Foundations of Applied Mathematics
Volume 2

Algorithms, Approximation, Optimization

Foundations of Applied Mathematics
Volume 2

Algorithms, Approximation, Optimization

JEFFREY HUMPHERYS
UNIVERSITY OF UTAH

TYLER J. JARVIS
BRIGHAM YOUNG UNIVERSITY

SOCIETY FOR INDUSTRIAL
AND APPLIED MATHEMATICS

PHILADELPHIA

10 9 8 7 6 5 4 3 2 1

PUBLICATIONS DIRECTOR	Kivmars H. Bowling
EXECUTIVE EDITOR	Elizabeth Greenspan
DEVELOPMENTAL EDITOR	Mellisa Pascale
MANAGING EDITOR	Kelly Thomas
PRODUCTION EDITOR	Louis R. Primus
COPY EDITOR	Susan Fleshman
PRODUCTION MANAGER	Donna Witzleben
PRODUCTION COORDINATOR	Cally Shrader
COMPOSITOR	Cheryl Hufnagle
GRAPHIC DESIGNER	Doug Smock
COVER DESIGNER	Sarah Kay Miller

Library of Congress Cataloging-in-Publication Data

Names: Humpherys, Jeffrey, author. | Jarvis, Tyler Jamison, author.
Title: Foundations of applied mathematics / Jeffrey Humpherys, Univerisity of Utah / Tyler J. Jarvis, Brigham Young University.
Description: Philadelphia : Society for Industrial and Applied Mathematics, [2017]- | Series: Other titles in applied mathematics ; 152 | Includes bibliographical references and index.
Identifiers: LCCN 2017012783 | ISBN 9781611974898 (v. 1)
Subjects: LCSH: Calculus. | Mathematical analysis. | Matrices.
Classification: LCC QA303.2 .H86 2017 | DDC 515--dc23 LC record available at *https://lccn.loc.gov/2017012783*

Contents

List of Notation

$\dagger$	indicates harder exercises xvi
$*$	indicates optional material xvi
$\oplus_\perp$	orthogonal direct sum 371
$\circledast$	floating-point version of the operation $*$ 487
$\oplus$	floating-point addition 487
$\ominus$	floating-point subtraction 487
$\otimes$	floating-point multiplication 487
$\oslash$	floating-point division 487
$\odot$	Hadamard product 358
$\sim$	asymptotic to 14, 96
$\equiv \pmod{n}$	congruent modulo n 56
$\subset$	is a subset of (not necessarily a proper subset)
$\leftarrow$	update value 703
$\rightarrow$	converges to 9, 706
> 0	(for matrices) positive definite 523, 524
$\succeq$	entrywise inequality 586
$\gg$	much greater than 511
$\langle \cdot, \cdot \rangle_n$	discrete inner product 347
$^\perp$	orthogonal complement 370, 371
$\times$	Cartesian product 44
$*$	convolution operator 356
$\mid$	divides 51
$\lceil \cdot \rceil$	ceiling: the least succeeding integer 12
$\lvert \cdot \rvert$	absolute value or modulus of a number; entrywise modulus of a matrix 326
$\lvert \cdot \rvert$	cardinality of a set
$[\![\cdot]\!]$	an equivalence class 56
2^S	power set of S 186
Γ_f	graph of f 473
$\Gamma(x)$	gamma function 87
Δ	difference operator 20
δ_{ij}	Kronecker delta 332, 357
∂E	boundary of the set E 580

$\varepsilon_{\text{machine}}$	machine epsilon or unit round-off 487
κ	relative condition number 492
$\hat{\kappa}$	absolute condition number 492
$\kappa(A)$	matrix condition number of A 496
$\sum_{k=1}^{n} x_k$	summation $x_1 + \cdots + x_n$ 19
Φ_n	sampling operator 346
ϕ_X	characteristic function of the random variable X 264
$(\Omega, \mathscr{F}, P)$	probability space 187
$\binom{a}{b_1,\ldots,b_k}$	multinomial coefficient 47
$\operatorname{argmax}_x f(x)$	value of x that maximizes f 520
$\operatorname{argmin}_x f(x)$	value of x that minimizes f 520
B_j^n	jth Bernstein polynomial of degree n 406
$\mathbb{C}$	complex numbers
$C(n,r)$	number of (unordered) combinations of r elements from a set of n elements, also denoted $\binom{n}{r}$ 45
$C^n(X;Y)$	space of Y-valued functions whose nth derivative is continuous on X 467, 476, 477, 523
$\operatorname{conv}(S)$	convex hull of S 576
$Df(\mathbf{x})$	total derivative of f at $\mathbf{x}$ 467
$Df(\mathbf{x})^{\mathsf{T}}$	gradient of $f : \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x}$ 467
$D^2 f(\mathbf{x})$	Hessian (second derivative) of f at $\mathbf{x}$ 476, 477
$D_i f(\mathbf{x})$	ith partial derivative of f at x 466
$D_{\mathbf{v}} f(\mathbf{x})$	directional derivative of f at $\mathbf{x}$ in the direction $\mathbf{v}$ 465
E°	set of interior points of E 698
E^c	complement $\Omega \setminus E$ of E 187
$\mathbb{E}[X]$	expected value of the random variable X 207
$\mathbf{e}_i$	ith standard basis vector 349, 375
effd	effective domain of a function to $\mathbb{R}_{\pm\infty}$ 668
$\mathbb{F}$	a field, always either $\mathbb{C}$ or $\mathbb{R}$ 19
$\mathbf{F}$	floating-point numbers 486
$\mathbb{F}[x]$	space of polynomials with coefficients in $\mathbb{F}$ 406
$\mathbb{F}[x;n]$	space of polynomials with coefficients in $\mathbb{F}$ of degree at most n 406, 424, 513
$f^{-1}(U)$	preimage $\{\mathbf{x} \mid f(\mathbf{x}) \in U\}$ of f 578
$\operatorname{fl}(x)$	floating-point representative of x 486
gcd	greatest common divisor 51
$\Im(z)$	imaginary part of z 326, 507
$\mathscr{K}_k(A, \mathbf{b})$	kth Krylov subspace of A generated by $\mathbf{b}$ 563
$k^{\overline{m}}$	rising Pochhammer symbol 39
$k^{\underline{m}}$	falling Pochhammer symbol 39
L	number of bits to encode a convex optimization problem 607
$L^p([a,b];\mathbb{F})$	space of p-integrable functions 332, 338
$\ell(\mathbf{a}, \mathbf{b})$	line segment from $\mathbf{a}$ to $\mathbf{b}$ 576

$\mathbb{N}$	natural numbers $\{0, 1, 2, \dots\}$, including 0 21, 50
$O(f)$	big-O of f 9
$o(f)$	little-o of f 9
$P(E\|F)$	probability of E given F 191
$P(E, F)$	probability of E and F; same as $P(E \cap F)$ 187
$P(n, r)$	number of permutations of r elements from a set of n elements 45
$\text{proj}_C(\mathbf{v})$	projection of $\mathbf{v}$ onto the convex set C 581
$\text{proj}_{\mathbf{x}}(\mathbf{v})$	orthogonal projection of $\mathbf{v}$ onto span($\mathbf{x}$) 333
$\mathbb{R}$	real numbers
$\mathbb{R}_{\pm\infty}$	real numbers extended by $\pm\infty$ 668
$\Re(z)$	real part of z 326
$\text{sign}(z)$	complex sign $z/\|z\|$ of z 329
$T[f]$	translation operator applied to f 38
W_n	nth Wallis integral 89, 94
$\mathbb{Z}$	integers $\{\dots, -2, -1, 0, 1, 2, \dots\}$
$\mathbb{Z}^+$	positive integers $\{1, 2, \dots\}$ 21
$\mathbb{Z}_n$	set of equivalence classes in $\mathbb{Z}$ modulo n 56

Preface

Overview

Why Algorithms, Approximation, and Optimization?

Moore's Law[1] has given us a half century of persistent and rapid innovation, where technological devices have been continually getting smaller, cheaper, and faster. As a result, our world today is stocked with computers. Not just desktops, laptops, cell phones, and tablets, but also digital health and wearable devices, video game consoles, smart appliances, and countless embedded systems.

Taking a broad view, we see a computer as a machine or network of machines that executes instructions in a systematic way to process and communicate information. When organized formally, these instructions take the of form of *algorithms* that are encoded into hardware or software. The process of executing algorithms is called *computing*.

Beyond the many and assorted electronic computing devices, algorithms are also found in biological systems. For example, the instructions for cell creation and reproduction are genetically encoded in DNA, which gets transcribed into RNA and then translated into rules for producing proteins—the building blocks of cells. These transcription and translation processes are also a type of computing.

Algorithms are also found in collective behavior. Sports teams call and execute plays with instructions so that every player knows what to do and how to adapt to varying circumstances on the field. Honey bees communicate the location of nectar to other bees through a waggle dance that encodes and transmits a recommended flight plan. In financial markets, a market maker on a trading floor clears trades and continuously reports prices for a variety of goods and securities being traded through a type of auctioning process. Wandering ants leave trails of home-finding pheromones when foraging so they can return the way they came, and when bringing back food to the nest, they leave food-finding pheromones to communicate to other ants where the food source is. As more ants follow the trail, they contribute to an increasingly stronger scent, which results in large self-organized trails of ants

[1]This refers to the observation that the number of transistors in computer chips has doubled approximately every two years for several decades. Although the doubling rate has slowed in recent years, there is still persistent growth in available computing power that is said to "extend" Moore's law in practical terms.

devouring the food source. In all of these cases, the collective behavior is really just *collective computing* of algorithms.

In each of the examples above, information is processed and communicated in a systematic way that can be codified into an algorithm. This broad and holistic view of algorithms encompasses electronic computing devices, biological systems, and collective behavior under a common umbrella of *computational science* and allows us to use the tools of mathematical and statistical analysis to explore the performance, complexity, and accuracy of algorithms. Algorithms are the focus of the first part of this book.

An important part of studying and using algorithms is the recognition that the world is too complex to be represented exactly. And most problems are too complex to be solved exactly. The way we make sense of the world is through imperfect representations, that is, through *approximation.* Every representation we make is an approximation that encodes some information without encoding all information. Knowing what information to keep and what to lose is essential to making an approximation useful.

For example, a map of a city is a very rough approximation of the reality it represents, containing only the essential information about locations and spatial relationships for key landmarks. But the very fact that it does not contain all the details of reality is what makes it useful. When I am lost, a map allows me to quickly identify where to go, whereas the full reality of all the buildings, streets, cars, people, noise, and lights can actually overwhelm me with unnecessary information and interfere with my ability to navigate. In this case the approximation is much more suitable for computing than a perfect representation of reality would be. The imperfection of the approximation is part of what allows it to be useful. As Leonard Cohen sings in "Anthem," "There is a crack in everything. That's how the light gets in." The second part of this book is focused on approximation and on using powerful mathematical tools for constructing, analyzing, and evaluating approximations.

Finally, the end goal of all our computing and approximating is to make the world better. Whether we want things to be faster, stronger, cheaper, smarter, easier, healthier, or kinder, we are perpetually engaged in the process of *optimization.* Nearly every problem in the world can be formulated as an optimization problem, so algorithms for optimization are almost universal in their applicability.

Most optimization algorithms are *iterative* in nature. This means that they start with an initial guess (or approximate solution) and compute incremental improvements with each iteration giving increasingly more accurate approximate solutions, repeating, again and again, until the solution is close enough that it is essentially indistinguishable from the exact solution. Thus, optimization requires a solid understanding of approximation and, of course, algorithms.

These three topics of algorithms, approximation, and optimization form the core of modern computational science, giving us a wide-angle lens to lift our attention beyond the latest devices and platforms. And computational science allows us to peer beyond the jargon-filled barriers of various disciplines and expose the fundamental ideas uniting and driving the world of science and technology. Our world is one of algorithms, approximation, and optimization.

To the Instructor

About This Text

This text gives a modern approach to computational science, by which we mean the fundamental mathematical ideas and tools of computing. The three main topics of algorithms, approximation, and optimization form the core.

The intent of this text and the associated computer labs is to attract students into the mathematical sciences and retain them by modernizing the curriculum and connecting theory to application in a way that makes them want to understand the theory, rather than just tolerate it. In short, a major goal of this text is to entice them to hunger for more.

The content in this volume could be reasonably described as upper-division undergraduate or first-year graduate-level mathematics. The mathematical prerequisites are vector calculus and linear algebra. The computational prerequisite is the equivalent of at least one semester of computer programming. Most of our students also have had a semester of undergraduate-level, single-variable real analysis as well.[2] However, mastery of the details of the undergraduate analysis class is less important than the mathematical maturity and mental discipline that comes from a rigorous study of analysis.

This volume can be taught as a stand-alone, two-semester sequence for advanced undergraduates or beginning graduate students. But it can also be part of a larger curriculum in applied and computational mathematics (for example, as currently used at Brigham Young University), taught in conjunction with the first volume of this series, *Foundations of Applied Mathematics: Volume* 1*, Mathematical Analysis* [HJE17], as two parallel, year-long courses.

There is a supplementary computer lab manual, containing over 25 computer labs to support this text. This text focuses more on the theory, while the labs cover application and computation. Although we recommend that the manual be used in a computer lab setting with a teaching assistant, it can be used without instruction. The concepts are developed thoroughly, with numerous examples and figures as pedagogical breadcrumbs, so that students can learn this material on their own, verifying their progress along the way. The labs and other classroom resources are open content and are available at

`https://bookstore.siam.org/ot166/bonus`.

Teaching from the Text

In our courses we teach each section in a 50-minute-long lecture. We require students to read the section carefully before each class so that class time can focus on the parts they find most confusing, rather than on just repeating to them the material already written in the book.

There are roughly five to seven exercises per section. We believe that students can realistically be expected to do all of the exercises in the text, but some are difficult and will require time, effort, and perhaps an occasional hint. Exercises

[2]Specifically, we assume the reader has had exposure to a rigorous treatment of continuity, convergence, differentiation, and Riemann integration in one dimension, as covered, for example, in [Abb15].

that are unusually hard are marked with the symbol †. Some of the exercises are marked with * to indicate that they cover advanced material. Although these are valuable, they are not essential for understanding the rest of the text, so they may safely be skipped, if necessary. Exercises not marked with * should generally not be skipped.

Throughout this book the exercises, examples, and concepts are tightly integrated and build upon each other in a way that reinforces previous ideas and prepares students for upcoming ideas. We find this helps students better retain and understand the concepts learned and helps achieve greater depth. Students are encouraged to do *all* of the exercises, as they reinforce new ideas and also revisit the core ideas taught earlier in the text.

Courses Taught from This Book

Full Year-Long Sequence

At BYU we teach a year-long advanced undergraduate-level course from this book, proceeding straight through the book, skipping only the sections marked with *. But this would also make a very good course at the beginning graduate level as well. Graduate students who are well prepared could be further challenged by covering advanced sections (marked with *) along the way.

One Semester: Algorithms with an Option of Approximation

The first seven chapters of the book make a good one-semester course on algorithms, including probabilistic algorithms.

As an alternative to the full algorithms course, Chapters 1–4 with 8–9 give a good one-semester course on classical algorithms and approximation without probability. This could be supplemented, as time permits, with some of the fundamentals of numerical computation from Chapter 11.

One Semester: Theory of Optimization

Chapters 11–17 (with a review of Chapter 10, as necessary, for those who are rusty on multivariate differentiation) form a good one-semester course on optimization. We have taught this course several times in various settings.

Advanced Sections

Some problems and sections are marked with the symbol * to indicate that they cover more advanced topics. Although this material is valuable, it is not essential for understanding the rest of the text, so it may safely be skipped, if necessary.

Instructors New to the Material

We've taken a tactical approach that combines professional development for faculty with instruction for the students. Specifically, the class instruction is where the theory lies and supporting media (labs, etc.) are provided so that faculty need not be computer experts nor be familiar with the applications in order to run the course.

Professors can teach the theoretical material in the text and use teaching assistants, who may be better versed in the latest technology, to cover the applications and computation in the labs, where the "hands-on" part of the course takes place. In this way professors can gradually become acquainted with the applications and technology over time, by working through the labs on their own time, without the pressures of staying ahead of the students. This approach has worked well for faculty at BYU who were unfamiliar with the material before they were assigned to teach from this book.

A more technologically experienced applied mathematician could flip the class if they wanted to, or change it in other ways. But we feel the current format is most versatile and allows instructors of all backgrounds to gracefully learn and adapt to the program. Over time, instructors will become familiar enough with the content that they can experiment with various pedagogical approaches and make the program theirs.

To the Student

Each section of the book has several exercises, all collected at the end of each chapter. Horizontal lines separate the exercises for each section from the exercises for the other sections. We have carefully selected these exercises. You should work them all (but your instructor may choose to let you skip some of the advanced exercises marked with *)—each is important for your ability to understand subsequent material.

Although the exercises are gathered together at the end of the chapter, we strongly recommend that you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter. Learning mathematics is like developing physical strength. It is much easier to improve, and improvement is greater, when exercises are done daily, in measured amounts, rather than doing long, intense bouts of exercise separated by long rests.

Origins

This curriculum evolved as an outgrowth of lecture notes and computer labs that were developed for a six-credit summer course in computational mathematics and statistics. This was designed to introduce groups of undergraduate researchers to a number of core concepts in mathematics, statistics, and computation as part of a National Science Foundation (NSF) funded mentoring program called CSUMS: Computational Science Training for Undergraduates in the Mathematical Sciences.

This NSF program sought out new undergraduate mentoring models in the mathematical sciences, with particular attention paid to computational science training through genuine research experiences. Our answer was the Interdisciplinary Mentoring Program in Analysis, Computation, and Theory (IMPACT), which took cohorts of mathematics and statistics undergrads and inserted them into an intense summer "boot camp" program designed to prepare them for interdisciplinary research during the school year. This effort required a great deal of experimentation, and when the dust finally settled, the list of topics that we wanted to teach blossomed into eight semesters of material—essentially an entire curriculum.

After we explained the boot camp concept to one visitor, he quipped, "It's the minimum number of instructions needed to create an applied mathematician." Our goal, however, is much broader than this. We don't want to train or create a specific type of applied mathematician; we want a curriculum that supports all types, simultaneously. In other words, our goal is to take in students with diverse and evolving interests and backgrounds and provide them with a common corpus of mathematical, statistical, and computational content so that they can emerge well prepared to work in *their own* chosen areas of specialization. We also want to draw their attention to the core ideas that are ubiquitous across various applications so that they can navigate fluidly across fields.

Python and Pseudocode

Throughout the book we give examples of algorithms. We generally use Python instead of pseudocode because it gives a certain degree of precision that pseudocode lacks, because it is useful for students to learn, and because it reads a lot like most pseudocode anyway. Most of the Python syntax we use should be clear to someone who has learned another programming language. When unusual syntax is used, we give a brief explanation.

Acknowledgments

We thank the National Science Foundation for their support through TUES grant DUE-1323785. We especially thank Ron Buckmire at the NSF for taking a chance on us and providing much-needed advice and guidance along the way. Without the NSF, this book would not have been possible. We also thank the Department of Mathematics at Brigham Young University for their generous support and for providing a stimulating environment in which to work.

Many colleagues and friends have helped shape the ideas that led to this text, especially Randy Beard, Rick Evans, Shane Reese, Dennis Tolley, and Sean Warnick, as well as Bryant Angelos, Jonathan Baker, Blake Barker, Mylan Cook, Casey Dougal, Abe Frandsen, Ryan Grout, McKay Heasley, Amelia Henricksen, Ian Henricksen, Rebecca Jones, Brent Kerby, Steven Lutz, Shane McQuarrie, Ryan Murray, Spencer Patty, Jared Webb, Matthew Webb, Jeremy West, and Alexander Zaitzeff, who were all instrumental in helping to organize this material.

We also thank the students of the BYU Applied and Computational Mathematics Emphasis (ACME) cohorts who suffered through our mistakes, corrected many errors, and never hesitated to tell us what they thought of our work.

We are deeply grateful to Emily Evans, Chris Grant, Paul Jenkins, Rachel Jenkins, and Robbie Snellman, who read various drafts of this volume very carefully, corrected many errors, and gave us a tremendous amount of helpful feedback. Of course, all remaining errors are entirely our fault. We thank Michael Hansen and Sierra Horst for their help with illustrations and Sarah Kay Miller for her outstanding graphic design work, including her beautifully designed covers. We also appreciate the patience, support, and expert editorial work of Elizabeth Greenspan and the other editors and staff at SIAM.

Finally, we thank the folks at Savvysherpa, Inc., for corporate sponsorship that greatly helped make the transition from IMPACT to ACME and for their help nourishing and strengthening the ACME development team.

Part I

Algorithms

1 Introduction to Algorithms and Analysis

The fundamental law of computer science: As machines become more powerful, the efficiency of algorithms grows more important, not less.
—Nick Trefethen

Before the advent of the modern computer, many mathematicians focused on finding closed-form solutions to highly idealized problems arising from such fields as classical mechanics, electromagnetism, quantum theory, thermodynamics, and fluid dynamics. Today, university libraries are still littered with dusty old volumes of encyclopedic texts of special functions and general solutions to these kinds of problems.

Over the last several decades, it has become increasingly clear that most of the important problems of modern science, technology, and even mathematics have no hope of a closed-form solution. In some cases this is because real-world problems are too messy or complex, but there are surprisingly many problems that are simple to state and yet it has been proved that no closed-form solution can exist.

As an example, recall that the quadratic equation

$$ax^2 + bx + c = 0$$

has the closed-form solution

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{1.1}$$

Similar, but more complicated, formulas exist for the cubic and quartic equations; however, Abel's theorem guarantees that no general algebraic solution exists for the quintic or for any higher-order polynomial equations [Art91, Theorem 9.9]. Thus, for example, there is no closed-form algebraic solution for solving the equation

$$x^5 + 2x^4 - x^3 - 3x^2 + x - 6 = 0, \tag{1.2}$$

but the intermediate value theorem easily shows that a root exists in the interval $[0, 2]$ because the polynomial gives a negative value when evaluated at $x = 0$ and a positive value when evaluated at $x = 2$.

Newton's method is an *algorithm* that accurately approximates a zero of a smooth function. It is an iterative procedure that, under fairly general conditions, converges to the zero and is terminated when the desired accuracy is met. Given an estimate x_n of the zero, the algorithm returns a new estimate x_{n+1} given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1.3}$$

With $f(x)$ as the left-hand side of (1.2) and using the initial guess $x_0 = 1.0$, the algorithm (1.3) produces the following sequence, computed to 15 digits of accuracy:

$$\begin{aligned}
x_0 &= 1.000000000000000,\\
x_1 &= 2.200000000000000,\\
x_2 &= 1.804654426169757,\\
x_3 &= 1.549707343960059,\\
x_4 &= 1.431481800966775,\\
x_5 &= 1.406763770052249,\\
x_6 &= 1.405779606478647,\\
x_7 &= 1.405778093756038,\\
x_8 &= 1.405778093752469,\\
x_9 &= 1.405778093752469.
\end{aligned}$$

Notice that the ninth iterate is the same as the eighth because the difference between the two is smaller than the 15 decimals of accuracy provided. This gives us a natural stopping rule, since we cannot improve the approximate solution without first increasing the decimal length. In other words, to 15 decimal places, x_8 is the best approximate solution of this zero of the polynomial.

The point of this example is to demonstrate that there are situations where an iterative algorithm can provide an arbitrarily close approximation to the solution of a problem even when there is no formula or closed-form expression. This situation is actually very common; thus we should adjust our thinking to accept an algorithm, even an iterative algorithm such as (1.3), as a "solution" to a problem.

What Is an Algorithm?

An *algorithm* is an unambiguous set of instructions for solving a problem or accomplishing a task. The set of rules taught to elementary school children for adding two integers is an algorithm, as is Newton's method for finding zeros of a function, as described above.

There are often many different algorithms for accomplishing the same task. For example, to compute the value of the polynomial x^2+3x+5 at a given point x, one naïve algorithm is to compute x^2 and then compute $3x$ and then sum the results and add 5. A faster algorithm that accomplishes the same task is to compute $3+x$, multiply that by x, and add 5. The result[3] is the same, because $(x + 3)x + 5 = x^2 + 3x + 5$, but the algorithms are different.

[3]This other algorithm is called *Horner's method.* In this case Horner's method requires only three arithmetic operations, while the naïve method requires four.

Any closed-form solution to a problem defines an algorithm in the sense that a formula gives a sequence of operations. For example, the quadratic formula (1.1) can be interpreted as the algorithm: compute b^2, subtract $4ac$, take both the positive and negative square roots, subtract b from each, and divide the results by $2a$. This illustrates a fundamental, but often underappreciated, idea of modern mathematics:

An algorithm is the natural extension of a formula.

Closed-form solutions are rare, since the limited vocabulary of polynomial and basic transcendental functions is woefully inadequate to reasonably describe the many functional relationships that exist in the world. Since so few problems have closed-form solutions, we suggest that the notion of a "solution" should be generalized to include more general algorithms, including iterative procedures, like Newton's method.

There is a caveat, however. For us to accept an algorithm as a solution, we must analyze it and prove that it will return the correct solution to the problem at hand. We may also want to know that the algorithm is computationally feasible, given the resources available. To prove such things, one must access the arsenal of mathematical analysis and leverage the theory of *algorithms, approximation, and optimization.* That is what this text is about.

What Do We Want from an Algorithm?

When adopting this algorithmic view, we should not consider a problem to be "solved" until we can rigorously demonstrate that the algorithm is both correct and feasible to employ. If an approximate solution isn't sufficiently close to the exact solution, or if the resources required to execute the algorithm are too great, then the algorithm is of little use and a better algorithm is needed. Until a better algorithm is found, we should think of the problem as unsolved.

Moreover, we don't just want algorithms that work—we want the best algorithms. Speaking broadly, the performance of an algorithm is usually characterized in terms of its accuracy and its efficiency. We want algorithms that are both accurate and efficient.

Accuracy

A high-quality algorithm should give a good approximation to the correct answer. Although solutions to some problems can be computed exactly, many cannot. Many important and useful algorithms instead compute *numerical* approximations, by which we mean finite-precision approximations (such as 0.666667 as an approximation for $\frac{2}{3}$ or 3.14159 as an approximation of π). Finite-precision arithmetic introduces small errors, called *round-off errors*, into almost every step of every computation. In some algorithms, these errors can compound into large, catastrophic errors in the final results. Such algorithms are said to be *numerically unstable.* We discuss floating-point arithmetic and stability in Chapter 11. A high-quality algorithm should be resistant to such errors and consistently give answers that are sufficiently close to the correct answer.

Example 1.0.1. One can show by hand that

$$450 \times 4054.5^4 - 50 \times 7022.6^4 + 7022.6^2 = 0.005.$$

But executing this on most computers gives a result of -33.24. This is an example of catastrophic round-off error. This is discussed in more depth in Section 11.3.

Efficiency

The cost of an algorithm could be measured in many ways, such as money, time, computer memory, labor, etc. We are generally most interested in how long an algorithm takes to run and its memory requirements. An algorithm is of no use if it takes too long to run or requires more memory than we have available. How long the algorithm takes to run depends on the specific computer being used, so instead of talking about run time, we often use as a proxy the number of primitive operations that must be executed. We call this the *temporal complexity*, and we call the amount of memory required the *spatial complexity*. Temporal and spatial complexity are discussed in more detail in Section 1.1 and are a major theme of this and the next several chapters.

1.1 Complexity

The *complexity* of a given algorithm is a measure of the resources required for it to execute. This could refer to execution time, memory requirements, the cost in dollars to pay for the equipment, the time or cost of programming labor necessary to develop the algorithms into software, or even the amount of electricity required. Generally speaking, however, complexity focuses on two main issues: the number of primitive operations required and the amount of memory required.

We define *primitive operations*[4] to be basic operations such as assigning a value to a variable (like `x = 5`), basic integer arithmetic (like `x+y`, but not `log(x)` or `cos(x)`), comparisons (like `x<y`) and basic Boolean operations (like `and`, and `or`), looking up an indexed value in an array (like `A[3]`), calling a method (like `myFunc(x,y)`),[5] returning from a method, and so forth. We treat accessing the values of scalar variables as having no cost (so if `x=5` and `y=6`, then computing `x+y` costs only one operation—the addition). We assume all primitive operations take approximately the same time to execute. Of course this is not true, but it gives a useful approximation.

[4] Our definition of primitive operations here is similar but not identical to some of the other standard models, such as the RAM model of [CLRS01, Section 2.2]. Our goal here is primarily pedagogical rather than computing the precise run time of an algorithm on a specific machine, so we don't want to get too bogged down in technical details.

[5] Note that this does not include all the primitive operations required by the method itself—just the cost of finding the method in memory and starting to execute it.

We define the *temporal complexity* of an algorithm to be the number of primitive operations needed for it to execute. This is a proxy for the execution time, but since execution time varies so much across platforms and hardware specifications, it is generally preferable to use this definition. We define *spatial complexity* to be the amount of memory required to execute the algorithm.

Example 1.1.1. If `L` is a list, then the number of primitive operations involved in the command `a = L[5] + 7` is three, corresponding to one lookup `L[5]`, one addition, and one assignment.

Example 1.1.2. Algorithm 1.1 is a simple implementation of a method for finding the largest element in a list `L` of integers. Because of the `if`-statement on Line 12, we cannot determine the exact number of primitive operations this algorithm will use without knowing more about the input list `L`, but we can compute the best- and worst-case complexity.

Setting the initial value of `max_val` on Line 7 requires two primitive operations, namely one lookup `L[0]` and one assignment. Line 8 involves computing `len(L)`, which we assume is one primitive operation, and one assignment. Initializing the counter `i` on Line 9 is one more primitive operation.

The `while`-statement on Line 11, involves just one comparison. But this comparison happens `n` times (the condition is true $n-1$ times and fails once, when $i = n$). Thus Line 11 contributes `n` primitive operations.

Inside the loop, we have an `if`-statement on Line 12 involving one lookup and one comparison, and if the comparison is true, then Line 13 contributes two more primitive operations (a lookup and an assignment). Thus if the conditional is true, the `if`-statement contributes four primitive operations, and otherwise it contributes two.

Finally, the incrementation of `i` on Line 14 requires two more operations (an addition `i + 1` and an assignment). Thus, the loop consists of either six or four operations, repeated $n-1$ times, for a total of $6n-6$ (worst case) or $4n-4$ (best case) from the loop.

The total number of operations, therefore, has temporal complexity of either $5+n+6n-6 = 7n-1$ (worst) or $5+n+4n-4 = 5n+1$ (best).

Note that spatial complexity influences the execution time, since it takes time to move data into and out of the CPU registers, various memory caches, random access memory (RAM), hard-disk space, and memory on other computers and storage devices that are accessed over an internet connection or through some communication port. To truly represent execution time, we would need to factor in the particular hardware and operating system specifications and understand how memory is managed by the system. If the CPU is sitting idle because it is waiting for a hard disk to retrieve a value, then time is being consumed even though no additional

```python
def find_max(L):
    """
    Find the largest element in a list L of integers.
    """

    # Set initial values
    max_val = L[0] # Max value.
    n = len(L)     # List length.
    i = 1          # Counter to iterate through L.

    while i < n:
        if L[i] > max_val:
            max_val = L[i]
        i += 1     # Increment i.

    return max_val
```

Algorithm 1.1. *Python implementation of a routine for finding the largest value in a list* `L` *of integers, as discussed in Example 1.1.2.*

operations are being executed. Despite this (or because of it), we focus our study on the temporal and spatial complexity as defined above.

Remark 1.1.3. It is important to recognize that temporal complexity can often be traded for spatial complexity and vice versa; that is, we can often make an algorithm use fewer primitive operations by having it store more values in memory. This reduces the temporal complexity but increases the spatial complexity of the algorithm.

1.1.1 Big-O and Little-o Notation

Since many algorithms are too intricate to easily account for the exact number of primitive operations or memory requirements, our primary interest is to give a reasonably sharp upper bound on how the various complexities increase as a function of the size of the inputs. This upper bound is what we need to understand how a given algorithm scales temporally and spatially as the size of the inputs grows.

As a simple example, consider the usual grade school algorithms for arithmetic. The standard long-addition algorithm (see Algorithm 1.2) has the property that when the number of digits to be added is doubled, the memory required also doubles, as does the number of primitive operations required. In contrast, doubling the number of input digits in a multiplication problem (see Algorithm 1.5) quadruples the number of primitive operations required. For small problems this quadrupling is not a big deal, but as the length of the inputs increases, it can quickly become significant.

It is often useful to think about the *asymptotic* growth of the temporal and spatial complexity, that is, how fast the time and space requirements of a problem

grow as the size of the inputs grow. We typically quantify asymptotic growth with *big-O* and *little-o* notation.

Definition 1.1.4. *Let f and g be real-valued functions on either the positive real numbers or the positive integers. We say that $f(x)$ is* big-O *of $g(x)$ as $x \to \infty$, denoted $f(x) \in O(g(x))$, if there exist $M > 0$ and $N > 0$ such that $|f(x)| \leq M|g(x)|$ whenever $x \geq N$. Similarly, we say that $f(x)$ is* little-o *of $g(x)$ as $x \to \infty$, denoted $f(x) \in o(g(x))$, if for each $\varepsilon > 0$ there exists $N > 0$ such that $|f(x)| \leq \varepsilon|g(x)|$ whenever $x \geq N$.*

Remark 1.1.5. When analyzing algorithms with discrete inputs, we typically use $f(n)$ instead of $f(x)$ to denote the discrete nature of the function.

Example 1.1.6. If the complexity of an algorithm is $T(n) = 3n^2 + 2n + 100$, where n is the size of the input, then

$$T(n) = 3n^2 + 2n + 100 \leq 3.3n^2$$

whenever $n \geq 22$. Thus $T(n) \in O(n^2)$. There's nothing special about 3.3. In fact, given any $\varepsilon > 0$ there exists an $n \geq N$ so that $T(n) \leq (3+\varepsilon)n^2$ whenever $n \geq N$. The smaller the ε, the sharper the bound, but the big-O rate of T is $O(n^2)$ regardless of the choice of ε.

Unexample 1.1.7. In the previous example $T(n) \notin O(n)$ because for any $M > 0$ we have

$$T(n) = 3n^2 + 2n + 100 > Mn$$

whenever $n > \frac{M}{3}$.

Example 1.1.8. Since the total number of primitive operations needed for Algorithm 1.1 is at most $7n - 1$ (see Example 1.1.2), the temporal complexity of this algorithm is $O(n)$.

Remark 1.1.9. Many computer science texts use the convention $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$, but this is a problematic abuse of notation. For example, it would imply that $O(n) = O(n^2)$, but $O(n^2) \neq O(n)$. Therefore, we use set membership instead of the equal sign to signify membership in a class of functions. Hence, we write $O(n) \subset O(n^2)$ and $O(n^2) \not\subset O(n)$, denoting that the class of linear functions is properly contained in the class of quadratic functions.

Example 1.1.10. The leading coefficient of a polynomial doesn't affect the big-O or little-o rate. For example:

(i) Let $f(n) = an + b$. Note that

$$|f(n)| = |an + b| \leq (|a| + 1)n$$

whenever $n \geq |b|$, and therefore $f(n) \in O(n)$. Moreover, $f(n) \in o(n^{1.1})$ because for each $\varepsilon > 0$, once $n > (|a| + 1)/\varepsilon^{10}$ we have

$$|f(n)| = |an + b| \leq (|a| + 1)n < \varepsilon n^{1.1}.$$

This argument can be extended to show that $f(n) \in o(n^{1+\delta})$ for any $\delta > 0$.

(ii) Let $f(n) = an^2 + bn + c$. Note that

$$|f(n)| = |an^2 + bn + c| \leq (|a| + 2)n^2$$

whenever $n \geq \max\{|b|, \sqrt{|c|}\}$. Thus, $f(n) \in O(n^2)$. Also, for every $\varepsilon > 0$ as n gets large we also have $n > |(a + 2)/\varepsilon|^{10}$, at which point

$$|f(n)| = |an^2 + bn + c| \leq |(a + 2)|n^2 < \varepsilon\, n^{2.1},$$

so $f(n) \in o(n^{2.1})$. This can be extended to show that $f(n) \in o(n^{2+\delta})$ for any $\delta > 0$.

Example 1.1.11. It is straightforward to identify some additional properties of the big-O and little-o notation:

(i) Let $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$. Exercise 1.2 shows that $f(n) \in O(n^k)$.

(ii) If $f(n) = \sum_{k=1}^{n} k$, then $f(n) \in O(n^2)$ because

$$f(n) = \sum_{k=1}^{n} k \leq \sum_{k=1}^{n} n = n^2.$$

More generally, Exercise 1.4 shows that if $f(n) = \sum_{k=1}^{n} k^m$, then $f(n) \in O(n^{m+1})$.

(iii) If $f(x) \in O(g(x))$ as $x \to \infty$, then $f(x) \in o(x^{\delta} g(x))$ for every $\delta > 0$.

(iv) Clearly $f(x) \in o(g(x))$ implies $f(x) \in O(g(x))$, but the converse is false; that is, $o(g)$ is a proper subclass of $O(g)$.

Common examples of big-O and little-o notation include powers, exponentials, and logs.

Definition 1.1.12. *If a real-valued function f on $\mathbb{Z}^+$ (or $\mathbb{R}^+$) is in*

(i) $O(\log n)$, *then we say it is* logarithmic*;*

(ii) $o(n)$, *then we say it is* sublinear*;*

(iii) $O(n)$, *then we say it is* linear*;*

(iv) $O(n^2)$, *then we say it is* quadratic*;*

(v) $O(n^3)$, *then we say it is* cubic*;*

(vi) $O(n^c)$ *for some $c \in \mathbb{N}$, then we say it is* polynomial*;*

(vii) $O(c^n)$ *for some $c > 1$, then we say it is* exponential.

Proposition 1.1.13. *Let f and g be real-valued functions on either the positive real numbers or the positive integers. If there exists $M > 0$ such that*

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} \leq M,$$

then $f(x) \in O(g(x))$. Also, we have

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = 0$$

if and only if $f(x) \in o(g(x))$. Finally, if

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty,$$

then $f(x) \notin O(g(x))$.

Proof. The proof is Exercise 1.3. □

Example 1.1.14. Let $f(n) = an^2 + bn + c$ with $a > 0$. To see that $f \notin O(n)$, we observe that

$$\frac{f(n)}{n} = an + b + \frac{c}{n} \to \infty$$

as $n \to \infty$. Thus, for large n there is no M such that $f(n) \leq Mn$. But it is straightforward to verify that $f(n) \in O(n^2)$.

1.1.2 Example: Complexity of Long Addition

One of the first algorithms taught in grade school is the standard algorithm (long addition) for adding two multidigit integers. It iterates through each column, from right to left, adding the corresponding single-digit numbers together, carrying a 1 to the next column as necessary. Algorithm 1.2 is an implementation of this algorithm in Python. For illustrative purposes we treat each positive integer as a list of digits so that we can manage arbitrarily long numbers. We assume that the rightmost entry in each list represents the ones digit, the next the tens digit, and so on.

Assume the longer of the two lists has length n. The algorithm begins by identifying which of the lists is shorter and prepending zeros to it so both lists are the same length and the place value of each digit matches the corresponding digit in the other list. Prepending d elements onto a list of length k requires $O(d+k)$ operations: first initializing a new list with $d+k$ entries, and then copying all the $d+k$ elements into the new list. In our case, we have $d+k \leq n$, so this contributes $O(n)$ to the temporal complexity.

After some key variables are initialized (in constant time), the `while`-loop (Lines 22–26) adds each digit in the second list to the corresponding digit of the first and accounts for the carrying digit as necessary. The number of primitive operations inside the loop is independent of the lengths of the lists, and therefore the n iterations of the loop contribute $O(n)$ to the temporal complexity of the algorithm. The final step is to prepend the carried digit, if necessary, at the beginning of the list. This, too, costs at most $O(n)$ primitive operations. Therefore, the temporal complexity of the algorithm is $O(n)$.

The data that must be stored are the two inputs, each of length at most n, the output list, of length at most $n+1$ (in this algorithm the output is stored in the same list as one of the inputs), one constant-length variable `carry` and two other variables `i` and `delta`. The values of `i` and `delta` are no more than n, so the memory required to store those values is bounded by the number of digits it takes to represent n, that is, $\log_{10} n$ in decimal notation or $\log_2(n)$ in binary. In either case, each of these contributes at most $\log_2 n < n$ to the spatial complexity, and thus the overall spatial complexity is no more than a constant times n, that is, $O(n)$.

Remark 1.1.15. As discussed above, if $n \in \mathbb{N}$ is arbitrarily large, then storing an integer (like the counter `i`) that is bounded by n requires $\lceil \log_2(n) \rceil$ digits[6] and hence adds $O(\log n)$ to the spatial complexity. However, for calculating spatial complexity of algorithms, we almost always assume that loop counters, array indices, and other such integers have a fixed size. This is not an unreasonable simplification because standard 64-bit integers (signed) can be as large as $2^{63}-1$, which is not likely to be very restrictive.

Remark 1.1.16. If your language of choice has a built-in data type for arbitrarily long integers, the addition algorithm for that data type has probably been carefully optimized and, therefore, should be much more efficient than the algorithm

[6]The notation $\lceil q \rceil$ denotes the least integer greater than or equal to q, also known as the *ceiling* function.

```
1  def add(a, b):
2      """
3      Add two numbers, where each number is input as a list of
4      single-digit integers, e.g., [1,2,3] = 123.
5
6      Returns a list of single-digit integers.
7      """
8
9      # Prepend zeros to the shorter list to
10     #   align with the longer list.
11     delta = abs(len(a)-len(b))
12     if len(a) <= len(b):
13         a = delta * [0] + a
14     else:
15         b = delta * [0] + b
16
17     # Set initial values.
18     carry = 0
19     i = len(a) - 1
20
21     # Add each pair of digits from right to left
22     while i >= 0:
23         a[i] = a[i] + b[i] + carry
24         carry = a[i] // 10
25         a[i] = a[i] % 10
26         i -= 1              # decrement i by 1
27
28     # Prepend the final carry digit
29     if carry > 0:
30         a = [carry] + a
31
32     return a
```

Algorithm 1.2. *Routine for adding two positive integers of arbitrary size. Here the integers are represented as lists, where each entry is a single digit. For example,* `add([1, 2, 3], [4, 5, 6])` *returns* `[5,7,9]`*. Note that in Python, the addition operator concatenates strings (that is,* `[0] + [1,2,3]` *returns* `[0,1,2,3]`*) and multiplication is repeated addition (thus,* `3 * [0]` *produces* `[0,0,0]`*). Also* `a//b` *is the integer part of* `a` *divided by* `b` *and* `a%b` *is the remainder of* `a` *when divided by* `b`*. Finally, note that Python indexing starts at* 0*, so the index of the last element of list* `a` *is* `len(a)` $-$ `1` *(see Line* 19*).*

presented here. However, it will still have temporal complexity $O(n)$, since every one of the n digits must be added to find the correct sum. An algorithm with an input of length n can only have complexity less than $O(n)$ if some input data can be skipped.

1.2 Leading-Order Behavior

In many situations we want to know more than just the big-O growth rate—we also want to know the leading coefficient of the growth rate. There's a big difference between an algorithm that requires $3n^2$ primitive operations and one that requires $3000n^2$ primitive operations. In this section we begin by defining leading-order behavior and giving several examples of how to analyze an algorithm for its leading-order complexity.

1.2.1 Leading-Order Behavior

Definition 1.2.1. *Let f and g be real-valued functions defined on the positive real numbers or the positive integers. We say that $f(n)$ is* asymptotically equivalent to *$g(n)$, denoted $f(n) \sim g(n)$ as $n \to \infty$, if*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

Informally one often says, "f grows like g," or "f is g to leading order," to mean $f \sim g$ as $n \to \infty$. Sometimes we drop the $n \to \infty$ designation when it is clear from the context.

Example 1.2.2. The function $T(n)$ given in Example 1.1.6 satisfies $T(n) \sim 3n^2$ since

$$\frac{T(n)}{3n^2} = 1 + \frac{2}{3n} + \frac{100}{3n^2} \to 1$$

as $n \to \infty$.

Remark 1.2.3. It is straightforward to show that the relation $\sim$ is an equivalence relation; that is, it is reflexive, symmetric, and transitive (see Exercise 1.10). For more about equivalence relations, see Volume 1, Appendix A.1.2.

Example 1.2.4. Since the maximum number of operations needed for Algorithm 1.1 is $7n - 1$, the leading order of the temporal complexity of this algorithm is $\sim 7n$.

1.2.2 Merging and Sorting

In this section we discuss how to merge two sorted lists and evaluate the complexity of this algorithm to leading order. Then we use the merging algorithm as the basis of a very naïve and inefficient sorting algorithm. Despite the inefficiencies of this sorting algorithm, it is an instructive example. In Section 1.10 we construct an efficient sorting algorithm from a simple recursive variant of this algorithm.

```
1  def merge(K, L):
2      """Merge two sorted lists into a new sorted list.
3       For example, merge([1, 4], [2, 3]) returns [1, 2, 3, 4].
4      """
5
6      # Initialization
7      merged = [None]*(len(K) + len(L))  # Preallocate output list
8      i = 0; j = 0 # Pointers to track location in each list
9
10     # Iterate over the two lists, terminate when one is empty
11     while i < len(K) and j < len(L):
12         if K[i] <= L[j]:
13             merged[i+j] = K[i]
14             i += 1
15         else:
16             merged[i+j] = L[j]
17             j += 1
18
19     merged[i+j:] = K[i:] + L[j:] # One of these is empty
20     return merged
```

Algorithm 1.3. *Routine for merging two sorted lists of numbers together into a single sorted list. This algorithm fails if the lists K and L are not sorted. Note that* `K[i:]` *refers to the list* `[K[i],K[i+1],...]` *of the elements of* `K` *starting from the element indexed by* `i` *and proceeding to the end of* `K`.

Merging

Merging combines two already sorted input lists into a single sorted output list. Although we allow the length of the lists to be arbitrarily long, the entries in the lists are assumed to be of a fixed size, so each takes the same, fixed, amount of memory. The basic merging algorithm compares the leading entries in each input list and extracts the smaller of the two entries, placing it into the resulting output list. The process then repeats, extracting the smaller of the leading entries of what remains of the two input lists into the output list, one at a time, until both input lists are empty.

A Python implementation of this merging procedure is given in Algorithm 1.3. Rather than actually extracting the smallest entry at each step, this implementation simply maintains placeholders on each of the input lists to track the leading entries in each remaining sublist; this is done with the variables `i` and `j`. At some point the end of one of the lists is reached and we append the remainder of the other input list to the final list. This is done in Line 19. Rather than trying to decide which of the two remaining lists is empty, we append both to the final list, which works fine because one of them is empty.

To evaluate the spatial complexity of this algorithm, note that the only data that must be stored are the initial lists `L` and `K`, whose combined storage is assumed to be n numbers; the merged list `merged` of length n; and the counters `i` and `j`. The

counters have length no more than the number of digits required to represent `i` and `j`, respectively, and these are $O(\log n) \subset o(n)$. Thus the total spatial complexity is $\sim 2n \in O(n)$.

The temporal complexity is also $O(n)$. To analyze the algorithm to leading order, first consider the loop. In Line 11, the `while`-loop computes two list lengths, makes two comparisons, and performs a conjunction for each iteration. Since one of `i` and `j` is incremented each iteration, the maximum number of iterations for which the condition could hold is $n-1$, after which the condition must fail, which terminates the loop. Hence Line 11 could cost as many as $5n$ primitive operations. Line 12 has two lookups and a comparison. That's three primitive operations in the loop, which iterates up to $n-1$ times. When the conditional is successful, the loop executes Lines 13–14; otherwise it executes Lines 16–17. In either case the first line is a sum, a lookup, and an assignment, while the next line is an incrementation, which is a sum and an assignment. Thus, after the conditional on Line 12, there are $5(n-1) \sim 5n$ more primitive operations. Adding these to the $3(n-1) \sim 3n$ from Line 12, and the $5n$ from Line 11, gives $\sim 13n$.

The operations outside of the loop that depend on n are the initial construction of `merged`, which costs $n+1 \sim n$ primitive operations (initialize the list, and make n assignments), and putting the lists `K[i:]` and `L[j:]` into the end of `merged`, which takes at most n lookups and n assignments. Thus the total temporal complexity of this algorithm is $\sim 16n$.

Naïve Sorting (Insertion Sort)

Sorting rearranges the entries of a list to produce a list that is arranged in order from least to greatest. Using the merging algorithm, we can design a sorting algorithm whose temporal complexity is $O(n^2)$ when the original list has length n.

We start with a list `sorted_list` consisting of just the first element `L[0]` of the input list `L`. For each successive entry of `L`, make a new list of length one (which is trivially sorted) and merge the new single-element list with `sorted_list`. The algorithm repeats until it runs out of new entries to merge; see Algorithm 1.4. This is a slight modification of the algorithm often called *insertion sort*.

Since merging two lists of total length k has temporal complexity $\sim 16k$, the naïve sorting method has temporal complexity $O(1+2+\cdots+n) = O(n^2)$; see Example 1.1.11(ii). More careful analysis (using the standard summation formula (1.9)) shows that to leading order, this is

$$\sim 16(1+2+\cdots+n) = 16\frac{(n+1)(n)}{2} \sim 8n^2.$$

The spatial complexity of this sorting algorithm is $\sim 2n$ because we need only store the lists `L` and `sorted_list`, each of length n, and one additional integer `i` whose value is bounded by n (and hence whose size is $O(\log n) \subset o(n)$).

Nota Bene 1.2.5. This naïve sorting method is not a good algorithm. Other sorting algorithms are much faster—for example, the merge sort algorithm, which we discuss in Section 1.10, is $O(n \log n)$.

```
1  def naive_sort(L):
2      """Sort a nonempty list L.
3      """
4
5      # Initialize values
6      sorted_list = [L[0]]
7      i = 1
8      n = len(L)
9
10     # Merge in the rest, one at a time
11     while i < n:
12         sorted_list = merge(sorted_list, [L[i]])
13         i += i
14
15     return sorted_list
```

Algorithm 1.4. *A naïve routine for sorting a list* `L`*. The sorted list begins as a single element and then is successively merged with single-element lists until every element in* `L` *has been merged into the sorted list.*

1.2.3 Leading Order for Long Addition and Multiplication

Long Addition

To analyze the leading-order temporal complexity of the addition problem in Algorithm 1.2, assume again that the longest list has length n. First, the initial prepending of `delta` zeros onto the front of the shorter list requires defining a new empty list of length n (one operation), putting the new zeros at the front of that list (`delta` assignments) and putting the remaining elements of the old, shorter list into the remaining positions in the new list ($n -$ `delta` lookups and $n -$ `delta` assignments), for a total $2n + 1 -$ `delta` $\sim 2n$ primitive operations.

The loop contained in Lines 22–26 executes 14 operations for each iteration. Specifically, Line 22 has a conditional operation, and Line 23 performs two lookups, adds three numbers together (two operations), and makes an assignment (one operation). That's six operations. Line 24 requires one lookup, computing an integer part, and making an assignment. That's three operations. One lookup, computing the remainder, and making an assignment gives three operations on Line 25. Finally the decrement in Line 26 is a subtraction and an assignment and therefore two operations. Hence, to leading order, the loop is $\sim 14n$ primitive operations.

At the end of the algorithm, in the worst case, the carry is positive and the carry digit is prepended to the list. This amounts to one conditional (Line 29) and a prepend operation (Line 30). Prepending an element onto a list of length n requires defining a new list of length $n+1$ (one operation), putting the new element at the front of that list (one assignment), and then moving all elements of the old list into the new list (n lookups and n assignments). Therefore, Lines 29–30 require $2n+3 \sim 2n$ additional operations. Combining this with the initial $\sim 2n$ for padding the short list, and the $\sim 14n$ operations of the earlier loop, makes this algorithm $\sim 18n$ primitive operations.

Long Multiplication

Algorithm 1.5 (below) gives an algorithm for long multiplication that is similar to the one taught in grade school, except that it has a small efficiency built in. To analyze the leading-order behavior, we must count the operations in the double

```
1  def mult(a, b):
2      """
3      Multiply two numbers, where each number is input as a list
4      of single-digit integers, e.g., [1,2,3] = 123.
5
6      Returns a list of single-digit integers.
7      """
8
9      # Set initial values
10     tens_shift = 0
11     product = []
12     j = len(b)-1
13
14     # Iterate over digits of b from right to left
15     while j >= 0:
16         sumstep = [None]*len(a) #preallocate list of len a
17         carry = 0
18
19         # Iterate over digits of a from right to left
20         i = len(a) - 1
21         while i >= 0:
22             temp = a[i] * b[j] + carry
23             sumstep[i] = temp % 10
24             carry = temp // 10
25             i -= 1
26         if carry > 0:
27             sumstep = [carry] + sumstep
28
29         # Shift sumstep by tens_shift places and add to
30         # final product, using the previous algorithm
31         product = add(product, sumstep + ([0]*tens_shift))
32         tens_shift += 1
33         j -= 1
34
35     return product
```

Algorithm 1.5. *Routine for multiplying two positive integers of arbitrary length. Numbers are represented as lists of single-digit integers. For example,* 12×34 *is calculated as* `mult([1, 2], [3, 4])`*. Line* 31 *uses the previously defined function* `add`*. Also in that line the Python syntax* `[0]*tens_shift` *constructs a list* `[0,0,...]` *with* `tens_shift` *zeros in it, which is then appended to the list* `mult_a` *using* `+`*.*

loop of Lines 15–33. The inner loop (Lines 21–25) consists of two parts. First, the `while`-statement of Line 21 has one comparison that is evaluated $n = \texttt{len(a)}$ times ($n-1$ successes and 1 failure). The second part (Lines 22–25) consists of $5+3+2+2=12$ operations and is repeated $n-1$ times. Thus, the inner loop contributes $12(n-1)+n \sim 13n$ operations for each iteration of the outer loop (Lines 15–33).

Preallocating the empty list `sumstep` in Line 16 requires initializing the list (one operation) and making n assignments for the list entries. Prepending `carry` to `sumstep` in Line 27 requires $2n+2$ operations, and the long addition of Line 31 requires $\sim 16n$ operations (see the previous subsection). The other operations in the outer loop contribute at most a constant number of operations. Therefore, the total number of operations in each iteration of the outer loop, including the inner loop, is $\sim 32n$.

The outer loop is repeated at most n times, so the total number of operations required by the outer loop is $\sim 32n^2$. The remaining operations outside the loop are repeated at most $O(n)$ times, so they don't contribute to the leading order. Therefore, the overall temporal complexity of Algorithm 1.5 is $\sim 32n^2$.

The spatial complexity of this algorithm is $O(n)$. Recall that the standard grade school algorithm constructs a stack of n summands that are summed at the end of the algorithm. That would have a spatial complexity $O(n^2)$, but our little efficiency is that we don't build the long stack of addition problems. Instead, the addition steps are done one at a time and thus don't need to be stored separately—we can use the same space in memory each time, as we add the running total. To analyze the leading order of the spatial complexity, we note that the spatial complexity is dominated by the inputs $\sim 2n$, the intermediate list `sumstep`, of size $\sim n$, and the result `product`, of size $\sim 2n$ (note that `ten_shift` can be as large as n), for a total of $\sim 5n$.

1.3 Summation

Analysis of the spatial and temporal complexity of an algorithm typically requires breaking it up into parts and then summing the costs of each part. As algorithms become increasingly sophisticated, it becomes increasingly important to have good tools for managing complicated sums. This section and the next three sections after it are dedicated to developing some of the most useful techniques of summation.

1.3.1 Basic Sums and Differences

Various types of sums occur in algorithm analysis, and the ability to work with these sums, including the ability to identify simple, closed-form expressions for many sums, is an important skill.

Definition 1.3.1. *Let $E = \{e_1, \ldots, e_n\}$. The summation operator $\sum$ maps any function $f : E \to \mathbb{F}$ into[7] $\mathbb{F}$ via the rule*

$$\sum_{e \in E} f(e) = \sum_{k=1}^{n} f(e_k) = f(e_1) + f(e_2) + \cdots + f(e_n).$$

If $E = \emptyset$, then the sum is defined to be 0.

[7]We use $\mathbb{F}$ to denote a field that could be either $\mathbb{R}$ or $\mathbb{C}$. For more about fields, see Volume 1, Appendix B.2.

Remark 1.3.2. In the case that $E = \{k \in \mathbb{Z} \mid a \leq k \leq b\}$ for integer values of a and b, we also write this as $\sum_{k=a}^{b} f(k)$ or $\sum_{a \leq k \leq b} f(k)$.

Remark 1.3.3. A sequence $(x_k)_{k=1}^{n}$ of (not necessarily distinct) numbers also defines a function by $f(k) = x_k$ for each $k \in E = \{1, \ldots, n\}$. The sum $x_1 + \cdots + x_n$ is equal to $\sum_{k=1}^{n} f(k) = \sum_{k=1}^{n} x_k$.

Nota Bene 1.3.4. The notation $\sum_{i=1}^{n} x_i + y$ is ambiguous, because it could mean either $(\sum_{i=1}^{n} x_i) + y$ or $\sum_{i=1}^{n}(x_i + y)$. In this text, it means the former. In other words, if the rightmost term does not have any dependence on the index, we assume that it is not part of the sum. If we want it to be part of the sum, then we use parentheses.

Proposition 1.3.5 (Summation Is Linear). *If $E = \{e_1, \ldots, e_n\}$ and f, g are $\mathbb{F}$-valued functions defined on E, then*

$$\sum_{e \in E} (rf(e) + sg(e)) = r \sum_{e \in E} f(e) + s \sum_{e \in E} g(e)$$

for any $r, s \in \mathbb{F}$.

Proof. The commutative, associative, and distributive laws give

$$\begin{aligned} \sum_{e \in E} (rf(e) + sg(e)) &= (rf(e_1) + sg(e_1)) + \cdots + (rf(e_n) + sg(e_n)) \\ &= (rf(e_1) + \cdots + rf(e_n)) + (sg(e_1) + \cdots + sg(e_n)) \\ &= r(f(e_1) + \cdots + f(e_n)) + s(g(e_1) + \cdots + g(e_n)) \\ &= r \sum_{e \in E} f(e) + s \sum_{e \in E} g(e). \quad \square \end{aligned}$$

1.3.2 Difference Operator

Summation can be thought of as the finite analogue of the definite integral. There is also a finite analogue of the derivative, namely the *difference operator* Δ, which takes a function (or sequence) f and defines a new function that is the difference of the consecutive terms.

Definition 1.3.6. *Let $E = \{a, a+1, \ldots, b\}$ and let $E' = \{a, a+1, \ldots, b-1\}$. The* difference operator Δ *takes any function $f : E \to \mathbb{F}$ and maps it to a new function $\Delta[f] : E' \to \mathbb{F}$ by*

$$\Delta[f](k) = f(k+1) - f(k).$$

Of course, this definition can also be modified in an obvious way to work for the case when the domain E is infinite, like $\mathbb{N}$ or $\mathbb{Z}$.

Remark 1.3.7. It is straightforward to show that the difference operator is linear; see Exercise 1.16.

Example 1.3.8.

(i) If $f(k) = r^k$ for some fixed r, then

$$\Delta[f](k) = r^{k+1} - r^k = r^k(r-1). \tag{1.4}$$

Notationally, we also write this as $\Delta r^k = r^k(r-1)$.

(ii) If $g(k) = r^{k^2}$ for fixed r, then

$$\Delta[g](k) = r^{(k+1)^2} - r^{k^2} = r^{k^2}(r^{2k+1} - 1). \tag{1.5}$$

Notationally, we also write this as $\Delta r^{k^2} = r^{k^2}(r^{2k+1} - 1)$.

Example 1.3.9. The derivative operator and the difference operator have many similarities. If f is constant, so that $f(k) = c$ for all k, then $\Delta[f](k) = c - c = 0$. So we have $\Delta c = 0$, just as $\frac{d}{dx}c = 0$. Conversely, if $\Delta[f] = 0$, then for every k, we have $f(k+1) - f(k) = 0$, so f must be constant on the whole domain (here the domain must be something like[a] $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Z}^+$, or a single, connected, interval $[a, b] \cap \mathbb{Z}$). This is analogous to the fact that when $\frac{d}{dx}f = 0$, then f is constant.

Similarly, if $f(k) = k$, then $\Delta[f](k) = (k+1) - k = 1$. Hence we have $\Delta k = 1$, just as $\frac{d}{dx}x = 1$.

[a]Note our convention that the natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$ include 0. We denote the positive integers by $\mathbb{Z}^+$.

1.3.3 Fundamental Theorem of Finite Calculus

Taking summation as the analogue of definite integration and the difference operator as the analogue of differentiation, the next theorem is an almost perfect match for the usual fundamental theorem of calculus. Its proof, however, is much easier.

Theorem 1.3.10 (The Fundamental Theorem of Finite Calculus). *Let $E = \{a, a+1, \dots, b\}$ and let $E' = \{a, a+1, \dots, b-1\}$. Given any function $f : E \to \mathbb{F}$, we have*

$$\sum_{k=a}^{b-1} \Delta[f](k) = f(b) - f(a). \tag{1.6}$$

Moreover, if we define

$$F(n) = \sum_{k=a}^{n-1} f(k) \qquad \text{for } n \in E, \tag{1.7}$$

then $\Delta[F](k) = f(k)$ for all $k \in E'$.

Nota Bene 1.3.11. The sum in (1.6) runs only to $b-1$ (not to b) and the sum in the definition of F runs only to $n-1$ (not to n).

Proof. For (1.6) we have

$$\sum_{k=a}^{b-1} \Delta[f](k) = \sum_{k=a}^{b-1} (f(k+1) - f(k)) = (f(a+1) - f(a)) + \cdots + (f(b) - f(b-1))$$
$$= -f(a) + f(b),$$

where the last equality comes from the fact that all the internal terms cancel. Sums like this are called *telescoping series.* To prove (1.7), note that for any $n \geq a$ we have

$$\Delta[F](n) = F(n+1) - F(n) = \sum_{k=a}^{n} f(k) - \sum_{k=a}^{n-1} f(k) = f(n). \quad \square$$

Example 1.3.12. Recall that if $f(k) = k$, then $\Delta[f](k) = 1$. By (1.6) we have

$$\sum_{k=a}^{b-1} 1 = \sum_{k=a}^{b-1} \Delta[f](k) = f(b) - f(a) = b - a. \tag{1.8}$$

Of course this sum is easily computed without (1.6), but this illustrates how to use the fundamental theorem.

Example 1.3.13. If $f(k) = k^2$, then $\Delta[f](k) = 2k+1$. By (1.6) we have

$$\sum_{k=a}^{b-1} (2k+1) = \sum_{k=a}^{b-1} \Delta[f](k) = f(b) - f(a) = b^2 - a^2.$$

Since summation is linear, we have

$$b^2 - a^2 = \sum_{k=a}^{b-1} (2k+1) = 2\sum_{k=a}^{b-1} k + \sum_{k=a}^{b-1} 1 = 2\sum_{k=a}^{b-1} k + (b-a).$$

This gives

$$\sum_{k=a}^{b-1} k = \frac{1}{2}(b^2 - a^2 - b + a) = \frac{b(b-1)}{2} - \frac{a(a-1)}{2}.$$

In the special case that $a = 1$ and $b = n + 1$, this gives the familiar sum

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}. \tag{1.9}$$

Example 1.3.14. If $f(k) = k^3$, then $\Delta[f](k) = 3k^2 + 3k + 1$. By (1.6) we have

$$\sum_{k=a}^{b-1} (3k^2 + 3k + 1) = b^3 - a^3.$$

An argument like the one in the previous example shows that

$$\sum_{k=1}^{n} k^2 = \frac{(2n+1)(n+1)n}{6}. \tag{1.10}$$

Remark 1.3.15. The method of the previous examples works in general, but the result gets increasingly cumbersome as the order gets higher and higher. For example,

$$\sum_{k=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2 \tag{1.11}$$

and

$$\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}. \tag{1.12}$$

There is no easily discernible pattern to these power sums. In Section 1.6.2 we discuss a slight variation to this problem and show that there is a generalization that does follow a nice pattern.

Corollary 1.3.16. *For any functions g and h defined on $E = \{a, a+1, \ldots, b\}$ with $\Delta[g] = \Delta[h]$ on $E' = \{a, a+1, \ldots, b-1\}$, the two functions differ by a constant:*

$$g = h + c \qquad \text{on } E,$$

where c is constant on E.

Proof. Let $f = g - h$. Since $\Delta[f] = \Delta[g - h] = \Delta[g] - \Delta[h] = 0$, we have $f(k) = c$ for some constant c (see Example 1.3.9). Thus, $g = h + c$. □

Equation (1.6) also gives a slick proof of the geometric series formula.

Corollary 1.3.17. *For any fixed value $r \neq 1$, we have the geometric series formula*

$$1 + r + \cdots + r^{b-1} = \sum_{k=0}^{b-1} r^k = \frac{r^b - 1}{r - 1}. \tag{1.13}$$

Moreover, taking the limit as $b \to \infty$ for $|r| < 1$ gives the familiar equation

$$\sum_{k=0}^{\infty} r^k = \frac{-1}{r-1} = \frac{1}{1-r}.$$

Proof. Note that $\Delta r^k = r^{k+1} - r^k = r^k(r-1)$. Thus, by the fundamental theorem we have

$$(r-1)\sum_{k=0}^{b-1} r^k = \sum_{k=0}^{b-1} r^k(r-1) = \sum_{k=0}^{b-1} \Delta r^k = r^b - 1,$$

which gives (1.13). □

Example 1.3.18. Choosing $r = 2$ we have $\Delta[2^k] = 2^k$ and (1.13) (or the fundamental theorem) shows that

$$\sum_{k=a}^{b-1} 2^k = 2^b - 2^a.$$

Remark 1.3.19. Example 1.3.18 shows that 2^k plays a role for differences and summation similar to that played by e^x for differentiation and integration.

Just as with integration and differentiation, it is usually more difficult to find a closed form for the summation of a function f than it is to find its first difference $\Delta[f]$. That means that even if it is difficult to write down a formula for $\sum_{k=a}^{b-1} f$, it is usually relatively easy to identify a g such that $\sum_{k=a}^{b-1} g = f$.

Example 1.3.20. Equation (1.5) gives $\Delta[r^{k^2}] = r^{k^2}(r^{2k+1} - 1)$, which implies that

$$\sum_{k=a}^{b-1} r^{k^2}(r^{2k+1} - 1) = r^{b^2} - r^{a^2}. \tag{1.14}$$

1.4 Reindexing and Changing Order of Summation

Two fundamental tools for computing integrals are changing variables and changing the order of integration. The natural analogues of these two techniques are also very important tools for computing sums. In this section we describe these techniques and give some examples.

1.4.1 Reindexing

When computing definite integrals in calculus, changing variables can change an integral into a more workable form. We can also change variables when doing summations. We focus here on the simplest change of variables, called *reindexing*.

Proposition 1.4.1 (Reindexing). *For any finite set $E \subset \mathbb{Z}$ and for any $c \in \mathbb{Z}$, let $E + c$ denote the set*

$$E + c = \{x \in \mathbb{Z} \mid x = e + c \text{ for some } e \in E\}.$$

For any function f defined on $E + c$, we have

$$\sum_{x \in E+c} f(x) = \sum_{e \in E} f(e + c).$$

In particular,

$$\sum_{j=a+c}^{b+c} f(j) = \sum_{k=a}^{b} f(k + c). \tag{1.15}$$

Proof. This follows immediately from writing out the sum

$$\sum_{x \in E+c} f(x) = f(e_1 + c) + f(e_2 + c) + \cdots + f(e_n + c) = \sum_{e \in E} f(e + c). \quad \square$$

Remark 1.4.2. Since j and k are dummy variables, it is common to reuse the index k and write (1.15) as

$$\sum_{k=a+c}^{b+c} f(k) = \sum_{k=a}^{b} f(k + c).$$

Example 1.4.3. The sum $\sum_{k=5}^{n}(k-4)$ looks a lot like (1.9), but the summands are all shifted by -4. That suggests that reindexing might be useful. Setting $j = k - 4$ means that j runs from $5 - 4 = 1$ to $n - 4$, and we have

$$\sum_{k=5}^{n}(k-4) = \sum_{j=1}^{n-4} j = \frac{(n-4)(n-4+1)}{2} = \frac{(n-4)(n-3)}{2}.$$

As described in Remark 1.4.2, the name of the dummy variable doesn't matter, so people often write the second sum as $\sum_{k=1}^{n-4} k$.

Example 1.4.4. Using (1.10) and reindexing, we compute $\sum_{k=1}^{n}(k+3)^2$ as

$$\sum_{k=1}^{n}(k+3)^2 = \sum_{k=4}^{n+3} k^2 = \sum_{k=1}^{n+3} k^2 - \sum_{k=1}^{3} k^2 = \frac{1}{6}[(n+3)(n+4)(2n+7)] - 14.$$

Example 1.4.5. Using (1.13) and reindexing, we compute $\sum_{k=m}^{n} r^k$ as

$$\sum_{k=m}^{n} r^k = \sum_{k=0}^{n-m} r^{m+k} = r^m \sum_{k=0}^{n-m} r^k = r^m \cdot \frac{r^{n-m+1}-1}{r-1} = \frac{r^{n+1}-r^m}{r-1}.$$

1.4.2 Changing Order of Summation

Just as multiple integrals are often simplified by changing the order of integration, multiple sums are often simplified by changing the order of summation. The next proposition is an immediate consequence of the commutativity and associativity of addition.

Proposition 1.4.6. *If $a, b, c, d \in \mathbb{Z}$, and $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{F}$, then*

$$\sum_{j=a}^{b} \sum_{k=c}^{d} f(j,k) = \sum_{k=c}^{d} \sum_{j=a}^{b} f(j,k). \tag{1.16}$$

Proof. Assume $a \leq b$ and $c \leq d$; otherwise both sums are zero. Consider the set $E = \{a, a+1, \ldots, b-1, b\} \times \{c, c+1, \ldots, d-1, d\}$. We see that both sums in (1.16) are equivalent to $\sum_{(j,k)\in E} f(j,k)$. This is illustrated in Figure 1.1. □

Nota Bene 1.4.7. When the sums are infinite, then (1.16) is not necessarily true. We need additional conditions on the convergence rate of the series before we can interchange the order of summation.

Notation 1.4.8. *Multiple sums can sometimes be written unambiguously with a single summation sign. For example, we can write*

$$\sum_{0\leq j,k\leq n} f(j,k) = \sum_{0\leq j\leq n} \sum_{0\leq k\leq n} f(j,k).$$

The proposition justifies this notation. Since it does not matter which index we put on the outside sum and which we put on the inside sum we can combine them.

Proposition 1.4.9. *Consider the domain $E = \{(j,k) \in \mathbb{Z} \times \mathbb{Z} \mid 0 \leq k \leq j \leq n\}$ and the function $f : E \to \mathbb{F}$. We have*

$$\sum_{j=0}^{n} \sum_{k=0}^{j} f(j,k) = \sum_{(j,k)\in E} f(j,k) = \sum_{k=0}^{n} \sum_{j=k}^{n} f(j,k). \tag{1.17}$$

Proof. The proof follows from Figure 1.2. The inner sum of the left side of (1.17) corresponds to summing over the jth column, while the outer sum adds the columns

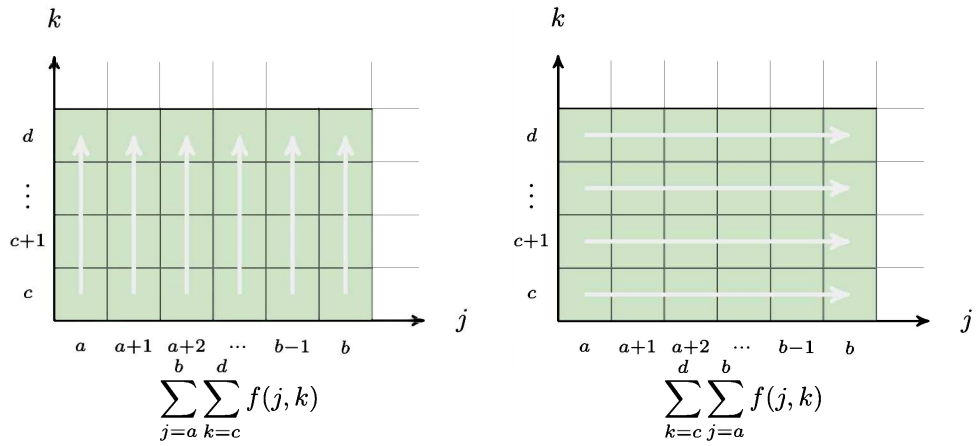


Figure 1.1. *The terms involved in the summation of* (1.16) *are those in the green rectangle. In the expression on the left, the inner sum runs over the terms in the* j*th column (that is,* $(j,c),(j,c+1),\ldots,(j,d)$*, for each* j*), and the outer sum adds the results of the columns together. In the expression on the right the inner sum runs over the terms in the* k*th row (that is,* $(a,k),(a+1,k),\ldots,(b,k)$*, for each* k*) and the outer sum adds the results of the rows together. In either case, the final result is the same.*

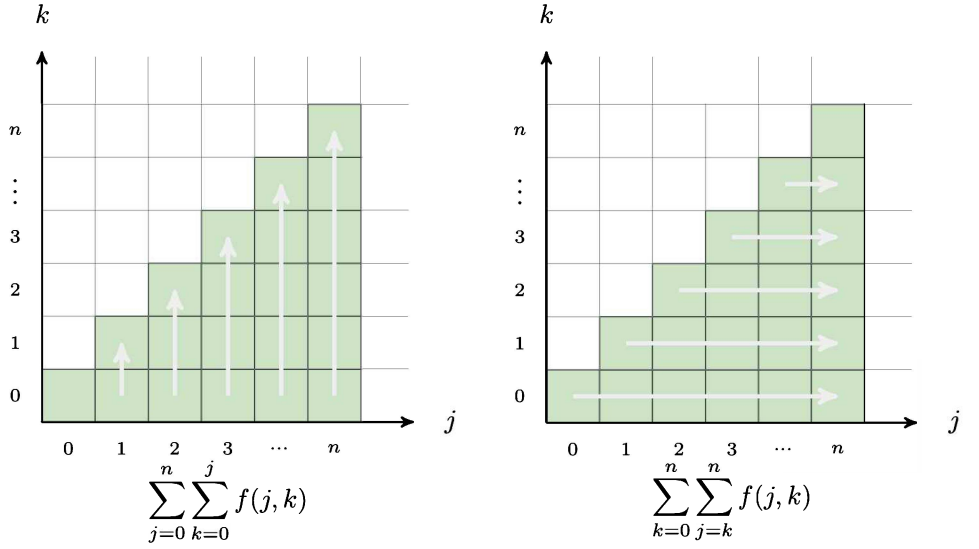


Figure 1.2. *The terms involved in the summation of* (1.17) *are those in the shaded triangular region. In the sum on the left of* (1.17)*, the inner sum runs over the terms in the* j*th column, that is,* $(j,0),(j,1),\ldots,(j,j)$*, and the outer sum adds the columns together. In the sum on the right of* (1.17)*, the inner sum runs over the terms in the* k*th row,* $(k,k),(k+1,k),\ldots,(n,k)$*, and the outer sum adds the rows together.*

together. The inner sum of the right side corresponds to summing the terms in the kth row, while the outer sum adds the rows together. In either case, every term corresponding to a shaded box appears exactly once in the full sum. □

Notation 1.4.10. *Given E and f in the previous proposition, we can also write*

$$\sum_{(j,k)\in E} f(j,k) = \sum_{0\le k\le j\le n} f(j,k).$$

Example 1.4.11. Using the identity in Exercise 1.21 (setting $\ell = n - j$), the double sum $\sum_{j=0}^{n-1}\sum_{k=j+1}^{n} 1$ can be computed directly as

$$\sum_{j=0}^{n-1}\sum_{k=j+1}^{n} 1 = \sum_{j=0}^{n-1} n - j = \sum_{\ell=1}^{n} \ell = \frac{n(n+1)}{2}.$$

But we can also compute it by changing the order of summation:

$$\sum_{j=0}^{n-1}\sum_{k=j+1}^{n} 1 = \sum_{k=1}^{n}\sum_{j=0}^{k-1} 1 = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

Example 1.4.12. Computing the double sum $\sum_{k=0}^{n}\sum_{j=k}^{n} r^k$ directly gives

$$\sum_{k=0}^{n}\sum_{j=k}^{n} r^k = \sum_{k=0}^{n}(n-k+1)r^k = (n+1)\sum_{k=0}^{n} r^k - \sum_{k=0}^{n} kr^k.$$

This last sum can be computed using summation by parts (see Section 1.6.1), but it is messy. However, interchanging the order of summation in the original problem makes the double sum easy to compute:

$$\sum_{k=0}^{n}\sum_{j=k}^{n} r^k = \sum_{j=0}^{n}\sum_{k=0}^{j} r^k = \frac{1}{r-1}\sum_{j=0}^{n}(r^{j+1}-1) = \frac{1}{r-1}\left(\frac{r^{n+2}-r}{r-1} - (n+1)\right).$$

There is nothing special about the particular shapes of the regions in Figure 1.1 or Figure 1.2. For any finite set E, the sum $\sum_{e\in E} f(e)$ can be computed by summing all the terms $f(e)$ in any order—row first or column first or even some other pattern. For example, the set $E = \{(j,k) \mid j,k \ge 0 \text{ and } j+k \le n\}$ of Figure 1.3 can be summed either rows first or columns first. This gives the equality

$$\sum_{(j,k)\in E} f(j,k) = \sum_{j=0}^{n}\sum_{k=0}^{n-j} f(j,k) = \sum_{k=0}^{n}\sum_{j=0}^{n-k} f(j,k). \tag{1.18}$$

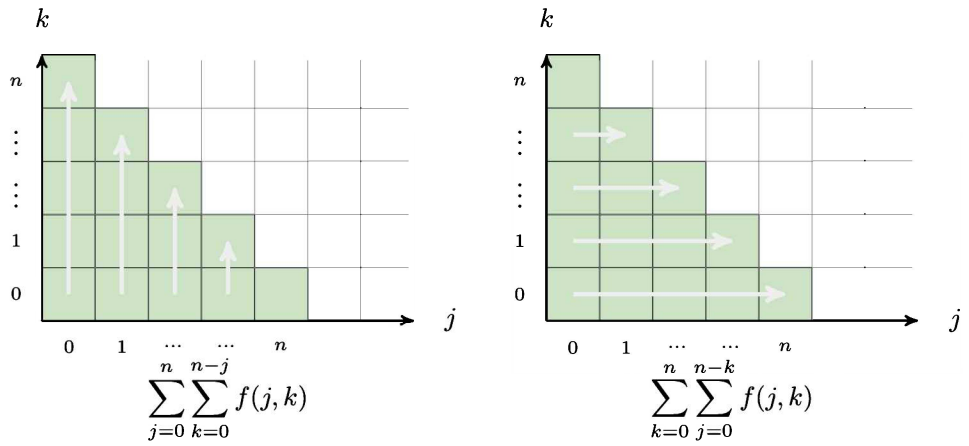


Figure 1.3. *Another example of changing the order of summation. Summing over the green region vertically first (left panel) gives the same result as summing horizontally first (right panel). This fact shows the two sums are equal, as given in equation* (1.18).

Example 1.4.13. To compute the double sum $\sum_{j=0}^{n}\sum_{k=0}^{n-j}\frac{j+1}{(n-k+1)(n-k+2)}$ initially appears difficult. But changing the order of summation gives

$$\begin{aligned}
&\sum_{j=0}^{n}\sum_{k=0}^{n-j}\frac{j+1}{(n-k+1)(n-k+2)} \\
&\quad=\sum_{k=0}^{n}\frac{1}{(n-k+1)(n-k+2)}\sum_{j=0}^{n-k}(j+1) \\
&\quad=\sum_{k=0}^{n}\frac{1}{(n-k+1)(n-k+2)}\frac{(n-k+1)(n-k+2)}{2} \\
&\quad=\sum_{k=0}^{n}\frac{1}{2}=\frac{n+1}{2}.
\end{aligned}$$

Example 1.4.14. Here is a more general example of changing the order of summation. To compute the sum $\sum_{0\le j\le 2n}\sum_{j/2\le k\le n} r^j r^{k^2}$, begin by interchanging the order of summation. To do this, note that the smallest value that k can ever take is 0 (when $j=0$), and the largest value that k can take is n, so the new outer sum will range over all values of $k\in\{0,\dots,n\}$. The inner variable j is bounded above by the constraints $j\le 2n$ (from the old outer sum) and $j/2\le k$ (from the old inner sum), so the new inner sum ranges over

all values of $j \in \{0, \ldots, 2k\}$. This gives

$$\sum_{0 \le j \le 2n} \sum_{j/2 \le k \le n} r^j r^{k^2} = \sum_{0 \le k \le n} \sum_{0 \le j \le 2k} r^j r^{k^2}.$$

Notice that r^{k^2} is independent of the index j, so it factors out of the sum to give

$$\sum_{0 \le k \le n} r^{k^2} \sum_{0 \le j \le 2k} r^j.$$

The inner sum is the geometric series (1.13), so the double sum reduces to

$$\sum_{0 \le k \le n} r^{k^2} \frac{r^{2k+1} - 1}{r - 1} = \frac{1}{r-1} \sum_{0 \le k \le n} r^{k^2} (r^{2k+1} - 1).$$

By (1.14) this sum becomes

$$\frac{1}{r-1} \sum_{0 \le k \le n} \Delta r^{k^2} = \frac{r^{(n+1)^2} - 1}{r - 1}.$$

1.5 Nested Loops

One important application of the double sums of the previous section is the analysis of nested loops, where one loop occurs within another. Nested loops occur frequently in scientific computing, especially in the algorithms of numerical linear algebra. We typically represent vectors and matrices as arrays of floating-point numbers, and moving through these arrays to perform the operations of matrix-vector and matrix-matrix multiplication uses nested loops. Thus, to analyze many algorithms in numerical linear algebra, we must understand how to analyze nested loops.

1.5.1 Aside: Floating-Point Operations

The long addition and long multiplication algorithms (Algorithms 1.2 and 1.5) dealt with integers of arbitrary size, but in most computational settings, including numerical linear algebra, we use floating-point numbers. Floating-point numbers are represented in a manner similar to scientific notation, except everything is carried out in base 2 instead of base 10, and they are all rounded to fit into 64 bits[8] (8 bytes) of memory; for details see Section 11.1.

Basic arithmetic operations for floating-point numbers are built into the hardware and can be performed in one or two clock cycles each. These include the

[8] A *bit* is a single binary digit (taking only a value of 1 or 0). A group of eight bits is commonly called a *byte*. Single-precision floating-point numbers are stored in 32-bit (4-byte) form, but there is little benefit to using single precision on modern computers, which mostly have 64-bit architectures. For that reason we focus on 64-bit (double-precision) floating-point arithmetic.

standard arithmetic operations $+$, $-$, $\times$, and $\div$ and are called *floating-point operations* (FLOPs).[9]

When analyzing the temporal complexity of many numerical algorithms, it is customary to count only the FLOPs, instead of using primitive operations. For example, when adding two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ of floating-point numbers, the sum $\mathbf{x} + \mathbf{y} = (x_1 + y_1, \ldots, x_n + y_n)$ requires n FLOPs (additions), and we ignore the other primitive operations like variable assignments, array lookups, and loop overhead. Similarly, scalar multiplication $\alpha\mathbf{x} = (\alpha x_1, \ldots, \alpha x_n)$ requires n FLOPs (multiplications), and the additional primitive operations of assignment, lookup, and loop overhead are likewise ignored. Generally each FLOP requires roughly two array lookups (the inputs) and one variable assignment (for the output), so those primitive operations are assumed to be part of the cost of performing one FLOP, whereas loop overhead is generally very small compared to the cost of all the FLOPs. Thus counting FLOPs can give a good measure of total complexity, at least to leading order, even without counting all the primitive operations.

The spatial complexity of both vector addition and scalar multiplication is $O(n)$ because the size of each floating-point number is fixed, and there are n of these numbers in each vector. The only other variables that depend on n are the indices required to loop through the vectors, and these have size at most $O(\log n)$, the number of digits required to represent n. As in the case of temporal complexity, when computing the spatial complexity of a numerical algorithm we usually track only the memory needed for floating-point numbers but ignore the memory needed for other aspects of the algorithm, like loop counters. These other, neglected, memory requirements are usually much smaller than the number of floating-point numbers used, so they rarely contribute anything to the leading order of the spatial complexity.

1.5.2 Matrix-Vector and Matrix-Matrix Multiplication

The inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i$ of two vectors in $\mathbb{R}^n$ is one of the most widely used operations in numerical linear algebra and scientific computing in general. Calculating it requires n multiplications and $n - 1$ additions, for a total of $2n - 1$ FLOPs.

Example 1.5.1. If $\mathbf{x} = (1, 2, 3, 4)$ and $\mathbf{y} = (5, 6, 7, 8)$, then the usual inner product $\langle \mathbf{x}, \mathbf{y} \rangle = 1 \times 5 + 2 \times 6 + 3 \times 7 + 4 \times 8$ requires four multiplications and three additions, for a total of seven FLOPs. Spatially, the algorithm must store the two vectors (eight values), and one more number for output (which can also be used for the intermediate calculation of the running total), for a total of nine floating-point numbers.

In matrix-vector multiplication, a matrix $A \in M_{m \times n}(\mathbb{R})$ and a vector $\mathbf{x} \in \mathbb{R}^n$ are multiplied together to form a new vector $A\mathbf{x} \in \mathbb{R}^m$. This can be thought of

[9] The acronym FLOPs should not be confused with FLOPS, which means *floating-point operations per second*. The latter is a measure of performance in hardware, namely, the number of floating-point operations a given computer can perform each second.

as m inner products between the rows of A and the vector $\mathbf{x}$. Thus, its temporal complexity is $m(2n-1) \sim 2mn$ FLOPs. Spatially, the inputs require $(m+1)n$ floating-point numbers, and the resulting vector requires m more. Thus the spatial complexity of matrix-vector multiplication is $\sim mn + n + m$.

In matrix-matrix multiplication, two matrices $A \in M_{\ell \times m}$ and $B \in M_{m \times n}$ are multiplied together to create a new matrix $AB \in M_{\ell \times n}$. This can be thought of as ℓn inner products between the rows of A and the columns of B. Thus, the temporal complexity of this algorithm is $\ell n(2m-1) \sim 2\ell mn$. For the spatial requirements, beyond the inputs (which have size $\sim \ell m + mn$), we need only store the output, which has spatial complexity $\sim \ell n$. Thus the total spatial complexity is $\sim \ell m + \ell n + mn$.

Remark 1.5.2. The basic operations in numerical linear algebra, including matrix-vector and matrix-matrix multiplication, are included in numerical libraries that are highly optimized for performance and therefore run much faster than a naïve implementation of the algorithms mentioned above. For this reason, it is rarely a good idea to code these algorithms yourself from scratch. One of the most famous numerical libraries is *Basic Linear Algebra Subprograms (BLAS)*, which is at the core of nearly every computing environment for numerical linear algebra. Numerical libraries like BLAS optimize the workflow of the algorithm by making clever use of the cache and pipelining. This allows for vectorization, meaning that several primitive and floating-point operations can be performed at once by different registers in the CPU. It also minimizes the latency, that is, the time wasted waiting for memory calls.

Vista 1.5.3. There are asymptotically faster algorithms for matrix-matrix multiplication than the one described here. For example, when $\ell = m = n$ *Strassen's algorithm* requires only $O(n^{\log_2 7}) \approx O(n^{2.8074})$ FLOPs, whereas the regular algorithm is $\sim 2n^3 \in O(n^3)$. As a trade-off, Strassen's algorithm has greater spatial complexity and generally more round-off error than the regular algorithm. Also, the overhead in Strassen's algorithm is large enough that the matrices must be rather large before it's actually faster to use it; see Section 1.10 for more details.

Row Reduction

Assume $A \in M_n(\mathbb{R})$ is an invertible matrix and $\mathbf{b} \in \mathbb{R}^n$. The canonical approach to solving a linear system of the form $A\mathbf{x} = \mathbf{b}$ is to use row reduction (see Volume 1, Section 2.7). This process consists of first performing a series of row operations to turn the augmented matrix $[A|\mathbf{b}]$ into an upper triangular matrix (row echelon form), and then performing back substitution to get the solution. For example, the row reduction step for solving the system

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 2 \\ 4 & 7 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 9 \end{bmatrix}$$

looks like

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 4 & 2 & 3 \\ 4 & 7 & 8 & 9 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 4 & 7 & 8 & 9 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 3 & 4 & 5 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 0 & 3 & 3 \end{array}\right].$$

Back substitution looks like

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{3} \\ 0 & 0 & 1 & 1 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{1}{3} \\ 0 & 1 & 0 & \frac{1}{3} \\ 0 & 0 & 1 & 1 \end{array}\right].$$

Now we show that row reduction as given in Algorithm 1.6 has a FLOP count of $\sim \frac{2}{3}n^3$. Back substitution is $O(n^2)$ because it requires a multiplication and an addition for each entry in the top half of the matrix, of which there are $\frac{1}{2}n(n+1)$. Thus, back substitution does not add to the leading-order behavior, and n-dimensional linear systems can be solved in $\sim \frac{2}{3}n^3$ FLOPs. It can be shown that this is roughly a third the cost of inverting A and computing $\mathbf{x} = A^{-1}\mathbf{b}$. For this reason (and for numerical stability[10] reasons) it is almost always preferable to compute the row reduction of a matrix rather than compute its inverse.

To add up all the FLOPs in Algorithm 1.6, we look at Lines 13–18. Inside the two loops one FLOP is needed for computing `c` (Line 15), and then the row operation on Line 17 is really another loop that repeats $n-k$ times and requires two FLOPs per iteration.

Summing the FLOPs over all the loops gives

$$\sum_{k=0}^{n-2} \sum_{j=k+1}^{n-1} \left(1 + \sum_{i=k+1}^{n} 2\right) = \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n. \tag{1.19}$$

The proof of the equality in (1.19) is Exercise 1.28. It is straightforward to see that the remaining parts of the algorithm require only $O(n^2)$ FLOPs, so they do not contribute to the leading order. Thus, row reduction costs $\sim \frac{2}{3}n^3$.

Remark 1.5.4. The industrial-grade approach to solving the linear system is to use the *LU decomposition*, which overwrites A with a lower triangular matrix L (with all ones on the diagonal) and an upper triangular matrix U that satisfies $A = LU$. Note that we can store the key parts of L and U in the space provided by A, dovetailing the two matrices together. For example,

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 2 \\ 4 & 7 & 8 \end{bmatrix}, \; L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 1 & 1 \end{bmatrix}, \; U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix}, \text{ and } lu(A) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 4 & 1 & 3 \end{bmatrix}.$$

The complexity of producing this factorization is the same as row reduction ($\sim \frac{2}{3}n^3$), and since $A\mathbf{x} = LU\mathbf{x} = \mathbf{b}$, we can find $\mathbf{x}$ by solving $L\mathbf{y} = \mathbf{b}$ by forward substitution (at a cost of $O(n^2)$) and then solving $U\mathbf{x} = \mathbf{y}$ by back substitution (also costing $O(n^2)$). The total complexity is dominated by the factorization $\sim \frac{2}{3}n^3$, but the

[10] Stability of an algorithm has to do with round-off error. For more on this see the introduction to Chapter 1 (page 5) and Section 11.3.

```python
import numpy as np # module for efficient linear algebra

def row_reduction(A):
    """
    Row-reduce an n x (n+1) matrix (without pivoting)
    and perform back substitution.

    Returns A in reduced row echelon form (RREF)
    """
    n = A.shape[0]

    # Row Reduction
    for k in range(n-1):        # Iterate over rows except the last
        for j in range(k+1,n):  # Iterate over rows below k
            c = A[j,k] / A[k,k] # Scalar to multiply row(k) by
            # Subtract c * row(k) from row(j)
            A[j,k+1:n+1] = A[j,k+1:n+1] - c * A[k,k+1:n+1]
            A[j,k] = 0

    # Back Substitution
    for j in range(1,n+1):       # Iterate from the bottom right
        # Divide row by its leading term (assume nonzero)
        A[n-j,n] = A[n-j,n] / A[n-j,n-j]
        A[n-j,n-j] = 1           # leading term always becomes 1
        for k in range(0,n-j):   # Rows above row(n-j)
            # Adjust the n-j th and n th columns
            A[k,n] = A[k,n] - A[k,n-j] * A[n-j,n]
            A[k,n-j] = 0
    return A
```

Algorithm 1.6. *A row reduction algorithm for an $n \times (n+1)$ matrix. This simplified method assumes the pivots (the diagonal elements) are nonzero so that the division at Lines 15 and 23 is well defined. This algorithm uses NumPy, a module for efficient linear algebra. The matrix* `A` *must be a NumPy array (for example,* `A = np.array([[1,1,1,1],[1,4,2,3],[4,7,8,9]])`*). The syntax* `A[j,k]` *gives the* `j,k` *element of* `A`*, and* `A[j,k:n]` *gives the elements of the* `j`*th row of* `A` *from* `k` *up to (but not including)* `n`*. Note that* `range(n-1)` *iterates through the values* $\{0, 1, \ldots, n-2\}$*, while* `range(k+1,n)` *iterates through the values* $\{k+1, k+2, \ldots, n-1\}$*.*

main advantage is that we can get this factorization without needing additional memory, which is vital when the size of the matrix is really big.

To improve numerical stability, most LU decomposition algorithms actually find a permutation matrix P that reorders the rows of A before doing the row reduction, so that $PA = LU$. Since a permutation matrix is equal to a reordering of the rows of the identity matrix, it can be stored as an array of length n, and thus it does not contribute to the leading-order spatial complexity of the LU decomposition.

Remark 1.5.5. There is a decomposition called the *QR decomposition* (see Volume 1, Sections 3.3–3.4), which is more stable than the LU decomposition and has complexity $\sim \frac{4}{3}n^3$. This can be used when stability is critical, but usually the LU decomposition is sufficient, at half the cost.

Vista 1.5.6. When the matrix A is sparse (that is, when most of the entries are zero), or when the matrix has some other special structure, there are some excellent iterative methods for solving the linear system $A\mathbf{x} = \mathbf{b}$. The asymptotic complexity of these algorithms is sometimes nearly as small as the number of nonzero entries in A, which is much less than n^2. Krylov subspace methods provide several of these solvers (see Chapter 13 of Volume 1).

The solution of a linear system can also be transformed into a quadratic optimization problem. This leads to some other powerful iterative methods for solving linear systems. We discuss some of these methods in Chapter 12, especially in Section 12.6.

1.5.3 *Loop Interchange

As shown in Section 1.4.2, changing the order of summation in a double sum can make the sum easier to compute. Similarly, changing the order of a nested loop (called *loop interchange*) can often reduce the complexity of the nested loop. To see how this works, we first must understand *loop-invariant code motion.*

Loop-Invariant Code Motion

A fundamental principle for designing efficient code is that no operation should be performed unnecessarily. In particular, no operation that can be performed outside a loop should be performed inside the loop. Otherwise, it is executed many more times than it needs to be.

Example 1.5.7. In the loop

```
for j in range(n):
    x += ham(j,k) + spam(k)
```

the function `spam(k)` is independent of the variables that change in the loop (like the counter `j`), but it is recomputed n times because it is inside the loop. A better approach would be to compute it only once, outside the loop, as follows:

```
spam_val = spam(k)
for j in range(n):
    x += ham(j,k) + spam_val
```

If computing `spam(k)` costs $s(k)$ FLOPs, then moving the computation outside the loop has reduced the temporal complexity by $(n-1)s(k)$ FLOPs. Even better would be to recognize that we can replace the n additions of `spam_val` inside the loop with one multiplication and one addition outside the loop:

```
x += n * spam(k)
for j in range(n):
    x += ham(j,k)
```

This further reduces the complexity by $n-2$ FLOPs.

Example 1.5.8. In the following loop, the value `3 * x**2 + 2` is computed with each iteration.

```
while 3 * x**2 + 2 < N:
    spam(x)
    x += 1
```

But the condition `3 * x**2 + 2 < N` is equivalent to `x < ((N - 2) / 3) **0.5`. Therefore, we can avoid computing `3 * x**2 + 2` each time and, instead, compute the value `((N - 2) / 3)**0.5` just once:

```
M = ((N - 2) / 3)**0.5
while x < M:
    spam(x)
    x += 1
```

This gives a savings of $3M-3$ FLOPs. Of course, if M is small, this is not meaningful, but if M is large, this could be a significant improvement in the temporal complexity of the algorithm.

Remark 1.5.9. In Example 1.5.8 we consider computing the square root of a floating-point number to be a single FLOP. This is a common practice because the square root function is often built into the hardware and can be computed in about the same amount of time as a floating-point multiplication.

The process of moving code outside of loops, as described in Examples 1.5.7 and 1.5.8, is called *loop-invariant code motion* because the part of the code that is independent of the loop (loop invariant) is moved out of the loop. In compiled languages this is often performed automatically by an optimizing compiler, but it does not happen automatically in interpreted languages like Python.

Loop Interchange

Loop interchange can allow us to move operations out of the inner loop or even eliminate the inner loop entirely, thereby reducing the complexity of the nested loop.

Example 1.5.10. In the double nested loop

```
for j in range(n):
    for k in range(m):
        x += ham(j,k) + spam(k)
```

the functions `ham(j,k)` and `spam(k)` are computed `nm` times. Changing which loop is inside and which is outside has no effect on the final result.

```
for k in range(m):
    for j in range(n):
        x += ham(j,k) + spam(k)
```

But now `spam(k)` does not depend on `j`, so `spam(k)` can be moved outside of the inner loop, as in Example 1.5.7, and we get the final result more efficiently:

```
for k in range(m):
    x += n * spam(k)
    for j in range(n):
        x += ham(j,k)
```

Changing the loop order and moving operations out of the inner loop has saved $\sum_{k=0}^{m-1}((n-1)s(k)+n-2) = m(n-2)+(n-1)\sum_{k=0}^{m-1}s(k)$ FLOPs, where $s(k)$ is the number of FLOPs required by `spam(k)`.

Remark 1.5.11. While good optimizing compilers automatically consider loop interchange for the purpose of improving the efficiency of memory access, they do not usually consider the other potential benefits of loop interchange, such as those shown in Example 1.5.10.

1.6 *Additional Techniques of Summation

In this section we discuss several additional techniques of summation. Many of these are discrete versions of fundamental techniques in traditional calculus, including discrete versions of the product rule and integration by parts. We also discuss the *Pochhammer symbols*, which provide the identities needed to adapt the power formulas in calculus to the difference and summation operators, as alluded to in Remark 1.3.15. We conclude this section with a discussion of the *inclusion-exclusion principle*, which generalizes the counting formula for unions of sets.

1.6.1 Product Rule and Summation by Parts

Just as the derivative has a product rule, the difference operator Δ also has a product rule. But to write this in a clean way, we must first define the translation operator T.

Definition 1.6.1. *The* translation operator T *takes any function* $f : \mathbb{N} \to \mathbb{F}$ *and maps it to a new function* $T[f] : \mathbb{N} \to \mathbb{F}$ *by* $T[f](k) = f(k+1)$. *The definition also works if the domain is* $\mathbb{Z}$ *or* $\mathbb{Z}^+$.

Remark 1.6.2. If I is the identity operator (that is, $I[f] = f$), then $\Delta = T - I$.

Theorem 1.6.3 (Finite Product Rule). *For any* $\mathbb{F}$*-valued functions* f, g *on* $\mathbb{N}$ *(or on* $\mathbb{Z}^+$ *or* $\mathbb{Z}$*) we have*

$$\Delta[fg] = \Delta[f] \cdot T[g] + f \cdot \Delta[g] = T[f] \cdot \Delta[g] + g \cdot \Delta[f]. \tag{1.20}$$

Proof. The proof uses the same trick used to prove the usual product rule—adding a fancy form of zero (shown in red below):

$$\begin{aligned}
\Delta[fg](k) &= f(k+1)g(k+1) - f(k)g(k) \\
&= f(k+1)g(k+1) + \color{red}{[f(k)g(k+1) - f(k)g(k+1)]} - f(k)g(k) \\
&= \Delta[f](k)g(k+1) + f(k)\Delta[g](k) \\
&= \Delta[f](k)T[g](k) + f(k)\Delta[g](k).
\end{aligned}$$

The other half of (1.20) follows by applying the same argument to gf. ∎

Remark 1.6.4. The presence of the operator T in (1.20) is a little disappointing. We don't see this in the product rule in calculus because the term $T[g](k)$ in the discrete case corresponds to $g(x+h)$ in the infinitesimal case, and $h \to 0$ means that $g(x+h) \to g(x)$.

Integrating the product rule gives integration by parts, and summing the discrete product rule gives the formula we call *summation by parts.*

Corollary 1.6.5. *For any* $\mathbb{F}$*-valued functions* u, v *on* $\mathbb{N}$ *(alternatively* $\mathbb{Z}^+$ *or* $\mathbb{Z}$*) we have*

$$\sum_{a \le k < b} u(k)\Delta[v](k) = u(b)v(b) - u(a)v(a) - \sum_{a \le k < b} T[v](k)\Delta[u](k). \tag{1.21}$$

Proof. This follows immediately from the product rule and the fundamental theorem. ∎

Example 1.6.6. Using summation by parts, we can compute $\sum_{k=1}^{n} ka^k$ for $a \neq 1$. Let $u(k) = k$ and $v(k) = \frac{a^k}{a-1}$, so that $\Delta[u](k) = 1$ and $\Delta[v](k) = a^k$. Thus,

$$\begin{aligned}
\sum_{k=0}^{n} ka^k &= \sum_{k=0}^{n} u(k)\Delta[v](k) \\
&= u(n+1)v(n+1) - u(0)v(0) - \sum_{k=0}^{n} \Delta[u](k)v(k+1) \\
&= \frac{(n+1)a^{n+1}}{a-1} - \sum_{k=0}^{n} \frac{a^{k+1}}{a-1} \\
&= \frac{(n+1)a^{n+1}}{a-1} + \frac{a-a^{n+2}}{(a-1)^2} \\
&= \frac{a}{(a-1)^2}(na^{n+1} - (n+1)a^n + 1).
\end{aligned}$$

1.6.2 Rising and Falling Powers

Remark 1.3.15 shows that sums of powers do not yield nice formulas. But there are two expressions that behave like powers, called the *rising* and *falling*, or *Pochhammer symbols*, that do behave nicely with respect to summation operators. They also behave nicely with difference operators.

Definition 1.6.7. *Assume that $m \in \mathbb{N}$ and $x \in \mathbb{R}$. The* rising Pochhammer symbol, *which reads as "x to the m rising," is given by the expression*

$$x^{\overline{m}} = x(x+1)(x+2)\cdots(x+m-1).$$

Similarly, the falling Pochhammer symbol *is defined to be*

$$x^{\underline{m}} = x(x-1)(x-2)\cdots(x-m+1).$$

In this case, we say "x to the m falling." For notational convenience, we define $x^{\overline{0}} = x^{\underline{0}} = 1$, since these both correspond to empty products, which are usually taken to be 1.

We show the following two identities for raising operators and leave the corresponding properties of the lowering operators to the reader to determine.

Lemma 1.6.8. *If $m \in \mathbb{N}$ and $x \in \mathbb{R}$, then*

$$\Delta \frac{x^{\overline{m+1}}}{m+1} = (x+1)^{\overline{m}}.$$

Proof.

$$\begin{aligned}
&\Delta x(x+1)(x+2)\cdots(x+m)\\
&\quad = (x+1)(x+2)\cdots(x+m+1) - x(x+1)(x+2)\cdots(x+m)\\
&\quad = ((x+m+1)-x)(x+1)(x+2)\cdots(x+m)\\
&\quad = (m+1)(x+1)(x+2)\cdots(x+m). \quad \square
\end{aligned}$$

Theorem 1.6.9. *If $m \in \mathbb{N}$, then*

$$\sum_{k=0}^{n} k^{\overline{m}} = \frac{n^{\overline{m+1}}}{m+1}. \tag{1.22}$$

Proof. From Lemma 1.6.8 and the fundamental theorem (Theorem 1.3.10), we have

$$\frac{n^{\overline{m+1}}}{m+1} = \sum_{k=0}^{n-1} \Delta \frac{k^{\overline{m+1}}}{m+1} = \sum_{k=0}^{n-1} (k+1)^{\overline{m}} = \sum_{k=1}^{n} k^{\overline{m}} = \sum_{k=0}^{n} k^{\overline{m}}. \quad \square$$

Example 1.6.10. Setting $m = 1$ we have

$$\sum_{k=1}^{n} k = \sum_{k=1}^{n} k^{\overline{1}} = \frac{n^{\overline{2}}}{2} = \frac{n(n+1)}{2},$$

which agrees with (1.9). Setting $m = 2$, we have

$$\sum_{k=1}^{n} k^{\overline{2}} = \sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3} = \frac{n^{\overline{3}}}{3}.$$

Expanding gives

$$\sum_{k=1}^{n} k^2 + \sum_{k=1}^{n} k = \frac{n(n+1)(n+2)}{3}.$$

Subtracting $\sum_{k=1}^{n} k$ gives

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(n+2)}{3} - \frac{n(n+1)}{2} = \frac{n(n+1)(2n+1)}{6},$$

which agrees with (1.10).

1.6.3 Inclusion-Exclusion

The *inclusion-exclusion principle* is a fundamental tool for counting and for working with unions of sets that overlap. The basic idea for a union of two sets is to first consider everything in both sets (inclusion) but then remove terms that were counted twice (exclusion).

Proposition 1.6.11 (Inclusion-Exclusion for Two Sets). *For any finite sets E, F and any $\mathbb{F}$-valued functions f, g defined on $E \cup F$, we have*

$$\sum_{k \in E \cup F} f(k) = \sum_{k \in E} f(k) + \sum_{k \in F} f(k) - \sum_{k \in E \cap F} f(k).$$

Proof. We partition $E \cup F$ into three sets:

$$E \cup F = (E \setminus (E \cap F)) \cup (F \setminus (E \cap F)) \cup (E \cap F),$$

which gives

$$\begin{aligned}\sum_{k \in E \cup F} f(k) &= \sum_{k \in (E \setminus (E \cap F))} f(k) + \sum_{k \in (F \setminus (E \cap F))} f(k) + \sum_{k \in (E \cap F)} f(k) \\ &= \sum_{k \in E} f(k) - \sum_{k \in (E \cap F)} f(k) + \sum_{k \in F} f(k) - \sum_{k \in (E \cap F)} f(k) + \sum_{k \in (E \cap F)} f(k) \\ &= \sum_{k \in E} f(k) + \sum_{k \in F} f(k) - \sum_{k \in E \cap F} f(k). \quad \square\end{aligned}$$

Example 1.6.12. Choosing f to be the constant function 1 means that the sum $\sum_{k \in S} f(k)$ is the cardinality of the finite set S. Thus, for any finite sets A and B, inclusion-exclusion (Proposition 1.6.11) implies that

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

This is a well-known counting formula for finite sets.

Proposition 1.6.13 (Inclusion-Exclusion for Three Sets). *Given three finite sets A, B, and C, we have*

$$\begin{aligned}\sum_{x \in A \cup B \cup C} f(x) = &\sum_{x \in A} f(x) + \sum_{x \in B} f(x) + \sum_{x \in C} f(x) - \sum_{x \in A \cap B} f(x) \\ &- \sum_{x \in A \cap C} f(x) - \sum_{x \in B \cap C} f(x) + \sum_{x \in A \cap B \cap C} f(x).\end{aligned}$$

Proof. This follows by writing $E = A \cup B$ and $F = C$ and applying the proposition twice. The details are Exercise 1.34. A graphical representation of how the three sets intersect is given in Figure 1.4. $\square$

Example 1.6.14. For finite sets A, B, and C, we have the counting formula

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|.$$

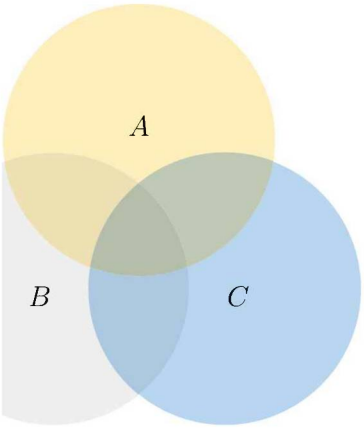


Figure 1.4. *A Venn diagram helps illustrate the ideas of inclusion-exclusion for three sets A, B, and C, as in Proposition 1.6.13. Including everything from A, B, and C means that the pairwise intersections $A \cap B$, $A \cap C$, and $B \cap C$ have all been included twice and the triple intersection has been included three times. Excluding one copy of each of the pairwise intersections means that the triple intersection, which was originally included three times, has now been excluded three times, so it must be reincluded once more.*

Example 1.6.15. Let S be the set $S = \{1, 2, \ldots, 1000\}$ and let E be the integers in S that are divisible by any of 5, 7, or 9; that is, let $E = E_5 \cup E_7 \cup E_9$, where E_n is the set of integers divisible by n in S. We have $|E_n| = \lfloor 1000/n \rfloor$, where $\lfloor x \rfloor$ denotes the integer part of $x \geq 0$. Note also that $E_n \cap E_m = E_{nm}$ whenever $\gcd(n, m) = 1$. Thus

$$\begin{aligned}
|E| &= |E_5| + |E_7| + |E_9| - |E_5 \cap E_7| - |E_5 \cap E_9| - |E_7 \cap E_9| + |E_5 \cap E_7 \cap E_9| \\
&= \lfloor 1000/5 \rfloor + \lfloor 1000/7 \rfloor + \lfloor 1000/9 \rfloor - \lfloor 1000/(5 \cdot 7) \rfloor - \lfloor 1000/(5 \cdot 9) \rfloor \\
&\qquad - \lfloor 1000/(7 \cdot 9) \rfloor + \lfloor 1000/(5 \cdot 7 \cdot 9) \rfloor \\
&= 200 + 142 + 111 - 28 - 22 - 15 + 3 \\
&= 391.
\end{aligned}$$

This can be extended to an arbitrary (finite) number of sets.

Theorem 1.6.16 (Inclusion-Exclusion). *For any finite collection $E_1, \ldots, E_m$ of finite sets and for any function $f : \bigcup_{i=1}^{m} E_i \to \mathbb{F}$ we have*

$$\sum_{e \in \bigcup_{i=1}^{m} E_i} f(e) = \sum_{k=1}^{m} (-1)^{k+1} \sum_{1 \leq i_1 < \cdots < i_k \leq m} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e). \tag{1.23}$$

Proof. This follows by induction on m. The base case is $m = 1$, which is trivial. Now assume that (1.23) holds for all m with $1 \leq m \leq n$. We prove that the equation also holds for $m = n + 1$.

Let $E = \bigcup_{i=1}^{n} E_i$ and $F = E_{n+1}$. By Proposition 1.6.11 and the induction hypothesis, we have

$$\begin{aligned}
\sum_{e \in \bigcup_{i=1}^{n+1} E_i} f(e) &= \sum_{e \in E \cup F} f(e) = \sum_{e \in E} f(e) + \sum_{e \in F} f(e) - \sum_{e \in E \cap F} f(e) \\
&= \sum_{e \in \bigcup_{i=1}^{n} E_i} f(e) + \sum_{e \in E_{n+1}} f(e) - \sum_{e \in \bigcup_{i=1}^{n} E_i \cap E_{n+1}} f(e) \\
&= \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k \le n} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e) + \sum_{e \in E_{n+1}} f(e) \\
&\quad - \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k \le n} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j} \cap E_{n+1}} f(e).
\end{aligned}$$

The last term can be rewritten as

$$\sum_{1 \le i_1 < \cdots < i_k \le n} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j} \cap E_{n+1}} f(e) = \sum_{1 \le i_1 < \cdots < i_k < i_{k+1} = n+1} \sum_{e \in \bigcap_{j=1}^{k+1} E_{i_j}} f(e).$$

Thus, we have

$$\begin{aligned}
\sum_{e \in \bigcup_{i=1}^{n+1} E_i} f(e) &= \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k \le n} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e) + \sum_{e \in E_{n+1}} f(e) \\
&\quad + \sum_{k=1}^{n} (-1)^{k+2} \sum_{1 \le i_1 < \cdots < i_k < i_{k+1} = n+1} \sum_{e \in \bigcap_{j=1}^{k+1} E_{i_j}} f(e) \\
&= \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k \le n} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e) + \sum_{e \in E_{n+1}} f(e) \\
&\quad + \sum_{k=2}^{n+1} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k = n+1} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e) \\
&= \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k < n+1} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e) \\
&\quad + \sum_{k=1}^{n+1} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k = n+1} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e) \\
&= \sum_{k=1}^{n+1} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k \le n+1} \sum_{e \in \bigcap_{j=1}^{k} E_{i_j}} f(e).
\end{aligned}$$

Thus by induction, (1.23) holds for all $m \ge 1$. □

Example 1.6.17. For any finite collection $E_1, \ldots, E_m$ of finite sets we have

$$\left| \bigcup_{i=1}^{m} E_i \right| = \sum_{k=1}^{m} (-1)^{k+1} \sum_{1 \le i_1 < \cdots < i_k \le m} \left| \bigcap_{j=1}^{k} E_{i_j} \right|. \tag{1.24}$$

Alternatively, we can write

$$\left| \bigcup_{i=1}^{m} E_i \right| = \sum_{J \subset \{1,2,\ldots,m\}} (-1)^{|J|-1} \left| \bigcap_{j \in J} E_j \right|. \tag{1.25}$$

1.7 Products and Counting

Computing the temporal and spatial complexities of an algorithm is essentially a big counting problem. Many computations in probability theory also boil down to counting problems (see Chapter 5). In this section we discuss some of the key tools for counting, including the multiplication rule, permutations, combinations, and the binomial theorem.

1.7.1 The Multiplication Rule

We begin by examining the cardinality of a product of sets. Let A and B be sets of finite cardinality $|A|$ and $|B|$, respectively. The *Cartesian product* of A and B, denoted $A \times B$, is the set of ordered pairs

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

A key observation is that the cardinality of $A \times B$ is $|A| \cdot |B|$. This is sometimes called the *multiplication rule*. It is a very useful concept even though its proof is trivial.

Example 1.7.1. Let the set of entrées be $E = \{\text{pizza}, \text{hamburger}, \text{salad}\}$ and the set of drinks be $D = \{\text{water}, \text{soda}\}$. If a meal is defined as a pair of exactly one entrée and exactly one drink, then the number of possible meals is the set of possible pairs $|E \times D| = |E| \cdot |D| = 3 \cdot 2 = 6$.

The multiplication rule generalizes as follows: Suppose that we have the finite sets $A_1, A_2, \ldots, A_n$. The Cartesian product $A_1 \times A_2 \times \cdots \times A_n$ is the set of n-tuples $(a_1, a_2, \ldots, a_n)$ with each $a_i \in A_i$. The cardinality of the Cartesian product is $|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdots |A_n|$.

Example 1.7.2. In the Land of Oz, a license plate has two letters followed by three numbers. By the multiplication rule, the total number of possible license plates in Oz is $26 \cdot 26 \cdot 10 \cdot 10 \cdot 10 = 676{,}000$.

1.7.2 Permutations

A permutation of a set S is an ordering of the elements of S. For example, there are six ways to permute the set $\{1,2,3\}$; they are $(1,2,3)$, $(1,3,2)$, $(3,2,1)$, $(3,1,2)$, $(2,3,1)$, and $(2,1,3)$. More generally, a set of n objects can be permuted $n! = 1 \cdot 2 \cdots (n-1) \cdot n$ ways. The proof follows by induction and is Exercise 1.37.

Example 1.7.3. If there are 19 students in a class, then there are 19! ways that the students can be ordered; that is, there are 19! permutations of the class. This is a very large number—approximately 10^{17}.

Remark 1.7.4. For notational convenience, we set $0! = 1$. This is a standard convention in mathematics.

Sometimes we are only interested in ordering r objects taken from a set of n elements. For example, suppose we wanted to elect a president, a vice president, and a secretary from the class of 19 students. In this case there are 19 choices for president, 18 choices for vice president after the president is chosen, and 17 choices for secretary after the other two are chosen. That gives $19 \cdot 18 \cdot 17 = \frac{19!}{16!} = 5814$ possible presidencies. This could also be thought of as taking the total number of orderings 19! and dividing out the unused orderings 16!. This pattern applies in general, as given in the following proposition.

Proposition 1.7.5. *The number of permutations of r objects from a set of n elements $(0 \leq r \leq n)$ is $P(n,r) = \frac{n!}{(n-r)!}$.*

Proof. The proof is Exercise 1.38. □

1.7.3 Combinations and Rearrangements

Suppose that instead of a presidency, we want to choose a committee of three people from the 19 students in the class. In this case, since none of the three are ranked above any other, we must also divide out the number of orderings of the three people in a given presidency. Thus the number of *combinations* of students on the committee is $\frac{19!}{16!3!} = 969$. This is denoted $C(19,3)$ or $\binom{19}{3}$ and is pronounced "19 choose 3."

Proposition 1.7.6. *The number of (unordered) combinations of r objects from a set of n elements $(0 \leq r \leq n)$ is $C(n,r) = \binom{n}{r} = \frac{n!}{r!(n-r)!}$. For notational convenience, we set $C(n,r) = \binom{n}{r} = 0$ whenever $r > n$ or $r < 0$.*

Proof. The proof is Exercise 1.38. □

Remark 1.7.7. The numbers $\binom{n}{r}$ are often called *binomial coefficients* because of their role in the binomial theorem (Theorem 1.7.16), below. Note that these are always integers because they count the number of times something can occur.

Example 1.7.8. The number of ways to choose two socks, not necessarily matching, from a drawer of 20 socks is $C(20,2) = \frac{20 \cdot 19}{2!} = 190$, because we don't care about the order in which the socks are chosen. But the number of ways to put these socks on your two feet is $P(20,2) = 20 \cdot 19 = 380$ because there are 20 choices for your left foot and 19 remaining for your right foot.

Example 1.7.9. In poker a player draws five cards, without replacement, from a standard deck of 52 cards (four suits and 13 ranks; each card has a rank and a suit). A three-of-a-kind is when there are three cards of the same rank, plus two cards which are not of this rank nor the same rank as each other. How many different ways can one get a three-of-a-kind? There are $C(13,1) = 13$ possible ranks for the triple. For each possible rank, there are $C(4,3) = 4$ different ways you can have three cards of that rank. Thus, there are $13 \cdot 4 = 52$ different triples of the same rank.

For the two extra cards, neither of them can be the same rank as the triple (or else you would have four of a kind), nor can they be the same as each other (or else you would have a full house). The remaining two cards must be from the remaining 12 ranks, which gives $C(12,2) = 66$ possibilities. For each of these two ranks, there can be four different suits. Therefore, the total number of unique three-of-a-kind combinations is $C(13,1)C(4,3)C(12,2)4^2 = 54{,}912$.

Example 1.7.10. In the Powerball Lottery players choose five distinct numbers ranging between 1 and 69 and also choose the Powerball, which is a single number ranging between 1 and 26. Although the balls are drawn one at a time, the numbers are always reported in ascending order and therefore the order drawn doesn't matter. The number of possible unique lottery tickets is the number of Powerball choices (26) times the number of ways to choose five numbers from 69, or $26 \cdot C(69,5) = 292{,}201{,}338$. Since there is only one jackpot, the odds of winning this lottery are 1 in 292,201,338.

Example 1.7.11. How many unique rearrangements are there of the word TOOTH? Five letters can be rearranged 5! ways, but there are two pairs of letters that are multiples; specifically, the letters O and T are represented twice. Thus, we must divide out the number of ways the multiple letters can be permuted among themselves. This gives $\frac{5!}{2!2!} = 30$ different rearrangements.

Example 1.7.12. The number of rearrangements of the word MISSISSIPPI can be counted by noting that the letters I and S are repeated four times, and P is repeated twice. Thus, there are $\frac{11!}{4!4!2!} = 34{,}650$ unique rearrangements.

Remark 1.7.13. An important generalization of the binomial coefficient is the *multinomial coefficient*. Let $n_1 + n_2 + \cdots + n_r = n$, where each $n_i \geq 0$. Define

$$\binom{n}{n_1, n_2, \ldots, n_r} = \frac{n!}{n_1! n_2! \cdots n_r!}. \tag{1.26}$$

It describes the number of ways that n elements can organized into r groups of $n_1, n_2, \ldots, n_r$ elements, respectively.

Example 1.7.14. If there are nine employees at a restaurant, how many ways can you choose four wait staff, two cooks, two bussers, and one host (assuming every employee is able to perform every job)? Using the multinomial, we have $\frac{9!}{4!2!2!1!} = 3780$.

1.7.4 Combinatorial Identities

The binomial coefficients satisfy some useful relations. Among the most famous is Pascal's rule, which is the foundation of Pascal's triangle (see Table 1.1). This is given algebraically in Lemma 1.7.15 and used in the proof of the binomial theorem (Theorem 1.7.16).

$n = 0$:					1					
$n = 1$:				1		1				
$n = 2$:			1		2		1			
$n = 3$:		1		3		3		1		
$n = 4$:	1		4		6		4		1	
$n = 5$:	1		5		10		10		5		1

Table 1.1. *Pascal's triangle can be used to determine the coefficients in binomial expansions. Pascal's lemma (Lemma 1.7.15) says that the rth element of the nth row can be found by adding the two elements (the rth and the $(r-1)$th) just above it on the $(n-1)$th row.*

Lemma 1.7.15 (Pascal's Rule). *For all $n, r \in \mathbb{Z}^+$ with $r \leq n$ we have*

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}. \tag{1.27}$$

Proof.

$$\begin{aligned}\binom{n-1}{r-1}+\binom{n-1}{r} &= \frac{(n-1)!}{(r-1)!(n-r)!}+\frac{(n-1)!}{r!(n-1-r)!}\\ &= \frac{(n-1)!r}{r!(n-r)!}+\frac{(n-1)!(n-r)}{r!(n-r)!}\\ &= \frac{(n-1)!n}{r!(n-r)!}=\frac{n!}{r!(n-r)!}=\binom{n}{r}. \quad \square\end{aligned}$$

Theorem 1.7.16 (Binomial Theorem). *For any $x, y \in \mathbb{F}$ and $n \in \mathbb{Z}^+$ we have*

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}. \tag{1.28}$$

Here, as in all similar sums, we use the convention that $x^0 = 1$, even when $x = 0$.

Proof. The proof follows by induction. We first prove the case $n = 1$. Note that

$$x + y = \binom{1}{0} x^0 y^1 + \binom{1}{1} x^1 y^0.$$

Now, assuming by the inductive hypothesis that the theorem is true for $n-1$, we prove that it holds for n. We have

$$\begin{aligned}(x+y)^n &= (x+y)(x+y)^{n-1}\\ &= (x+y)\sum_{k=0}^{n-1}\binom{n-1}{k}x^k y^{n-1-k}\\ &= \sum_{k=0}^{n-1}\binom{n-1}{k}x^{k+1}y^{n-1-k} + \sum_{k=0}^{n-1}\binom{n-1}{k}x^k y^{n-k}\\ &= \sum_{k=1}^{n}\binom{n-1}{k-1}x^k y^{n-k} + \sum_{k=0}^{n-1}\binom{n-1}{k}x^k y^{n-k}\\ &= x^n + \sum_{k=1}^{n-1}\binom{n-1}{k-1}x^k y^{n-k} + y^n + \sum_{k=1}^{n-1}\binom{n-1}{k}x^k y^{n-k}\\ &= \binom{n}{0}x^0 y^n + \sum_{k=1}^{n-1}\left[\binom{n-1}{k-1}+\binom{n-1}{k}\right]x^k y^{n-k} + \binom{n}{n}x^n y^0\\ &= \sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k}.\end{aligned}$$

Thus (1.28) holds for all $n \in \mathbb{Z}^+$. $\square$

Remark 1.7.17. Note that the binomial theorem gives another way to see that the binomial coefficients must always be integers, because each coefficient of $(x+y)^n$ must be an integer.

Corollary 1.7.18. *For any $n \in \mathbb{Z}^+$, the following combinatorial identity holds:*

$$2^n = \sum_{k=0}^{n} \binom{n}{k}.$$

Proof. From the binomial theorem we have

$$2^n = (1+1)^n = \sum_{k=0}^{n} \binom{n}{k} 1^k 1^{n-k} = \sum_{k=0}^{n} \binom{n}{k}. \quad \square$$

There is also a multinomial theorem that expresses $(x_1 + \cdots + x_d)^n$ in terms of multinomial coefficients.

Theorem 1.7.19. *For any $x_1, \ldots, x_d \in \mathbb{R}$ and $n \in \mathbb{Z}^+$ we have*

$$\left(\sum_{i=1}^{d} x_i \right)^n = \sum_{k_1 + \cdots + k_d = n} \binom{n}{k_1\, k_2 \cdots k_d} x_1^{k_1} x_2^{k_2} \cdots x_d^{k_d}, \tag{1.29}$$

where the sum on the right runs over all d-tuples of nonnegative integers $k_1, \ldots, k_d$ that sum to n, and $\binom{n}{k_1\, k_2 \cdots k_d}$ is the multinomial coefficient (1.26).

Proof. This is proved by inducting on d and using the binomial theorem. The details are Exercise 1.45. $\square$

1.8 Division and Divisors

Some of the most useful algorithms depend on divisibility properties of integers. In this section, we develop these ideas and also discuss the Euclidean algorithm, an ancient and very efficient algorithm for finding the greatest common divisor of two positive integers.

1.8.1 Divisibility and the Division Theorem

The fundamental tool for working with integers is the well-ordering axiom of the natural numbers (see Volume 1, Appendix A, Sections 3–4).

Definition 1.8.1. *A binary relation $\leq$ on a set X is called an* ordering *(or a* total order*) if it satisfies the following properties:*

(i) *For every $x, y \in X$ either $x \leq y$ or $y \leq x$.*

(ii) *$x \leq x$ for every $x \in X$.*

(iii) *If $x \leq y$ and $y \leq x$, then $x = y$.*

(iv) *If $x \leq y$ and $y \leq z$, then $x \leq z$.*

A set X with an ordering $\leq$ is well ordered *if every nonempty subset $S \subset X$ has a least element, that is, if there exists an element $x \in S$ such that $x \leq y$ for every $y \in S$.*

Unexample 1.8.2.

(i) The interval $[0,1] \in \mathbb{R}$ with the usual ordering $\leq$ is not well ordered, because the set $(0,1]$ does not have a least element. For example, given any $x \in (0,1]$ the number $\frac{x}{2}$ is strictly less than x.

(ii) The set of integers $\mathbb{Z}$ with the usual ordering is not well ordered because there is no least element. For every $n \in \mathbb{Z}$ the number $n-1 \in \mathbb{Z}$ is less than n.

Axiom 1.8.3 (Well-Ordering Axiom for Natural Numbers). *The set of natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$ with the usual ordering $\leq$ is well ordered.*

Example 1.8.4. Any subset of a well-ordered set is also well ordered. Hence, the set $\mathbb{Z}^+$ of positive integers is also well ordered.

The well-ordering axiom guarantees that any nonempty subset of $\mathbb{N}$ has a least element. For example, given some property of interest characterized by a Boolean-valued function $P(n)$ on $\mathbb{N}$, that is, $P(n) \in \{\text{True}, \text{False}\}$ for each $n \in \mathbb{N}$, we can let $S = \{n \in \mathbb{N} \mid P(n)\}$ be the set of all natural numbers satisfying that property. The well-ordering axiom guarantees that if S is nonempty, then there is a least element of S, and this must be the smallest natural number satisfying the desired property.

Lemma 1.8.5 (Archimedean Property). *If $a, b \in \mathbb{Z}^+$, then there exists $n \in \mathbb{Z}^+$ such that $bn \geq a$.*

Proof. Suppose no such n exists. Thus, $0 < a - bn$ for each $n \in \mathbb{Z}^+$. Let $S = \{a - bn \mid n \in \mathbb{Z}^+\} \subset \mathbb{Z}^+$. Since S is nonempty, the well-ordering axiom guarantees that S has a least element, say, $a - bm$. It follows that $a - bm \leq a - b(m+1)$, which implies that $b \leq 0$, which is a contradiction. Hence, there exists $n \in \mathbb{Z}^+$ such that $bn \geq a$. □

The well-ordering axiom is also the key to proving several divisibility properties of the integers. The first of these is the division theorem.

Theorem 1.8.6 (Division Theorem). *Given any integer $a \in \mathbb{Z}$ and any nonzero $b \in \mathbb{Z}$, there exist unique integers q, r with $0 \leq r < |b|$ such that*

$$a = bq + r.$$

Moreover, if $a, b > 0$, then $q \geq 0$, and if $a \geq b > 0$, then $q > 0$. We call a the dividend*, b the* divisor*, q the* quotient*, and r the* remainder*.*

Proof. Let $S = \{a - bx \mid x \in \mathbb{Z}\} \subset \mathbb{Z}$ and $S^+ = S \cap \mathbb{N}$. By Exercise 1.47 the subset $S^+ \subset \mathbb{N}$ is nonempty, so the well-ordering axiom implies that S^+ has a least

element $r \geq 0$. Thus, there exists a $q \in \mathbb{Z}$ such that $r = a - bq$. Assume, by way of contradiction, that $r = a - bq \geq |b|$. If $b > 0$, then $r - |b| = a - b(q+1) \geq 0$ is an element of S^+ that is less than r, a contradiction. If $b < 0$, then $r - |b| = a - b(q-1) \geq 0$ is an element of S^+ that is less than r, which is also a contradiction. Therefore, $0 \leq r < |b|$.

To see uniqueness, consider any $r' = a - bq'$ with $0 \leq r' < |b|$. Without loss of generality, assume that $r \leq r'$, and so $0 \leq r' - r < |b|$. But $r' - r = a - bq' - (a - bq) = b(q - q')$ is a multiple of b, and the only nonnegative multiple of b less than $|b|$ is 0. Thus $r' = r$ and $q' = q$.

Finally, if $a \geq b > 0$, then $a - r \geq b - r > 0$, which implies that $q = (a-r)/b > 0$. If $b > a > 0$, then $a = 0 \cdot b + a$, so $q = 0$. □

1.8.2 Greatest Common Divisors

Definition 1.8.7. *Given any $a, b \in \mathbb{Z}$ with $b \neq 0$, we say that b* divides *a (denoted $b|a$) if there exists $c \in \mathbb{Z}$ such that $bc = a$. In this case, we say that b is a* divisor *of a. If b does not divide a we write $b \nmid a$.*

Example 1.8.8. We have $2|18$ but $6 \nmid 13$.

Theorem 1.8.9. *Given $a, b \in \mathbb{Z}$, not both zero, there is a unique $d \in \mathbb{Z}^+$ satisfying the following properties:*

(i) *d is the least positive integer that can be written in the form $ax + by$ for some $x, y \in \mathbb{Z}$.*

(ii) *The integer d divides both a and b, that is, $d|a$ and $d|b$.*

(iii) *For any integer d' with $d'|a$ and $d'|b$, we have $d'|d$.*

(iv) *d is the greatest positive integer that divides both a and b.*

The integer d is called the greatest common divisor (gcd) *of a and b and is denoted* $\gcd(a, b)$.

Proof. Let $S = \{an + bm \mid n, m \in \mathbb{Z}\}$ and let $S^+ = S \cap \mathbb{Z}^+$. It is straightforward to see that $S^+ \neq \emptyset$. By the well ordering of $\mathbb{Z}^+$, there must be a least element in S^+; let d be that least element.

(i) By definition, $d = ax + by$ for some $x, y \in \mathbb{Z}$ and is the least such element.

(ii) By the division theorem, there are integers q, r with $0 \leq r < d$ such that $a = dq + r$, but $r = a - dq = a(1 - xq) - byq$ is either 0 or an element of S^+. If r is an element of S^+, then since d is the least element in S^+, we must have $d \leq r$, which is a contradiction. Therefore $r = 0$, and d divides a. Exchanging a for b in the previous argument proves that d also divides b.

(iii) Since $d'|a$ and $d'|b$ we must have $a = d's$ and $b = d't$ for some $s, t \in \mathbb{Z}$, and hence $d = ax + by = d'sx + d'ty = d'(sx + ty)$, so $d'|d$.

(iv) This follows from (ii) and (iii). □

Example 1.8.10. In elementary school, we find the gcd by factoring the two numbers into products of primes and then collecting the common factors. For example, to compute the $\gcd(12, 20)$ we write $12 = 2^2 \cdot 3$ and $20 = 2^2 \cdot 5$, and so the $\gcd(12, 20) = 2^2 = 4$. But factoring is a very expensive algorithm. In fact its high complexity is the foundation of many cryptosystems; see Section 1.9.7 for more on this. In Section 1.8.3 we present a much faster way to find the gcd.

Proposition 1.8.11. *Let $a, b, c \in \mathbb{Z}$. If $a|bc$ and $\gcd(a, b) = 1$, then $a|c$.*

Proof. Since $\gcd(a, b) = 1$ there exist $x, y \in \mathbb{Z}$ such that $ax + by = 1$. Multiplying by c gives $axc + byc = c$. Since $a|bc$ we have $az = bc$ for some $z \in \mathbb{Z}$, and hence

$$c = axc + byc = a(xc + zy).$$

Therefore $a|c$ as required. □

1.8.3 The Euclidean Algorithm

The gcd can be found very efficiently by way of the *Euclidean algorithm*. This is one of the most ancient algorithms still in modern use. It was described by Euclid in his book *Elements* around 300 BCE, but many scholars believe it was known earlier.

Theorem 1.8.12 (The Euclidean Algorithm). *Given $a, b \in \mathbb{Z}$ with $b \neq 0$, define $q_0, r_0 \in \mathbb{Z}$ as in the division theorem (Theorem 1.8.6) to get*

$$a = bq_0 + r_0$$

with $0 \leq r_0 < |b|$. If $r_0 = 0$, then $\gcd(a, b) = b$. Otherwise, divide b by r_0 to get $q_1, r_1 \in \mathbb{Z}$ by the division algorithm; that is,

$$b = r_0 q_1 + r_1$$

with $0 \leq r_1 < r_0$. Repeating the process eventually gives a remainder of zero:

$$\begin{aligned} a &= bq_0 + r_0, \\ b &= r_0 q_1 + r_1, \\ r_0 &= r_1 q_2 + r_2, \\ r_1 &= r_2 q_3 + r_3, \\ &\vdots \\ r_{n-2} &= r_{n-1} q_n + r_n, \\ r_{n-1} &= r_n q_{n+1} + 0. \end{aligned} \tag{1.30}$$

The penultimate remainder r_n is the gcd of a and b, that is,

$$\gcd(a, b) = r_n.$$

Proof. For any two integers m and n with $n \neq 0$, let $m = nq + r$ with $0 \leq r < |n|$. Let $d = \gcd(m, n)$ and $e = \gcd(n, r)$. Notice that $r = m - nq$, and $d|m$ and $d|n$, so $d|r$, and hence $d \leq e$. Conversely, since $m = nq + r$, we have $e|m$ and $e|n$, so $e \leq d$. Therefore $\gcd(m, n) = \gcd(n, r)$. Applying this result to each successive division in (1.30) shows that

$$\gcd(a, b) = \gcd(b, r_0) = \cdots = \gcd(r_{n-1}, r_n) = \gcd(r_n, 0) = r_n.$$

The algorithm terminates with $n < |b| - 1$, because at each stage we have $0 \leq r_{k+1} < r_k$, and so $0 < r_n < \cdots < r_1 < r_0 < |b|$. □

Example 1.8.13. The gcd of 14562 and 348 is computed as follows:

$$\begin{aligned} 14562 &= 348 \cdot 41 + 294, \\ 348 &= 294 \cdot 1 + 54, \\ 294 &= 54 \cdot 5 + 24, \\ 54 &= 24 \cdot 2 + 6, \\ 24 &= 6 \cdot 4 + 0. \end{aligned}$$

Thus, $\gcd(14562, 348) = 6$.

The bound $n < |b| - 1$ in the previous proof can be improved a lot, as the following lemma shows.

Lemma 1.8.14. *Following the notation in the previous theorem, we have $b > 2r_1$ and $r_k > 2r_{k+2}$ for each $k \in \{0, 1, 2, \ldots, n-2\}$.*

Proof. Since $r_{k+1} = q_{k+3}r_{k+2} + r_{k+3}$ and $r_k = q_{k+2}r_{k+1} + r_{k+2}$, we have that $r_k = r_{k+2}(1 + q_{k+3}q_{k+2}) + q_{k+2}r_{k+3}$. Thus, $r_k > r_{k+2}(1 + q_{k+3}q_{k+2})$. Since $r_{j+1} < r_j$ for every $j \in \{0, \ldots, n-1\}$, we have $q_{k+2} \geq 1$ and $q_{k+3} \geq 1$, hence $r_k \geq 2r_{k+2}$. To show that $b > 2r_1$, set $r_{-1} = b$ and use the same argument. □

Theorem 1.8.15. *Using the notation in the previous theorem, assume $a, b \in \mathbb{Z}^+$ satisfy $b < a$. The number $n+1$ of iterations of the Euclidean algorithm for $\gcd(a, b)$ is bounded above by $2(\log_2 b) + 1$; that is, $n \leq 2 \log_2 b$.*

Proof. Choose $m \in \mathbb{Z}^+$ so that $2^{m-1} \leq b < 2^m$. Suppose that $n \geq 2m - 1$. From the lemma, we have

$$b > 2r_1 > 4r_3 > \cdots > 2^m r_{2m-1} \geq 2^m r_n.$$

Thus, we have that $r_n \leq 2^{-m} b < 1$, which is impossible. Thus, $n < 2m - 1$, which implies $n \leq 2(m-1) \leq 2 \log_2 b$. □

Remark 1.8.16. The number of iterations of the Euclidean algorithm is at most $\sim 2\log_2 b$. Normally we think of complexity in terms of the number of digits required to store or represent the inputs, not the numerical value of the input. So in a base-2 representation of b, the size of the input is $\ell = \lceil \log_2 b \rceil$. If the complexity of each iteration is Q, then the temporal complexity of the Euclidean algorithm is $\sim 2\ell Q$. If the input is represented in base 10 instead, then

$$n \leq 2\log_2 b = 2\frac{\log_{10} b}{\log_{10} 2} \approx 6.644 \log_{10} b \leq 6.644 \times \text{number of digits of } b.$$

This is still a crude estimate—there are much sharper bounds in the literature.

1.8.4 Extended Euclidean Algorithm

Theorem 1.8.9 guarantees that, for any nonzero integers a and b, the element $\gcd(a, b)$ can be written as $ax+by$ for some $x, y \in \mathbb{Z}$. Knowing the actual values of x and y is useful in many applications. This can be found easily, by back-substituting in the original Euclidean algorithm.

Example 1.8.17. In Example 1.8.13, we computed $\gcd(14562, 348) = 6$ as

$$14562 = 348 \cdot 41 + 294, \tag{1.31}$$
$$348 = 294 \cdot 1 + 54, \tag{1.32}$$
$$294 = 54 \cdot 5 + 24, \tag{1.33}$$
$$54 = 24 \cdot 2 + 6, \tag{1.34}$$
$$24 = 6 \cdot 4 + 0. \tag{1.35}$$

We work from the bottom up and solve for each remainder in terms of the other parts and then back substitute. Equation (1.34) gives

$$6 = 54 - 24 \cdot 2 \tag{1.36}$$

and (1.33) gives

$$24 = 294 - 54 \cdot 5. \tag{1.37}$$

Substituting (1.37) into (1.36) gives

$$6 = 54 - (294 - 54 \cdot 5) \cdot 2 = 11 \cdot 54 - 2 \cdot 294. \tag{1.38}$$

Solving for 54 in (1.32) gives

$$54 = 348 - 294 \cdot 1, \tag{1.39}$$

and substituting (1.39) into (1.38) gives

$$6 = 11 \cdot (348 - 294 \cdot 1) - 2 \cdot 294 = 11 \cdot 348 - 13 \cdot 294. \tag{1.40}$$

Solving for 294 in (1.31) gives

$$294 = 14562 - 348 \cdot 41, \tag{1.41}$$

and substituting into (1.40) gives

$$6 = 11 \cdot 348 - 13 \cdot (14562 - 348 \cdot 41) = 544 \cdot 348 - 13 \cdot 14562. \tag{1.42}$$

This gives the desired expression for $6 = \gcd(14562, 348)$ as an integer combination of 14562 and 348.

Writing out the equations for this procedure, we have $r_n = r_{n-2} - r_{n-1}q_n$, $r_{n-1} = r_{n-3} - r_{n-2}q_{n-1}$, and so forth, up to $r_0 = a - bq_0$. The initial Euclidean algorithm finds all the values of q_k, so back substituting gives

$$\begin{aligned} \gcd(a,b) = r_n &= r_{n-2} - r_{n-1}q_n \\ &= r_{n-2} - (r_{n-3} - r_{n-2}q_{n-1})q_n \\ &= (r_{n-4} - r_{n-3}q_{n-2}) - (r_{n-3} - (r_{n-4} - r_{n-3}q_{n-2})q_{n-1})q_n \\ &\;\;\vdots \end{aligned}$$

and this gives an explicit expression for $\gcd(a,b) = r_n$ as $ax + by$. This is called the *extended Euclidean algorithm*. It may feel painful to write out all the equations for the extended Euclidean algorithm, but it is simple to program.

1.9 Primes and Remainders

In this section, we treat basic properties of prime numbers and modular arithmetic. We prove *Fermat's little theorem*, which follows from the binomial theorem and gives a very fast algorithm for determining when a given number is likely to be prime. We also discuss the Rivest–Shamir–Adleman (RSA) cryptosystem.

1.9.1 Primes

Definition 1.9.1. *If two integers $a, b \in \mathbb{Z}$ satisfy $\gcd(a,b) = 1$, then we say that a and b are* relatively prime. *An integer $p > 1$ is* prime *if it is relatively prime to every $a \in \{1, 2, \ldots, p-1\}$.*

Example 1.9.2.

(i) We have $\gcd(6,9) = 3$ and $\gcd(4,9) = 1$, so 4 and 9 are relatively prime, but 6 and 9 are not.

(ii) If p is any prime, and $b \in \mathbb{Z}$, then either $\gcd(p,b) = 1$ or $\gcd(p,b) = p$, since the only positive divisors of p are 1 and p.

1.9.2 Modular Arithmetic

Definition 1.9.3. *Given $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$, we say a* is congruent to b modulo n, *denoted $a \equiv b \pmod{n}$, if $n|(a-b)$.*

Example 1.9.4. The following statements are true:

$$\begin{aligned} 37 &\equiv 25 \pmod{12}, \\ 37 &\equiv 1 \pmod{12}, \\ -9 &\equiv 31 \pmod{10}, \\ 5 &\not\equiv 7 \pmod{3}. \end{aligned}$$

Example 1.9.5. You've been using modular arithmetic since you learned to tell time. The minutes on a clock are measured modulo 60. For example, 45 minutes after the hour is the same as 15 minutes before the hour; that is, $45 \equiv -15 \pmod{60}$.

Theorem 1.9.6. *Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$. The relation $a \equiv b \pmod{n}$ is an equivalence relation on $\mathbb{Z}$.*

Proof. It suffices to show that $\equiv$ is reflexive, symmetric, and transitive. Reflexivity follows because $n|0$ always holds. Symmetry follows because $n|(a-b)$ holds if and only if $n|(b-a)$. Finally transitivity follows from the fact that if $n|(a-b)$ and $n|(b-c)$, then $n|[(a-b)+(b-c)]$, hence $n|(a-c)$. □

Definition 1.9.7. *The set of equivalence classes in $\mathbb{Z}$ defined by congruence $\pmod{n}$ is denoted $\mathbb{Z}_n$. The equivalence classes are also called* cosets.

The equivalence class of x is denoted $[\![x]\!]$. The equivalence classes are the sets

$$\begin{aligned} [\![0]\!] &= \{0, \pm n, \pm 2n, \pm 3n, \dots\}, \\ [\![1]\!] &= \{1, 1 \pm n, 1 \pm 2n, 1 \pm 3n, \dots\}, \\ [\![2]\!] &= \{2, 2 \pm n, 2 \pm 2n, 2 \pm 3n, \dots\}, \\ &\vdots \\ [\![n-1]\!] &= \{n-1, (n-1) \pm n, (n-1) \pm 2n, (n-1) \pm 3n, \dots\}. \end{aligned}$$

Each equivalence class in $\mathbb{Z}_n$ has a unique representative in the set $\{0, 1, \dots, n-1\}$. As a result, when it can be done without introducing ambiguity, we often abuse notation and leave off the $[\![\cdot]\!]$ and write 1 to mean $[\![1]\!]$, 5 to mean $[\![5]\!]$, etc.

Remark 1.9.8. It is important to remember that each element of $\mathbb{Z}_n$ is an entire coset of numbers. We can write these cosets by choosing any element of the coset, e.g., $[\![-1]\!] = [\![n-1]\!] = [\![-3n-1]\!]$, $[\![-2]\!] = [\![n-2]\!] = [\![6n-2]\!]$, etc.

Theorem 1.9.9. *If $a, b, c \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$, then*

(i) $(a+b)+c \equiv a+(b+c) \pmod{n}$,

(ii) $(ab)c \equiv a(bc) \pmod{n}$,

(iii) $a+b \equiv b+a \pmod{n}$,

(iv) $ab \equiv ba \pmod{n}$,

(v) $a(b+c) \equiv ab+ac \pmod{n}$.

Proof. The proof is Exercise 1.54. □

The previous theorem, combined with the substitution rule, below, makes computation in $\mathbb{Z}_n$ much simpler than computation in $\mathbb{Z}$.

Theorem 1.9.10 (Substitution Rule). *Let $a, b, a', b' \in \mathbb{Z}$, and $n \in \mathbb{Z}^+$. If $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then*

(i) $a+b \equiv a'+b' \pmod{n}$,

(ii) $ab \equiv a'b' \pmod{n}$.

Proof. If $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then $n|(a-a')$ and $n|(b-b')$. This implies there exist $c, d \in \mathbb{Z}$ such that $a = a' + nc$ and $b = b' + nd$.

(i) Adding gives $a+b = a'+b'+n(c+d)$. Thus, $a+b \equiv a'+b' \pmod{n}$.

(ii) Multiplying gives $ab = a'b' + n(cb' + a'd + ncd)$. Thus, $ab \equiv a'b' \pmod{n}$. □

Example 1.9.11. Since $31 \equiv 4 \pmod{9}$ and $66 \equiv 3 \pmod{9}$, we have

$$97 = 31 + 66 \equiv 4 + 3 \equiv 7 \pmod{9},$$
$$2046 = 31 \cdot 66 \equiv 4 \cdot 3 \equiv 12 \equiv 3 \pmod{9}.$$

1.9.3 Fast Modular Exponentiation

We can also compute $m^k \pmod{n}$ using Theorem 1.9.10. For example, to compute $37^{81} \pmod{11}$, note that $37 \equiv 4 \pmod{11}$. So it suffices to find $4^{81} \pmod{11}$. We apply the theorem multiple times in an improvised way to get

$$4^{81} \equiv (4^3)^{27} \equiv (-2)^{27} \equiv (-2)(-2)^{26} \equiv (-2) \cdot 4^{13} \equiv (-8) \cdot 4^{12} \equiv (-8) \cdot 16^6$$
$$\equiv (-8) \cdot 5^6 \equiv (-8) \cdot 25^3 \equiv -8 \cdot 3^3 \equiv (-8) \cdot 5 \equiv -40 \equiv 4 \pmod{11}.$$

We can do this more efficiently using a technique called *fast modular exponentiation*. To compute $m^k \pmod{n}$, find $m \equiv a_0 \pmod{n}$ and then square both sides to get $m^2 \equiv (a_0)^2 \equiv a_1 \pmod{n}$, and keep squaring to get each $a_{j+1} \equiv a_j^2 \pmod{n}$ until $k < 2^{j+1}$. At this point we can write m^k as a product of some combination[11] of the a_j.

[11] A little thought shows that the particular terms a_j appearing in the product are determined by the binary expansion of k.

Example 1.9.12. We compute $37^{81} \equiv 4 \pmod{11}$. We begin with $37 \equiv 4 \pmod{11}$, which gives $a_0 = 4$. Taking powers of both sides yields

$$\begin{aligned} 37^2 &\equiv 4^2 \equiv 16 \equiv 5 = a_1, \\ 37^4 &\equiv 5^2 \equiv 25 \equiv 3 = a_2, \\ 37^8 &\equiv 3^2 \equiv 9 = a_3, \\ 37^{16} &\equiv 9^2 \equiv 81 \equiv 4 = a_4, \\ 37^{32} &\equiv 4^2 \equiv 16 \equiv 5 = a_5, \\ 37^{64} &\equiv 5^2 \equiv 25 \equiv 3 = a_6. \end{aligned}$$

Thus, $37^{81} \equiv 37^{64+16+1} \equiv 37^{64} \cdot 37^{16} \cdot 37^1 \equiv a_6 \cdot a_4 \cdot a_1 = 3 \cdot 4 \cdot 4 \equiv 4 \pmod{11}$.

Remark 1.9.13. Fast modular exponentiation requires $O(\lfloor \log_2 k \rfloor)$ integer multiplications. This is considered fast because it uses many fewer multiplications than the naïve approach of multiplying by m repeatedly (which requires $O(k)$ multiplications).

1.9.4 Finding Inverses in $\mathbb{Z}_n$

If a and n are relatively prime, then there exist $x, y \in \mathbb{Z}$ such that $ax + ny = 1$. This can be rewritten as $ax - 1 = -ny$, and therefore $ax \equiv 1 \pmod{n}$. This implies that x is a *multiplicative inverse* to a in $\mathbb{Z}_n$. In particular, given any relation of the form $az \equiv w \pmod{n}$, we can easily find z by multiplying both sides by x, that is,

$$z \equiv (xa)z \equiv x(az) \equiv xw \pmod{n}.$$

Example 1.9.14. To find an integer z satisfying

$$31z \equiv 17 \pmod{56},$$

note that 31 and 56 are relatively prime, so there exist integers x and y such that $31x + 56y = 1$. We can use the extended Euclidean algorithm to find $x = -9$ and $y = 5$. This implies that

$$(31)(-9) \equiv 1 \pmod{56},$$

and hence

$$(31)(-9)(17) \equiv 17 \pmod{56}.$$

Therefore $z = (-9)(17) \equiv 15 \pmod{56}$ is a solution.

1.9.5 Fermat's Little Theorem

Fermat's little theorem is much more useful than his last theorem. It's also much easier to prove.

Lemma 1.9.15. *Assume $p \in \mathbb{Z}^+$ is prime. If $k \in \mathbb{Z}^+$ with $k < p$, then $p|\binom{p}{k}$. In other words, $\binom{p}{k} \equiv 0 \pmod{p}$ for $k = 1, 2, \ldots, p-1$.*

Proof. We have $\binom{p}{k} = \frac{p!}{k!(p-k)!} \in \mathbb{Z}$, and p divides $p! = \binom{p}{k}k!(p-k)!$. But p is relatively prime to $k!(p-k)!$, so by Proposition 1.8.11 we must have $p|\binom{p}{k}$. □

Corollary 1.9.16 (Freshman's Dream). *If p is prime, then for $a, b \in \mathbb{Z}$ we have $(a+b)^p \equiv a^p + b^p \pmod{p}$.*

Proof. By the binomial theorem and the previous lemma we have

$$(a+b)^p = \sum_{j=0}^{p} \binom{p}{j} a^{p-j}b^j \equiv a^p + 0a^{p-1}b + \cdots + 0ab^{p-1} + b^p \equiv a^p + b^p \pmod{p}. \quad \square$$

Theorem 1.9.17 (Fermat's Little Theorem). *If $p \in \mathbb{Z}^+$ is prime, then $a^p \equiv a \pmod{p}$ for all $a \in \mathbb{Z}$.*

Proof. Assume p is prime. Define $S = \{a \in \mathbb{Z} \mid a^p \equiv a \pmod{p}\}$. We first prove the theorem for nonnegative integers a by using induction to show $\mathbb{N} \subset S$. We know $0, 1 \in S$ because $0^p \equiv 0 \pmod{p}$ and $1^p \equiv 1 \pmod{p}$. Assuming $k \in S$, we show that $k+1 \in S$. By the freshman's dream, we have

$$(k+1)^p \equiv 1^p + k^p \equiv 1 + k^p \pmod{p}.$$

By the inductive hypothesis, $k^p \equiv k \pmod{p}$; therefore, $k+1 \in S$. By induction, we have $a \in S$ for all $a \in \mathbb{N}$.

The theorem also holds for negative integers because if $a < 0$, then $a \equiv r \pmod{p}$ for some $0 \leq r < p$; thus $a^p \equiv r^p \equiv r \equiv a \pmod{p}$. □

Corollary 1.9.18. *If $p \in \mathbb{Z}^+$ is prime and $a \in \mathbb{Z}$ with $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof. By the theorem, we have that $a^p \equiv a \pmod{p}$. Thus, there exists $n \in \mathbb{Z}$ such that $np = a(a^{p-1} - 1)$. Since $\gcd(a, p) = 1$, we must have $p|(a^{p-1} - 1)$ or, equivalently, $a^{p-1} \equiv 1 \pmod{p}$. □

Corollary 1.9.19. *If $p \in \mathbb{Z}^+$ is prime, and $x \equiv 1 \pmod{p-1}$, then $a^x \equiv a \pmod{p}$ for all $a \in \mathbb{Z}$.*

Proof. The proof is Exercise 1.61. □

Example 1.9.20. Consider again the problem from Example 1.9.12, where we compute $37^{81} \pmod{11}$. Since 11 is prime, Corollary 1.9.18 implies that $37^{10} \equiv 1 \pmod{11}$. Thus, $37^{81} \equiv 37 \cdot (37^{10})^8 \equiv 4 \cdot 1^8 \equiv 4 \pmod{11}$.

1.9.6 *Application: Primality Testing

In cryptography, it is often important to determine whether a given number n is prime and, if not, then to factor it into a product of primes. One obvious way to do both of these tasks is to attempt to factor n by each of the primes of size less than or equal to $\sqrt{n}$. If none of these primes is a factor, then n is also prime. This algorithm is prohibitively time consuming when n is large, as are the most sophisticated factoring algorithms currently available.

The strength of the widely used Rivest–Shamir–Adleman public-key encryption method is based on the presumption that it is very time consuming to factor a product of two large primes. For example, if the product is a few hundred digits long, then the fastest factoring methods will take years to factor n with today's fastest supercomputers.

But there are much faster tests that merely determine whether a given integer n is prime, rather than factoring it. And tests that determine whether a given number is likely (but not guaranteed) to be prime are faster still. One probabilistic test for primality is to use Corollary 1.9.18, which shows that $a^{n-1} \equiv 1 \pmod{n}$ holds whenever n is prime and $a < n$. If this equivalence fails for even a single value of a, then n is composite and the test is concluded. If equivalence holds for several values of a, we start to gain confidence that n is likely to be prime. Of course, this approach does not prove that n is prime. So even though many people call this a test of primality, it's really a test of whether a given number is composite, and repeated failure to show that a number is composite suggests that it is likely prime.

There is a class of composite numbers, called *Carmichael numbers*, that satisfy $a^{n-1} \equiv 1 \pmod{n}$ for all a; the smallest of these is $n = 561 = 3 \cdot 11 \cdot 17$. However, these numbers are very rare. For example, there are only 8,220,777 Carmichael numbers between 1 and 10^{20}, but there are roughly 2.17×10^{18} prime numbers in the same range. Thus, even if we don't explicitly account for the Carmichael numbers, the chances of accidentally getting a Carmichael number are extremely small when testing random integers for primality. Note that there are even better probable primality tests with no Carmichael equivalent.

1.9.7 *Application: RSA Cryptography

The Rivest–Shamir–Adleman (RSA) cryptosystem is widely used in network security. It works by choosing a pair of distinct, large primes p and q, setting $n = pq$, and finding two positive integers e and d so that $m^{ed} \equiv m \pmod{n}$ for every $m \in \mathbb{Z}$ (as described below). The numbers n and e are made public, but d is kept private. Assuming a message is expressed as an integer m, anyone can encrypt the message by computing the ciphertext $c \equiv m^e \pmod{n}$, but only the person holding the secret key d can decrypt the message by computing $c^d \equiv (m^e)^d \equiv m \pmod{n}$.

Given $n = pq$, the number e can be any integer that is relatively prime to $\ell = (p-1)(q-1)$. The private key d is chosen to satisfy $de \equiv 1 \pmod{\ell}$ (see Section 1.9.4). In the theorem below, we show that for any $x \equiv 1 \pmod{\ell}$ we have $m^x \equiv m \pmod{n}$ for all $m \in \mathbb{Z}$. This yields $c^d \equiv (m^e)^d \equiv m^{ed} \equiv m \pmod{n}$. Thus the original message is recovered.

The system relies upon the fact that it is easy to compute $m^e \pmod{n}$ and $c^d \pmod{n}$ using fast modular exponentiation, but it is numerically prohibitive to compute m when all you know is c, e, and n. In particular, for large n there is no

known feasible way to factor it into the two primes $n = pq$, and therefore there is no known way to compute ℓ.

The next theorem is the key to decoding RSA. It is an easy corollary of Fermat's little theorem (or, rather, of Corollary 1.9.19).

Theorem 1.9.21. *If $p, q \in \mathbb{Z}^+$ are distinct primes and $x \in \mathbb{Z}$ is such that $x \equiv 1 \pmod{(p-1)(q-1)}$, then for any $m \in \mathbb{Z}$ we have $m^x \equiv m \pmod{pq}$.*

Proof. Since $x \equiv 1 \pmod{(p-1)(q-1)}$, we have $x \equiv 1 \pmod{p-1}$ and $x \equiv 1 \pmod{q-1}$. By Corollary 1.9.19 we have $m^x \equiv m \pmod{p}$, which implies that $p|(m^x - m)$. Similarly, we have $q|(m^x - m)$, and thus, since $\gcd(p, q) = 1$, we must have $pq|(m^x - m)$. □

Example 1.9.22. Let $p = 17$ and $q = 13$, which gives $n = 221$ and $\ell = 192 = 2^6 \cdot 3$. We choose $e = 7$ and verify that $\gcd(7, 192) = 1$. The extended Euclidean algorithm gives $1 = (55)7 + (-2)192$, which implies $d = 55$.

If the message is $m = 191$, then using fast modular exponentiation we find $c \equiv m^e \equiv 191^7 \pmod{221}$. To do this we compute

$$191^2 \equiv 16 \pmod{221} \qquad \text{and} \qquad 191^4 \equiv 16^2 \equiv 35 \pmod{221},$$

which implies $c \equiv 191^7 \equiv 191^{4+2+1} \equiv 35 \cdot 16 \cdot 191 \equiv 217 \pmod{221}$.

To decrypt the message we compute $c^d \equiv m^{ed} \equiv m \pmod{221}$ via fast modular exponentiation, again, which gives

$$\begin{aligned} 217^2 &\equiv 16 \pmod{221}, \\ 217^4 &\equiv 16^2 \equiv 35 \pmod{221}, \\ 217^8 &\equiv 35^2 \equiv 120 \pmod{221}, \\ 217^{16} &\equiv 120^2 \equiv 35 \pmod{221}, \\ 217^{32} &\equiv 35^2 \equiv 120 \pmod{221}. \end{aligned}$$

Thus $217^{55} \equiv 217^{32+16+4+2+1} \equiv 191 \pmod{221}$.

1.10 Divide and Conquer

An algorithm is *recursive* when it divides a larger problem into one or more subproblems and then reapplies itself on the subproblems, dividing them further, and so on, until the individual pieces are reduced to some simple base cases. Recursive algorithms are sometimes called *divide-and-conquer* algorithms because of the way they continually divide larger problems into smaller, more conquerable problems.

In this section we examine a few recursive algorithms and then present the master theorem, which gives a general rule for computing big-O bounds on divide-and-conquer algorithms. We prove the master theorem in the next section, after giving a few more examples of its use. The master theorem does not tell us anything

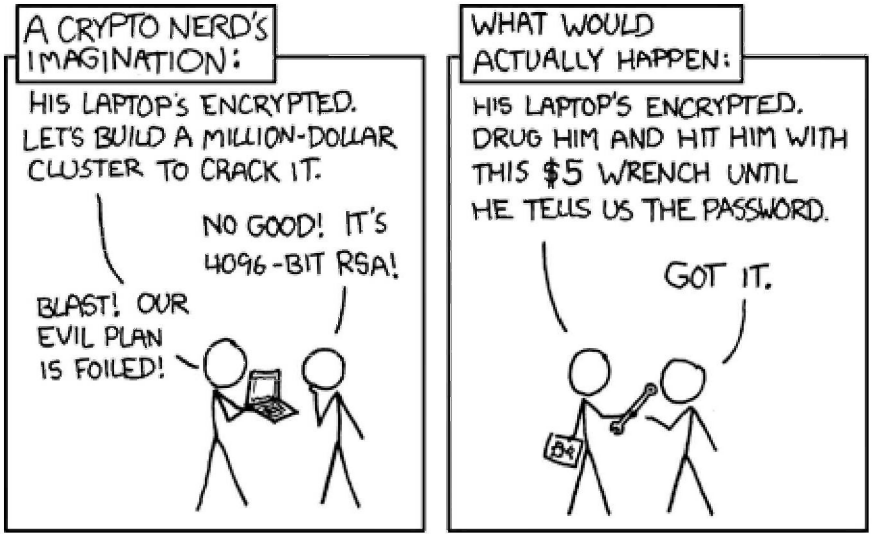


Figure 1.5. *Cryptographic security. Source: XKCD, Randall Munroe.* ***http://xkcd.com/538/***

about the leading-order behavior of these algorithms beyond giving a big-O bound. For the leading-order behavior more in-depth analysis is necessary.

1.10.1 Examples of Recursive Algorithms

Recursive Merge

Recall the merge algorithm (Algorithm 1.3) in Section 1.2.2. We can use recursion to give a different algorithm for merging together two ordered lists, as follows: Take the first entry of each list, make a comparison, take the smaller of the two entries off its list, and then reapply the merge function to the two lists again; see Algorithm 1.7, below. The temporal complexity $T(n)$ of this algorithm satisfies the equation

$$T(n) = T(n-1) + c, \tag{1.43}$$

where c is a constant representing the temporal complexity of one recursion step and n is the sum of the lengths of the two lists. It is easy to see that (1.43) has temporal complexity of $O(n)$ (see Exercise 1.66) since we have

$$T(n) = T(n-1) + c = T(n-2) + 2c = \cdots = T(0) + nc.$$

The spatial complexity is a little more difficult to compute, because it depends on whether the algorithm makes a copy of the data each time it is called. If the data are duplicated each time, then the spatial complexity $S(n)$ satisfies

$$S(n) = S(n-1) + O(n). \tag{1.44}$$

Here the $O(n)$ occurs because we need to store the initial two lists and the output list, all of which are $O(n)$. This shows that the total spatial complexity of this

```python
def merge(K, L):
    """Merge two sorted lists K and L into a new sorted list.
    """

    # Base case: a list is empty
    if K == [] or L == []:
        return K + L

    # Recursive cases
    elif K[0] <= L[0]:
        return [K[0]] + merge(K[1:], L)
    else:
        return [L[0]] + merge(K, L[1:])
```

Algorithm 1.7. *Recursive routine for merging two sorted lists of numbers together into a single sorted list.*

algorithm is $O(\sum_{k=1}^{n} k) = O(n^2)$. There are ways to do this more efficiently, for example, by passing only some pointers to a location in the original lists—this can bring the spatial complexity back down to $O(n)$.

Recursive Addition

Addition can also be written as a recursive algorithm that cuts off the first two digits (in the ones place), adds them together, and then appends the result to the sum of the truncated addends, carrying if necessary; see Algorithm 1.8. If the larger of the two addends has no more than n digits, then the temporal complexity $T(n)$ of this recursive addition algorithm satisfies (1.43). As a result, the temporal complexity is also $O(n)$. It is straightforward to see that the spatial complexity $S(n)$ satisfies (1.44), and thus $S(n) \in O(n^2)$, but, again, this can be changed to $O(n)$ by using pointers instead of duplicating the data at each step.

Binary Search

A linear search algorithm is one that starts at the beginning of the list and checks each entry in succession until the desired element is found or until the list is exhausted. It is usually assumed that the input list is unsorted, and thus one has no choice but to search sequentially for the entry.

If a list is length n, then, on average, half the list must be examined in order to find the match and so the average run time grows linearly in n. The worst-case scenario is that the desired entry is either the last entry or nowhere present in the list. In either case, every entry is checked and thus the algorithm is $O(n)$, both spatially and temporally.

Linear search is a relatively slow method for searching a list. If the input list is sorted, there is a much faster way to search through it, called *binary search*, which works as follows. First check whether the target value is greater than, equal to, or less than the middle entry in the list. If equal, terminate the search and return the

```python
def add(a, b, carry=0):
    """Add two numbers together recursively, where
    each input is a list of single-digit integers.
    """

    # Recursive case 1: both lists are nonempty
    if a != [] and b != []:
        # Add the rightmost digits and recurse on the rest
        temp = a[-1] + b[-1] + carry
        return add(a[:-1], b[:-1], temp//10)+[temp%10]

    # Recursive case 2: one list empty but must carry
    elif carry:
        return add(a+b, [carry], 0)

    # Base case: one list is empty and carry is 0
    else:
        return a + b
```

Algorithm 1.8. *Recursive routine for adding two lists of digits together.*

location. If less, then do a binary search on the first half of the list. If larger, then do a binary search on the second half. Repeat, halving the list at each step, until the match is found or until the list is exhausted. This is implemented in Algorithm 1.9.

This algorithm has temporal complexity $O(\log n)$ because it needs at most k iterations, where $2^{k-1} \leq n < 2^k$. To leading order, the spatial complexity at the first iteration is $\sim n$ because the initial list must be stored, plus a few constant-length variables (`left`, `right`, and `midpoint`). At each subsequent iteration the same list is passed to the algorithm, and, at least in Python, this does not require more memory, so the spatial complexity satisfies $S(n) \sim n + \sum_{j=1}^{k} c$, where the number k of iterations is less than $\log_2(n)$, and c is a constant (corresponding to the constant number of constant-length variables). Hence $S(n) \sim n + c\log_2(n) \sim n$.

Remark 1.10.1. This particular implementation of the binary search algorithm reuses the same list at each iteration, which makes it much more efficient than it would be if it passed a new list (or sublist) to each subsequent iteration (as is done in Algorithms 1.7 and 1.8). If, instead, it passed new sublists at each iteration (for example, `my_list[left:midpoint-1]`), then a new copy of the sublist would be stored at each step and the spatial complexity would instead satisfy

$$S(n) \sim n + S\left(\frac{n}{2}\right) = n + \sum_{j=1}^{k} 2^{-j} n = n + n(1 - 2^{-k}) \sim 2n.$$

1.10.2 Master Theorem

The master theorem gives general upper bounds on the complexity of a large class of recursive algorithms. This theorem applies to many of the most important classical

```python
def binary_search(mylist, target, left=0, right=None):
    """
    Search a sorted list 'mylist' for 'target'.
    Return the index if a match and -1 if no match
    """

    # Set initial variables
    if right is None:
        right = len(mylist) - 1
    midpoint = (left + right) // 2

    # Failed search of entire list
    if left > right:
        return -1

    # If found the target in the list
    if target == mylist[midpoint]:
        return midpoint

    # Search the left half of the list
    elif target < mylist[midpoint]:
        return binary_search(mylist, target, left, midpoint-1)

    # Search the right half of the list
    else:
        return binary_search(mylist, target, midpoint+1, right)
```

Algorithm 1.9. *An implementation of the binary search algorithm. This is an example of a recursive algorithm because, after breaking the problem into two halves, the algorithm calls itself again on one of the halves.*

algorithms in computer science. We discuss the main ideas and present several examples in this section. We prove the master theorem in Section 1.11.

Theorem 1.10.2 (Master Theorem). *Consider a function $T : \mathbb{Z}^+ \to [0, \infty)$ satisfying the recursion rule*

$$T(n) \leq \begin{cases} aT(\lceil \frac{n}{b} \rceil) + f(n) & \text{if } n > 1, \\ T_1 & \text{if } n = 1, \end{cases} \tag{1.45}$$

where $a > 0$ and $T_1 > 0$ are real constants, $b \geq 2$ is an integer constant, and $f(n)$ is nonnegative, with $f \in O(n^d)$ for some $d \geq 0$.

(i) *If $b^d > a$, then $T(n) \in O(n^d)$.*

(ii) *If $b^d = a$, then $T(n) \in O(n^d \log n)$.*

(iii) *If $b^d < a$, then $T(n) \in O(n^{\log_b a})$.*

Example 1.10.3.

(i) Let $T(n) \leq 9T(\lceil \frac{n}{3} \rceil) + n$. We have $a = 9$, $b = 3$, $\log_b a = \log_3 9 = 2$, and $d = 1$. Since $b^d = 3 < 9 = a$, it follows that $T(n) \in O(n^{\log_b a}) = O(n^2)$.

(ii) Let $T(n) \leq T(\lceil \frac{n}{2} \rceil) + 1$. We have $a = 1$, $b = 2$, and $d = 0$. Since $b^d = 1 = a$, it follows that $T(n) \in O(n^0 \log n) = O(\log n)$.

(iii) Let $T(n) \leq 3T(\lceil \frac{n}{4} \rceil) + n \log n$. We have $a = 3$ and $b = 4$. Note that $n \log n \in O(n^{1+\varepsilon})$ for any $\varepsilon > 0$. Thus, the master theorem applies with $d = 1 + \varepsilon$. Since $b^d = 4^{1+\varepsilon} > 3$, it follows that $T(n) \in O(n^{1+\varepsilon})$.

Remark 1.10.4. The master theorem does not give us the sharpest possible bound for Example 1.10.3(iii). Exercise 1.68 shows that $T(n) \in O(n \log n)$.

Unexample 1.10.5. In the case of recursive addition (Algorithm 1.8), if we let $T(n)$ be the number of operations required by the algorithm for two lists of length n, then $T(n) = T(n-1) + c$ for some constant c. The master theorem does not apply because $n - 1 \neq \lceil n/b \rceil$ for any integer $b \geq 2$. However, Exercise 1.66 shows, without using the master theorem, that $T(n) \in O(n)$.

Example 1.10.6. The binary search of Algorithm 1.9 checks to see if the middle of the list is the number it is looking for, and if not, it calls itself again on one half of the list. The number $T(n)$ of operations required satisfies the relation

$$T(n) \leq T(\lceil n/2 \rceil) + c,$$

where c is a constant. The master theorem applies with $a = 1$, $b = 2$, and $d = 0$. Since $2^d = 1 = a$, the master theorem implies that $T(n) \in O(\log n)$.

1.10.3 Algorithms

The master theorem is useful for understanding the asymptotics of many important algorithms. In this subsection we demonstrate this on a few examples.

Multiplication

One way to multiply recursively is to separate each number into right and left halves and multiply each half separately. Let x, y denote two numbers in base 10, each with $n = 2^m$ digits. Thus, $x = x_L 10^{n/2} + x_R$ and $y = y_L 10^{n/2} + y_R$, where the

```
def mult(a, b):
    """Recursively multiply two numbers 'a' and 'b'
       together, where each number is given as a list
       of single-digit integers.
    """

    m = len(a); n = len(b)

    # Base case 1: one of the lists is empty
    if a = [] or b = []:
        return []

    # Base case 2: single digit multiplication
    elif m == 1 and n == 1:
        product = a[0] * b[0]
        return [product // 10, product % 10]

    # Recursive case
    else:
        aRbL=mult(a[m//2:],b[:n//2])+[0]*(n - n//2)
        aLbR=mult(a[:m//2],b[n//2:])+[0]*(m - m//2)
        aLbL=mult(a[:m//2],b[:n//2])+[0]*(n+m-n//2-m//2)
        aRbR=mult(a[m//2:],b[n//2:])
        return add(aRbL, add(aLbR, add(aLbL, aRbR)))
```

Algorithm 1.10. *A recursive routine for multiplying two long integers together when represented as lists.*

subscripts L and R denote the left and right halves, respectively. This gives

$$\begin{aligned} xy &= (x_L 10^{n/2} + x_R)(y_L 10^{n/2} + y_R) \\ &= x_L y_L 10^n + (x_R y_L + x_L y_R) 10^{n/2} + x_R y_R. \end{aligned} \tag{1.46}$$

So to multiply x and y, the function calls itself four times on new numbers with $\frac{n}{2}$ digits each (see Algorithm 1.10 for details). The temporal complexity of the addition step is in $O(n)$, so the complexity of this recursion satisfies the relation

$$T(n) \leq 4T(\lceil n/2 \rceil) + f(n),$$

where $f(n) \in O(n)$. The master theorem applies with $a = 4$, $b = 2$, and $d = 1$, and since $2^d = 2 < 4 = a$, we have $T(n) \in O(n^{\log_2 4}) = O(n^2)$.

Faster Multiplication

We can break up (1.46) differently, as follows:

$$xy = x_L y_L 10^n + [(x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R] 10^{n/2} + x_R y_R. \tag{1.47}$$

This expression consists of three long multiplications and six long additions, instead of four long multiplications and three long additions. Since addition is $O(n)$ and multiplication is $O(n^2)$, this is more efficient when n is large than our earlier multiplication algorithms. In fact, we have the recursive relationship

$$T(n) \leq 3T(\lceil n/2 \rceil) + f(n),$$

where $f(n) \in O(n)$. The master theorem applies with $a = 3$, $b = 2$, and $d = 1$. Since $2^d = 2 < 3$, we have $T(n) \in O(n^{\log_2 3}) \approx O(n^{1.585})$. For large values of n, this multiplication algorithm is much faster than the algorithms discussed earlier.

Remark 1.10.7. Note that in both (1.46) and (1.47) we assumed $n = 2^m$. This made the algorithms easier to implement. They can be adapted to the more general case, but it's rather messy and not very enlightening to analyze the algorithms. Instead, we can pad the lists with zeros until their lengths are powers of 2.

Merge Sort

Section 1.2.2 shows that the naïve sorting algorithm has temporal complexity $O(n^2)$. A much better sorting algorithm is the *merge sort*. The algorithm splits the list in half, calls itself on each half, and then merges the two resulting lists. The details are given in Algorithm 1.11. Exercise 1.66 shows that the merge step has complexity $O(n)$, so the complexity of this algorithm satisfies the relation

$$T(n) = 2T(\lceil n/2 \rceil) + cn, \tag{1.48}$$

where c is a constant. The master theorem applies with $a = 2$, $b = 2$, and $d = 1$, and since $b^d = 2 = a$, we have $T(n) \in O(n \log n)$.

```
def mergesort(L):
    """Recursively sort a list 'L' by merging sorted
       sublists.
    """

    n = len(L)

    # Recursive case: split L into halves
    if n > 1:
        return merge(mergesort(L[:n//2]), mergesort(L[n//2:]))

    # Base case: L has length 1 or 0
    else:
        return L
```

Algorithm 1.11. *Recursive routine for merge sort. This algorithm can use either of the previously defined* `merge` *routines, that is, either Algorithm 1.3 or Algorithm 1.7.*

Matrix Multiplication

Matrix multiplication can be defined recursively by subdividing each matrix into blocks. Let A and B be $n \times n$ matrices where $n = 2^m$. Write A and B in block form as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

where each A_{ij} and B_{ij} is a matrix of size $\frac{n}{2} \times \frac{n}{2}$. We have

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

Thus, the product of two $n \times n$ matrices is broken up into eight multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices and added together as above. Since addition of $n \times n$ matrices is $O(n^2)$, the number $T(n)$ of operations used by this algorithm satisfies the relation

$$T(n) = 8T(\lceil n/2 \rceil) + cn^2,$$

where c is a constant. The master theorem applies with $a = 8$, $b = 2$, and $d = 2$. Since $2^d = 4 < 8 = a$, we have that $T(n) \in O(n^{\log_2 8}) = O(n^3)$.

Remark 1.10.8. This recursive matrix multiplication can be adapted to matrices whose dimensions are not powers of 2 by padding the rows and columns A and B with zeros.

Faster Matrix Multiplication

Just as there is a faster multiplication algorithm (1.47) there is also a faster matrix multiplication algorithm due to Strassen, based on the following observation (see Exercise 1.63). As before, we write the $n \times n$ matrices A and B, where $n = 2^m$, in block form as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

where each A_{ij} and B_{ij} is a matrix of size $\frac{n}{2} \times \frac{n}{2}$. The key observation is that AB can be written as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}, \tag{1.49}$$

where

$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}), \\ P_2 &= (A_{11} + A_{12})B_{22}, \\ P_3 &= (A_{21} + A_{22})B_{11}, \\ P_4 &= A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}), \\ P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}). \end{aligned} \tag{1.50}$$

This allows us to compute AB by doing only seven matrix multiplications of half size and several matrix additions. Thus, the recursive equation is given by

$$T(n) = 7T\left(\lceil n/2 \rceil\right) + f(n),$$

where $f(n) \in O(n^2)$. The master theorem applies with $a = 7$, $b = 2$, and $d = 2$. Since $b^d = 4 < 7 = a$, we have $T(n) \in O(n^{\log_2 7}) \approx O(n^{2.8074})$.

Nota Bene 1.10.9. It is important to remember that this is an asymptotic result and a smaller big-O rate doesn't necessarily mean the algorithm is always faster. Indeed, Strassen's algorithm requires n to be moderately large (roughly $n > 3000$) before it overtakes regular matrix multiplication in run-time performance.

1.11 Proof of the Master Theorem

In this section, we prove the master theorem (Theorem 1.10.2). We first prove it when n is an exact power of b, and then we prove it generally.

Recall that the master theorem states that a function $T : \mathbb{Z}^+ \to [0, \infty)$ satisfying

$$T(n) = \begin{cases} aT\left(\lceil n/b \rceil\right) + f(n) & \text{if } n > 1, \\ T_1 & \text{if } n = 1 \end{cases} \tag{1.51}$$

has its asymptotic bounds determined by the relationship between b^d and a as

$$T(n) \in \begin{cases} O(n^d) & \text{if } b^d > a, \\ O(n^d \log n) & \text{if } b^d = a, \\ O(n^{\log_b a}) & \text{if } b^d < a. \end{cases}$$

1.11.1 Proof for $n = b^m$

In the special case that $n = b^m$, we have the following lemmata.[12]

Lemma 1.11.1. *Assume that* (1.51) *holds for some nonnegative integer* $b \geq 2$, *constants* $a > 0$ *and* $T_1 > 0$, *and a nonnegative function* f, *such that for all* $n = b^m$ *with* $m \in \mathbb{N}$ *we have*

$$T(b^m) = \begin{cases} aT(b^{m-1}) + f(b^m) & \text{if } m > 0, \\ T_1 & \text{if } m = 0. \end{cases}$$

In this case, for any exact power $n = b^m$ *with* $m \in \mathbb{N}$ *we have*

$$T(b^m) = a^m T_1 + \sum_{k=0}^{m-1} a^k f(b^{m-k}). \tag{1.52}$$

[12] The traditional plural of the word *lemma* is *lemmata*. People will think you are smarter if you purse your lips and raise your eyebrows when you say it.

Proof. Expanding the recursion we get

$$\begin{aligned} T(b^m) &= aT(b^{m-1}) + f(b^m) \\ &= a^2T(b^{m-2}) + af(b^{m-1}) + f(b^m) \\ &= a^3T(b^{m-3}) + a^2f(b^{m-2}) + af(b^{m-1}) + f(b^m) \\ &\vdots \\ &= a^mT(b^0) + \sum_{k=0}^{m-1} a^k f(b^{m-k}). \quad \square \end{aligned}$$

Remark 1.11.2. Since $m = \log_b n = \log_a n \cdot \log_b a$, we have

$$a^m = (a^{\log_a n})^{\log_b a} = n^{\log_b a}.$$

If $n = b^m$ for some $m \in \mathbb{N}$, then (1.52) becomes

$$T(n) = n^{\log_b a}T_1 + \sum_{k=0}^{(\log_b n)-1} a^k f\left(\frac{n}{b^k}\right). \tag{1.53}$$

The next lemma gives an asymptotic bound on the sum in (1.53).

Lemma 1.11.3. *Let $a > 0$ be a real constant, $b \geq 2$ an integer, and $f : \mathbb{Z}^+ \to [0, \infty)$. Assume that $f(n) \in O(n^d)$ for some $d \geq 0$ and that g is a function defined on exact powers of b by*

$$g(n) = \sum_{k=0}^{(\log_b n)-1} a^k f\left(\frac{n}{b^k}\right) \tag{1.54}$$

for any $n = b^m$ with $m \in \mathbb{N}$.

(i) *If $b^d > a$, then $g(n) \in O(n^d)$.*

(ii) *If $b^d = a$, then $g(n) \in O(n^d \log n)$.*

(iii) *If $b^d < a$, then $g(n) \in O(n^{\log_b a})$.*

Proof. If $f(n) \leq cn^d$ for $n \in \mathbb{N}$ sufficiently large, then

$$g(n) \leq cn^d \sum_{k=0}^{(\log_b n)-1} \left(\frac{a}{b^d}\right)^k.$$

This gives the following three cases:

(i) If $b^d > a$, then

$$g(n) \leq cn^d \sum_{k=0}^{(\log_b n)-1} \left(\frac{a}{b^d}\right)^k < cn^d \sum_{k=0}^{\infty} \left(\frac{a}{b^d}\right)^k = cn^d \cdot \frac{1}{1 - \frac{a}{b^d}} \in O(n^d).$$

(ii) If $b^d = a$, then since $m = \log_b n$, we have

$$g(n) \leq cn^d \log_b n \in O(n^d \log_b n) = O(n^d \log n).$$

(iii) If $b^d < a$, then we have

$$g(n) \leq cn^d \frac{\left(\frac{a}{b^d}\right)^{\log_b n} - 1}{\frac{a}{b^d} - 1} = c\frac{a^{\log_b n} - n^d}{\frac{a}{b^d} - 1} = c\frac{n^{\log_b a} - n^d}{\frac{a}{b^d} - 1} \in O(n^{\log_b a}). \quad \square$$

When n is an exact power of b, the master theorem follows from the two lemmata. In particular, under the hypotheses of the master theorem, since

$$T(n) = n^{\log_b a} T_1 + g(n),$$

we have

(i) $T(n) \in O(n^{\log_b a}) + O(n^d) = O(n^d)$, when $b^d > a$;

(ii) $T(n) \in O(n^{\log_b a}) + O(n^d \log_b n) = O(n^d \log_b n)$, when $b^d = a$;

(iii) $T(n) \in O(n^{\log_b a}) + O(n^{\log_b a}) = O(n^{\log_b a})$, when $b^d < a$.

1.11.2 Proof for General $n \in \mathbb{Z}^+$

If T satisfies the recursion relation (1.45) (or equivalently (1.51)) for all $n \in \mathbb{Z}^+$, then the fractional values in the argument of T are rounded up with the ceiling operator, leading to a sequence of recursion arguments

$$n, \left\lceil \frac{n}{b} \right\rceil, \left\lceil \frac{\left\lceil \frac{n}{b} \right\rceil}{b} \right\rceil, \ldots.$$

This is a nonincreasing sequence that starts with n and goes down to 1 (and then is always 1 thereafter, but that part is not important). Define the sequence as $n_0, \ldots, n_m$, that is,

$$n_j = \begin{cases} n & \text{if } j = 0, \\ \left\lceil \frac{n_{j-1}}{b} \right\rceil & \text{if } j > 0. \end{cases} \tag{1.55}$$

Let m be the smallest integer such that $n_m = 1$. We call m the *recursion depth* of $T(n)$. In the special case that $n = b^m$, the previous subsection shows that the recursion depth is $m = \log_b n$. When n is not an exact power of b, the length of the sequence is not quite as simple to find, but we can still bound its size.

Proposition 1.11.4. *If $T : \mathbb{Z}^+ \to [0, \infty)$ satisfies the recursion relation (1.45), with integer $b \geq 2$, then the depth m of the recursion, as given by the sequence $(n_k)_{k=0}^m$ defined in (1.55), is bounded above by $\lceil \log_b n \rceil$. In other words, $m \leq \lceil \log_b n \rceil$. Moreover, for any $k \geq 0$, we have $n_k < nb^{-k} + 1$.*

Proof. Begin by bounding each term in the sequence generated by (1.55). Since the remainder of n divided by an integer b is at most $b-1$, we have

$$\begin{aligned}
n_0 &= n \le n+1,\\
n_1 &= \left\lceil \frac{n}{b} \right\rceil \le \frac{n}{b} + \frac{b-1}{b} < \frac{n}{b} + 1,\\
n_2 &= \left\lceil \frac{n_1}{b} \right\rceil \le \frac{1}{b}\left(\frac{n}{b} + \frac{b-1}{b}\right) + \frac{b-1}{b} = \frac{n}{b^2} + \frac{b-1}{b}\left(1 + \frac{1}{b}\right) < \frac{n}{b^2} + 1,\\
&\vdots\\
n_k &= \left\lceil \frac{n_{k-1}}{b} \right\rceil \le \frac{1}{b}\left(\frac{1}{b}\left(\cdots \frac{1}{b}\left(\frac{n}{b} + \frac{b-1}{b}\right) + \frac{b-1}{b}\right)\cdots + \frac{b-1}{b}\right) + \frac{b-1}{b}\\
&= \frac{n}{b^k} + \frac{b-1}{b}\sum_{j=0}^{k-1}\left(\frac{1}{b}\right)^j < \frac{n}{b^k} + \frac{b-1}{b}\sum_{j=0}^{\infty}\left(\frac{1}{b}\right)^j = \frac{n}{b^k} + 1.
\end{aligned}$$

Suppose, by way of contradiction, that $m > \lceil \log_b n \rceil$. Thus, $m - 1 \ge \lceil \log_b n \rceil \ge \log_b n$, which implies $n \le b^{m-1}$. Since

$$n_{m-1} < \frac{n}{b^{m-1}} + 1 \le 2,$$

we have $n_{m-1} \le 1$, which contradicts the minimality of m. Thus $m \le \lceil \log_b n \rceil$. ∎

Remark 1.11.5. If b is not an integer, the previous proof does not work, because $\lceil n/b \rceil - n/b$ is not necessarily bounded by $\frac{b-1}{b}$.

We now complete the proof of the master theorem. The previous two lemmata can be adapted to the case where n is not an exact power of b. As in the case of Lemma 1.11.1, expanding the recursion step by step gives

$$\begin{aligned}
T(n) &= T(n_0)\\
&= aT(n_1) + f(n_0)\\
&= a^2T(n_2) + af(n_1) + f(n_0)\\
&\vdots\\
&= a^mT(n_m) + a^{m-1}f(n_{m-1}) + \cdots + af(n_1) + f(n_0)\\
&= a^mT_1 + g(n),
\end{aligned}$$

where $m \le \lceil \log_b n \rceil$ is the recursion depth and

$$g(n) = \sum_{k=0}^{m-1} a^k f(n_k).$$

Note that for each $k \in \{0, \ldots, m\}$ we have $b^k \le b^m < b^{\log_b n + 1} = bn$, so, by Proposition 1.11.4, we have

$$\frac{n_k}{\frac{n}{b^k}} < \frac{\left(\frac{n}{b^k} + 1\right)}{\frac{n}{b^k}} = 1 + \frac{b^k}{n} < 1 + b.$$

Thus, for each $k \in \{0, \ldots, m\}$, we have $n_k < (1+b)\frac{n}{b^k}$. This implies there exists a constant c such that $f(n_k) \leq c(\frac{n}{b^k})^d$. This gives

$$g(n) < \sum_{k=0}^{m-1} a^k c \left(\frac{n}{b^k}\right)^d = cn^d \sum_{k=0}^{m-1} \left(\frac{a}{b^d}\right)^k.$$

So we have the following cases:

(i) If $d > \log_b a$, or equivalently $a < b^d$, then

$$g(n) \leq cn^d \sum_{k=0}^{m-1} \left(\frac{a}{b^d}\right)^k < cn^d \sum_{k=0}^{\infty} \left(\frac{a}{b^d}\right)^k = cn^d \cdot \frac{1}{1-\frac{a}{b^d}} \in O(n^d).$$

(ii) If $d = \log_b a$, or equivalently $a = b^d$, then

$$g(n) \leq cn^d m \leq cn^d \lceil \log_b n \rceil < cn^d (\log_b n + 1) \in O(n^d \log_b n).$$

(iii) If $d < \log_b a$, or equivalently $a > b^d$, then

$$g(n) \leq cn^d \cdot \frac{\left(\frac{a}{b^d}\right)^m - 1}{\frac{a}{b^d} - 1} = c \cdot \frac{\left(\frac{n}{b^m}\right)^d a^m - n^d}{\frac{a}{b^d} - 1}.$$

In Exercise 1.70, we show that $\frac{n}{b^m} \leq 1$, which implies that

$$g(n) \leq c \cdot \frac{a^m - n^d}{\frac{a}{b^d} - 1}.$$

Since $m \leq \lceil \log_b n \rceil = \log_b n + \varepsilon = \log_a n \cdot \log_b a + \varepsilon$, where $0 \leq \varepsilon < 1$, and $a \geq b^d \geq 1$, we have

$$a^m \leq a^{\lceil \log_b n \rceil} = a^{\log_a n \cdot \log_b a + \varepsilon} = n^{\log_b a} \cdot a^{\varepsilon} \in O(n^{\log_b a}). \tag{1.56}$$

Thus, $g(n) \in O(n^{\log_b a})$.

This completes the proof.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in

this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

1.1. Prove or disprove each of the following:

(i) $3n - 1 \in O(n)$.

(ii) $3n - 1 \in o(n)$.

(iii) $3n - 1 \in O(n^2)$.

(iv) $1 \in O(n)$.

1.2. ⚠ Prove the following:

(i) If $f_1(n), h(n) \in O(g(n))$ and $f_2(n) \in O(h(n))$, then the sum satisfies $f_1(n) + f_2(n) \in O(g(n))$.

(ii) If $f_1(n), h(n) \in o(g(n))$ and $f_2(n) \in O(h(n))$, then the sum satisfies $f_1(n) + f_2(n) \in o(g(n))$.

(iii) For any $k \in \mathbb{N}$ and any coefficients $a_k, a_{k-1}, \ldots, a_0 \in \mathbb{R}$, the function $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ is in $O(n^k)$.

1.3. ⚠ Prove Proposition 1.1.13.

1.4. For $m \in \mathbb{N}$, prove that $f(n) = \sum_{k=0}^{n} k^m \in O(n^{m+1})$.

1.5. Show that for every $p > 0$ and every $a > 1$ we have

(i) $O(\log n) \subset o(n^p)$, but $\log n \notin O(1)$;

(ii) $n \log n \in o(n^{1+p})$, but $n \log n \notin O(n)$;

(iii) $O(n^p) \subset o(a^n)$.

Hint: Use l'Hôpital's rule.

1.6. Consider the standard elementary school algorithm for subtraction of multi-digit integers.

(i) Code up the algorithm. Your code should accept two lists of single-digit integers and return a list of single-digit integers. Explain the algorithm carefully in the comments of your code.

(ii) Determine the asymptotic temporal and spatial complexity (big-O) of this algorithm and explain why your answer is correct.

1.7.* The *Fibonacci sequence* $\{F_n\}_{n=0}^{\infty}$ is defined by the rule $F_{n+1} = F_n + F_{n-1}$, for $n \in \mathbb{Z}^+$, where $F_0 = 0$ and $F_1 = 1$.

(i) Assuming that the sequence $x_n = F_{n+1}/F_n$ converges to some point in $\mathbb{R}$, prove that it converges to the *golden ratio*

$$\phi = \frac{1 + \sqrt{5}}{2}. \tag{1.57}$$

Prove that (1.57) satisfies $\phi^2 = \phi + 1$, and use this to prove (inductively) that $\phi F_n + F_{n-1} = \phi^n$. Use this fact to prove that $F_n \in O(\phi^n)$.

(ii)† That F_n/F_{n-1} converges to ϕ is not enough to show that $F_n \in O(\phi^n)$, as you will now show. Let $a > 0$, and let $G_n = a^n e^{1+\frac{1}{2}+\cdots+\frac{1}{n}}$. Prove that $\lim_{n\to\infty} G_n/G_{n-1} = a$ but that $G_n \notin O(a^n)$.

1.8. Prove the following:

(i) For any $k \in \mathbb{N}$ and any coefficients $a_k, a_{k-1}, \ldots, a_0 \in \mathbb{R}$, the function $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ satisfies $f \sim a_k n^k$.

(ii) If $g \in o(f)$, then $f + g \sim f$.

1.9. An algorithm with leading-order temporal complexity $\sim 100n^2$ will not necessarily take longer to run than an algorithm with leading-order temporal complexity ~ 1. Give an example of two functions $f \sim 100n^2$ and $g \sim 1$ such that $f(n) < g(n)$ for all $n < 10^5$.

1.10. Prove that $\sim$ is an equivalence relation, as mentioned in Remark 1.2.3.

1.11. Find the leading-order spatial and temporal complexity of Algorithm 1.1 and explain why your answer is correct.

1.12. Find the leading-order spatial and temporal complexity of the long subtraction algorithm in Exercise 1.6 and explain why your answer is correct.

1.13. Construct an algorithm for finding the index of the smallest element in a list L of length n, using only primitive operations, that is, assigning a value to a variable or to a given position in a list, looking up the value of a particular element at a given position in a list, comparing two values, incrementing a value, etc.

(i) Code up your algorithm.

(ii) Give the leading-order temporal and spatial complexity of this algorithm as a function of n.

1.14. Consider the following sorting algorithm, called *selection sort*. Given a list L of length n, first find the smallest element and swap it with the element $L[0]$. Then find the second smallest element of L and swap it with the element $L[1]$, and so forth for the first $n-1$ elements of the list.

(i) Explain why the algorithm needs only to run through $n-1$ elements instead of all n.

(ii) Code up this sorting algorithm without using any built-in sorting or indexing functions. You may use the minimum function you wrote in the previous problem.

(iii) Give the leading-order temporal and spatial complexity for this algorithm as a function of n.

1.15.* Given n points in the plane, consider the problem of finding the pair of points that is closest together. One algorithm for doing this is the brute force method: list all the pairs, compute their Euclidean distances, and take the smallest one.

(i) Explain how this brute force algorithm can be implemented without ever computing a square root.

(ii) Code up this algorithm without using square roots. Your code should accept a list of points as ordered pairs (x_i, y_i) of scalars and return the two points that are closest together.

(iii) Show that this algorithm has temporal complexity in $O(n^2)$, where the primitive operations include the basic arithmetic operations $+, -, \times, \div$, assigning a value to a variable or to a given position in a list, looking up the value of a particular element at a given position in a list, comparing two values, and incrementing a value.

1.16. Prove that the difference operator is linear; that is, if $f, g : \mathbb{N} \to \mathbb{F}$ and $a, b \in \mathbb{F}$, then

$$\Delta[af + bg](k) = a\Delta[f](k) + b\Delta[g](k).$$

1.17. Prove the following:

(i) $\sum_{i=1}^{n} \frac{1}{i(i+1)} = 1 - \frac{1}{n+1}$.

(ii) $\sum_{k=1}^{n} \frac{1}{4k^2-1} = \frac{n}{2n+1}$.

Hint: Consider using the fundamental theorem.

1.18. Prove that

$$\sum_{k=a}^{b-1} \log\left(1 + \frac{1}{k}\right) = \log(b) - \log(a)$$

for $b \geq a \geq 1$.

1.19. Derive the formula (1.11).

1.20. ⚠ For any $\beta \in (-1, 1)$ show that $\sum_{t=0}^{\infty} t\beta^t = \frac{\beta}{(1-\beta)^2}$ by differentiating the geometric series $\sum_{t=0}^{n} \beta^t = \frac{\beta^{n+1}-1}{\beta-1}$ with respect to β and then taking the limit as $n \to \infty$.

1.21. Show that

$$\sum_{k=0}^{n} f(n-k) = \sum_{\ell=0}^{n} f(\ell).$$

1.22. Find closed-form expressions (no summation) as a function of n for each of the following:

(i) $\sum_{k=5}^{n+5} (k-5)^2$.

(ii) $\sum_{k=0}^{n} (k+2)^3$.

(iii) $\sum_{k=5}^{n} \sum_{j=-3}^{k-8} (k-4)$.

(iv) $\sum_{j=-3}^{n-3} \sum_{k=j+3}^{n+3} (k-3)$.

1.23. Compute the following double sum in two ways: first, as written, and second, by changing the order of summation:

$$\sum_{k=0}^{n} \sum_{j=k}^{n} j.$$

1.24. Give another proof of the relation in Exercise 1.20 as follows:

(i) Show that $\sum_{t=0}^{n} t\beta^t$ is equal to the double sum $\sum_{t=1}^{n} \sum_{s=0}^{t-1} \beta^t$.

(ii) Change the order of summation.

(iii) Compute the inner sum as a geometric series.

(iv) Use the previous result to give a closed-form expression for $\sum_{t=0}^{n} t\beta^t$.

(v) Compute the limit as $n \to \infty$.

1.25. Show that the double sum of (1.18) satisfies

$$\sum_{j=0}^{n} \sum_{k=0}^{n-j} f(j,k) = \sum_{a=0}^{n} \sum_{b=0}^{a} f(a-b,b).$$

Give a geometric description or picture (in the style of Figures 1.1, 1.2, and 1.3) of how this summation proceeds.

1.26. For each $k = 1, 2, \ldots, 11$ do the following:

(i) Define random matrices A and B of size $2^k \times 2^k$ and a column vector $\mathbf{x}$ of length 2^k.

(ii) Time the computation of $(AB)\mathbf{x}$ and the computation of $A(B\mathbf{x})$.

For each k, find the ratio of the time it takes to compute $(AB)\mathbf{x}$ versus $A(B\mathbf{x})$. When k increases by one, does the ratio of the times of the two computations change? By how much? Explain this in terms of what we have discussed about the complexity of matrix-matrix and matrix-vector multiplication.

1.27. Verify algebraically that $(I_n + \mathbf{u}\mathbf{v}^\mathsf{T})\mathbf{x} = \mathbf{x} + \mathbf{u}(\mathbf{v}^\mathsf{T}\mathbf{x})$ for any $\mathbf{u}, \mathbf{v}, \mathbf{x} \in \mathbb{R}^n$. For each choice of $n = 1, 2, \ldots, 11$ do the following:

(i) Create the $2^n \times 2^n$ identity matrix I and random vectors $\mathbf{u}, \mathbf{v}, \mathbf{x}$ of dimension 2^n.

(ii) Time the computation $(I + \mathbf{u}\mathbf{v}^\mathsf{T})\mathbf{x}$ versus $\mathbf{x} + \mathbf{u}(\mathbf{v}^\mathsf{T}\mathbf{x})$.

(iii) Compare the computation times, describe how the ratio of the two grows as n gets larger, and explain this in terms of the asymptotic temporal complexity of the two computations.

1.28. Write out the details to prove the equality in (1.19).

1.29. Carefully compute the leading-order temporal and spatial complexity of the back-substitution part of Algorithm 1.6 (Lines 21–29).

1.30.* Consider the following Python code, which calculates

$$S = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (i + f(j)),$$

where $f(j)$ is a function that depends only on j:

```
S = 0
for i in range(n):
    for j in range(m):
        S += i + f(j)
```

(i) Assuming that computing `f(j)` requires F FLOPs for every `j`, calculate the number of FLOPs that this code uses, as a function of n, m, and F.

(ii) Move any computations possible out of the inner loop and find the number of FLOPs used after this change is made.

(iii) Going back to the original code, change the order of the loops, so that the `i`-loop is the inner one, and move any computations possible out of the inner loop. Find the number of FLOPs required by the modified code.

(iv) Eliminate the inner `i`-loop entirely by finding a closed-form expression for the sum it computes. Find the number of FLOPs required now.

(v) The closed-form expression of the last step does not depend on `j` so it can be moved outside the `j`-loop. Make this change to the code and move any additional calculations outside the `j`-loop. Your new code should correspond to computing the sum $\frac{mn(n-1)}{2} + (n-1)\sum_{j=0}^{m-1} f(j)$. Find the number of FLOPs used in this version of your code.

(vi) Show that you can save two more FLOPs by factoring $(n-1)$ out of the previous expression. Adjust your code correspondingly.

1.31.* Use summation by parts to compute the sum

$$\sum_{k=0}^{n} k^2 x^k$$

in closed form.

1.32.* Using equation (1.22), derive equation (1.11).

1.33.* Prove that

$$\sum_{i=1}^{n-1} i^{\underline{k}} = \frac{n^{\underline{k+1}}}{k+1}.$$

1.34.* Prove the inclusion-exclusion formula for three sets; that is, prove Proposition 1.6.13.

1.35.* Suppose that $(a_k)_{k=0}^{\infty}$ is a sequence of complex numbers with uniformly bounded partial sums; that is, there exists $M > 0$ such that

$$\left| \sum_{k=0}^{n} a_k \right| \leq M < \infty$$

for every $n \in \mathbb{Z}^+$. Prove: If $(b_k)_{k=0}^{\infty} \subset \mathbb{R}$ is a monotonically decreasing[13] sequence converging to zero, then the sum

$$\sum_{k=0}^{\infty} a_k b_k$$

converges. Moreover, $|\sum_{k=0}^{\infty} a_k b_k| \leq 2Mb_1$. Hint: Use summation by parts.

1.36.* Use the previous result to prove that the sequence

$$\sum_{k=1}^{\infty} \frac{z^k}{k}$$

converges for any complex number $|z| = 1$ with $z \neq 1$.

1.37. Prove that there are $n!$ permutations of a set with n elements. Note: An informal proof suffices.

1.38. (i) Prove Proposition 1.7.5. An informal proof suffices.

(ii) Prove Proposition 1.7.6. An informal proof suffices.

1.39. A group of friends, Alice, Bob, Carlos, Dan, Eve, and Fakhira, are going to a movie. In how many different ways can they be seated together in a single row of six seats if

(i) there are no restrictions on the seating assignment;

(ii) Alice and Bob must sit next to each other;

(iii) Alice, Bob, and Carlos must sit together;

(iv) the six seats must alternate between genders (Alice, Eve, and Fakhira are female, while Bob, Carlos, and Dan are male).

1.40. Show that there are 123,552 different ways to draw a two-pair hand in five-card poker; see Example 1.7.9 for details.

1.41. In a certain lottery, five distinct numbers (balls) are drawn randomly from the set $\{1, 2, \ldots, 59\}$ and a "superball" is drawn from the set $\{1, 2, \ldots, 35\}$. You win the \$100 prize when you match three regular balls and the superball. How many unique draws qualify for the \$100 prize?

1.42. Prove that for any $n \in \mathbb{Z}^+$ and any $x, y \in \mathbb{F}$, we have

$$\sum_{k=1}^{n} \binom{n}{k} k x^{k-1} y^{n-k} = n(x+y)^{n-1}.$$

Use this to show that

$$\sum_{k=1}^{n} \binom{n}{k} k = 2^{n-1} n.$$

Hint: Compute the derivative $\frac{\partial}{\partial x}$ of the binomial formula.

---

[13] A sequence $(b_k)_{k=0}^{\infty}$ is *monotonically decreasing* if $b_{k+1} \leq b_k$ for all $k \in \mathbb{N}$.

1.43. Prove that for any $x, y \in \mathbb{F}$ and $n \in \mathbb{N}$ with $n \geq 2$ and $x \neq 0$ we have

$$\sum_{k=1}^{n} \binom{n}{k} k^2 x^{k-2} y^{n-k} = \left(n^2 + \frac{ny}{x}\right)(x+y)^{n-2},$$

and use this to show that

$$\sum_{k=1}^{n} \binom{n}{k} k^2 = n(n+1)2^{n-2}.$$

1.44.* Prove that

$$\binom{n+m}{r} = \sum_{k=0}^{r} \binom{n}{k}\binom{m}{r-k}$$

for integers r, n, and m with $r \leq n+m$. Hint: Look at the binomial expansions for $(1+x)^m$ and $(1+x)^n$, and compare their product to the binomial expansion for $(1+x)^{m+n}$.

1.45.* Prove Theorem 1.7.19.

1.46.* Prove that the Pochhammer symbols (see Definition 1.6.7) satisfy a form of the binomial theorem:

$$(x+y)^{\overline{n}} = \sum_{k=0}^{n} \binom{n}{k} x^{\overline{n-k}} y^{\overline{k}} \qquad \text{and} \qquad (x+y)^{\underline{n}} = \sum_{k=0}^{n} \binom{n}{k} x^{\underline{n-k}} y^{\underline{k}}.$$

1.47. Given any integer $a \in \mathbb{Z}$ and any nonzero $b \in \mathbb{Z}$, show that the set $S^+ = \{a - bx \mid x \in \mathbb{Z}\} \cap \mathbb{N}$, in the proof of the division theorem (Theorem 1.8.6), is nonempty.

1.48. Let $a = 323$ and $b = 204$. Use the Euclidean algorithm (by hand) to find $\gcd(a, b)$. Show all the intermediate steps.

1.49. Find $x, y \in \mathbb{Z}$ such that $323x + 204y = 17$.

1.50. Prove: If $d = \gcd(a, b)$, then $\gcd(a/d, b/d) = 1$.

1.51. Code up the extended Euclidean algorithm from scratch, without importing any additional libraries or methods. Your code should accept two integers a and b and return $\gcd(a, b)$ as well as x, y, satisfying $ax + by = \gcd(a, b)$.

1.52. Prove that $\gcd(n, n+1) = 1$ for all $n \in \mathbb{Z}^+$. Conclude from this that if a prime p divides n then it does not divide $n+1$.

1.53.* Using the previous exercise prove there are infinitely many prime numbers. Hint: If there are only finitely many primes, say $p_1, p_2, \ldots, p_m$, then set $n = p_1 p_2 \cdots p_m$ and consider $n+1$.

1.54. Prove Theorem 1.9.9.

1.55. Given an integer $a = \sum_{k=0}^{n} a_k 10^k$, prove that a is divisible by

(i) 3 if and only if the sum $\sum_{k=0}^{n} a_k$ is divisible by 3;

(ii) 9 if and only if the sum $\sum_{k=0}^{n} a_k$ is divisible by 9;

(iii) 11 if and only if the sum $\sum_{k=0}^{n} (-1)^k a_k$ is divisible by 11.

1.56. Prove: If $a \equiv b \pmod{c}$ and $d | c$, then $a \equiv b \pmod{d}$.

1.57. Determine necessary and sufficient conditions on x and c so that

$$ax \equiv bx \pmod{c} \implies a \equiv b \pmod{c}.$$

Prove your answer is correct.

1.58. By hand, find the remainder when dividing 34^{34} by 12.

1.59. Use the extended Euclidean algorithm to find

(i) the element $a \in \mathbb{Z}_{72}$ such that $35a \equiv 1 \pmod{72}$;

(ii) the element $b \in \mathbb{Z}_{72}$ such that $35b \equiv 67 \pmod{72}$.

1.60. Compute the following by hand:

(i) $14^{128} \pmod{127}$.

(ii) $18^{254} \pmod{127}$.

(iii) $25^{640} \pmod{127}$.

1.61. Prove Corollary 1.9.19.

1.62. For each of the following recurrence relations, determine whether the master theorem applies. If it applies, use it to provide the big-O bounds, and if not, explain why not.

(i) $T(n) = 8T\left(\lceil \frac{n}{2} \rceil\right) + n$.

(ii) $T(n) = 16T\left(\lceil \frac{n}{8} \rceil\right) + n$.

(iii) $T(n) = 3T\left(\lceil \frac{n}{2} \rceil\right) + n^2$.

(iv) $T(n) = 4T\left(\lceil \frac{n}{2} \rceil\right) + n^2$.

(v) $T(n) = 5T\left(\lceil \frac{n}{2} \rceil\right) + n^2$.

(vi) $T(n) = T(n-5) + \sqrt{n}$.

(vii) $T(n) = T\left(\lceil \frac{n}{4} \rceil\right) + 2^n$.

(viii) $T(n) = 2^n T\left(\lceil \frac{n}{2} \rceil\right) + n^n$.

1.63. Expand the terms to show that (1.49) is correct.

1.64. A sequence $(x_i)_{i=0}^n$ is *unimodal* if it consists of an increasing sequence followed by a decreasing sequence; that is, there is some $k \in \{0, \ldots, n\}$ such that $x_{i-1} < x_i$ when $0 < i \leq k$ and $x_{i+1} < x_i$ when $k \leq i < n$.

(i) Give an algorithm with temporal complexity $O(\log n)$ that finds the maximal element x_m in a unimodal sequence.

(ii) Code up your algorithm and explain the details in the comments.

(iii) Prove the $O(\log n)$ bound on the temporal complexity.

1.65. Assume $a > 0$ and g is a nonnegative function. Prove that the recurrence

$$T(n) = aT(n-1) + g(n), \quad n \in \mathbb{Z}^+,$$

with $T(0) = T_0 > 0$, has the solution

$$T(n) = a^n T_0 + \sum_{k=1}^{n} a^{n-k} g(k). \tag{1.58}$$

1.66. Use (1.58) from the previous problem to show that the temporal complexity of both recursive merge and recursive addition, Algorithms 1.8 and 1.7, respectively, is $O(n)$.

1.67. Find an exact closed-form formula for T in each of the following recursions when $n = 2^m$ for $m \in \mathbb{Z}^+$:

(i) $T(n) = 8T(\frac{n}{2}) + n$.

(ii) $T(n) = 3T(\frac{n}{2}) + n$.

(iii) $T(n) = 3T(\frac{n}{2}) + n^3$.

1.68. Prove that the recurrence

$$T(n) = 3T\left(\left\lceil \frac{n}{4} \right\rceil\right) + n \log n, \quad n \geq 2,$$

with $T(1) = T_1 > 0$, satisfies $T(n) \in O(n \log n)$.

1.69. Assume that $T(n)$ satisfies the recurrence (1.45) for $n = b^m$ for $m \in \mathbb{N}$. Generalize Theorem 1.10.2(ii) by proving the following theorem: If $d = \log_b a$ and $f(n) \in O(n^d (\log n)^\ell)$, then $T(n) \in O(n^d (\log n)^{\ell+1})$. Hint: Show that $T(n) \in O(n^d m^{\ell+1})$ and then use the fact that $m = \log_b n$.

1.70. Show that the sequence (1.55) satisfies $\frac{n}{b^k} \leq n_k$ for each $k \in \{0, 1, 2, \ldots, m\}$. In particular, $\frac{n}{b^m} \leq n_m = 1$.

1.71. Prove that recursion depth m, given by the sequence (1.55), is bounded below by $\lfloor \log_b n \rfloor$, that is, $m \geq \lfloor \log_b n \rfloor$.

Notes

Exercise 1.14 is from [CLRS01, Exercise 2.2-2]. Exercise 1.15 is discussed in [KT05, Section 5.4]; surprisingly, there is a closest pair algorithm developed by Shamos and Hoey that is $O(n \log n)$. Exercise 1.64 is from [DL05, 1-3]. Our treatment of the master theorem is inspired by [BHS+78] and [CLRS01].

For a comparison of matrix multiplication algorithms at various dimensions, see [BB14]. For more details on solving linear systems, see [TB97, Section 20]. For more about loop interchange in optimizing compilers, as mentioned in Section 1.5.3, see [SS07, TdD14]. For more on the origins of the Euclidean algorithm, see [BBC+99, Hea49, vdW83].

2 Asymptotic Integrals

You know my methods, Watson.
—Sherlock Holmes

Big-O and little-o notation convey important information about the limiting behavior of a function or algorithm by bounding its growth relative to another function. The leading-order behavior provides more information about the limiting behavior of a function or algorithm growth in absolute terms. In this section, we expand on this theme by examining asymptotic behavior in richer detail.

One area of considerable interest is the asymptotic behavior of combinatorial functions and algorithms, that is, those having factorial terms. As a first step to analyzing these, we generalize the factorial function and extend it to the positive real numbers and beyond. Additionally, we expand on the asymptotic behavior of the factorial function and develop some tools for analyzing functions (and therefore algorithms) that are combinatorial in nature.

2.1 The Gamma Function and Stirling's Approximation

The complexity of many algorithms is expressed most naturally in terms of binomial coefficients or other formulas involving factorials. For example, a binary search tree is an important data structure that lies at the heart of many important algorithms (see Section 3.3.1). One can show that the number of binary search trees with n nodes is $\frac{1}{n+1}\binom{2n}{n}$; see Exercise 3.14. It is useful to compare the asymptotic growth of these combinatorial expressions to the growth of other expressions involving simpler functions, like exponentials.

In this section we study the asymptotic behavior of the factorial function, we define the *gamma function*, which is a continuous analogue and generalization of the factorial function, and we describe an important asymptotic formula called *Stirling's approximation* for $n!$ or $\log(n!)$ when n is large. Stirling's approximation is useful in many areas of mathematics, including probability theory and asymptotic analysis.

2.1.1 Simple Approximation of the Factorial Function

It is easy to see that $n! = 1 \cdot 2 \cdots (n-1) \cdot n$ is bounded above by $n^n = n \cdot n \cdots n \cdot n$. For a lower bound, the nth term $n^n/n!$ of the power series $e^n = \sum_{k=0}^{\infty} n^k/k!$ satisfies $n^n/n! < e^n$, which we can rewrite as $n^n/e^n < n!$. Thus we have

$$\frac{n^n}{e^n} < n! < n^n. \tag{2.1}$$

We can do better than this with a little more work. Since $\log(x)$ is a strictly increasing function on $[1, \infty)$, we can show (see Exercise 2.1) that

$$\sum_{k=1}^{n-1} \log(k) \le \int_1^n \log(x)\, dx \le \sum_{k=1}^{n} \log(k),$$

from which we get

$$\int_1^n \log(x)\, dx \le \sum_{k=1}^{n} \log(k) \le \log(n) + \int_1^n \log(x)\, dx$$

and, thus, the following proposition.

Proposition 2.1.1. *For any integer $n \in \mathbb{Z}^+$ we have*

$$n\log(n) - n + 1 \le \log(n!) \le n\log(n) - n + \log(n) + 1 \tag{2.2}$$

and

$$\frac{n^n}{e^{n-1}} \le n! \le \frac{n^{n+1}}{e^{n-1}}. \tag{2.3}$$

Proof. The proof is Exercise 2.1. □

This proposition can be reformulated as an asymptotic formula:

$$\log(n!) - n\log(n) + n - 1 \in O(\log(n)). \tag{2.4}$$

These expressions show up in many different places, including the calculation of entropy in statistical mechanics and the proof of the prime number theorem, which says that the number of primes less than n is $\sim \frac{n}{\log(n)}$. Below we improve these approximations with the famous *Stirling's approximation.*

Example 2.1.2. When $n = 22{,}026$, the approximation (2.2) for $\log(n!)$ gives $198{,}235 \le \log(n!) \le 198{,}245$, which is correct to four digits of accuracy. This approximation is sufficiently close in some situations but inadequate in others. We get a more accurate estimate in Example 2.1.9.

Example 2.1.3. Using (2.3) we can give upper and lower bounds for $\binom{2n}{n}$. We have

$$\left(\frac{1}{en^2}\right)4^n \leq \frac{(2n)!}{(n!)^2} \leq \left(\frac{2n}{e}\right)4^n.$$

This bound can be improved with Stirling's approximation (2.6), as shown below in Example 2.1.10.

2.1.2 The Gamma Function

The *gamma function* is the continuous analogue and generalization of the factorial function. There are, of course, infinitely many continuous functions that match the factorial function at the positive integers, but the gamma function is natural in applications. Its graph is pictured in Figure 2.1 and it is defined as

$$\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}\,dt, \tag{2.5}$$

which is well defined for all positive real numbers. This function can be extended to the complex numbers and to negative real numbers (except the nonpositive integers) using an important technique from complex analysis called *analytic continuation*, but the details of that extension are outside the scope of this book. The following proposition shows that $\Gamma(n+1) = n!$ for any nonnegative integer n, so it really is a generalization of the factorial function.

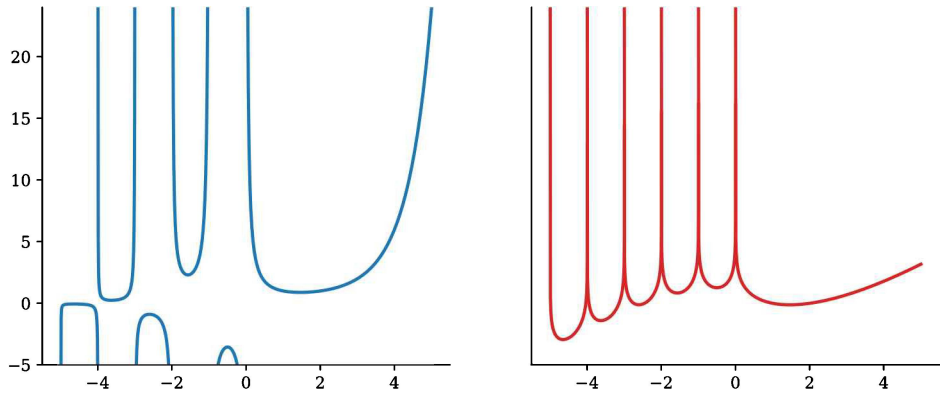


Figure 2.1. *A plot of* $\Gamma(x)$ *(left) and* $\log|\Gamma(x)|$ *(right). Notice that the graphs have vertical asymptotes at zero and at the negative integers, where the gamma function is not defined.*

Proposition 2.1.4. *If* $x \in (0,\infty)$*, then* $\Gamma(x+1) = x\Gamma(x)$.

Proof. For any $x > 0$, integrate by parts to get

$$\Gamma(x+1) = \int_0^\infty e^{-t}t^x\,dt = -e^{-t}t^x\Big|_0^\infty + x\int_0^\infty e^{-t}t^{x-1}\,dt = x\Gamma(x). \quad \square$$

Corollary 2.1.5. *If $n \in \mathbb{N}$, then $\Gamma(n+1) = n!$.*

Proof. When $n = 0$ we have

$$\Gamma(1) = \int_0^\infty e^{-t}\,dt = 1 = 0!.$$

The desired equality $\Gamma(n+1) = n!$ follows immediately by induction on n, using Proposition 2.1.4. □

Remark 2.1.6. The gamma function is special for many reasons, but one reason to think it is the right continuous version of the factorial function is the Bohr–Mollerup theorem, which guarantees that $\Gamma(x)$ is the unique function defined for all $x > 0$ that satisfies

(i) $x\Gamma(x) = \Gamma(x+1)$ for all $x > 0$;

(ii) $\Gamma(1) = 1$;

(iii) $\frac{d^2}{dx^2}\log(\Gamma(x)) > 0$ on $x \in (0, \infty)$, that is, $\log(\Gamma(x))$ is twice differentiable and convex on the positive real numbers.

Proposition 2.1.7. $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$.

Proof. The proof is Exercise 2.5. □

2.1.3 Stirling's Approximation

Stirling's approximation gives a sharper estimate of the asymptotic growth of the factorial function than the simple bounds in (2.2).

Theorem 2.1.8 (Stirling's Approximation). *As $x \to \infty$, we have*

$$\Gamma(x+1) \sim x^{x+1}e^{-x}\sqrt{\frac{2\pi}{x}}. \tag{2.6}$$

Alternatively,

$$\log(\Gamma(x+1)) \sim x\log(x) - x + \frac{1}{2}\log(2\pi x).$$

We give the proof of Stirling's approximation for positive integers in Section 2.1.4 and, more generally, for positive real numbers in Section 2.2.5.

Example 2.1.9. When $n = 22{,}026$, Stirling's approximation for $\log(n!)$ is 198,239.45313 (evaluated to 11 digits). The correct value is 198,239.45314 (also evaluated to 11 digits). Compare this approximation to the one given in Example 2.1.2, which was accurate only to four digits.

Example 2.1.10. Using (2.6) we can improve the bound on $\binom{2n}{n}$ given in Example 2.1.3, at least asymptotically:

$$\binom{2n}{n} \sim \frac{\sqrt{2\pi 2n}(2n)^{2n}}{e^{2n}} \left(\frac{e^n}{\sqrt{2\pi n}(n^n)} \right)^2 = \frac{4^n}{\sqrt{\pi n}}.$$

Remark 2.1.11. A more careful analysis gives an improved version of Stirling's approximation, which includes an additional lower-order term. In Section 2.3.4 we show that

$$\Gamma(x+1) \sim x^{x+1} e^{-x} \sqrt{\frac{2\pi}{x}} \left(1 + \frac{1}{12x} \right) \quad \text{as } x \to \infty. \tag{2.7}$$

In Table 2.1 we compare the logarithms of the gamma function, Stirling's approximation (2.6), and the improved Stirling's approximation (2.7). Notice how much more accurate the improved version is.

n	$\log \Gamma(n+1)$	Leading Order (2.6)	Improved (2.7)
8	10.6046029	10.5941916	10.6045544
16	30.6718601	30.6666524	30.6718472
32	81.5579594	81.5553553	81.5579561
64	205.1681994	205.1668974	205.1681986
128	496.4054784	496.4048274	496.4054782
256	1167.2572785	1167.2569530	1167.2572785
512	2686.0604716	2686.0603088	2686.0604716

Table 2.1. *The logarithms of* $\Gamma(n+1)$*, Stirling's approximation* (2.6)*, and the improved Stirling's approximation* (2.7)*.*

2.1.4 Proof of Stirling's Approximation

In this section we prove Stirling's approximation for positive integers. We prove the general case for all real numbers in Section 2.2.5. We begin with a few lemmata.

Lemma 2.1.12. *The* Wallis integrals

$$W_n = \int_0^{\pi/2} \sin^n(x)\, dx$$

form a positive, strictly decreasing sequence $(W_n)_{n=0}^{\infty}$ *satisfying*

(i) $W_0 = \pi/2$,

(ii) $W_1 = 1$,

(iii) $W_n = \frac{n-1}{n} W_{n-2}$ *for all* $n \geq 2$.

Proof. The proof is Exercise 2.6. □

Lemma 2.1.13. *The Wallis integrals $(W_n)_{n=0}^{\infty}$ satisfy the following identity:*

$$\lim_{n\to\infty} \prod_{k=1}^{n} \frac{4k^2}{4k^2-1} = \frac{\pi}{2} \lim_{n\to\infty} \frac{W_{2n+1}}{W_{2n}} = \frac{\pi}{2}. \tag{2.8}$$

Proof. Since $(W_n)_{n=0}^{\infty}$ is strictly decreasing, we have $W_n > W_{n+1} > W_{n+2}$, which gives

$$1 > \frac{W_{n+1}}{W_n} > \frac{W_{n+2}}{W_n} = \frac{n+1}{n+2},$$

and thus by the squeeze theorem we have

$$\lim_{n\to\infty} \frac{W_{n+1}}{W_n} = 1.$$

By Lemma 2.1.12(iii), we have

$$\frac{W_{2n+1}}{W_{2n}} = \frac{\left(\frac{2n}{2n+1}\right)\left(\frac{2n-2}{2n-1}\right)\cdots\left(\frac{2}{3}\right) W_1}{\left(\frac{2n-1}{2n}\right)\left(\frac{2n-3}{2n-2}\right)\cdots\left(\frac{1}{2}\right) W_0} = \frac{2}{\pi} \prod_{k=1}^{n} \frac{(2k)^2}{(2k+1)(2k-1)} = \frac{2}{\pi} \prod_{k=1}^{n} \frac{4k^2}{4k^2-1},$$

and thus (2.8) holds. ∎

Lemma 2.1.14. *For any $n \in \mathbb{N}$, we have*

$$\prod_{k=1}^{n} \frac{4k^2}{4k^2-1} = \frac{(n!)^4\, 2^{4n}}{(2n)!\,(2n+1)!}. \tag{2.9}$$

Proof. The proof is Exercise 2.7. ∎

We now finish the proof of Stirling's approximation (2.6) for positive integers; that is, we show

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n, \tag{2.10}$$

where $n \in \mathbb{Z}^+$. The proof for real numbers is given in Section 2.2.5.

The idea of the proof is to approximate the integral $\int_1^n \log(x)\,dx$ with trapezoids on the intervals $[k, k+1]$ (see Example 9.6.3) and integrate by parts with some clever choices of integration constants.

Proof of Stirling's approximation (integer case). We have

$$n\log(n) - n + 1 = \int_1^n \log(x)\,dx = \sum_{k=1}^{n-1} \int_k^{k+1} \log(x)\,dx.$$

Integrating the integrals by parts ($\int u\,dv = uv - \int v\,du$) with $u(x) = \log(x)$ and $v(x) = x - k - \frac{1}{2}$ gives

$$n\log(n) - n + 1 = \sum_{k=1}^{n-1} \left[\frac{1}{2}(\log(k+1) + \log(k)) - \int_k^{k+1} \left(x - k - \frac{1}{2}\right)\frac{dx}{x}\right].$$

Integrating by parts again gives

$$-\int_k^{k+1}\left(x-k-\frac{1}{2}\right)\frac{dx}{x}=\int_k^{k+1}\frac{f(x)}{x^2}\,dx,$$

where $f(x)=\frac{1}{8}-\frac{1}{2}\left(x-k-\frac{1}{2}\right)^2$. Since $\frac{1}{(k+1)^2}\le\frac{1}{x^2}\le\frac{1}{k^2}$, we have

$$\int_k^{k+1}\frac{f(x)}{(k+1)^2}\,dx\le\int_k^{k+1}\frac{f(x)}{x^2}\,dx\le\int_k^{k+1}\frac{f(x)}{k^2}\,dx.$$

Hence, for each $k\in\mathbb{Z}^+$, the intermediate value theorem (see Volume 1, Corollary 5.9.14) and Exercise 2.8 show there exists some $\xi_k\in[k,k+1]$ such that

$$\int_k^{k+1}\frac{f(x)}{x^2}\,dx=\frac{1}{\xi_k^2}\int_k^{k+1}f(x)\,dx=\frac{1}{12\xi_k^2}.$$

Moreover we have

$$\sum_{k=1}^{n-1}\frac{1}{2}(\log(k+1)+\log(k))=-\frac{1}{2}\log(n)+\sum_{k=1}^{n}\log(k)=-\frac{1}{2}\log(n)+\log(n!).$$

Therefore, we have

$$n\log(n)-n+1=-\frac{1}{2}\log(n)+\log(n!)+R_n,\tag{2.11}$$

where

$$R_n=\frac{1}{12}\sum_{k=1}^{n-1}\frac{1}{\xi_k^2}<\frac{1}{12}\sum_{k=1}^{n-1}\frac{1}{k^2}<\frac{1}{12}\sum_{k=1}^{\infty}\frac{1}{k^2}.$$

Since R_n is monotone increasing and bounded (see Exercise 2.8), it converges to some R. Moreover, for $r_n=\exp(1-R_n)$, we have $r_n\to r=\exp(1-R)$. From (2.11) we have

$$r_n=\frac{n!e^n}{n^{n+1/2}}.$$

We complete the proof by proving that $r_n\to\sqrt{2\pi}$. Lemmata 2.1.14 and 2.1.13 give that

$$\frac{r_n^4}{(r_{2n})^2}=\frac{(n!)^4(2n)^{4n+1}}{n^{4n+2}(2n)!^2}=4\cdot\frac{2n+1}{2n}\frac{(n!)^4 2^{4n}}{(2n)!\,(2n+1)!}=4\cdot\frac{2n+1}{2n}\prod_{k=1}^{n}\frac{4k^2}{4k^2-1}.$$

Thus, we have

$$r^2=\lim_{n\to\infty}\frac{r_n^4}{(r_{2n})^2}=4\lim_{n\to\infty}\frac{2n+1}{2n}\prod_{k=1}^{n}\frac{4k^2}{4k^2-1}=2\pi.$$

Therefore (2.10) holds. □

2.2 *The Beta Function and Laplace's Method

In this section we describe the *beta function*, which provides a continuous analogue and generalization of the binomial coefficient. In addition to being important in probability and statistics, the beta function also allows for the generalization of the binomial theorem to real-valued powers that are not positive integers.

We also give an informal treatment of *Laplace's method*, which is a very useful tool in asymptotic analysis. Laplace's method is key to proving the general version of Stirling's approximation. We give a rigorous treatment of Laplace's method in Section 2.3.

2.2.1 The Beta Function

Definition 2.2.1. *The* beta function[14] *is defined to be*

$$\mathrm{B}(x,y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}, \tag{2.12}$$

which is well defined for all real[15] $x, y \notin \{0, -1, -2, -3, \dots\}$. *If* $x+y$ *is a nonpositive integer, but* x *and* y *are not, then we set* $\mathrm{B}(x,y) = 0$.

The beta function can also be written as an integral, as follows.

Proposition 2.2.2. *For all* $x, y > 0$ *we have*

$$\mathrm{B}(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,dt. \tag{2.13}$$

Proof. We have

$$\begin{aligned}\Gamma(x+y)\int_0^1 t^{x-1}(1-t)^{y-1}\,dt &= \int_0^\infty e^{-z}z^{x+y-1}\,dz \int_0^1 t^{x-1}(1-t)^{y-1}\,dt \\ &= \int_0^\infty \int_0^1 e^{-z}(zt)^{x-1}(z(1-t))^{y-1} z\,dt\,dz.\end{aligned}$$

Let $(u,v) = (zt, z(1-t))$; this function has Jacobian determinant $|\det D(u,v)| = z$. Using this to change variables, the previous expression becomes (see Exercise 2.9)

$$\int_0^\infty\int_0^\infty e^{-u-v}u^{x-1}v^{y-1}\,du\,dv = \left(\int_0^\infty e^{-u}u^{x-1}\,du\right)\left(\int_0^\infty e^{-v}v^{y-1}\,dv\right) = \Gamma(x)\Gamma(y).$$

This gives us the required identity:

$$\int_0^1 t^{x-1}(1-t)^{y-1}\,dt = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} = \mathrm{B}(x,y). \qquad \square$$

[14] The capital Greek letter *Beta* looks identical to the Roman letter *B*.

[15] As with the gamma function, this definition can be extended to complex values of x and y, but treating that carefully would take us beyond the scope of this book.

2.2.2 Combinatorial Identities Revisited

When a and b are nonnegative integers, we have

$$\binom{a}{b} = \frac{\Gamma(a+1)}{\Gamma(a-b+1)\Gamma(b+1)} = \frac{1}{(a+1)\,\mathrm{B}(a-b+1, b+1)}. \tag{2.14}$$

Note that the right side of this equation is defined even when a and b are not integers, so this gives an extension of the binomial coefficients to all real numbers a, b such that a, b, and $a-b$ are not negative integers. Many of the properties of binomial coefficients hold for these generalized binomial coefficients.

Proposition 2.2.3. *The following identities hold for any $a, b \in \mathbb{R}$, provided the various terms are defined.*

(i) $\displaystyle\binom{a}{0} = 1.$

(ii) $\displaystyle\binom{a}{b+1} = \binom{a}{b}\frac{a-b}{b+1}.$

(iii) $\displaystyle\binom{a}{b} = \binom{a-1}{b-1} + \binom{a-1}{b}.$

(iv) *If $k \in \mathbb{Z}^+$ and $a \in \mathbb{R}$, with $a \geq 0$, then* $\displaystyle\binom{a}{k} = a(a-1)(a-2)\cdots(a-k+1)/k!.$

Proof. The proof is Exercise 2.10. □

Nota Bene 2.2.4. Not all properties of the integer binomial coefficients hold in the general case. For example, the coefficient $\binom{a}{b}$ does not vanish for $b > a$ unless $a - b \in \mathbb{Z}$.

Theorem 2.2.5 (Binomial Series). *Given any $x \in (-1, 1)$ and any $a \geq 0$ the following series converges absolutely:*

$$(1+x)^a = \sum_{k=0}^{\infty} \binom{a}{k} x^k. \tag{2.15}$$

Proof. Convergence follows from the ratio test since $\binom{a}{k} \to -1$ as $k \to \infty$. Hence the series converges for $x \in (-1, 1)$. To show that the series converges to $(1+x)^a$, compute the Taylor series of the right-hand side and match coefficients. The details are Exercise 2.11. □

Remark 2.2.6. As an alternative proof of the theorem, we can show that $f(x) = \sum_{k=0}^{\infty} \binom{a}{k} x^k$ satisfies a simple linear differential equation; see Exercise 2.14.

Example 2.2.7. By Proposition 2.2.3(ii) we have

$$\binom{\frac{1}{2}}{0} = 1, \quad \binom{\frac{1}{2}}{1} = \frac{1}{2}, \quad \binom{\frac{1}{2}}{2} = -\frac{1}{8}, \quad \binom{\frac{1}{2}}{3} = \frac{1}{16}, \quad \text{and } \binom{\frac{1}{2}}{4} = -\frac{5}{128}.$$

Thus for any $x \in (-1, 1)$ we have

$$(1+x)^{1/2} = \sum_{k=0}^{\infty} \binom{\frac{1}{2}}{k} x^k = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} - \frac{5x^4}{128} + \cdots.$$

This agrees with the Taylor expansion of $f(x) = (1+x)^{1/2}$ around $x = 0$.

Example 2.2.8. By Proposition 2.2.3(ii) we have

$$\binom{\frac{1}{3}}{0} = 1, \quad \binom{\frac{1}{3}}{1} = \frac{1}{3}, \quad \binom{\frac{1}{3}}{2} = -\frac{1}{9}, \quad \binom{\frac{1}{3}}{3} = \frac{5}{81}, \quad \text{and } \binom{\frac{1}{3}}{4} = -\frac{10}{243}.$$

Thus for any $x \in (-1, 1)$ we have

$$(1+x)^{1/3} = \sum_{k=0}^{\infty} \binom{\frac{1}{3}}{k} x^k = 1 + \frac{x}{3} - \frac{x^2}{9} + \frac{5x^3}{81} - \frac{10x^4}{243} + \cdots.$$

This agrees with the Taylor expansion of $f(x) = (1+x)^{1/3}$ around $x = 0$.

2.2.3 Trigonometric Integrals

The beta function can be rewritten as a trigonometric integral.

Proposition 2.2.9. *For any $a, b > 0$ we have*

$$\mathrm{B}(a,b) = 2\int_0^{\frac{\pi}{2}} \sin^{2a-1}(u)\cos^{2b-1}(u)\,du. \tag{2.16}$$

Proof. The proof is Exercise 2.13. □

As a special case of the previous proposition, we can express the Wallis integrals W_n (see Lemma 2.1.12) in terms of the beta function.

Corollary 2.2.10. *For any $n \in \mathbb{N}$, we have*

$$W_n = \int_0^{\pi/2} \sin^n(x)\,dx = \frac{1}{2}\,\mathrm{B}\left(\frac{n+1}{2}, \frac{1}{2}\right) = \frac{\sqrt{\pi}}{2}\,\frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2}+1)}.$$

Proof. This follows immediately from (2.16) and the definition of $\mathrm{B}(x, y)$. □

2.2.4 Laplace's Method: Simple Version

Here we give a common form of Laplace's method and a heuristic argument for it. A rigorous proof is given in Section 2.3.

Theorem 2.2.11 (Laplace's Method). *Assume $f : [a, b] \to \mathbb{R}$ is smooth and has a unique global maximum at $t_0 \in (a, b)$. As $x \to \infty$, we have*

$$\int_a^b e^{xf(t)}\,dt \sim e^{xf(t_0)}\sqrt{\frac{2\pi}{x|f''(t_0)|}}. \tag{2.17}$$

Rough Argument. Consider the Taylor expansion of f at t_0 (see Theorem 10.3.7). Since f has a local maximum at t_0, it follows that $f'(t_0) = 0$ and $f''(t_0) < 0$. Hence, in a neighborhood of t_0, we have that $f(t) \approx f(t_0) - \frac{1}{2}|f''(t_0)|(t - t_0)^2$. Thus,

$$\begin{aligned}\int_a^b e^{xf(t)}\,dt &\approx \int_a^b e^{xf(t_0) - x|f''(t_0)|(t-t_0)^2/2}\,dt \\ &= e^{xf(t_0)}\int_a^b e^{-x|f''(t_0)|(t-t_0)^2/2}\,dt \\ &= \frac{e^{xf(t_0)}}{\sqrt{x|f''(t_0)|}}\int_{-\sqrt{x|f''(t_0)|}(t_0-a)}^{\sqrt{x|f''(t_0)|}(b-t_0)} e^{-u^2/2}\,du.\end{aligned}$$

From Exercise 2.5 we know that $\int_{-\infty}^{\infty} e^{-t^2/2}\,dt = \sqrt{2\pi}$, so as $x \to \infty$, we have

$$\int_{-\sqrt{x|f''(t_0)|}(t_0-a)}^{\sqrt{x|f''(t_0)|}(b-t_0)} e^{-u^2/2}\,du \to \int_{-\infty}^{\infty} e^{-u^2/2}\,du = \sqrt{2\pi},$$

which yields (2.17). □

2.2.5 Proof of Stirling's Approximation (Theorem 2.1.8)

In the integral defining Γ, make the substitution $\tau = xt$ to get

$$\begin{aligned}\Gamma(x+1) &= \int_0^\infty \tau^x e^{-\tau}\,d\tau = \int_0^\infty e^{x\log\tau - \tau}\,d\tau \\ &= \int_0^\infty x e^{x\log xt - xt}\,dt = x^{x+1}\int_0^\infty e^{x(\log t - t)}\,dt.\end{aligned}$$

We can use (2.17) to approximate the integral with $f(t) = \log t - t$, which has its maximum at $t_0 = 1$ and satisfies $f''(t_0) = -1$. For $x \gg 1$ we have

$$\int_0^\infty e^{x(\log t - t)}\,dt \sim e^{-x}\sqrt{\frac{2\pi}{x}}. \tag{2.18}$$

Thus (2.6) holds. □

2.2.6 Asymptotic Expansions

Remark 2.1.11 mentions an improved version of Stirling's approximation (2.7), which includes a lower-order term. The leading-order version (2.6) gives the asymptotic limit, but the inclusion of lower-order terms provides greater accuracy for intermediate values of x. In this subsection we describe how lower-order terms can be accounted for, using what are called *asymptotic expansions*.

Definition 2.2.12. *Let $\{\phi_k\}_{k=0}^{\infty}$ be a sequence of real-valued continuous functions defined for x sufficiently large in $\mathbb{R}$, satisfying $\phi_{k+1}(x) \in o(\phi_k(x))$ as $x \to \infty$ for each $k \in \mathbb{N}$. The series $\sum_{k=0}^{\infty} a_k \phi_k(x)$ is an* asymptotic expansion *of the function f as $x \to \infty$ if for each $n \in \mathbb{N}$ the remainder function r_n satisfies*

$$r_n(x) = f(x) - \sum_{k=0}^{n-1} a_k \phi_k(x) \in o(\phi_n(x)) \quad \text{as } x \to \infty. \tag{2.19}$$

Remark 2.2.13. The leading-order behavior of an asymptotic expansion is given by $a_0\phi_0(x)$. The additional terms in the series are all of lower order and do not affect the leading-order behavior. Therefore it is correct to write $f(x) \sim \sum_{k=0}^{\infty} a_k \phi_k(x)$ as $x \to \infty$. Throughout the remainder of this chapter, we use the symbol $\sim$ to mean "has the asymptotic expansion" when aligning a function to a series.

Example 2.2.14. Although it is algebraically complicated (see Exercise 2.21) to compute additional lower-order terms of Stirling's approximation, we can expand the gamma function even further to get

$$\Gamma(x+1) \sim x^{x+1} e^{-x} \sqrt{\frac{2\pi}{x}} \left(1 + \frac{1}{12x} + \frac{1}{288x^2} - \frac{139}{51840x^3} + \cdots\right) \tag{2.20}$$

as $x \to \infty$. This is an asymptotic expansion because each term is of the form $\phi_k(x) = e^{-x} x^{x-1/2-k}$, which satisfies $\phi_{k+1}(x) \in o(\phi_k(x))$, for each $k \in \mathbb{N}$.

Example 2.2.15. We prove that

$$\frac{1}{1+x} \sim \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{x^k} \quad \text{as } x \to \infty$$

by showing for all $n \in \mathbb{N}$ that (see Exercise 2.15 for details)

$$r_n(x) = \frac{1}{1+x} - \sum_{k=1}^{n} \frac{(-1)^{k-1}}{x^k} = \frac{(-1)^n}{x^n} \left(\frac{1}{1+x}\right) \in o(x^{-n}) \quad \text{as } x \to \infty.$$

2.3 *Laplace's Method and Stirling Improved

Laplace's method has several different formulations. Section 2.2.4 gives a fairly simple, yet common, version of Laplace's method and a rough sketch of why you should believe it. Although that heuristic argument is commonly given as a proof, it is not rigorous. Here we state and carefully prove a much stronger form of Laplace's method. The main tool used in this proof is Watson's lemma, which we also prove carefully. We finish the section by using these results to prove the more refined version of Stirling's approximation (2.7).

2.3.1 Extending Big-O and Little-o

In order to treat Laplace's method and Watson's lemma, we must first extend big-O and little-o notation to describe the convergence of a function at a point $t_0 \in \mathbb{R}$. We also describe the notion of an asymptotic expansion as $t \to t_0$.

Definition 2.3.1. *Let f and g be real-valued functions defined in a neighborhood of $t_0 \in \mathbb{R}$. We say that $f(t)$ is* big-O *of $g(t)$ as $t \to t_0$, denoted $f(t) \in O(g(t))$, if there exist $M > 0$ and $\delta > 0$ such that $|f(t)| \leq M|g(t)|$ whenever $|t - t_0| < \delta$. Similarly, we say that $f(t)$ is* little-o *of $g(t)$ as $t \to t_0$, denoted $f(t) \in o(g(t))$, if for each $\varepsilon > 0$ there exists $\delta > 0$ such that $|f(t)| \leq \varepsilon|g(t)|$ whenever $|t - t_0| < \delta$.*

Remark 2.3.2. When we talk about big-O and little-o as $x \to \infty$, we can think of it as convergence at infinity. In this sense, Definitions 1.1.4 and 2.3.1 are the same.

Definition 2.3.3. *Let $\{\phi_k\}_{k=0}^{\infty}$ be a sequence of real-valued continuous functions defined in a neighborhood of $t_0 \in \mathbb{R}$, satisfying $\phi_{k+1}(t) \in o(\phi_k(t))$ as $t \to t_0$ for all $k \in \mathbb{N}$. We say that the series $\sum_{k=0}^{\infty} a_k \phi_k(t)$ is an* asymptotic expansion *of the function f in a neighborhood of t_0 if for each $n \in \mathbb{N}$ the remainder function r_n satisfies*

$$r_n(t) = f(t) - \sum_{k=0}^{n-1} a_k \phi_k(t) \in o(\phi_n(t)) \quad \text{as } t \to t_0. \tag{2.21}$$

Following the justification in Remark 2.2.13, we denote this as $f(t) \sim \sum_{k=0}^{\infty} a_k \phi_k(t)$ as $t \to t_0$.

Example 2.3.4. We prove that

$$\log(1+t) \sim \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} t^k \quad \text{as } t \to 0^+$$

by showing for all $n \in \mathbb{Z}^+$ that (see Exercise 2.16 for details)

$$r_n(t) = \log(1+t) - \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} t^k \in o(t^n) \quad \text{as } t \to 0^+.$$

2.3.2 Watson's Lemma

To give a rigorous proof of Laplace's method, we need *Watson's lemma.* This describes the asymptotic behavior of integrals of the form

$$I(x) = \int_0^\infty e^{-xt} f(t)\, dt \quad \text{as } x \to \infty. \tag{2.22}$$

Watson's lemma says that the main contribution of $f(t)$ in (2.22) comes from its behavior near $t = 0$; the behavior of $f(t)$ elsewhere is wiped out by exponential decay as $x \to \infty$. However, before we prove Watson's lemma, we need the following.

Lemma 2.3.5 (Small Laplace Tail). *Let $f : [0, \infty) \to \mathbb{R}$ be a continuous, real-valued function and $\bar{x} \in \mathbb{R}$. If $I(\bar{x})$ in (2.22) converges absolutely (meaning that $\int_0^\infty e^{-\bar{x}t}|f(t)|dt$ is finite), then for all $\delta > 0$ there exists $M > 0$ such that*

$$J(x) = \int_\delta^\infty e^{-xt} f(t)\, dt$$

satisfies $|J(x)| \le M e^{-\delta(x-\bar{x})}$ whenever $x \ge \bar{x}$. It follows that $|J(x)| \in o(x^p)$ for all $p \in \mathbb{R}$, as $x \to \infty$.

Proof. For a given $\delta > 0$ and each $T \in [\delta, \infty)$, define

$$K(T) = \int_\delta^T e^{-\bar{x}t} f(t)\, dt.$$

Let $\varepsilon > 0$ be given. Since $I(\bar{x})$ is absolutely convergent, we choose $T_0 > 0$ so that

$$\left| \int_T^\infty e^{-\bar{x}t} f(t)\, dt \right| < \varepsilon$$

whenever $T \ge T_0$. Since K is continuous, it is bounded on $[\delta, T_0]$. Therefore, we can set $M_0 = \sup_{T \in [\delta, T_0]} |K(T)|$; moreover, K is bounded on $[T_0, \infty)$ by $M = M_0 + \varepsilon$. Thus, for $x > \bar{x}$, we have

$$J(x) = \int_\delta^\infty e^{-(x-\bar{x})t} e^{-\bar{x}t} f(t)\, dt = \int_\delta^\infty e^{-(x-\bar{x})t} K'(t)\, dt = (x-\bar{x}) \int_\delta^\infty e^{-(x-\bar{x})t} K(t)\, dt.$$

And this gives

$$|J(x)| \le |x-\bar{x}| \int_\delta^\infty e^{-(x-\bar{x})t} |K(t)|\, dt \le |x-\bar{x}| M \int_\delta^\infty e^{-(x-\bar{x})t}\, dt = M e^{-\delta(x-\bar{x})}. \qquad \square$$

Theorem 2.3.6 (Watson's Lemma). *If $f(t) \sim t^\alpha \sum_{k=0}^\infty a_k t^{\beta k}$ as $t \to 0^+$, where $\alpha > -1$ and $\beta > 0$, then*

$$\int_0^\infty e^{-xt} f(t)\, dt \sim \sum_{k=0}^\infty \frac{a_k \Gamma(\alpha + \beta k + 1)}{x^{\alpha+\beta k+1}} \quad \text{as } x \to \infty, \tag{2.23}$$

provided the integral converges absolutely for all sufficiently large x.

Proof. It suffices to show for each $n \in \mathbb{N}$ that

$$R_n(x) = \int_0^\infty e^{-xt} f(t)\,dt - \sum_{k=0}^{n} \frac{a_k \Gamma(\alpha+\beta k+1)}{x^{\alpha+\beta k+1}} \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right) \quad \text{as } x \to \infty.$$

Exercise 2.4 gives

$$\int_0^\infty e^{-xt} t^{\alpha+\beta k}\,dt = \frac{\Gamma(\alpha+\beta k+1)}{x^{\alpha+\beta k+1}}, \tag{2.24}$$

which yields

$$\begin{aligned} |R_n(x)| &= \left| \int_0^\infty e^{-xt} f(t)\,dt - \sum_{k=0}^{n} \frac{a_k \Gamma(\alpha+\beta k+1)}{x^{\alpha+\beta k+1}} \right| \\ &= \left| \int_0^\infty e^{-xt} f(t)\,dt - \sum_{k=0}^{n} a_k \int_0^\infty e^{-xt} t^{\alpha+\beta k}\,dt \right| \\ &\le \int_0^\infty e^{-xt} r_n(t)\,dt, \end{aligned}$$

where

$$r_n(t) = \left| f(t) - t^\alpha \sum_{k=0}^{n} a_k t^{\beta k} \right| \in o(t^{\alpha+\beta n}) \quad \text{as } t \to 0^+.$$

Thus, given $\varepsilon > 0$, there exists $\delta_n > 0$ such that $|r_n(t)| \le \varepsilon t^{\alpha+\beta n}$ for all $t \in [0, \delta_n]$. Hence, we can decompose $R_n(x)$ as $R_n(x) = I_n(x) + J_n(x)$, where

$$I_n(x) = \int_0^{\delta_n} e^{-xt} r_n(t)\,dt \quad \text{and} \quad J_n(x) = \int_{\delta_n}^\infty e^{-xt} r_n(t)\,dt.$$

From (2.24), we have

$$|I_n(x)| \le \varepsilon \int_0^{\delta_n} e^{-xt} t^{\alpha+\beta n}\,dt < \varepsilon \frac{\Gamma(\alpha+\beta n+1)}{x^{\alpha+\beta n+1}} \quad \text{as } x \to \infty,$$

which implies that $|I_n(x)| \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right)$ as $x \to \infty$. By Lemma 2.3.5, we also have that $|J_n(x)| \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right)$ as $x \to \infty$. Thus, $R_n(x) \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right)$ as $x \to \infty$. ∎

Example 2.3.7. Using Example 2.3.4 with Watson's lemma, we have

$$\int_0^\infty e^{-xt} \log(1+t)\,dt \sim \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\Gamma(k+1)}{k x^{k+1}} = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\Gamma(k)}{x^{k+1}} \quad \text{as } x \to \infty.$$

Remark 2.3.8. As mentioned above, Watson's lemma says that essentially all of the contribution to the integral (2.22) from $f(t)$ comes from its behavior near zero. In the corollary below, we see that we don't even need to integrate all the way to infinity to get the exact same asymptotic expansion.

Corollary 2.3.9. *Let $\delta > 0$ be given. If $f(t) \sim t^\alpha \sum_{k=0}^\infty a_k t^{\beta k}$ as $t \to 0^+$, where $\alpha > -1$ and $\beta > 0$, then*

$$I_\delta(x) = \int_0^\delta e^{-xt} f(t)\, dt \sim \sum_{k=0}^\infty \frac{a_k \Gamma(\alpha + \beta k + 1)}{x^{\alpha+\beta k+1}} \quad \text{as } x \to \infty, \tag{2.25}$$

provided the integral (2.22) converges absolutely for all sufficiently large x.

Proof. For each $n \in \mathbb{N}$ the proof of Watson's lemma gives

$$R_n(x) = \int_0^\infty e^{-xt} f(t)\, dt - \sum_{k=0}^n \frac{a_k \Gamma(\alpha + \beta k + 1)}{x^{\alpha+\beta k+1}} \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right) \quad \text{as } x \to \infty.$$

From Lemma 2.3.5, we have

$$J_\delta(x) = \int_\delta^\infty e^{-xt} f(t)\, dt \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right) \quad \text{as } x \to \infty.$$

Combining these gives

$$\int_0^\delta e^{-xt} f(t)\, dt - \sum_{k=0}^n \frac{a_k \Gamma(\alpha + \beta k + 1)}{x^{\alpha+\beta k+1}} = R_n(x) - J_\delta(x) \in o\left(\frac{1}{x^{\alpha+\beta n+1}}\right)$$

as $x \to \infty$. Thus, (2.25) holds. ∎

Remark 2.3.10. When splitting up (2.22) into the two parts

$$\int_0^\infty e^{-xt} f(t)\, dt = \int_0^\delta e^{-xt} f(t)\, dt + \int_\delta^\infty e^{-xt} f(t)\, dt,$$

it is remarkable that the first term on the right-hand side has the asymptotic contribution as $x \to \infty$ and that the second term decays exponentially. As shown in the next subsection, when considering integrals of the form

$$\int_0^\infty e^{xh(t)} f(t)\, dt,$$

the relevant part, asymptotically speaking, is near the maximum of $h(t)$. In the case of (2.22) with $h(t) = -t$, the maximum occurs when $t = 0$, so that's the most relevant part.

2.3.3 Laplace's Method

Laplace's method is a generalization of Watson's lemma. There are a few variations in the literature. We prove a fairly general version here.

Theorem 2.3.11 (Laplace's Method, General Version). *Let $f : [a,b] \to \mathbb{R}$ be continuous and $h : [a,b] \to \mathbb{R}$ be continuously differentiable. Assume that h has a unique maximum at a, that is, assume there exists $c \in (a,b)$ and constant M such that $h'(t) < 0$ on $(a,c]$ and $h(t) \leq M < h(a)$ for $t \in [c,b]$. Hence the function $\xi(t) = h(a) - h(t)$ is strictly increasing on $[a,c]$ and is invertible with a continuously differentiable inverse denoted $t(\xi)$. If $F(\xi) = f(t(\xi))t'(\xi)$ has the asymptotic expansion*

$$F(\xi) \sim \xi^{\alpha} \sum_{k=0}^{\infty} \gamma_k \xi^{\beta k} \quad \text{as } \xi \to 0^+,$$

with $\alpha > -1$ and $\beta > 0$, then

$$I(x) = \int_a^b e^{xh(t)} f(t)\, dt \sim e^{xh(a)} \sum_{k=0}^{\infty} \frac{\gamma_k \Gamma(\alpha + \beta k + 1)}{x^{\alpha+\beta k+1}} \quad \text{as } x \to \infty, \tag{2.26}$$

provided that the integral converges absolutely for all x sufficiently large.

Proof. Assume the integral $I(x)$ converges absolutely for all $x > L$. Note that

$$\begin{aligned} I(x) &= e^{xh(a)} \int_a^c e^{-x\xi(t)} f(t)\, dt + \int_c^b e^{xh(t)} f(t)\, dt \\ &= e^{xh(a)} \int_0^{\xi(c)} e^{-x\xi} f(t(\xi)) t'(\xi)\, d\xi + \int_c^b e^{(x-L)h(t)} e^{Lh(t)} f(t)\, dt. \end{aligned}$$

Thus, for $x > L$ we have

$$\left| I(x) - e^{xh(a)} \int_0^{\xi(c)} e^{-x\xi} F(\xi)\, d\xi \right| \leq e^{(x-L)M} \int_c^b e^{Lh(t)} |f(t)|\, dt \leq C e^{xM},$$

where $C > 0$ is constant. Since $M < h(a)$, we have that $Ce^{xM} \in o\left(\frac{e^{xh(a)}}{x^{\alpha+\beta k+1}}\right)$ as $x \to \infty$ for all $k = \mathbb{N}$. Thus by Corollary 2.3.9, we have

$$I(x) \sim e^{xh(a)} \int_0^{\xi(c)} e^{-x\xi} F(\xi)\, d\xi \sim e^{xh(a)} \sum_{k=0}^{\infty} \frac{\gamma_k \Gamma(\alpha + \beta k + 1)}{x^{\alpha+\beta k+1}} \quad \text{as } x \to \infty. \qquad \square$$

Corollary 2.3.12. *For $\lambda > 0$ and $\alpha > -1$, we have*

$$\int_0^{\infty} e^{-xt^{\lambda}} t^{\alpha}\, dt = \frac{1}{\lambda} \frac{\Gamma\left(\frac{\alpha+1}{\lambda}\right)}{x^{\frac{\alpha+1}{\lambda}}}. \tag{2.27}$$

Proof. This is Exercise 2.17. $\square$

2.3.4 Stirling's Approximation Refined

Using the more general form of Laplace's method (Theorem 2.3.11), we can derive the improved version of Stirling's approximation (2.7).

Theorem 2.3.13 (Asymptotic Expansion of Stirling's Approximation).

$$\Gamma(x+1) \sim x^{x+1}e^{-x}\sqrt{\frac{2\pi}{x}}\left(1+\frac{1}{12x}+\cdots\right) \quad as\ x\to\infty. \tag{2.28}$$

Proof. As in Section 2.2.5, letting $\tau = xt$ gives

$$\Gamma(x+1)=\int_0^\infty \tau^x e^{-\tau}\,d\tau=\int_0^\infty e^{x\log\tau-\tau}\,d\tau=\int_0^\infty xe^{x\log xt-xt}\,dt=x^{x+1}\int_0^\infty e^{x(\log t-t)}\,dt.$$

To prove (2.28) it suffices to show that

$$\int_0^\infty e^{x(\log t-t)}\,dt \sim e^{-x}\sqrt{\frac{2\pi}{x}}\left(1+\frac{1}{12x}+\cdots\right) \quad \text{as } x\to\infty. \tag{2.29}$$

Note that $\log t - t$ has a maximum at $t=1$, and the substitution $s = t-1$ moves the maximum to $s=0$. Thus, we have

$$\int_0^\infty e^{x(\log t-t)}\,dt = e^{-x}\int_{-1}^\infty e^{xh(s)}\,ds \sim e^{-x}\int_{-1}^1 e^{xh(s)}\,ds \quad \text{as } x\to\infty,$$

where $h(s)=\log(1+s)-s$. Write $h(s) = -s^2/2+g(s)$, where $g(s)=\sum_{k=3}^\infty \frac{(-1)^{k+1}s^k}{k}$. Expanding the exponential $e^{xg(s)}$ gives

$$\int_{-1}^1 e^{xh(s)}\,ds=\int_{-1}^1 e^{-xs^2/2}e^{xg(s)}\,ds=\int_{-1}^1 e^{-xs^2/2}\left(1+xg(s)+\frac{x^2}{2}g(s)^2+\cdots\right)ds.$$

The terms of odd order in s integrate to zero, thus reducing the expansion to

$$\int_{-1}^1 e^{-xs^2/2}\left(1-x\left(\frac{s^4}{4}+\frac{s^6}{6}+\cdots\right)+\frac{1}{2}x^2\left(\frac{s^6}{9}+\frac{47s^8}{240}+\cdots\right)+\cdots\right)ds. \tag{2.30}$$

Corollary 2.3.12 shows that the terms above that will contribute to the first lower-order term ($\frac{1}{x}$) in (2.28) are of the form

$$\int_{-1}^1 e^{-xs^2/2}x^a s^{2b}\,ds, \tag{2.31}$$

where $b-a=1$. The only terms in (2.30) satisfying this condition have $a=1, b=4$ or $a=2, b=6$. By even symmetry we can halve the domain and double the integral. Further, the asymptotic expansion is the same when we integrate from 0 to infinity instead of to 1. Thus we have

$$\int_{-1}^1 e^{xh(s)}\,ds \sim 2\int_0^\infty e^{-xs^2/2}\left(1-x\frac{s^4}{4}+x^2\frac{s^6}{18}+\cdots\right)ds.$$

Changing variables to $\xi(s) = s^2/2$ and following Theorem 2.3.11, we have

$$\begin{aligned}
\int_{-1}^{1} e^{xh(s)}\,ds &\sim 2\int_0^\infty e^{-x\xi}\left(1 - x\frac{(2\xi)^2}{4} + x^2\frac{(2\xi)^3}{18} + \cdots\right)\frac{d\xi}{\sqrt{2\xi}} \\
&= \sqrt{2}\int_0^\infty e^{-x\xi}\left(\xi^{-1/2} - x\xi^{3/2} + x^2\frac{4}{9}\xi^{5/2} + \cdots\right)d\xi \\
&= \sqrt{2}\left(\frac{\Gamma(1/2)}{x^{1/2}} - x\frac{\Gamma(5/2)}{x^{5/2}} + x^2\frac{4}{9}\frac{\Gamma(7/2)}{x^{7/2}} + \cdots\right) \\
&= \sqrt{\frac{2\pi}{x}}\left(1 - x\frac{3}{4x^2} + x^2\frac{5}{6x^3} + \cdots\right) \\
&= \sqrt{\frac{2\pi}{x}}\left(1 + \frac{1}{12x} + \cdots\right). \quad \square
\end{aligned}$$

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

2.1. Prove Proposition 2.1.1 as follows:

(i) Use the fact that $\log(x)$ is a strictly increasing function on $[1,\infty)$ to show that

$$\sum_{k=1}^{n-1}\log(k) \le \int_1^n \log(x)\,dx \le \sum_{k=1}^{n}\log(k).$$

(ii) Use the previous step to show that

$$n\log(n) - n + 1 \le \log(n!) \le (n+1)\log(n) - n + 1$$

and hence

$$\frac{n^n}{e^{n-1}} \le n! \le \frac{n^{n+1}}{e^{n-1}}.$$

2.2. Let $y > 0$ be fixed. Prove that the binomial coefficient satisfies

$$\binom{x+y}{x} \sim \frac{x^y}{y!} \quad \text{as } x \to \infty.$$

2.3. Find the leading-order behavior of $\binom{3n}{n}$ as $n \to \infty$.

2.4. Let $p > -1$ and $x > 0$. Prove that

$$\int_0^\infty e^{-xt} t^p \, dt = \frac{\Gamma(p+1)}{x^{p+1}}.$$

2.5. ⚠ Perform the following steps:

(i) Show that

$$\int_{-\infty}^\infty e^{-x^2/2} \, dx = \sqrt{2\pi}.$$

Hint: Use polar coordinates to compute

$$\left(\int_{-\infty}^\infty e^{-x^2/2} \, dx \right)^2 = \int_{-\infty}^\infty \int_{-\infty}^\infty e^{-(x^2+y^2)/2} \, dx dy.$$

(ii) Using the substitution $t = u^2/2$ show that

$$\Gamma(x) = \frac{1}{2^{x-1}} \int_0^\infty e^{-u^2/2} u^{2x-1} \, du.$$

(iii) Show that $\Gamma(\frac{1}{2}) = \sqrt{\pi}$.

(iv) Show that

$$\int_0^\infty e^{-xt^2} \, dt = \frac{1}{2}\sqrt{\frac{\pi}{x}}.$$

2.6. Prove Lemma 2.1.12 as follows:

(i) Prove that $W_n > 0$ for all $n \in \mathbb{N}$.

(ii) Prove that $W_n - W_{n+1} = \int_0^{\pi/2} \sin^n(x)(1-\sin(x)) \, dx > 0$ for every $n \in \mathbb{N}$.

(iii) Prove that $W_0 = \pi/2$ and that $W_1 = 1$ by direct computation.

(iv) For $n \geq 2$, show that $W_n = (n-1)(W_{n-2} - W_n)$. Hint: Use integration by parts.

2.7. Prove Lemma 2.1.14 as follows:

(i) Show that

$$\prod_{k=1}^n \frac{4k^2}{4k^2-1} = (n!)^2 4^n \prod_{k=1}^n \frac{1}{(2k-1)(2k+1)}.$$

(ii) Show that

$$\prod_{k=1}^n (2k-1) = \frac{(2n)!}{2^n (n!)}.$$

(iii) Combine the previous two results to show that

$$\prod_{k=1}^n \frac{4k^2}{4k^2-1} = \frac{(n!)^4 2^{4n}}{(2n)!\,(2n+1)!}.$$

2.8. Complete the proof of Stirling's approximation by justifying the following steps:

(i) Go through the second integration by parts argument and show that

$$\int_k^{k+1} f(x)\,dx = \frac{1}{12}.$$

(ii) Show that $\sum_{k=1}^{\infty} \frac{1}{k^2}$ is bounded.

2.9.* Complete the proof of Proposition 2.2.2 by showing that

$$\int_0^\infty \int_0^1 e^{-z}(zt)^{x-1}(z(1-t))^{y-1} z\,dt\,dz = \int_0^\infty \int_0^\infty e^{-u-v}u^{x-1}v^{y-1}\,du\,dv.$$

Hint: Remember the change-of-variables formula (see Volume 1, Section 8.7).

2.10.* Prove Proposition 2.2.3.

2.11.* Give all the details for a careful proof of Theorem 2.2.5 as follows:

(i) Prove that the series converges.

(ii) Prove that the coefficients in the Taylor series expansion of $(1+x)^a$ around $x = 0$ satisfy the same relations as the binomial coefficients.

(iii) Prove these relations imply that the Taylor coefficients are the same as the binomial coefficients.

2.12.* Prove the following identities:

(i) If $k \in \mathbb{Z}^+$, then

$$\binom{\frac{1}{2}}{k} = \binom{2(k-1)}{k-1}\frac{(-1)^{k-1}}{2^{2k-1}k}.$$

(ii) As an alternative to Example 2.2.7, we have

$$(1+x)^{1/2} = 1 - 2\sum_{k=0}^{\infty} \frac{1}{k+1}\binom{2k}{k}\left(\frac{-x}{4}\right)^{k+1}. \tag{2.32}$$

2.13.* Prove Proposition 2.2.9 by substituting $t = \sin^2(u)$, $1-t = \cos^2(u)$, and $dt = 2\sin(u)\cos(u)\,du$ in the integral formula (2.13) for B.

2.14.* Give an alternative proof of Theorem 2.2.5 by proving that both sides of (2.15) satisfy the differential equation $(1+x)y' = ay$ subject to $y(0) = 1$.

2.15.* Assuming that $n \in \mathbb{Z}^+$, prove the following:

(i) If $|z| < 1$, then

$$\frac{1}{1+z} - \sum_{k=0}^{n}(-1)^k z^k = (-1)^{n+1}\left(\frac{z^{n+1}}{1+z}\right). \tag{2.33}$$

(ii) If $|x| > 1$, then

$$\frac{1}{1+x} - \sum_{k=1}^{n}\frac{(-1)^{k-1}}{x^k} = \frac{(-1)^n}{x^n}\left(\frac{1}{1+x}\right).$$

Hint: Set $x = 1/z$ in (2.33).

2.16.* Integrate (2.33) from 0 to $t > 0$ and reindex to show that

$$\log(1+t) - \sum_{k=1}^{N} \frac{(-1)^{k+1}}{k} t^k \in o(t^N) \quad \text{as } t \to 0^+.$$

2.17.* Prove Corollary 2.3.12.

2.18.* Find the leading-order behavior of the integral

$$\int_{-1}^{1} e^{-x \cosh t}\, dt \quad \text{as } x \to \infty.$$

2.19.* Find the leading-order behavior of the integral

$$\int_{-1}^{\frac{1}{2}} e^{-xt^2}\, dt \quad \text{as } x \to \infty.$$

2.20.* Find the leading-order behavior of the integral

$$\int_{0}^{\pi+\varepsilon} e^{-x \cos t}\, dt \quad \text{as } x \to \infty$$

for any $0 < \varepsilon < \pi/2$. Note that this behavior does not change as $\varepsilon \to 0$.

2.21.* Extend Theorem 2.3.13 to one more term; that is, show that

$$\Gamma(x+1) \sim x^{x+1} e^{-x} \sqrt{\frac{2\pi}{x}} \left(1 + \frac{1}{12x} + \frac{1}{288x^2} + \cdots\right).$$

Notes

For more on the Bohr–Mollerup theorem and the uniqueness of the gamma function see [BB08, Theorem 5.10] and [Art64, Theorem 2.1]. Our proof of the short form of Stirling's approximation follows that of [BB08], with additional ideas about Wallis integrals from [Wik18b]. For more on Stirling's formula, see [Con16, Wik18a]. For more on the coefficients of the Stirling series, see [Nem10]. Our proofs of Watson's lemma and Lagrange's method were inspired in part by [vRB12].

3 Data Structures

Bad programmers worry about the code. Good programmers worry about data structures and their relationships.
—Linus Torvalds

A *data structure* is a specialized format for organizing, storing, and processing data. Seemingly inconsequential differences between similar data structures can have a profound impact on the complexity of the algorithms that use them. In this chapter we discuss a few widely used data structures and prove complexity bounds for several important algorithms.

One of the most fundamental low-level data structures is the *array*. An array consists of a collection of several elements of a specific data type (integer, floating-point number, character, memory address, etc.) stored together in a contiguous block of allocated memory. Because the data types of the entries in an array are all the same, each one uses the same amount of memory, so we can easily compute the memory address of the kth entry, for any k, and can access or modify its value in constant time. Arrays are convenient to use, since most modern programming languages have (highly optimized) built-in functions for working with them. For some applications (like many linear algebra algorithms) arrays are a very efficient data structure.

However, arrays have the disadvantage that their length cannot be modified dynamically. We cannot add more data to an array than was originally allocated, and there's no graceful way to make the array bigger. Instead, we must construct a new, larger array elsewhere in memory and copy the existing data to it, along with the new data being added. This can be very costly, especially if it happens frequently. Arrays also cannot support mixed data types in the same array.

Applications that need to accommodate collections of dynamically varying size often benefit from more versatile data structures that can dynamically and gracefully expand and contract, as needed, to accommodate the underlying application and can do so without having to waste time and space shuffling data around unnecessarily.

Most dynamic data structures spread their data across several disparate blocks of memory and manage the blocks by keeping track of their memory addresses. Data

that consists of a memory address for other data is called a *pointer*. In some cases the pointers are all managed centrally in some kind of manifest or lookup table, and in other cases the pointers are distributed across the various blocks, forming chains, trees, or other network structures. In many cases these dynamic data structures can gracefully store and dynamically process large quantities of data in memory, on a hard drive, on an array of hard drives, or even in a large network of distributed storage devices.

The sewing together of blocks of data into a sophisticated network structure allows for the efficient insertion, deletion, and search across an entire collection of data elements. This network structure relies on the mathematical *theory of graphs*, which provides the rigorous abstraction needed to describe these data structures and facilitate their analysis. Graphs can also be used to describe other kinds of networks such as those used in communications, multiagent systems, and sophisticated supply chains.

3.1 Theory of Graphs

In this section we describe some basic elements of graph theory. Graph theory is a fundamental tool for analyzing and constructing data structures and algorithms.

3.1.1 Graphs

Definition 3.1.1. *A* directed graph $G = (V, E)$ *is a pair consisting of a nonempty, finite set* V *of* vertices *(or* nodes*) and a set* $E \subset V \times V$ *of ordered pairs, called the* edges *of the graph. A pair* $(v_i, v_j) \in E$ *is an edge from vertex* v_i *to vertex* v_j*.*

Example 3.1.2. Figure 3.1 depicts the directed graph $G_1 = (V_1, E_1)$ with vertices $V_1 = \{1, 2, 3, 4\}$ and edges $E_1 = \{(1, 2), (1, 3), (2, 1), (4, 2), (4, 3)\}$.

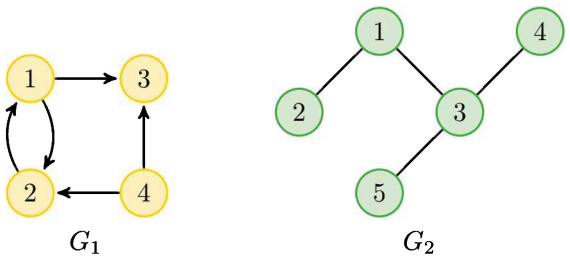


Figure 3.1. *Depiction of a directed graph* (G_1) *and an undirected graph* (G_2)*, as described in Examples 3.1.2 and 3.1.6. A directed graph has arrows that identify the directions of the edges, but an undirected graph does not specify a direction for its edges.*

Remark 3.1.3. Note that we defined the collection of edges E in a graph to be a set of pairs, so there can be at most one edge $(v_i, v_j) \in E$ from v_i to v_j. Some definitions of graphs allow more than one edge from one vertex to another, but we do not consider such graphs here.

Application 3.1.4. As mentioned in the introduction, graphs can be used to describe a data structure where each piece of data is stored along with pointers identifying where to look for more of the data. Each node of the graph corresponds to an object in memory (for example, an integer, a string, a list, or even an entire data file), and each edge (v, v') of the graph corresponds to a pointer stored at node v, pointing to the location of node v'.

Definition 3.1.5. *An* undirected graph $G = (V, E)$ *is a pair consisting of a nonempty, finite set V of vertices (or nodes) and a set E of unordered pairs of elements of V corresponding to the edges of a graph. We denote the undirected edge*[16] *from v to v' as $\{v, v'\}$. A graph (whether directed or undirected) is called* simple *if it has no edges connecting a vertex to itself.*

Example 3.1.6. Figure 3.1 also depicts the undirected graph $G_2 = (V_2, E_2)$ with vertices $V_2 = \{1, 2, 3, 4, 5\}$ and edges $E_2 = \{\{1, 2\}, \{1, 3\}, \{3, 4\}, \{3, 5\}\}$.

Remark 3.1.7. As with directed graphs, we allow at most one edge between any two vertices in an undirected graph.

Remark 3.1.8. Every simple undirected graph has an associated directed graph, defined by replacing every unordered edge $\{v, v'\}$ in the undirected graph by the pair of directed edges (v, v') and (v', v). Conversely, if the set E of edges in a simple directed graph is symmetric (that is, $(v_i, v_j) \in E$ if and only if $(v_j, v_i) \in E$), then it naturally defines an associated undirected graph corresponding to replacing every matching pair $(v, v'), (v', v)$ of directed edges with a single unordered edge $\{v, v'\}$.

Definition 3.1.9. *Let $G = (V, E)$ and $G' = (V', E')$ be graphs (either directed or undirected). We say that G' is a* subgraph *of G if $V' \subset V$ and $E' \subset E$.*

Example 3.1.10. Consider the undirected graphs represented in Figure 3.2. Note that G_3 is a subgraph of G_4, G_5, and G_6. Also, G_4 is a subgraph of G_6 with the same vertices as G_6 but fewer edges. However, G_5 is not a subgraph of G_6, despite the fact that its vertices are all in G_6.

[16] We recognize that this notation could be confusing when there is an undirected edge from a vertex to itself, but the meaning should be clear from context. Moreover, we rarely discuss graphs that are not simple.

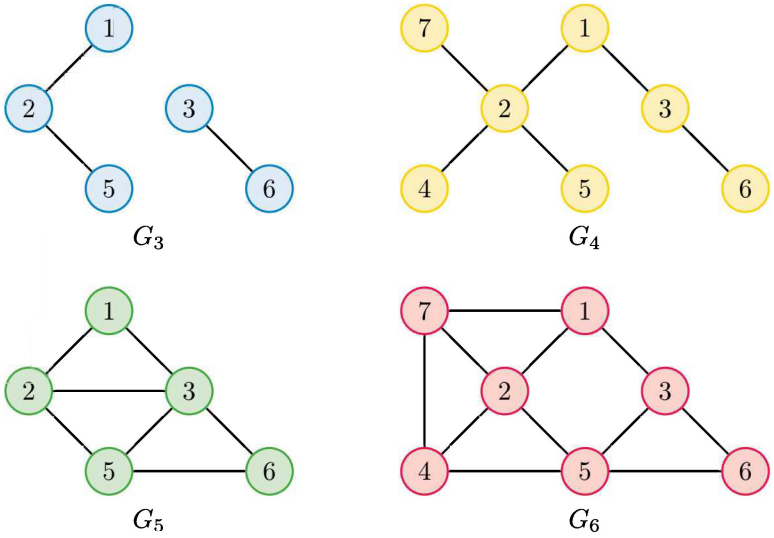


Figure 3.2. *Several graphs and subgraphs, as described in Example 3.1.10.*

Remark 3.1.11. When a definition, example, proposition, etc., does not specify whether a graph is directed or undirected, it is usually because we want to adapt the statement to both cases.

3.1.2 Walks, Paths, Cycles, and Connectedness

Definition 3.1.12. *Let $G = (V, E)$ be a graph. A* walk *of length m is any sequence $(v_{i_0}, v_{i_1}, \ldots, v_{i_m})$ of vertices, where $(v_{i_k}, v_{i_{k+1}}) \in E$ (or $\{v_{i_k}, v_{i_{k+1}}\} \in E$ for an undirected graph) for each $k = 0, 1, \ldots, m-1$. A walk is* closed *if the first and last vertices are the same; otherwise it is* open. *A* path *is an open walk in which no vertex or edge is repeated. A* cycle *is a closed walk in which no vertex or edge is repeated except for the first and last vertices, which are the same.*

Example 3.1.13. In graph G_5 of Figure 3.2, there are four paths from node 2 to node 3, namely $(2, 3)$, $(2, 1, 3)$, $(2, 5, 3)$, and $(2, 5, 6, 3)$.

Example 3.1.14. In Figure 3.2, the walk $(2, 3, 2)$ is not a cycle of G_5 because the edge $\{2, 3\}$ is the same as $\{3, 2\}$, and edges cannot be repeated in a cycle. However, in Figure 3.1, the walk $(1, 2, 1)$ is a cycle in G_1 since in a directed graph the edge $(1, 2)$ is a different edge than $(2, 1)$ and therefore no edge is repeated.

Example 3.1.15. Consider the graphs in Figure 3.3. In graph G_7, there are two paths from node 4 to node 1, namely $(4, 3, 1)$ and $(4, 2, 1)$. Graph G_8 has three different cycles starting (and ending) at 4, namely $(4, 3, 5, 6, 4)$, $(4, 3, 1, 4)$, and $(4, 2, 1, 4)$.

Definition 3.1.16. *A graph $G = (V, E)$, is* connected[17] *if for any two distinct vertices $v_i, v_j \in V$, there exists a path from v_i to v_j. A graph is* disconnected *if it is not connected.*

Remark 3.1.17. A connected, directed graph $G = (V, E)$ with more than one vertex must have a cycle. To see this, note that there must be at least one edge; denote it by $(v, v') \in E$. Since G is connected, there is a path connecting v' to v. Appending the edge (v, v') to the end of that path gives a cycle that starts and ends at v'.

Remark 3.1.18. For undirected graphs, combining the two paths as described in Remark 3.1.17 does not necessarily give a cycle, because some edges might be repeated; for example, the second path could just be the first path traversed backward.

Unexample 3.1.19. The directed graph G_7 in Figure 3.3 has no cycles and hence is not connected. If G_7 were changed to an undirected graph, so that each edge was replaced with an undirected edge, then it would be connected.

Example 3.1.20. The directed graph G_8 in Figure 3.3 is connected. Note that G_8 has several cycles.

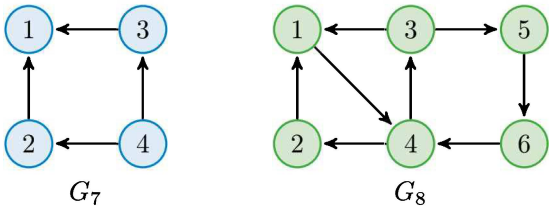


Figure 3.3. *A disconnected directed graph G_7, described in Unexample 3.1.19, and a connected directed graph G_8, with several cycles, as described in Example 3.1.20.*

[17] Many texts call this property *strongly connected* to distinguish it from a lesser property called *weakly connected*. However, we do not consider weakly connected graphs in this text, and thus we just use the term *connected*.

Proposition 3.1.21. *Any connected undirected graph with n vertices must have at least $n - 1$ edges.*

Proof. We prove this by induction on n. It is trivially true if $n = 1$. Now suppose the proposition is true for all $n < N$. Assume by way of contradiction that there exists a connected undirected graph of N vertices with at most $N - 2$ edges. If every vertex has at least two edges attached to it, then $|E| \geq |V| = N$. But since $|E| < N$, we must have at least one vertex v with a single edge (if it had no edges, then the graph would be disconnected). Removing v and its lone edge from the graph will produce a connected subgraph because any path between two vertices other than v could not have passed through v, and thus the path still remains after removing v and its edge. However, the subgraph has $N - 1$ vertices and no more than $N - 3$ edges, which contradicts the induction hypothesis. □

3.1.3 Adjacency Matrices

There are many ways to represent a graph. One important method is to represent a graph as an *adjacency matrix*.

Definition 3.1.22. *Let $G = (V, E)$ be a directed graph with vertices $V = \{v_1, v_2, \ldots, v_n\}$. The* adjacency matrix *is the $n \times n$ matrix $A(G) = [a_{ij}]$, where*

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

For an undirected graph we use the adjacency matrix of the corresponding directed graph (see Remark 3.1.8). As a result, the adjacency matrix of an undirected graph is always symmetric.

Example 3.1.23. The adjacency matrices for the graphs depicted in Figure 3.1 are

$$A(G_1) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \text{and} \quad A(G_2) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Notice that $A(G_2)$ is a symmetric matrix, since G_2 is an undirected graph.

To determine whether a walk of length k from vertex i to vertex j exists, we can look at the (i, j) entry of powers A^k of the adjacency matrix.

Proposition 3.1.24. *Given any $k \in \mathbb{Z}^+$ and any graph G with adjacency matrix A, the (i, j) entry of A^k is the number of walks of length k in G from vertex i to vertex j.*

Proof. This follows by induction on k. The details are Exercise 3.5. □

Example 3.1.25. The adjacency matrix of the graph G_7 from Figure 3.3 and the square of the adjacency matrix are

$$A(G_7) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \text{and} \quad A(G_7)^2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}.$$

By Proposition 3.1.24 the 2 in the lower left corner of $A(G_7)^2$ shows there are two walks of length 2 from node 4 to node 1, and the fact that all other entries in the matrix are zero shows there are no walks of length 2 between any other two nodes in the graph.

Example 3.1.26. Consider the graph G_8 from Figure 3.3. The adjacency matrix and its eighth power are given by

$$A(G_8) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{and} \quad A(G_8)^8 = \begin{bmatrix} 0 & 4 & 4 & 4 & 0 & 1 \\ 2 & 0 & 0 & 4 & 1 & 2 \\ 6 & 1 & 1 & 4 & 3 & 2 \\ 8 & 4 & 4 & 1 & 4 & 0 \\ 2 & 0 & 0 & 4 & 1 & 2 \\ 0 & 4 & 4 & 4 & 0 & 1 \end{bmatrix}.$$

This shows there are six walks of length 8 from node 3 to node 1; that is, $b_{31} = 6$ for $A(G_8)^8 = [b_{ij}]$. Since the diagonals correspond to closed walks, $b_{33} = 1$ means there is exactly one closed walk of length 8 starting and ending at node 3. The details of that walk are not clear from $A(G_8)^8$, but in this case it is easy to verify that $(3, 5, 6, 4, 3, 5, 6, 4, 3)$ is the length-8 walk from 3 to 3.

3.2 Trees and Tree-Based Data Structures

In this section we describe two of the most common and important kinds of graphs, namely *undirected trees* and *directed rooted trees*. Many of the most useful data structures are based on these trees, and most recursive algorithms can be described in terms of trees. As the godfather of algorithms, Donald Knuth, says, "Trees sprout up just about everywhere in computer science."

3.2.1 Undirected Trees

Definition 3.2.1. *A connected, undirected graph without cycles is called a* tree. *An undirected graph without cycles (not necessarily connected) is called a* forest.

Example 3.2.2. The graph G_3 in Figure 3.2 is not connected and has no cycles; that is, it is a forest of two trees. The graph G_4 in the same figure has no cycles and is connected, so it is a single tree.

Unexample 3.2.3. The connected graphs G_5 and G_6 in Figure 3.2 have cycles and are neither forests nor trees.

Example 3.2.4. Although it may seem counterintuitive, the graph consisting of a single vertex and no edges is a tree, since it is a connected, undirected graph without cycles. It is also a forest since every tree is also a forest.

Proposition 3.2.5. *An undirected graph is a tree if and only if for any two distinct vertices there exists exactly one path connecting them.*

Proof. If G is an undirected graph having the property that any two vertices are connected by exactly one path, then the graph is clearly connected. If the graph contained a cycle $(v_0, v_1, v_2, \ldots, v_k, v_0)$, then there would be two paths $(v_0, v_1, v_2, \ldots, v_k)$ and (v_0, v_k) from v_0 to v_k, contradicting the hypothesis of unique paths. Hence G has no cycles and is a tree.

Conversely, if an undirected graph G is a tree, then it is connected, and hence any two distinct vertices have at least one path between them. Suppose G contains two distinct paths connecting the same pair of vertices v and w, say, $P = (v_0, v_1, v_2, \ldots, v_k)$ and $P' = (v'_0, v'_1, v'_2, \ldots, v'_\ell)$, where $v_0 = v = v'_0$ and $v_k = w = v'_\ell$. Let $i \geq 1$ be the smallest index such that $v_i \neq v'_i$. And let n be the smallest index greater than or equal to i such that $v'_n = v_m$ for some m. Since $v_k = w = v'_\ell$, such an n must exist. The closed walk $v_{i-1}, v_i, \ldots, v_m, v'_{n-1}, \ldots, v'_i, v'_{i-1}$ is a cycle, since the only repeated vertex is $v_{i-1} = v'_{i-1}$. The existence of this cycle contradicts the hypothesis that G is a tree. Therefore, there can be only one path between any two vertices. □

Proposition 3.2.6. *Any tree T with more than one vertex has at least two vertices with only one edge each, that is, each of these vertices has only one edge connecting to it.*

Proof. Among all paths in T there must be at least one path P of maximal length. Denote its endpoints by u and v, respectively. The path P has exactly one edge connecting to u and exactly one edge connecting to v. Suppose, by way of contradiction, that either u or v has more than one edge. Adding that additional edge in T to P cannot make a cycle, so it must connect to a new vertex not in P.

Thus, adding the new edge to P makes a path that is longer than P, which is a contradiction. Hence, u and v have exactly one edge each. □

Proposition 3.2.7. *Let $G = (V, E)$ be a connected undirected graph with n vertices. The graph G is a tree if and only if $|E| = n - 1$.*

Proof. Assume that $|E| = n - 1$ and suppose that G has a cycle. Removing one edge from the cycle will yield a subgraph G' that is still connected, with n nodes, but has $n - 2$ edges, which is a contradiction to Proposition 3.1.21. Hence G has no cycles and is therefore a tree.

Conversely, assume G is a tree. We proceed by induction on n, noting that the case $n = 1$ holds trivially. Assume the result holds for all graphs of at most $n - 1$ vertices, and consider one with n vertices. Since G is a tree, it follows from Proposition 3.2.6 that there must be at least one vertex with exactly one edge. Removing that vertex and its lone edge gives a new tree G' with $n - 1$ vertices, which must have $n - 2$ edges. Therefore G must have had $n - 1$ edges. □

3.2.2 Linked Lists, Stacks, and Queues

As mentioned above, using an array with n slots to accommodate a dynamic list of $k \leq n$ objects can be problematic. Adding $\ell > n - k$ new elements to the list, so that the total number of elements in the list exceeds n, requires a new, larger array to be created, and all the elements, old and new, must be copied to the new array. This costs roughly $O(k+\ell) \approx O(n)$ primitive operations. Similarly, removing an object from the list can require up to $O(n)$ primitive operations, depending on where the object is located, due to reshuffling of the remaining objects to the front of the array. When n is large or there are many such lists to manage, the temporal cost of using arrays can become prohibitively expensive. A better alternative in this setting is a data structure that can dynamically and gracefully expand and contract, as needed. One example of such a data structure links blocks together in a chain called a *linked list*.

A linked list is among the most basic of the graph-based data structures. It is a finite, ordered sequence of n nodes, each with an edge pointing to the next node in the sequence. A linked list $[v_1, \ldots, v_n]$ corresponds to the directed graph with $n-1$ edges $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$, as depicted in Figure 3.4(a). The list can be searched by starting with the first node, which is called the *root*,

Figure 3.4. *Depiction of two linked lists. List* (a) *is a singly linked list, where each node is a block of data (white) followed by another block (green) containing the address of its successor. List* (b) *is a doubly linked list, with an address at the end of each block pointing to its successor and another address at the beginning of each block pointing to its predecessor.*

and following the edges until the desired node is found. Since there are at most n nodes to visit, the temporal complexity of searching for an arbitrary node is bounded by $O(n)$.

To add a node to the beginning of a linked list, create the new node and point it to the root of the old list. This can be done in constant time. Compare this with the array, where insertion is $O(n)$. In general, a new node can be added anywhere in a linked list in constant time if the location of the node we wish to have precede it is known: make the preceding node point to the new node that's being inserted, and then have the new node point where the preceding node was originally pointing. Note, however, that in many settings one must search the linked list to find where to insert the new item. Since the complexity of a search is $O(n)$, any insertion requiring a search is also $O(n)$.

Similarly, to remove a node, once the location of the node preceding it is known, redirect the previous node to point where the removed node had been pointing to. So again the operation can be performed in constant time once the predecessor of the desired removal location is known, but finding that predecessor may require a search.

Remark 3.2.8. In some applications, nodes point back to their predecessors. Such linked lists are called *doubly linked*; see Figure 3.4(b).

Linked lists can be used to define other kinds of data structures. *Stacks* and *queues* are special types of lists where modifications to the list are more restricted than a regular list.

Definition 3.2.9. *A* stack *is a list where data may only be inserted and removed from the root (first node) of the list. A* queue *is a list where data may only be inserted at the tail (last node) and only removed from the root.*

Nota Bene 3.2.10. Don't confuse the abstract data structure called *a stack* with the pool of memory in the computer called *the stack*. The latter is an example of the former, but it is certainly not the only example.

Example 3.2.11. To help remember the difference between a stack and a queue, consider a stack of plates in the kitchen cabinet. Plates are taken from the top of the stack, and returned to the top of the stack (which is the root of the stack). This means that the last plate put in will be the first one taken out. Hence a stack is said to satisfy the *last in, first out* (LIFO) principle.

Contrast this with a queue of polite people standing in line. Here the root is the front of the line and the tail is the back of the line. The first one to arrive is the first one served. Hence a queue satisfies the *first in, first out* (FIFO) principle.

Remark 3.2.12. It is common to use the word *push* to describe the operation of placing an element on the stack or queue and the word *pop* to describe the operation of removing an element and returning it to the user.

A stack can be implemented by creating a linked list and adding the restriction that all insertions and deletions must happen at the root. A queue can also be implemented with a linked list, where insertions happen only at the tail (whose location should also be stored separately), and removals happen only at the root. For further instruction on the coding of a linked list, see the computer labs that accompany this volume.

If the maximum size n of the stack is known in advance, then we can also use an array of size n to implement the stack. To do this, simply store a pointer that tracks the location of the root as a counter. Start with an empty stack and push the first node onto the stack by putting it in the first position in the array and setting the root pointer to point to that position. Push each subsequent node onto the stack by appending it to the existing data in the array (the immediate successor of the old root) and resetting the root counter to point to the new root position. Data is popped off the stack by removing the node at the root position and updating the root pointer by subtracting the counter accordingly. This makes both pushing and popping very efficient and avoids having to store all the links that point from one node to the next. But searching for a node with a specific value still has temporal complexity $O(k)$, where $k \leq n$ is the size of the stack.

A queue whose size never exceeds n can also be implemented as an array of size n, but now two counters must be kept—both the root and the tail. New elements are pushed at the point of the root counter (just like stacks) and then the root counter is increased accordingly. Elements are popped off the queue by increasing the tail counter. When either counter gets to $n - 1$ (the last entry in the array, assuming indexing starts at zero), incrementing it starts the counter over at zero. It helps to think of the counters modulo n. Of course, it is important to be careful when the head and the tail are the same, since that could mean the queue is either empty or full.

3.2.3 Directed Rooted Trees

Another type of graph that underlies many useful data structures is the *directed rooted tree*.

Definition 3.2.13. *A* directed rooted tree *is a directed graph with no cycles, having exactly one node (the root) with no incoming edges, and where every other vertex in the graph has a unique path from the root to that vertex. The outgoing neighbors of a given node are called its* children*, and the given node is called the* parent *of those children. A node with no children is called a* leaf node*.*

When talking about data structures and algorithms, it is common to refer to a *directed rooted tree* simply as a *tree*. When there is little chance of confusion, we also use this terminology. An example of a directed rooted tree is shown in Figure 3.5.

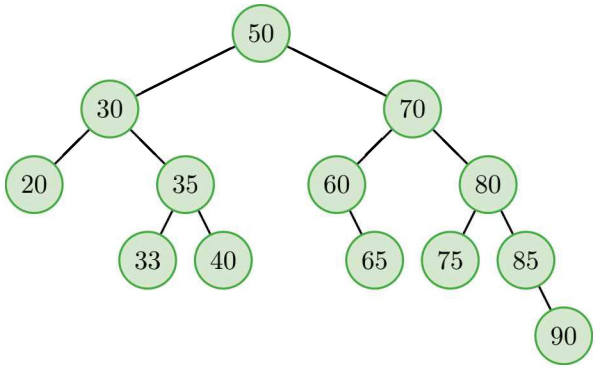


Figure 3.5. *An example of a binary directed rooted tree. It is traditional to draw directed rooted trees upside down, that is, with the root at the top and the leaves below. With this convention, it is assumed that the direction of the edge goes from top to bottom and we don't need to draw arrows. This particular tree is also a binary search tree (see Section 3.3.1).*

Data structures arising from trees give a generalization of linked lists, where the nodes of a tree can link to multiple children, instead of just one. A linked list corresponds to a tree in which every node has at most one child. We say that a tree is *binary* if every parent has at most two children.

Some of the most common tree-based data structures are *search trees*, which are organized in a way to facilitate rapid searching. Binary search trees are particularly important. We describe these in the next section.

A natural way to implement any data structure based on a tree is like a linked list: store each node separately, but have each node also store pointers to each of its children. This makes it easy to add and remove new nodes, provided the parent is known.

3.3 Search Trees

Search trees are special tree-based data structures that are designed to facilitate rapid searching. In a search tree, each node has a value, called a *key*, that uniquely identifies the node and is the basis upon which the tree is organized and searched. For example, Figure 3.5 depicts a tree with integer-valued keys. Of course, the keys need not be integers—they can be any objects that are ordered (for example, strings, ordered lexicographically).

In addition to the key, the node can contain other data relevant to the node, but for the purposes of searching and sorting the data, the key is all that matters. For example, suppose each node represents a different student's data, such as their date of birth, address, student identification number, etc. Using the (unique) student identification number as the key would allow for rapid search for a given student's data, provided we know their identification number. The other information could be accessed once the node is found, but it wouldn't be relevant for purposes of the search or for building the tree.

3.3.1 Binary Search Trees

A *binary search tree* (BST) is a tree-based data structure that allows finding any key in the tree in $O(\log n)$ time (assuming the tree is balanced; see Definition 3.3.6). A BST has a maximum of two children per node and no duplicate keys, and the nodes are organized so that the subtree to the left of each child contains only keys that are less than the parent node, whereas the subtree to the right contains only keys that are greater than the parent. See Figure 3.5 for an example of a BST.

One of the main benefits of a BST is that it allows for more rapid searching than a linked list. However, this comes at a cost for insertion and deletion, as shown below.

To find a certain node in a BST, start by comparing the target to the key at the root. If the target is greater than the root key, move to the child on the right; if it's equal to the root, stop; otherwise move to the child on the left. Repeating this process eventually reaches the target value, if it is in the BST. If the target value is not in the BST, this process arrives at a leaf, at which point the search terminates and reports an unsuccessful search.

Example 3.3.1. The tree in Figure 3.5 is a BST. To find the target value 75, compare with the root 50. Since $75 > 50$, move to node 70 on the right. Since $75 > 70$ move right again to 80. Since $75 < 80$, move left to the desired node.

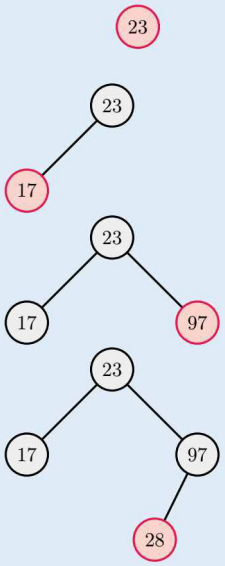
Example 3.3.2. One spatially efficient way of implementing a BST with n elements is to simply sort the keys (temporal complexity of $O(n \log n)$ with `mergesort` of Algorithm 1.11) and store the nodes, in order, in an array. To find a given key in the sorted array, do a binary search (see Algorithm 1.9). Specifically, compare the key in the middle (position $\lfloor \frac{n}{2} \rfloor$) of the array with the target value. If the target is less than the middle key, then compare with the key in the center of the left half (the key that is in position $\lfloor \frac{n}{4} \rfloor$). Similarly, if the target is greater than the middle key, compare now to the key at position $\lfloor \frac{3n}{4} \rfloor$; continue in a similar fashion until the desired key is found. Any such search will terminate in at most $\log_2 n$ steps.

Using a sorted array in this way is an implementation of a BST. The root is in position $\lfloor \frac{n}{2} \rfloor$, the left child of the root is in position $\lfloor \frac{n}{4} \rfloor$, the right child is in position $\lfloor \frac{3n}{4} \rfloor$, and so forth. This is a space-efficient implementation, since pointers need not be stored. However, adding a new key to the sorted list requires finding the correct insertion point and shifting everything that is greater than the new element one place to the right (temporal complexity of $O(n)$). And, again, if the original array is not large enough to hold all the inserted elements, the entire array must be copied to a larger array.

To add a node to such a BST, simply move down the tree (at each stage move to the right if the new key is greater than the current node, and otherwise move to

the left) until no further movement is possible. At this point we are at a node with one or no children. This will be the new parent, and a new child is spawned below it on the appropriate side. An insertion in a BST always creates a new leaf.

Example 3.3.3. In this example, we build a BST containing the keys 23, 17, 97, and 28 by adding the keys consecutively to the tree.



Start with an empty tree, and add the root node: 23.

Now add 17. The node for 17 becomes the left child of 23 because $17 < 23$.

Now add 97. The node for 97 becomes the right child of 23 because $97 > 23$.

Finally add 28. Starting at the root, move to the right because $28 > 23$. Since $28 < 97$, the new node is placed as the left child of 97.

To delete a node from a BST, there are three different cases to consider. First, if the node is a leaf, delete it. Second, if the node is a parent with only one child, replace it with its child. Finally, it could happen that the node is a parent with two children. In this case first find its *in-order predecessor node*, which is the rightmost child of the left subtree (or alternatively one can use the in-order successor node). This predecessor will have at most one child. Swap the node to be deleted with its in-order predecessor, and now the node to be deleted has at most one child, so it can be deleted, as in the previous two cases. Examples of these three cases are shown in Figure 3.6. For further instruction on the coding of a BST, see the computer labs that accompany this volume.

3.3.2 Balance

The order in which numbers arrive when inserting and deleting nodes in a BST affects the shape, or *balance*, of the tree, and that affects the efficiency of searching.

Definition 3.3.4. *The* height *of a node in a directed rooted tree is the number of edges in the longest path from the node to a leaf. The height of a directed rooted tree is the height of the root.*

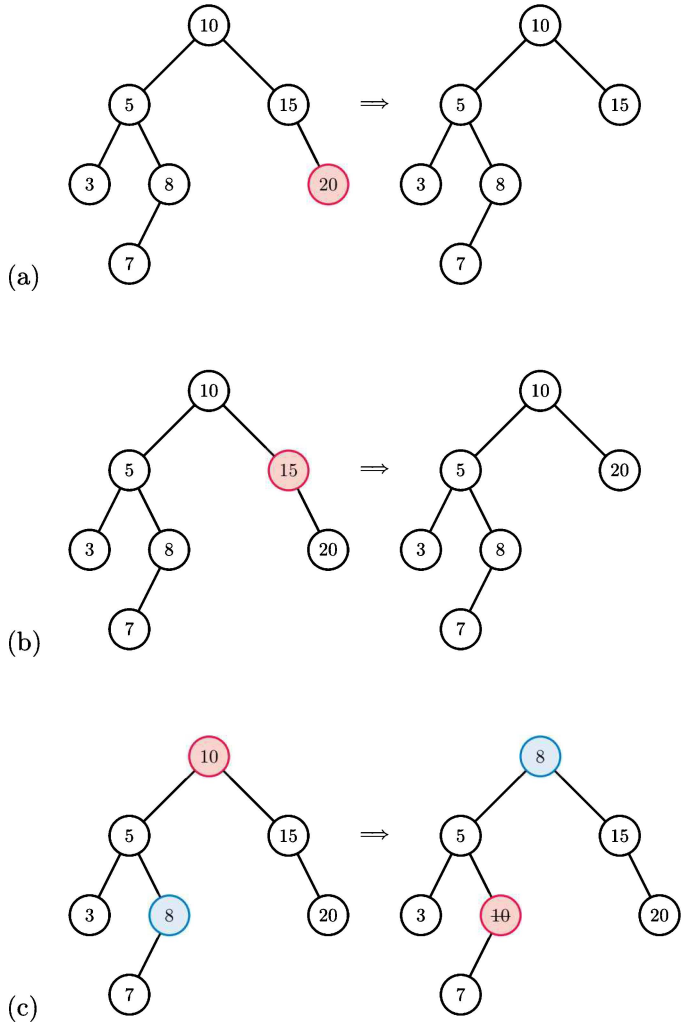


Figure 3.6. *The three cases for deleting a node (red) in a BST. In the first case* (a)*, the node to delete (*20*) has no children. In the second case* (b)*, the node to delete (*15*) has exactly one child. In the last case* (c) *the node to delete (*10*) has two children. In this last case, swap the node to delete with its immediate predecessor (*8*, blue). The node to delete now has at most one child, so it falls into one of the other, easier, cases and can be removed.*

Example 3.3.5. The height of the tree in Figure 3.5 is 4, corresponding to the path $50 < 70 < 80 < 85 < 90$.

Definition 3.3.6. *The* balance *of a given node in a BST is the height of the left child minus the height of the right child. A tree is* balanced *if every node has balance* -1, 0, *or* 1.

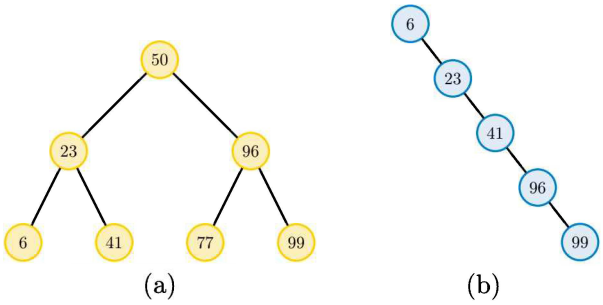


Figure 3.7. *Depiction of* (a) *a well-balanced BST of height* 2 *and* (b) *a pathologically unbalanced BST of height* 4*, which is really just a linked list.*

Example 3.3.7. The BST constructed by successively adding the sequence $50, 23, 6, 41, 96, 77, 99$ is shown in Figure 3.7(a). Every node in this example has balance 0, because for any node in the tree, its left subtree has the same height as its right subtree. In other words, the graph is perfectly balanced.

At the other extreme, constructing a BST by successively adding the (already sorted) sequence 6, 23, 41, 96, 99 gives the graph in Figure 3.7(b). This is so unbalanced that it is a linked list. The balance of the root node is -4 since the subtree to the left has height zero and the subtree to the right has height 4.

Remark 3.3.8. A perfectly balanced binary tree (one whose balance at each node is 0) can exist only if the number of nodes is exactly $2^k - 1$ for some $k \in \mathbb{Z}^+$. If the number of nodes is anything other than this, then at least one node must have a nonzero balance; that is why the definition of balanced trees allows nodes to have balance 1 or -1 as well.

The temporal complexity of searching a BST is determined by the height of the root. If the root has height h, the search will take up to $h + 1$ steps. In the best case every node is perfectly balanced, with a total of $n = 2^{h+1} - 1$ nodes, so the best possible temporal complexity is $O(h) = O(\log n)$. The closer the BST is to being perfectly balanced, the faster the searches will be. The worst case is a BST that is a linked list with n nodes and n levels. In this case the temporal complexity of searching the BST is $O(n)$.

Since additions and deletions consist of a search plus a bounded number of operations, their temporal complexity is likewise bounded between the best case $O(\log n)$ and the worst case $O(n)$. In the next subsection, we examine a method of rebalancing a BST so that the search complexity (and hence also insertions and deletions) is always $O(\log n)$.

3.3.3 AVL Trees

A balanced BST is called an *AVL tree*, named for two Russian mathematicians, Adelson-Velsky and Landis, who first described them and the AVL balancing al-

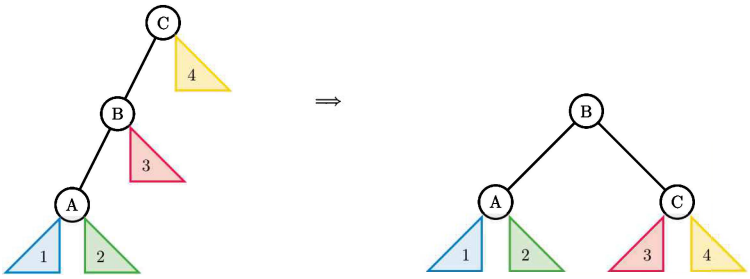


Figure 3.8. *An AVL right rotation on B to correct a left-left imbalance. Here each circle indicates a single node, and each triangle indicates a subtree (possibly empty). A left-left imbalance means that the balance of C is 2 and the balance of B is not negative. All other subtrees are assumed to satisfy the AVL condition. To perform the rotation, the node C is rotated clockwise around B, and the right subtree 3 below B becomes the left subtree of C. This rotation reduces the balance of C so that it and all other nodes and subtrees in this diagram satisfy the AVL condition.*

gorithm in 1962. Their algorithm allows us to rebalance the tree after adding or removing a node.

In an AVL tree, deleting a node or inserting a new node could throw off the balance, making some nodes in the new tree have balance 2 or -2. Whenever this occurs the tree must be rebalanced by rearranging the subtrees using an operation called *rotation*. When a rebalancing is necessary, begin at the lowest level and first rebalance the lowest subtrees that do not meet the AVL criterion. Then work up one level at a time, rebalancing any unbalanced subtrees as follows:

There are four cases to consider.

(i) If a node C has balance at least 2, then the left subtree is deeper than the right. Let B be the left child of C. If the left subtree of B is deeper than (or the same depth as) the right subtree, we call this a *left-left imbalance.* In this case perform a *right rotation* on B, as in Figure 3.8. This operation takes the node C and its left child B, moves the node B up to where C was, makes C the right child of B, and makes B's old right subtree into C's new left subtree. Note that the ordering of the subtrees from left to right has not changed, so the resulting tree is still a BST. Since we are working upward from the bottom, we may assume the subtrees below C are all AVL trees. Using this assumption, it is straightforward to check that in the new tree the nodes B and C satisfy the AVL condition and the balance of all the other subtrees has not changed; see Exercise 3.16.

(ii) If the node C has balance at least 2, but instead of a left-left imbalance the left child of C has its left subtree shallower than the right (so its balance is negative), then we call this a *left-right imbalance.* Denote the left child of C by A and the right child of A by B, as depicted in the leftmost BST of Figure 3.9. To correct the imbalance, first perform a left rotation on B (first arrow of Figure 3.9) and then a right rotation on B (second arrow of Figure 3.9). For the left rotation, the node A is rotated counterclockwise

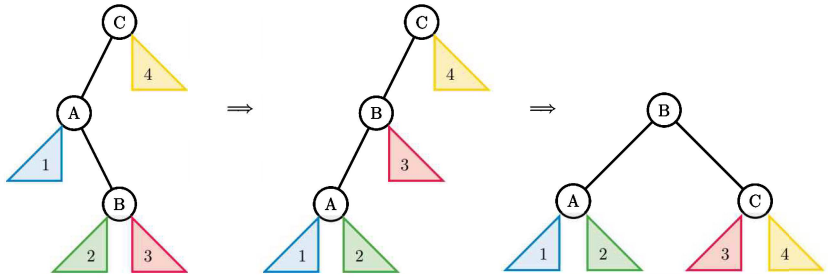


Figure 3.9. *A left-right imbalance (in the tree on the left) is corrected by first performing a left rotation on B (first arrow) and then a right rotation on B (second arrow). For the left rotation, the node A is rotated counterclockwise around B, and the old left subtree 2 below B becomes the new right subtree of A in the center BST. For the right rotation on B, the node C is rotated clockwise around B and the right subtree 3 of B becomes the new left subtree of C in the final BST.*

around B, and the old left subtree 2 of B becomes the new right subtree of A (in the center BST). For the right rotation on B, the node C is rotated clockwise around B and the right subtree 3 of B becomes the new left subtree of C (in the right BST).

Again, ordering of the subtrees from left to right has not changed, so the resulting tree is still a BST. And since we are working upward from the bottom, we may assume the subtrees below C are all AVL trees. The intermediate step of this rebalancing does not meet the AVL condition, but it is straightforward to check that in the final BST the nodes A, B, and C all satisfy the AVL condition and the balance of all the other subtrees has not changed; see Exercise 3.17.

(iii) A right-right imbalance is the mirror of the left-left case. The node C has balance at most -2, and its right child B has nonpositive balance. This is rebalanced by a left rotation on B.

(iv) A right-left imbalance is the mirror of the left-right case. The node C has balance -2, and its right child A has positive balance. This is rebalanced by a right rotation on the left child B of A and then a left rotation on B.

Definition 3.3.9. *An* AVL tree *is a balanced BST (every node has balance* 1, 0, *or* -1*).*

After a node is added or deleted, the tree is rebalanced, if needed, as described above. This guarantees that the final BST constructed in this manner is close to being balanced.

Theorem 3.3.10. *The minimal number $m(h)$ of nodes possible in an AVL tree of height h satisfies*

$$m(h) = 1 + m(h-1) + m(h-2) > 2^{h/2},$$

and the maximal height of an AVL tree with n nodes is bounded by $2\log_2 n$.

Proof. It is straightforward to see that the minimal number of nodes in an AVL tree of height 0 is $m(0) = 1$, and the minimal number of nodes in an AVL tree of height 1 is 2. Given an AVL tree of height $h-1$ with $m(h-1)$ nodes and another AVL tree of height $h-2$ with $m(h-2)$ nodes, we can construct a new AVL tree of height h by adding one new vertex R as the root and making the root of each tree into a child of R. This shows that $m(h) \leq 1 + m(h-1) + m(h-2)$.

Conversely, given any AVL tree of height h and $m(h)$ nodes, removing the root produces two subtrees. One of these must have height $h-1$ and at least $m(h-1)$ nodes. The other must have height at least $h-2$ by the AVL criterion and thus at least $m(h-2)$ nodes. This shows that $m(h) \geq 1 + m(h-1) + m(h-2)$, and hence equality holds.

If $h \geq 2$, then $m(h-1)$ and $m(h-2)$ are both positive, and hence $m(h) = 1 + m(h-1) + m(h-2) > m(h-1)$. Thus if $h \geq 3$ we have $m(h-1) > m(h-2)$, which gives

$$m(h) = 1 + m(h-1) + m(h-2) > 2m(h-2) \geq 2^{h/2}.$$

Since $n \geq m(h)$ we have $n > 2^{h/2}$ and $2\log_2 n > h$. □

Since an AVL tree has height at most $2\log_2 n$, searching it has a worst-case temporal complexity of $O(\log n)$. Inserting a new node or deleting a node are done the same way as with any BST (which requires a search), but then the result may no longer be balanced, so one must also rebalance. Each rotation has a constant time complexity, but to make the tree fully balanced may require $O(\log n)$ rotations (proceeding up the tree to the root from the point of the insertion or deletion); therefore the temporal complexity of insertion or deletion, even if the correct location is already known, is also $O(\log n)$.

The complexity of constructing an AVL tree, by inserting elements one at a time, involves n insertions, costing at most $O(\log n)$ each, so the total cost of constructing the tree is $O(n \log n)$. For further instruction on the coding of an AVL tree, see the computer labs that accompany this volume.

3.4 Priority Queues and Heaps

An important problem in computing is to find the minimal element in an unordered collection (or list) of data.[18] This is particularly challenging in a dynamic situation, where data are continually being added to and removed from the collection. In this section, we frame this problem as a *priority queue* and show how to solve it efficiently with what is called a *heap*.

3.4.1 Priority Queues

A *priority queue* is a data structure where the keys have an order, and the node with the lowest key value is removed and returned (popped) from the queue first. Instead of a LIFO or FIFO rule for returning elements, as discussed in Example 3.2.11, a

[18] Since the algorithms for deciding which element is maximal are nearly identical to those for identifying which is minimal (invert all the inequalities), all the arguments can be easily translated to that of finding the maximal element, but for simplicity we just consider the minimal case.

priority queue returns the node with the minimal key, regardless of when it was added to the queue. In other words, first "priority" is given to the node with the smallest key value. The combined operation of identifying, removing, and returning the node with the minimal key is called `pop_min` or just `pop`. We call the operation of adding a node `insert` or `put`.

Naïve Implementations

A naïve way to implement a priority queue would be to save the data in a linked list, and then with each request for the minimal element, simply search the entire list for the minimum. This implementation has a cost of only $O(1)$ for each insertion, so it has a maximum total temporal cost of $O(n)$ for building a priority queue with n elements. Unfortunately, it also has a cost of $O(n)$ for `pop_min`.

Alternatively, we could implement a priority queue as a balanced BST (for example, as an AVL tree). The minimal element is easily found by moving down the tree to the leftmost leaf. In this case each `pop_min` costs only $O(\log n)$. But each insertion also costs $O(\log n)$, so constructing the tree by successively inserting n unordered elements has a temporal complexity of $O(n \log n)$.

The standard implementation of a priority queue uses a different type of balanced tree called a *heap*, which we discuss in the next subsection. The advantage of a heap is that it can be constructed in $O(n)$ operations and yet `pop_min` still costs only $O(\log n)$ operations.

Applications

Priority queues are very useful for handling sorting and optimization problems, like the problem of finding the shortest path between two locations. This and other examples are found in Chapter 4.

A priority queue can be used to construct a sorting algorithm as follows: Put all the data into a priority queue, and then pop each key back off the priority queue. The resulting sequence of elements will be sorted least to greatest. The temporal complexity of this sorting algorithm on a data set of n elements is precisely the cost of creating the priority queue and then the cost of removing (popping) all n elements.

Several well-known sorting algorithms are constructed in this way, including the *selection sort*, which uses the naïve implementation of a priority queue as an unordered linked list (or an array); *tree sort*, which uses a self-balancing tree as the implementation of the priority queue; and *heap sort*, which uses a heap as the implementation of the priority queue. Among the priority-queue-based sorting algorithms, heap sort is generally preferred.

3.4.2 Heaps

A *heap*[19] is a special type of tree providing an efficient implementation of a priority queue. A heap is not a BST, but, like a BST, its temporal complexity for insertion and deletion is $O(\log n)$; yet the temporal complexity of constructing a heap from an

[19]This should not be confused with the pool of memory in your computer called *the heap*. While the stack *is* an example of a stack, the heap is *not* usually structured as a heap.

unordered array of n elements is only $O(n)$. Moreover, the construction can happen in place, which means that we need only a small amount of additional memory—the initial array plus $O(1)$—for the construction.

Definition 3.4.1. *A* binary heap *is a binary tree satisfying the following three properties:*

(i) *Every level is full except, possibly, the lowest level.*

(ii) *Each parent is less than or equal to its children.*

(iii) *All leaves are located as far left as possible.*

Remark 3.4.2. The minimal element of a heap is easy to find because it is always the root.

Unexample 3.4.3. The three trees in Figure 3.10 are almost, but not quite, heaps. The left tree fails (i). The center tree satisfies (i) and (iii) but fails (ii). The right tree satisfies (i) and (ii) but fails (iii).

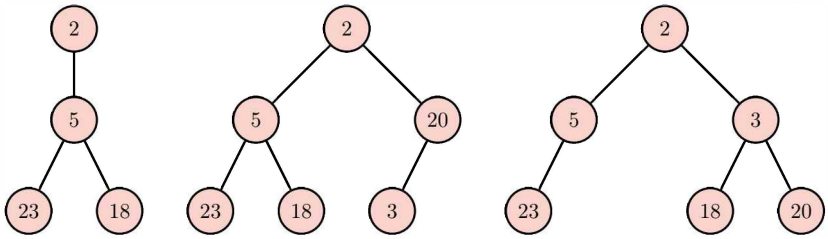


Figure 3.10. *Some trees that are almost, but not quite, heaps. The left tree fails condition* (i). *The center tree fails condition* (ii). *The right tree fails condition* (iii).

A binary heap is often implemented as an array by storing the root (the minimal element) in the first position of the array (position 0), its children in the next two positions (positions 1 and 2), its grandchildren in the next four positions, and so forth, so that the node in position k has its children in positions $2k+1$ and $2k+2$. This avoids storing pointers for each node and instead allows traversal of the tree with simple arithmetic computations; see Figure 3.11.

When using the array implementation of a heap, every new added node is placed in the leftmost empty position. However, this new node is not necessarily greater than its parent and so Definition 3.4.1(ii) might not be satisfied. To remedy this, we must *sift up*; that is, the new node swaps places with its parent whenever it is less than its parent. Repeat this process until either the new node is greater than its parent or it becomes the new root. In either case, the new tree is now a heap and we have the following result.

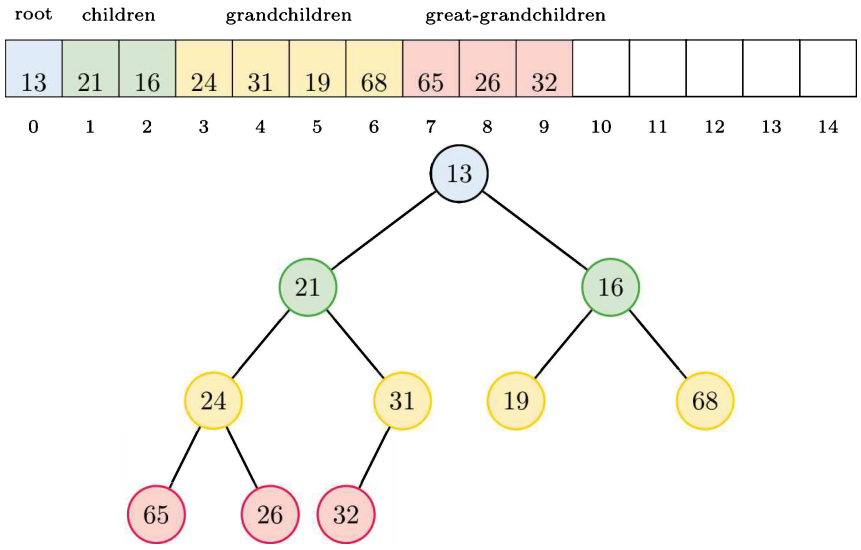


Figure 3.11. *An array implementation of a heap. The root (blue) is in the* 0*th position, its two children (green) are in positions* 1 *and* 2*, and, in general, the children of the node in the* k*th position are in positions* $2k+1$ *and* $2k+2$*.*

Proposition 3.4.4. *If a heap has one new leaf added that does not satisfy the condition that the leaf is greater than its parent, then sifting that leaf up until it is greater than its parent will produce a heap.*

Proof. The proof is Exercise 3.22. □

Remark 3.4.5. The process of inserting a new entry to a heap (that is, adding it to the end of the heap and then sifting up until the new tree is a heap) has a temporal complexity of $O(\log n)$ since there are at most $\log_2 n$ ancestors to swap with while sifting up.

To remove an element from a heap, simply replace the key to delete with the key in the rightmost leaf of the heap. If that rightmost key is greater than one of its children, *sift down*, by trading places with its smallest child and repeating as necessary until it satisfies the ordering property in Definition 3.4.1(ii); see Figure 3.12 for an illustration.

Proposition 3.4.6. *If the key in one node of a heap is replaced with a different key that is greater than one (or both) of its children, then sifting that node down (trading places with its smallest child) until it is less than all of its children will produce a heap.*

Proof. The proof is Exercise 3.23. □

Again, since there are at most $\lceil \log_2 n \rceil$ levels, deletion requires at most $O(\log n)$ steps (down- or up-sifts). Also, updating the priority of a given node and then

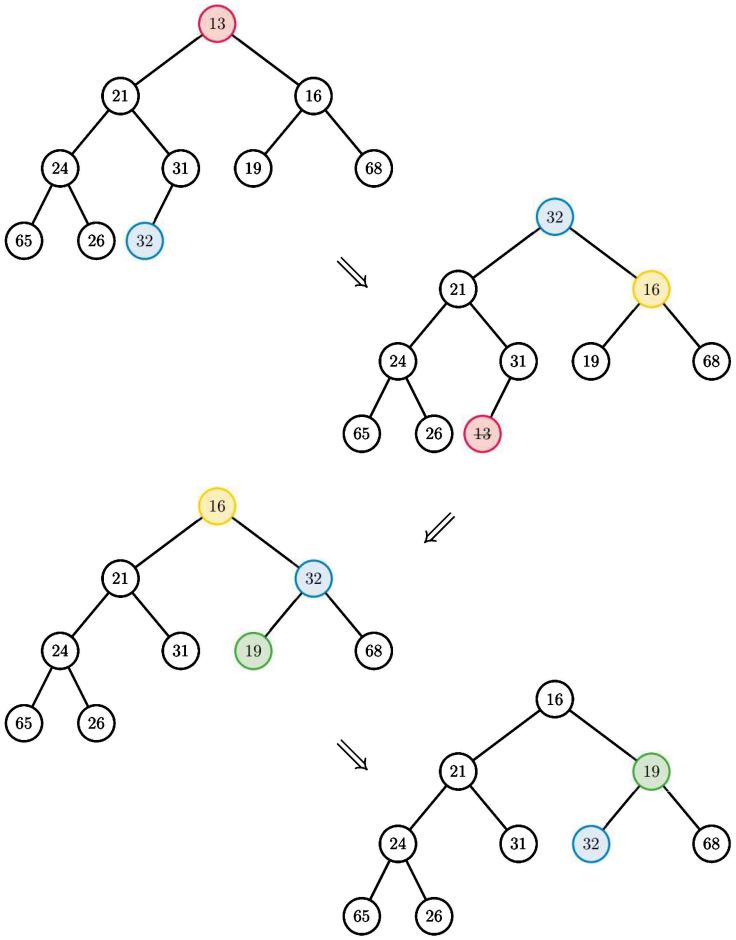


Figure 3.12. *Removing a key from a heap. To remove the key* 13 *(red) from the heap at the top of the figure, trade places with the bottom rightmost key* 32 *(blue), and then remove* 13*. The result is no longer a heap, because the root* 32 *is larger than a child (actually, it is larger than both children). To fix this, sift down by trading places with the smallest child, in this case* 16 *(yellow). The result is still not a heap because* 32 *is still larger than one of its children,* 19 *(green). Sift down again, by trading places with* 19*. The process is complete because there are now no children of* 32 *that are smaller than it, thus the last tree satisfies the heap-ordering property of Definition 3.4.1*(ii).

sifting in the appropriate way will require at most $\log_2 n$ up- or down-sifts to make the result into a heap again.

3.4.3 Constructing a Heap

Since adding a new node to an existing heap of n nodes costs $O(\log n)$ operations, you might think that creating a new heap from an unordered array of n elements by

adding the nodes in succession would take $O(n \log n)$ operations. In this subsection, we show how to grow a heap in $O(n)$ time through a process called *heapifying* the array, which goes as follows.

Start with all n elements in an array. Treating the array as a tree with the root in position 0 and the children of node k in positions $2k+1$ and $2k+2$ means that the tree automatically satisfies (i) (all but the bottom level is full) and (iii) (all the nodes are as far left as possible) of Definition 3.4.1. Therefore, the only property that is not necessarily satisfied is the heap-ordering property, (ii). The main tool in the heapifying process is sifting down, and the success of the process relies on Proposition 3.4.6, which guarantees that if all the elements of a subtree except the root satisfy the heap-ordering condition (that is, the root may be larger than one of its children, but all other nodes in the subtree are less than their children), then sifting the root down until it satisfies the ordering condition ensures that the entire subtree satisfies the ordering condition.

The strategy is to start at the bottom and work upward, sifting each node down until it satisfies the heap-ordering property. The leaf nodes are those in the range from $\lceil \frac{n}{2} \rceil$ to the end, and all leaves vacuously satisfy the property that they are less than their children (since they have no children). Starting at the next level, the rightmost node that has a child is in position $\lceil \frac{n}{2} \rceil - 1$. Running through the nodes in positions $\lceil \frac{n}{2} \rceil - 1$ down to 0, sift each node down, as necessary. Once each node has been sifted, then everything below that point is a heap.

Proposition 3.4.7. *The process described above of building a heap from an unordered array with n elements has temporal complexity $O(n)$.*

Proof. The temporal complexity $T(n)$ is determined by the total number of downsifts. The bottom layer has no more than 2^{k-1} leaves, where $k = \lceil \log_2 n \rceil$, and these leaves need no sifting down. The next layer has exactly 2^{k-2} nodes, and they need to be sifted down at most once. The next layer of 2^{k-3} nodes need to be sifted down at most twice, and so forth. Therefore, we have

$$T(n) \leq \sum_{\ell=0}^{k-1} \ell 2^{k-\ell-1} < 2^{k-1} \sum_{\ell=0}^{\infty} \ell \left(\frac{1}{2}\right)^{\ell} \leq n \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2n,$$

where the penultimate inequality follows from Exercise 1.20 and from the fact that $2^{k-1} \leq 2^{\log_2 n} = n$. □

Remark 3.4.8. The previous description and proposition show that a heap can be built in place, using the original array for memory with only a few additional temporary variables for the sifting process. Moreover, the total number of sift-down operations required for the build is less than n, so the build process is very efficient. For these reasons this array-based binary heap is usually the preferred implementation for a priority queue. Indeed, many people use the terms *heap* and *priority queue* interchangeably, although this is not technically correct, since, as we have seen, there are many other (less efficient) ways to implement a priority queue.

Remark 3.4.9. Since a heap can be built in $O(n)$ time, it should not be surprising that, on average, a heap insertion (with all necessary up-sifting) has complexity only $O(1)$—intuitively, most of the elements of the heap belong near the bottom, since the bottom has exponentially more elements than the top. Thus, most elements inserted at the bottom need very few up-sifts after insertion to satisfy the heap ordering.

Remark 3.4.10. In some settings it is useful for a priority queue to have an additional operation that allows us to update the priority of an existing node. In the heap implementation of a priority queue, this is easy to do if the location of the node is known. In that case, simply delete the node (and heapify) and then insert the same data back into the heap with the new priority (and heapify). The problem, of course, is that we generally do not know the location of the node we want to update, and a search for a general node in a heap costs $O(n)$ because every branch of the tree must be searched.

One possible approach to this problem is to simply duplicate the data. That is, make a new node with the same data but with a new key matching the new priority. This can work in situations where the additional memory cost is not a problem, provided data with old priorities are not a threat to the success of the application. If actually updating the priorities, rather than duplicating data, is important, then a BST may be better suited than a heap, since finding an arbitrary key in a BST can be done in $O(\log(n))$ time.

3.5 *B-Trees

In Section 3.3.3, we showed that AVL trees were of logarithmic temporal complexity for searches, inserts, and deletes. In this section we introduce the B-tree, which is a more general self-balancing tree that also allows searches, inserts, and deletes in logarithmic time, but each node can support several children and thus store large blocks of data contiguously. This allows the tree to operate more efficiently with some types of large data sets; for example, B-trees are used heavily to store data in both relational databases and many file systems.

Definition 3.5.1. *Fix two positive integers m and h, and let $k = 2m$. A* balanced B-tree *of order k and height h is a tree where each node may contain multiple keys, where edges from a parent node to its children are separated by the keys of the parent, and where edges are arranged so that all the keys in a child lie between the keys of the parent that separate that edge from the other edges. Moreover, the following must hold:*

(i) *The distance from the root to every leaf is h.*

(ii) *Each node contains no more than k keys and no more than $k + 1$ children.*

(iii) *Every node except the root has at least m keys.*

(iv) *The root may never have only one child—unless the root is also a leaf, it must have at least two children.*

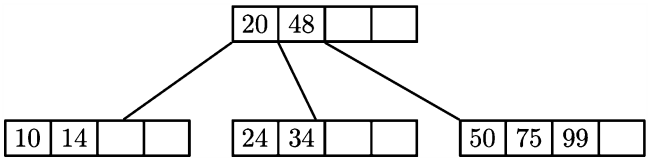


Figure 3.13. *A B-tree of order $k = 4$ and height $h = 1$, as described in Example 3.5.2.*

Example 3.5.2. An example of a B-tree is given in Figure 3.13. Note that all the keys (10 and 14) in the leftmost leaf are less than the first key (20) of the root, all the keys (24 and 34) of the middle leaf lie between the first two keys (20 and 48) of the root, and all the keys of the rightmost leaf (50, 75, and 99) are greater than the last key (48) of the root. Also, every node including the root has at least $m = 2$ keys. It is allowable for the root to have fewer than m keys, but every other node must have at least m.

Remark 3.5.3. As with other data structures, we may associate a large amount of data to each key, but each datum is only identified and ordered by its key, so we only talk about the keys themselves, as if they were the data.

Proposition 3.5.4. *The number of keys that a B-tree can hold grows exponentially in h. More precisely, a B-tree of order k and height h can store up to $k(k+1)^h$ keys.*

Proof. The proof is Exercise 3.24. □

Example 3.5.5. A B-tree of order $k = 99$ and height $h = 2$ can store $99(100)^2 = 990{,}000$ keys. Adding a new level ($h = 3$) increases the total capacity a hundredfold to $(99)(100)^3 = 99{,}000{,}000$ keys.

A B-tree has spatial complexity $O(n)$, where n is the total number of keys. Searching a B-tree of order k and height h has worst-case temporal complexity $O(k(h+1))$, so if n is the total number of keys, and the order k is fixed, then searching has temporal complexity $O(\log(n))$.

To insert a new key, first search for the appropriate leaf node to insert into. If the leaf is not full, insert the key into the correct place in the leaf (this will require rearranging the keys in the leaf, but will not require changing the rest of the tree). If the leaf is full, temporarily add the new key to the list of keys in that leaf, find the median of that list, and promote it to the parent node. All keys greater than the median key belong to the new right leaf and all keys smaller than the median key belong to the new left leaf; see Figure 3.14 for an example of this procedure.

If the parent node is full, split it in a similar way—by finding the median and promoting it—and continue this process until you reach the root of the tree. If the

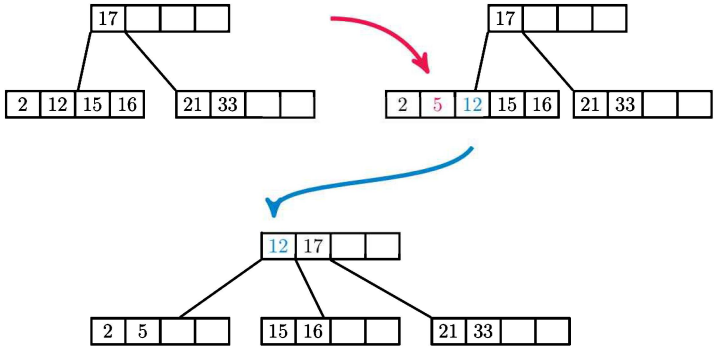


Figure 3.14. *Inserting the key 5 into a full leaf makes the leaf overfull. To rebalance, the median 12 of the overfull leaf is moved into the parent node and the overfull leaf is split.*

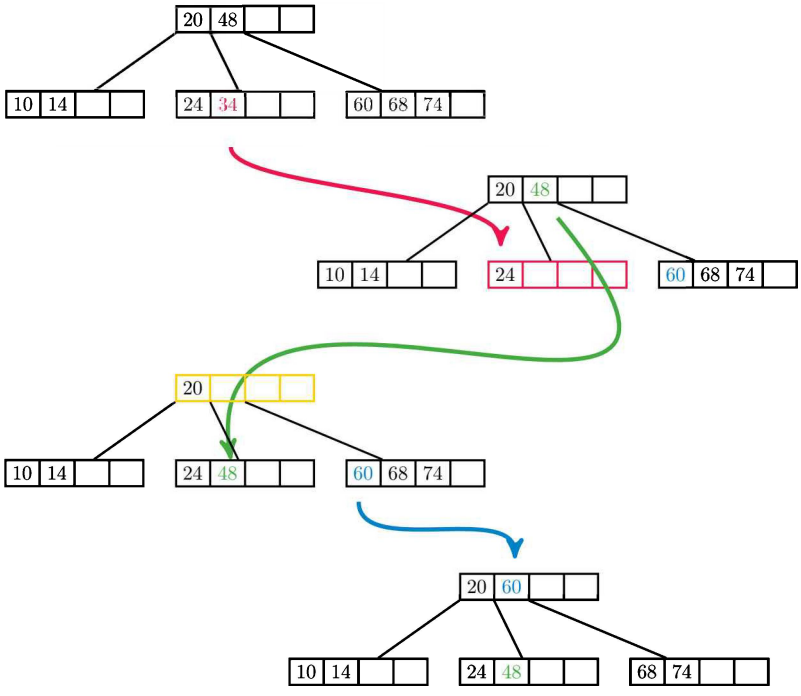


Figure 3.15. *Deleting from a B-tree leaf with only m keys, and taking from a sibling: Deleting the key 34 leaves the middle leaf (red) underfull. Since the right sibling can spare a key, move the right separator in the parent (48) down to the middle leaf. This leaves the parent (yellow) underfull. This can be repaired by moving the immediate successor 60 of the moved key (48) from the right sibling up to the parent.*

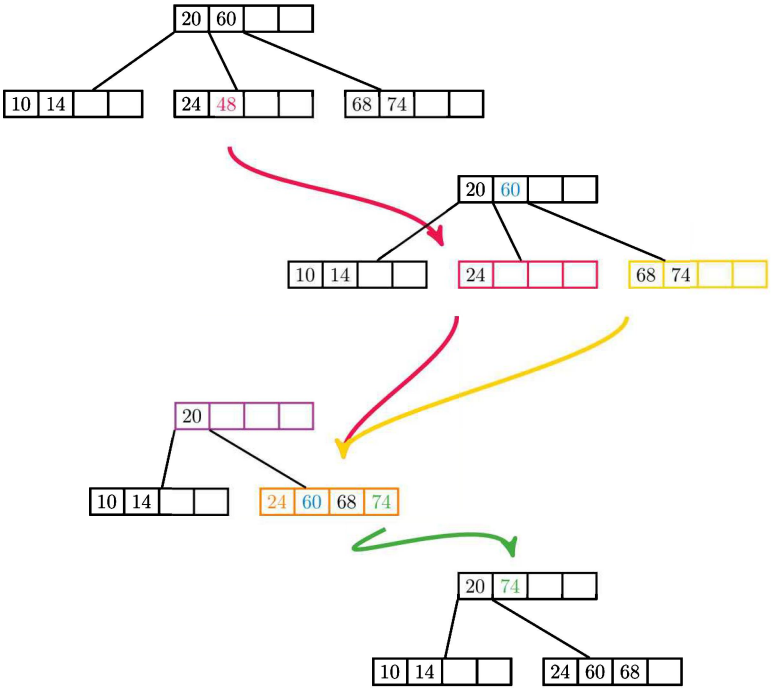


Figure 3.16. *Deleting from a B-tree leaf with m keys, but siblings can't spare keys: Deleting the key 48 leaves the middle leaf (red) underfull. Since neither sibling can spare a key, merge with the right sibling (yellow) and move the right separator 60 in the parent down to the merged leaf (orange). This leaves the parent (purple) underfull. If the parent is the root, this is not a problem, but if it is not the root, it will have to be rebalanced by taking from (or merging with) a sibling. In this particular case it can also be rebalanced by moving the largest key (74) from the newly merged leaf to the parent.*

root is full, split it and create a new root (increasing the overall height by 1), with the median as the only key in the new root. The number of times we must perform this splitting procedure for a given insertion is never more than $h+1$, so the temporal complexity of the insertion with rebalancing is at worst $O(\log n)$. Note, however, that insertion can increase the height of the B-tree to $h+1$.

To delete a key from a leaf with at least $m+1$ keys, remove the key. If the leaf has only m keys, removing the key will cause the leaf to be underfull. In this case take a key from an adjacent sibling, if the sibling can spare it. To do this, move the parent key down to the underfull leaf and move the sibling's key up to the parent; see Figure 3.15 for an illustration. If neither adjacent sibling can spare a key, merge with one of the adjacent siblings. This is always possible, because the total number of keys in the merged leaf will be $(m-1)+1+m = 2m$. This reduces the number of the parent's keys by 1, so if the parent is underfull, it will need to be rebalanced. In some cases, this can be done by moving either the largest or the smallest key up from the newly merged leaf to the parent, but in general this requires taking from a sibling again; see, for example, Figure 3.16.

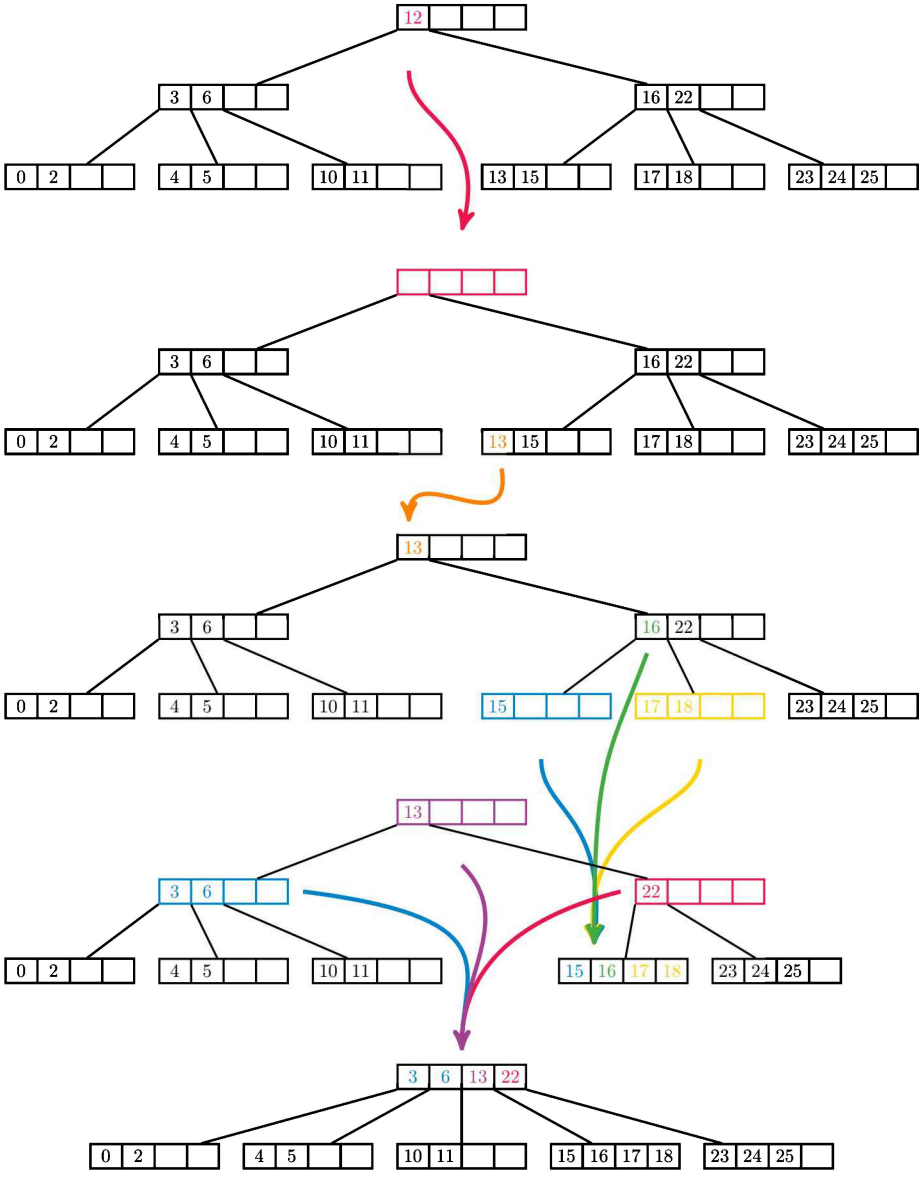


Figure 3.17. *Deleting from a nonleaf node of a B-tree: Deleting the key 12 leaves the root (red) underfull. Move 12's immediate successor 13 up to the root. This leaves the affected leaf (blue) underfull. Since its sibling (yellow) can spare no keys, merge them (and the parent key 16 between them) to produce a completely full leaf. But this makes the parent (red) underfull. To remedy this, since its sibling (blue) cannot spare keys, merge the underfull node with its sibling and parent (purple).*

To delete a key from a nonleaf node, if the key is not separating two children, remove the key and rebalance as above. If the key is separating two children, then its immediate in-order predecessor (the largest element in the left subtree) is still less than the key we wish to remove, and it lies in a leaf. Similarly, its immediate in-order successor (the smallest element in the right subtree) is still greater than the key we wish to remove, and it lies in a leaf. Move one of these two keys (the immediate predecessor or successor) up to fill the spot vacated by the deleted key. If the leaf from which the key was taken is underfull, rebalance as before. An example of this is given in Figure 3.17.

The operation of deletion involves a search for the key to delete, and at worst another search for the in-order predecessor or successor and a rebalancing. Each of these steps has temporal complexity at most $O(\log n)$, so the entire operation also has temporal complexity $O(\log n)$.

Remark 3.5.6. There are several common variants of the B-tree. For example, a *B+tree* is like a B-tree, but the actual data of the tree is stored only in the leaf nodes; all other nodes in the tree are index nodes.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

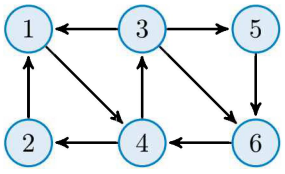
You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

3.1. Let $G = (V, E)$ with $V = \{a, b, c, d\}$ and $E = \{(a, b), (a, c), (a, d), (b, c)\}$. List all the subgraphs of G.

3.2. How many distinct undirected graphs can be created with seven vertices and 13 edges? What if the graphs are directed?

3.3. Consider the graph in the figure below. Use the adjacency matrix to determine the number of walks of length 4 from node 1 to node 4. How many of those walks are paths?



3.4. Consider the graph G_5 in Figure 3.2. Use the adjacency matrix to determine the number of closed walks of length 3 starting and ending at node 3. How many of those walks are cycles?

3.5. Let A be the adjacency matrix of a directed graph G.

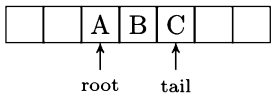
(i) Explain why the (i, j) entry of A is the number of walks of length 1 from node i to node j.

(ii) Prove that the (i, j) entry of A^2 is the number of walks of length 2 from i to j.

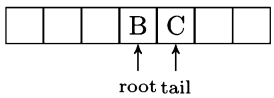
(iii) Complete the proof of Proposition 3.1.24. Hint: Use induction.

3.6.* † An *automorphism* of a graph $G = (V, E)$ is a bijection $\phi : V \to V$ such that for each edge $(v, v') \in E$ the pair $(\phi(v), \phi(v'))$ is also in E.
Show that if ϕ is an automorphism of an undirected (finite) tree with a finite number of vertices, then either there exists a vertex $v \in V$ with $\phi(v) = v$ or there exists an edge $(v, v') \in E$ with $\phi(v) = v'$ and $\phi(v') = v$.

3.7. We encode a sequence of instructions using letters and asterisks—a letter means `push` that letter, and $*$ means `pop`. For each operation of the sequence, illustrate the result of applying the operation to the corresponding data structure. For example, if the data structure is a stack, and the existing state was



then applying $*$ (`pop`) results in



(i) Starting with an empty queue, show the result of each step of the following sequence of operations (parsed from left to right): $AB * CD * *$

(ii) Starting with an empty stack, show the result of each step of the following sequence of operations (parsed from left to right): $AB * CD * *$

3.8. Describe how to use two stacks to implement a queue. What is the temporal complexity of the operations `push` and `pop`?

3.9. A *double-ended queue*, usually called a *deque* (pronounced "deck"), is a data structure like a queue, but where the data can be added to (pushed) or removed from (popped) either end of the queue. Describe how to implement a deque using one or more of the data structures described in Section 3.2.2. The temporal complexity of pushing or popping from either end should be $O(1)$.

3.10. Describe how to use a stack, queue, or deque to construct a palindrome verifier: Given any input $(a_0, \ldots, a_n)$, your algorithm should determine whether $a_i = a_{n-i}$ for every $i \in \{0, \ldots, n\}$. What is the temporal complexity of your algorithm? Justify your answer.

3.11. Prove that for an undirected graph G the following statements are equivalent:

(i) G is a tree.

(ii) G is connected, but for every edge e of G, removing the edge e makes the resulting graph disconnected.

(iii) G contains no cycle, but if any edge is added between any pair of vertices of G, the resulting graph has a cycle.

3.12. Prove that a perfectly balanced tree of height h has $n = 2^{h+1} - 1$ nodes.

3.13. Start with an empty BST.

(i) Show the result and all intermediate steps of adding the sequence 5, 3, 7, 2, 4, 6, in order (from left to right).

(ii) Show the result and all intermediate steps of adding the sequence 2, 3, 4, 5, 6, 7.

3.14. Let $S_n = \{1, 2, \ldots, n\}$ be a space of keys. Let C_n be the distinct number of BSTs for S_n. By the multiplication rule, there are $C_{k-1} \cdot C_{n-k}$ BSTs with root $k \in S_n$.

(i) Prove that

$$C_n = \sum_{k=1}^{n} C_{k-1} C_{n-k} = \sum_{k=0}^{n-1} C_k C_{n-1-k}.$$

(ii) Let $C(x) = \sum_{k=0}^{\infty} C_k x^k$. Prove that $C(x) = 1 + x\, C(x)^2$.

(iii) Since $C(0) = 1$, we have that

$$C(x) = \frac{1 - \sqrt{1 - 4x}}{2x}.$$

(iv) Using (2.32), prove that

$$C(x) = \sum_{k=0}^{\infty} \frac{1}{k+1} \binom{2k}{k} x^k.$$

Conclude for $n \in \mathbb{Z}^+$ that $C_n = \frac{1}{n+1}\binom{2n}{n}$.

3.15. Start with an empty AVL tree.

(i) Show each step of the process of adding and then rebalancing for the sequence 2, 3, 4, 5, 6, 7.

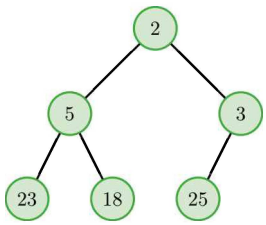
(ii) Now show each step of the process of deleting and then rebalancing for the sequence 2, 7, 5, 6.

3.16. For a left-left imbalance, as in Figure 3.8, prove that if all the nodes below C satisfy the AVL condition, but the balance of C is 2, then after a right rotation on B the nodes B and C in the new tree satisfy the AVL condition and the balance of all the other subtrees has not changed.

3.17. For a left-right imbalance, as in Figure 3.9, prove that a left rotation on B, followed by a right rotation on B, gives a new tree in which the nodes A, B, and C all satisfy the AVL condition and the balance of all the other subtrees has not changed.

3.18.* Give an example to show that the order in which elements are added to an AVL tree, and rebalanced by the AVL algorithm, can affect the final structure of the tree.

3.19. Beginning with the following heap, insert a node with key 1 and sift, as necessary, until the result is a new heap. Draw the corresponding tree at each intermediate step of the process.



3.20. Starting with the array $[7, 6, 5, 4, 3, 2, 1]$, heapify the array, showing the status of the array at each intermediate step of the process.

3.21. Given the array in the previous exercise, is there a permutation of the array that would require more steps to heapify? What permutation would require the most sifting? What permutation would require the least sifting?

3.22. Prove Proposition 3.4.4.

3.23. Prove Proposition 3.4.6.

3.24.* Prove Proposition 3.5.4.

3.25.* Starting with an empty B-tree of order $k = 4$, show each step resulting from adding (and rebalancing) the sequence $1, 2, 3, 4, 5, 6, 7, 8, 9$ (in order, from left to right).

3.26.* For the B-tree produced in the previous problem, show each step of the result of deleting (and rebalancing) the sequence $5, 2, 3, 1$.

3.27.* Consider an initially empty B-tree of order 3. Draw the B-tree after the insertion of the keys 27, 33, 39, 1, 3, 10, 7.

3.28.* Take your answer from the previous problem and then draw the tree after deleting the following keys: 33, 39, and 10.

3.29.* Find all legal B-trees of any order that contain only the keys 1, 2, 3, 4, 5.

Notes

A standard reference for much of the material in this chapter is [CLRS01]. AVL trees were first described in [AVL62] and have inspired many other methods for balancing BSTs. Our proof of Theorem 3.3.10 on the depth of AVL trees was inspired by [Pat17].

It's easy to see how to build a priority queue out of a sorting algorithm, but it can also be shown that any sorting algorithm with temporal complexity of $O(n\, s(n))$ can be used to implement a priority queue such that the cost of the `find_min` and `pop_min` function are $O(s(n))$. For more on this see [Tho07, WY12]. For more information about the average complexity of insertion in a heap, see [PS75].

4 Combinatorial Optimization

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.
—John von Neumann

Combinatorial optimization is about finding the best choice among a discrete collection of choices. Many important problems, both in mathematics and in life, can be formulated as combinatorial optimization problems. In this chapter we describe a few of these problems and some common techniques for attacking them.

The problems we consider here include finding the shortest route between two locations in a network of roads, the most efficient way to compress a message, and the most efficient or valuable combination of items that meet a given budget (which could be time, space, money, or something else). Many real-world problems can be reformulated in terms of these and other standard combinatorial optimization problems. You will find many of the ideas from this chapter cropping up over and over again, not only in this text, but whenever you want to do something faster, cheaper, or better.

Because combinatorial optimization problems are usually finite, we could, at least in theory, check every possible combination and see what works best. We call this naïve approach the *brute force* or *exhaustive* method. One example using this method is given in Example 4.0.1.

Example 4.0.1. Consider the *change-making* problem, which consists of finding the smallest number n of coins necessary to achieve a given value v. We assume the standard American coinage system, consisting of pennies ($0.01), nickels ($0.05), dimes ($0.10), quarters ($0.25), half dollars ($0.50), and dollars ($1.00).

To solve the problem for $v = \$0.19$ by the exhaustive method, we list all the ways to get $0.19:

Configuration	n
19 pennies	19
14 pennies, 1 nickel	15
9 pennies, 2 nickels	11
9 pennies, 1 dime	10
4 pennies, 3 nickels	7
4 pennies, 1 nickel, 1 dime	6

Since this table is complete, we see that the minimum number of coins for $v = 0.19$ is 6.

In practice, the exhaustive method works only in situations where there are relatively few possibilities to pick from. But, unfortunately, most combinatorial optimization problems have far too many possibilities for anyone to check them all. Example 4.0.2 is typical of the size of a combinatorial problem.

Example 4.0.2. The traveling salesman problem (TSP) asks for the best route to visit a given collection of n cities. It is straightforward to show that there are $n!/2$ possible routes. So if $n = 20$, then the number of possible routes is $20!/2 = 1{,}216{,}451{,}004{,}088{,}320{,}000$. Hence, even if you had a machine that could check a billion routes per second, it would still take more than 38 years to check all the possible routes. And in many applications, we need to solve this problem for much larger values of n, which means that the naïve, exhaustive approach is simply not feasible.

One of the most useful tools for solving optimization problems is called *dynamic programming*, which computes and keeps track of solutions of smaller, easier versions of the problem and then combines them to construct the solution of the full problem. We discuss dynamic programming in Section 4.1 and again in Chapter 16.

Another useful approach to solving discrete optimization problems is to use a *greedy algorithm*. These are algorithms that develop a candidate solution stepwise, always choosing the next step to be the option that appears at the moment to be best. Greedy algorithms are not always successful because the optimal solutions to many problems have intermediate steps that do not appear optimal at the intermediate stages of development. But for a surprising number of problems, there is a greedy algorithm that always produces the optimal solution, and for many other problems, there are greedy algorithms that usually produce solutions that are nearly optimal. We explore some of each of these sorts of problems in this chapter.

Finally, it is important to note that reformulating problems slightly to allow the possibility of a small amount of error in a solution can vastly reduce the complexity of the algorithms required to solve them. A good approximate answer can often be found very rapidly, even for many optimization problems that are essentially impossible to solve exactly. We discuss this briefly in Section 4.5.3 and further in Chapter 7.

4.1 Dynamic Programming

Some recursive algorithms are of low complexity because they "divide and conquer" efficiently. For example, the `mergesort` algorithm (see Algorithm 1.11) breaks a list of n elements into two subproblems of approximate size $\frac{n}{2}$. This gives us the recurrence relation (1.48) and results in $O(\log n)$ temporal complexity.

By contrast, in many combinatorial optimization problems the size of the subproblems encountered in a divide-and-conquer method is not necessarily much smaller than the original problem. For example, computing the lengths of all the paths among n cities in the TSP (see Example 4.0.2) involves a subproblem of computing the lengths of all the paths for $n-1$ cities, and the obvious divide-and-conquer approach to the TSP has temporal complexity satisfying the recurrence relation $T(n) = nT(n-1)$, which results in factorial complexity $O(n!)$. Note, however, that in this approach the various subparts of the recursive algorithm have substantial overlap, and we end up solving the same subproblems over and over again. This is common in many combinatorial optimization problems.

When problems have such an overlapping structure among their subproblems, the algorithms that solve them can often be improved by storing the commonly recomputed subproblems the first time they are computed and then looking up their answers the next time they are encountered. Depending on the amount of overlap among subproblems, this can significantly improve the temporal complexity of an algorithm, but it often comes at the cost of some increase in spatial complexity.

The strategy of storing solutions of subproblems, rather than recomputing, is called *dynamic programming*. The term *program* in this context does not refer to a computer program but rather a schedule or table, corresponding to the idea that the computed values are stored in some sort of lookup table.

Dynamic programming is primarily done in one of two ways:

(i) Top down (often called *memoization*[20]).

(ii) Bottom up (sometimes called *iterative* dynamic programming).

4.1.1 Top-Down Dynamic Programming

The top-down (memoization) approach consists of running a recursive process as usual, but at each stage, when a subproblem is encountered in the recursion, the results of that subcomputation are saved for possible future use. The next time the same subproblem is encountered, the answer is looked up instead of recomputed.

Example 4.1.1. Consider the change-making problem of Example 4.0.1. To use memoization on this problem, first formulate it recursively. Let $n(v)$ be the minimal number of coins required to achieve value $v \geq 0$. Denote the set of coins as $C = \{0.01, 0.05, 0.10, 0.25, 0.50, 1.00\}$. We can write $n(v)$ in terms of $n(v-c)$ for each $c \in C$ as

$$n(v) = \min_{c \in C} \{1 + n(v-c)\}. \tag{4.1}$$

[20] Not to be confused with *memorization*.

Each smaller problem $n(v-c)$ can then be attacked in the same way, repeating until the amount to solve for is 0.

We have

$$n(0.19) = 1 + \min_c n(0.19 - c) = 1 + \min\{n(0.18), n(0.14), n(0.09)\},$$
$$n(0.18) = 1 + \min_c n(0.18 - c) = 1 + \min\{n(0.17), n(0.13), n(0.08)\},$$
$$n(0.14) = 1 + \min_c n(0.14 - c) = 1 + \min\{n(0.13), n(0.09), n(0.04)\},$$
$$n(0.09) = 1 + \min_c n(0.09 - c) = 1 + \min\{n(0.08), n(0.04)\},$$

and so on. In a naïve recursion some of the subproblems (like $n(0.08)$) would be computed multiple times. In a memoized (top-down dynamic programming) algorithm, these values are stored the first time they are computed and then just looked up each subsequent time they are needed; see Algorithm 4.1 for details

```
# global variables
C = [1,5,10,25,50,100]    # Currency system
lookup={0:0}              # Known solutions

def makechange(v):
    """ Return the minimum number of coins that add to v cents.
    """

    if v in lookup.keys():
        return lookup[v]
    else:
        ans = 1 + min([makechange(v-c) for c in C if c <= v])
        lookup[v] = ans
        return ans
```

Algorithm 4.1. *Implementation of the change-making algorithm; note the use of the dictionary* `lookup` *for memoization. As expected, calling* `makechange(19)` *returns* 6. *After calling* `makechange(19)` *the dictionary* `lookup` *contains all the solutions for* $v \in \{0, \ldots, 19\}$.

Remark 4.1.2. Knowing the optimal number of coins $n(v)$ in the previous example does not necessarily tell you how to achieve that optimal number. But it is easy to adjust the algorithm to track the optimal configuration of coins (the *optimizer*) at each step. This is true of most combinatorial optimization problems. The approach to solving the problem usually produces the optimizer itself along the way, and so algorithms that find the optimal value ($n(v)$ in this example) can be adapted to also return the optimizer as part of the solution.

Example 4.1.3. The nth Fibonacci number $F(n)$ is defined by the recurrence

$$F(n) = F(n-2) + F(n-1) \tag{4.2}$$

with starting values $F(0) = F(1) = 1$. A naïve divide-and-conquer algorithm computes both $F(n-2)$ and $F(n-1)$ recursively. Therefore it has temporal complexity

$$T(n) = T(n-2) + T(n-1) + O(1) \geq 2T(n-2) + O(1) \in O(2^{n/2}),$$

which is terrible. Saving the result of each $F(k)$ the first time it is computed, speeds up the algorithm considerably. Indeed, the computation of $F(n-1)$ computes $F(n-2)$ along the way, so, using memoization, the value of $F(n-2)$ used in the sum (4.2) requires only a single lookup (hence $O(1)$), and so the new recurrence relation for the temporal complexity takes the form

$$T(n) = T(n-1) + O(1),$$

which means the memoized computation has temporal complexity in $O(n)$.

Remark 4.1.4. Although memoization offers a huge temporal savings and is generally easy to implement, memoization can come at a significant memory cost. In any divide-and-conquer algorithm, each step must be stored while the lower steps are being computed. For the naïve Fibonacci algorithm that means each level increases the spatial complexity by at least one, and so the overall spatial complexity is at least $O(n)$. Memoizing requires us to store the previously computed results, so it is also at least $O(n)$. When computing F for large values like $n = 10^6$, these algorithms are seriously constrained by limited memory.

Example 4.1.5. Consider the problem of multiplying three matrices together. Matrix multiplication is associative, so we have two choices of how to compute the product.

$$ABC = (AB)C = A(BC)$$

Exercise 1.26 shows that the choice of grouping can have a significant effect on the complexity of the computation. More precisely, if $A \in M_{k,\ell}(\mathbb{R})$, $B \in M_{\ell,m}(\mathbb{R})$, and $C \in M_{m,n}(\mathbb{R})$, then the temporal complexity of computing AB is $\sim 2k\ell m$ (see Section 1.5), and the complexity of computing $(AB)C$, given (AB), is $\sim 2kmn$ for a total complexity of $\sim 2(k\ell m + kmn)$ for the first grouping. Performing the multiplication using the second grouping has complexity $\sim 2(k\ell n + \ell mn)$.

If k, ℓ, and m are large but not too similar in size, we can often save a lot of time by comparing the complexities of the two groupings and then performing the multiplication using the grouping with lower complexity. When there

are many matrices to be multiplied, careful grouping can make a significant improvement in efficiency.

Assume that $A_0, \dots A_{n-1}$ are to be multiplied, with $A_k \in M_{m_k \times m_{k+1}}(\mathbb{R})$ for each $k \in \{0, \dots, n-1\}$, The optimal choice of grouping depends only on the dimensions $m_0, \dots, m_n$. Denote the complexity of the optimal choice of grouping by $C(m_0, m_1, \dots, m_n)$. This quantity satisfies the recursion relation

$$C(m_0, m_1, \dots, m_n) = \min_{0<k<n} \{C(m_0, \dots, m_k) + 2m_0 m_k m_n + C(m_k, \dots, m_n)\}. \tag{4.3}$$

This leads to a naïve divide-and-conquer algorithm that involves recursively computing $C(m_0, \dots, m_{k-1})$ and $C(m_k, \dots, m_n)$ for each choice of k. But many of these quantities are computed more than once. For example, $C(m_1, m_2)$ must be computed to find the quantity $C(m_0, m_1, m_2)$ and again to find $C(m_1, m_2, m_3)$. With memoization each of these is computed only once, which reduces the temporal complexity of this algorithm substantially.

4.1.2 Bottom-Up Dynamic Programming

With bottom-up dynamic programming we still take advantage of the fact that many of the subproblems are being solved repeatedly, but instead of using recursion, we use an iterative algorithm, starting with the simplest computations and then assembling the results into solutions of progressively more complicated problems, until the desired solution has been computed. Most of the previously computed values need to be remembered only for a few steps, after which they can be discarded. This means that the spatial complexity is usually less for bottom-up dynamic programming than for top-down, but the temporal complexity is similar between the two.

Example 4.1.6. The bottom-up approach to the change-making problem of Examples 4.0.1 and 4.1.1 is to compute $n(0) = 0$ and store that value. Now use that to compute $n(1) = n(0)+1 = 1$, and again store that value. Continuing in this way gives $n(2) = n(1) + 1 = 2$, $n(3) = n(2) + 1 = 3$, $n(4) = n(3) + 1 = 4$, and $n(5) = 1 + \min\{n(0), n(4)\} = 1$. To solve the problem $n(v)$ involves computing every value $n(a)$ for $a \leq v$.

These values need only be stored until we are sure they are no longer needed. Since 1.00 is the largest coin value, the smallest that $a - c$ could ever be is $a - 1.00$. Therefore, once $a - 1.00 > m$, we will never need the value $n(m)$ and can safely discard it. This means we need only store at most 100 values at a time, and the spatial complexity of this algorithm is $O(1)$.

Example 4.1.7. The bottom-up approach to the Fibonacci problem first computes $F(0)$ and $F(1)$, then uses those to compute $F(2) = F(0) + F(1)$. Since $F(0)$ will never be needed again, it is discarded. Now compute $F(3) = F(1)+F(2)$, at which point $F(1)$ is discarded. The algorithm continues in this way until reaching $F(n)$. This dynamic optimization algorithm still involves only n additions and has temporal complexity $T(n) \in O(n)$, but its spatial complexity is $O(1)$, which is much better than memoization.

Remark 4.1.8. Not all recursive algorithms can be improved with a dynamic program. The merge sort and binary search algorithms cannot be improved by these techniques because no subproblem is the same as any other subproblem (usually). In merge sort, for example, we don't expect any of the sublists that we must sort to be identical to any of the other sublists. Looking to see whether a sublist has already been sorted is a waste of time, and storing previously sorted sublists is a waste of memory.

Remark 4.1.9. Memoization is usually easy to implement, and it greatly improves temporal complexity if there are many repeated computations. This is often more than enough to achieve the performance needed from the algorithm. But it comes at the cost of some added spatial complexity. In many situations the spatial cost is not enough to matter, but when it does matter, bottom-up dynamic programming is often a better choice. Bottom-up dynamic programming usually takes more thought to implement than memoization, but the extra thought can produce significant savings in spatial complexity while still achieving the temporal savings of memoization.

4.1.3 Bellman Optimality

A fundamental idea that allows many problems to be solved rapidly with dynamic programming is *Bellman's optimality principle*. Bellman's principle is essentially a generalization of the observation that any part of a shortest path is itself a shortest path. For example, if the shortest path from Boston to Salt Lake City passes through Chicago, then the last part of that route, from Chicago to Salt Lake City, must be the shortest path from Chicago to Salt Lake City. More generally, many problems have the property that subparts of the optimal solution are optimal solutions for a subproblem.

Example 4.1.10. In the change-making problem, if the optimal number of coins $n(v)$ for v is achieved with coins of value $c_1, c_2, \ldots, c_{n(v)}$, then the optimal number of coins $n(v-c_1)$ is achieved by removing coin c_1 to get $c_2, c_3, \ldots, c_{n(v)}$. That is the essence of the recursion relation (4.1), which we call the *Bellman equation* for the change-making problem. This relation is the key to using bottom-up dynamic programming to solve the problem, as described in Example 4.1.6.

Example 4.1.11. The Bellman relation for the matrix-grouping problem of Example 4.1.5 is given by (4.3). It shows that if the best way to group n matrices includes a grouping of the first $k < n$ matrices, then that grouping is the best grouping for those k matrices. Again, this means we can use bottom-up dynamic programming to solve this problem by first computing $C(m_i, m_{i+1}, m_{i+2}) = 2m_i m_{i+1} m_{i+2}$, for every i, and then using the relation (4.3) to compute $C(m_i, m_{i+1}, m_{i+2}, m_{i+3})$ for every i, and so on, using (4.3) to assemble the previous results into the optimal value for a slightly more complicated problem, until finally reaching the optimal value $C(m_0, m_1, \ldots, m_n)$ of the original problem.

We revisit Bellman's optimality principle several times throughout this book, including in the next section, where we talk about Dijkstra's algorithm, and again in Chapter 16.

Remark 4.1.12. Bellman's optimality principle is so important in dynamic programming that people will often say *dynamic programming* or *dynamic optimization* when they actually mean the optimality principle.

4.2 Graph Search Algorithms

Many important combinatorial optimization problems can be formulated as graph search algorithms. In this section we discuss a few important examples of graph search problems and the algorithms for solving them.

Two key questions about paths in a graph (including trees) are

(i) is there is a path between two given nodes? and, if so,

(ii) what is the shortest path?

These are examples of *graph search* problems, and the two standard methods to solve them are *depth-first search (DFS)* and *breadth-first search (BFS)*.

4.2.1 Depth-First Search

The idea of DFS is simple: Start at the initial node, follow the first edge out of that node to a new node, follow the first edge out of that node to the next node (deeper), and so forth until arriving at a node with no neighbors that have not been visited. Then return up one step to the previous node, and follow the next edge and repeat until arriving at a node with no outgoing edges that have not already been traversed. Continue this process until the desired target node has been reached or you run out of available nodes; see, for example, Figure 4.1.

Remark 4.2.1. For a given graph there are many different ways to perform a DFS, depending on the different ways of ordering the outgoing edges from each node. The ordering is usually not important.

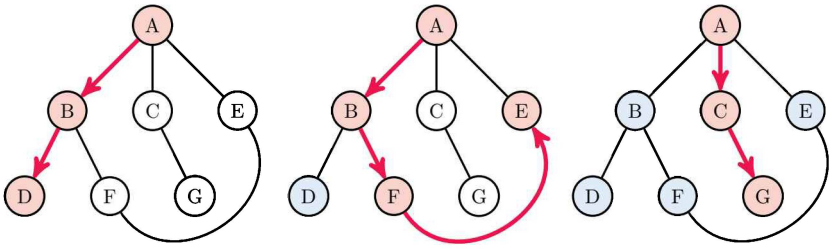


Figure 4.1. *Example of a DFS to find a path from A to G in an undirected graph. The algorithm begins at A and follows the first edge to B, and then to D (left panel). Since there are no edges out of D to unvisited (white) nodes, it backs up to B. Following the next edge out of B leads to F and then to E (center panel). Since there are no edges to unvisited node from E, it backs up to F, and then B, finding no unvisited neighbors at each step. Finally, backing up to A gives an edge to a new node C and then to G (right panel), where the search completes.*

To construct a DFS we need ways to keep track of which nodes have been visited and which nodes to visit next. To do this, we use two data structures:

- A stack S of the partial paths that have been generated.
- A *set*,[21] M (marked), of the nodes already visited.

Begin by putting the initial (starting) node on the stack S. At each stage, pop a partial path off the stack S, examine the last node N of that path, and add it to M. For each neighbor P of N that is not in M, add a new path to the stack consisting of the old path (to N) with P added to the end. Repeat the process by popping the next path from S and moving to its last node. If there is no neighbor for the last node of a path, then discard that path and pop the next path from S. Note that the paths in the stack S are processed in reverse order of arrival (LIFO), so we always move down first, as required in a DFS. The algorithm terminates when S has no more paths or when the target node is found; see Algorithm 4.2.

Example 4.2.2. The following table lists the main variables and their states at the beginning of each step of a DFS (using Algorithm 4.2), searching for a path from A to G in the graph in Figure 4.1. The left panel of Figure 4.1 corresponds to step 2 here, the center panel corresponds to step 4, and the right panel corresponds to step 6.

Step	0	1	2
S	`[[A]]`	`[[A,E],[A,C],[A,B]]`	`[[A,E],[A,C],[A,B,F], [A,B,D]]`
path	`[A]`	`[A,B]`	`[A,B,D]`
M	{A}	{A,B}	{A,B,D}

[21] The data type called a *set* is similar to a mathematical set, in that each element occurs at most once and there is no ordering of the elements. Most implementations of the set data type have a very efficient method for identifying whether an element is in the set. For more on the set data type, see Section 7.3.

Step	3	4
S	[[A,E],[A,C],[A,B,F]]	[[A,E],[A,C],[A,B,F,E]]
path	[A,B,F]	[A,B,F,E]
M	{A,B,D,F}	{A,B,D,F,E}

Step	5	6
S	[[A,E],[A,C]]	[[A,E],[A,C,G]]
path	[A,C]	[A,C,G]
M	{A,B,D,F,E,C}	{A,B,D,F,E,C,G}

```
1  def dfs(graph, start, end):
2      """Find a path from start to end with the DFS algorithm.
3      `graph` is a dictionary mapping each node to the set of
4              its neighbors.
5      """
6
7      # Initialization
8      M = set(start) # Set of marked (visited) nodes.
9      S = [[start]]  # Stack of partial paths.
10
11     while S:                  # While S is not empty:
12         path = S.pop()
13         M.add(path[-1])       # Add the path's last node to M.
14         if path[-1] == end: # If the end node is found.
15             return path
16         for node in graph[path[-1]] - M: # Difference of sets.
17             S.append(path+[node])        # Update the stack.
```

Algorithm 4.2. *Implementation of DFS. Here* `graph` *must be a dictionary mapping each node of the graph to the set of its neighbors. Here the Python list* `S` *functions as a stack, where* `append()` *plays the role of the operation* `push`*, and* `pop()` *pops the last element off list (the end of the list is the top of the stack). To change this implementation into a BFS, simply replace* `pop()` *in Line* 12 *with* `pop(0)`*, which pops the first element off the list* `S` *(instead of the last) and thus makes* `S` *into a queue. For details on applying this code to the graph in Figure 4.1, see Example 4.2.3.*

Example 4.2.3. An implementation of the DFS algorithm in Python is given in Algorithm 4.2. In that implementation the graph is stored as a dictionary `graph` mapping each vertex to its neighbors. For the graph in Figure 4.1 we have `graph = {'A':{'E','C','B'}, 'B':{'A','D'}, 'C':{'A','G'}, 'D':{'B','E'}, 'E':{'A','D','F'}, 'F': {'E'}, 'G': {'C'}}`. For this example, calling `dfs(graph,'A','G')` returns the list `['A', 'C', 'G']`.

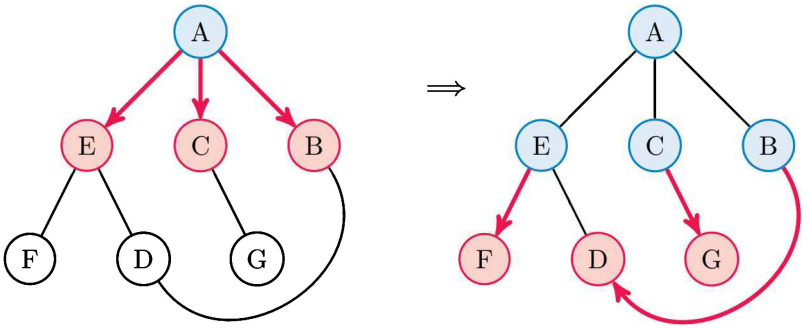


Figure 4.2. *An example of a BFS to find a path from A to F in an undirected graph. First the nodes nearest to A are visited: B, C, and E. Then the nodes that are two steps away are visited: D, G, and finally F.*

4.2.2 Breadth-First Search

BFS is very similar to DFS, except it first visits all the nodes adjacent to the starting node, and then visits all the nodes that are two steps away from the starting node, and so on. An example of a BFS is given in Figure 4.2.

From an algorithmic perspective, the only change needed to convert the DFS algorithm into the BFS algorithm is to replace the stack S with a queue, so that the nodes are processed in the order they arrived (FIFO). To implement this in Python, simply change Line 12 of Algorithm 4.2 to `path = S.pop(0)`, which pops the first element of S instead of the last, and makes S into a queue instead of a stack.

4.2.3 BFS versus DFS

The choice of which of these two graph search algorithms to use depends on both the graph and the question being asked. If the goal is to determine whether the graph is connected, then every node that is connected to the starting node must be visited, and potentially every edge examined, regardless of the algorithm used. In this case both DFS and BFS could have complexity as bad as $O(|V| + |E|)$, where $|V|$ is the total number of nodes and $|E|$ is the total number of edges. So, in this case it does not matter much which algorithm is used.

If the goal is to find the shortest path between two nodes, then BFS is really the only choice, because the first path found using BFS will be the shortest, while the first path found by DFS could easily be the longest, and DFS is not well suited to finding another, shorter path. For example, in Figure 4.1 the path from A to E found by DFS has length 3, while the first path from A to E found by BFS (in Figure 4.2) has length 1.

If the goal is to find whether there exists a path between two given nodes of a graph, then both of these algorithms have worst-case temporal complexity of $O(|V| + |E|)$. They both could end up visiting every other node first, before finally reaching the target node. So for the existence-of-path problem the choice of which algorithm to use depends on which one is most likely to avoid the worst cases for the given graph.

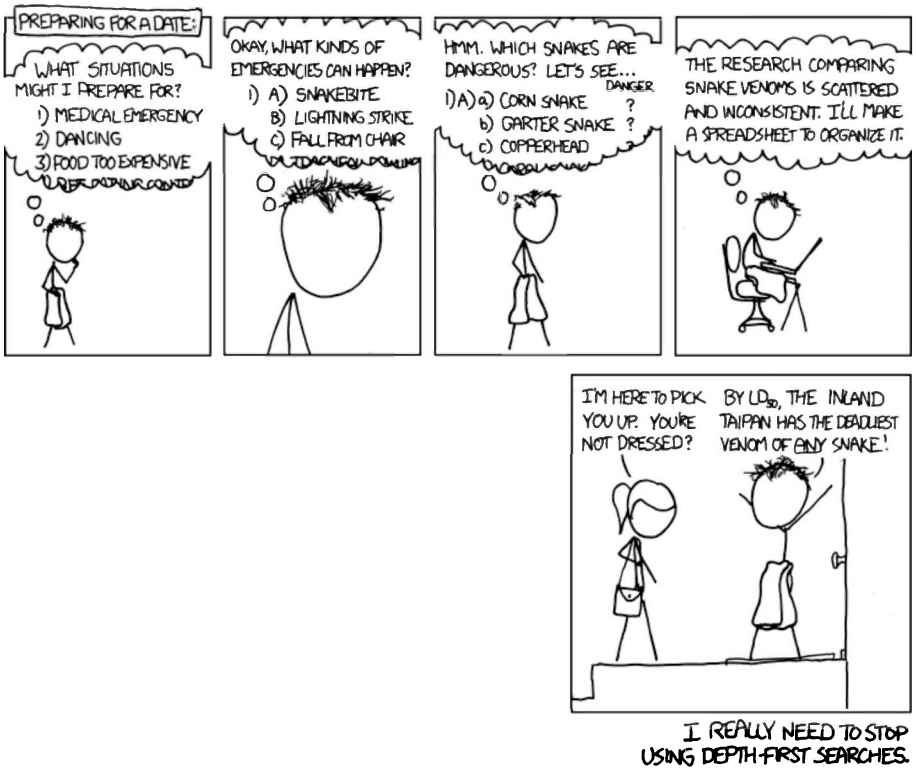


Figure 4.3. *Some potential problems with DFS. Source: XKCD, Randall Munroe.* `http://xkcd.com/761/`

The DFS algorithm is generally most effective for graphs where the target node is likely to be far from the start. It also tends to perform better when the available memory is limited relative to the size of the graph. The BFS algorithm is generally better when the target node is likely to be near the starting node and sufficient memory is readily available.

4.2.4 Shortest Path via Dijkstra's Algorithm

It is often useful to attach a weight to each edge of a graph. For example, these weights could represent physical lengths if the nodes represent locations on a map and the edges represent roads between them. Or they could represent costs associated to traversing the edge, if the graph represents a utility network, and some edges have more capacity or lower cost than others. See Figure 4.4 for an example of a weighted graph. In this section we always require that the weights be nonnegative.

The BFS algorithm finds the shortest path between two nodes if all the edges have the same weight, but now we want to extend to the case that edges have different weights. Dijkstra's algorithm is the canonical method for finding the minimal-weight path from a given starting node to a given target node in a weighted directed graph. It can also be easily adapted to find the minimal-weight path to every node in the graph from a given starting node.

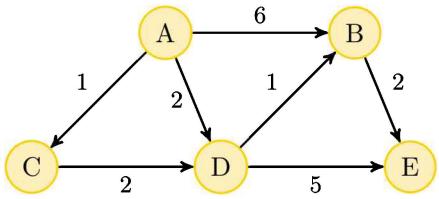


Figure 4.4. *An example of a weighted graph, where each edge has a nonnegative weight. In this figure the weights all happen to be integers, but the algorithms of this section work for any nonnegative real weights.*

Dijkstra's algorithm is essentially a weighted variant of the BFS, but nodes are visited in a locally minimizing way. Thus it is a greedy algorithm, but it is still guaranteed to give the optimal solution. The key to Dijkstra's algorithm is the Bellman optimality principle: any segment of a shortest path is itself a shortest path. The main idea uses this principle to do a sort of bottom-up dynamic programming, computing the minimal distance to nodes, one after another, until reaching the desired target.

The minimal distance to the starting node s is, of course, 0. Proceed now to the nodes that are adjacent to the starting node, noting that the minimal path between the starting node and a target node t must pass through one of these adjacent nodes. Let v be the node adjacent to s that has the smallest weight $w(s, v)$. There can be no shorter path from s to v because the first step of any path must pass through one of the adjacent nodes and hence must have weight at least $w(s, v)$. So the optimal path to v has weight $0 + w(s, v)$. Now repeat the process with the collection of all the nodes adjacent to either s or v (but not including s and v). Continuing in this way eventually gives the optimal path from s to the target node t.

The main data structure used in Dijkstra's algorithm is a priority queue Q containing all the nodes whose optimal path has not yet been found, prioritized by the length of the current shortest path to those nodes. In detail, the algorithm proceeds as follows: For each node u set $d(u) = \infty$, except for the starting node s, which is set to 0. The value of $d(u)$ represents the length of the shortest path found so far from s to u. Push the node s onto the priority queue Q with priority $d(s)$. Repeat the following steps until the minimal path is found or Q is empty. Pop the minimal element off of Q, and call this v. Because it is minimal, the optimal path to v has been found. If v is the target node, we are done. If not, then for each neighbor u of v whose optimal path has not been found, check whether $d(v) + w(v, u) < d(u)$, where $w(v, u)$ is the weight of the edge from v to u. If the inequality holds, then set the predecessor of u to v, update $d(u) = d(v) + w(v, u)$, and push u onto Q with priority $d(u)$. Repeat the process, popping the next minimal element off Q and evaluating its neighbors. If Q has no remaining nodes to pop, then there is no path from s to t.

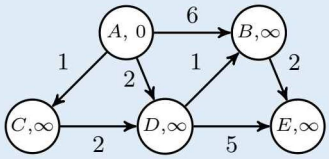
If the node u is already in Q, the last step of the iteration, pushing u onto Q with priority $d(u)$, means that Q has multiple copies of u with different priorities. In all future steps of the algorithm the copy with the lowest value of d will be popped first, so the other copies do not interfere with the identification of the best value of $d(u)$, but some copies of u could be popped off Q before all the nodes have been

processed. These should be discarded once the first copy of u has been processed. Alternatively, if the priority queue also has an efficient method for updating the priority of an existing node, then instead of pushing a new copy of u onto Q, we could just update the priority of u in Q.

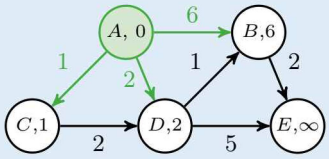
The optimal path from s to t is recovered by working backward from t by identifying its predecessor and then the predecessor of the predecessor, and so forth until reaching s. An implementation of the algorithm in Python is given in Algorithm 4.3.

Remark 4.2.4. If every edge in a graph has the same weight, then Dijkstra's algorithm traverses the edges in the same order as the BFS.

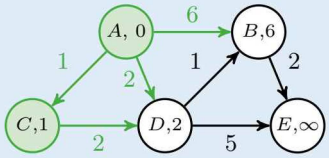
Example 4.2.5. We use Dijkstra's algorithm to find the shortest path from A to E in the graph in Figure 4.4. Throughout this example, nodes that have been processed are green, while those still in the priority queue Q are white.



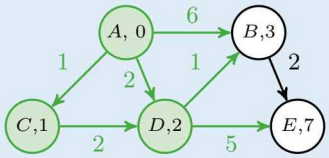
The algorithm is initialized with every node u having priority $d(u) = \infty$, except the starting node A, which has $d(A) = 0$.



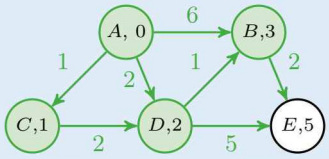
Node A is popped off Q. Distances between A and its neighbors are updated, giving $d(B) = 6$, $d(C) = 1$, and $d(D) = 2$. These nodes are pushed onto Q with priorities given by d.



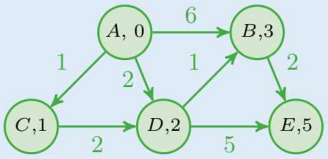
Node C is popped off Q. The priority of D remains unchanged because $d(D) < d(C) + w(C, D)$.



Node D is popped off Q. The priorities of B and E are updated to $d(B) = 3$ and $d(E) = 7$, and these are pushed onto Q.



Node B is popped off Q. The priority of E is updated to $d(E) = 5$ (and E is pushed onto Q with this priority).



Node E is popped off Q. The minimum distance between A and E is 5.

The path can be reconstructed by stepping back through the predecessors, which can be stored for each node and updated at each step when the priorities are updated. For our example, they are $E \leftarrow B \leftarrow D \leftarrow A$.

Example 4.2.6. An implementation of Dijkstra's algorithm in Python is given in Algorithm 4.3. In that implementation the graph `graph` is given as a dictionary of dictionaries, where the outer dictionary maps nodes to neighbors and the inner dictionary maps nodes to weights. For example, we could let `graph = {'A':{'B':6, 'C':1, 'D':2}, 'B':{'E':2}, 'C':{'D':2}, 'D':{'B':1, 'E':5}, 'E':{}}`, corresponding to the graph in Figure 4.4 and Example 4.2.5. Calling `dijkstra(graph, 'A', 'E')` returns `(5, ['A', 'D', 'B', 'E'])`.

Theorem 4.2.7. *On a finite, weighted directed graph G, Dijkstra's algorithm always terminates, and if there is a path from the start to the target, Dijkstra's algorithm returns the shortest path.*

Proof. The algorithm always terminates because each vertex v can be added to the queue Q only when an edge into v is processed, and each edge is processed only once, so the maximum number of entries in Q is the number $|E|$ of edges in G, which is finite.

We prove by induction on the number k of visited nodes (remember that duplicates are discarded, not revisited) that the following hypotheses hold:

(i) If v is the node currently being visited, then the shortest path to v has length $d(v)$.

(ii) For every unvisited node u, the shortest path to u *passing only through the visited nodes* has length $d(u)$, if such a path exists at all. If no path exists, then $d(u) = \infty$.

If $k = 1$, then only the starting node has been visited, and the hypotheses are clearly true. Assume the hypotheses hold for the first k nodes visited, and let v be the kth node visited. Denote the next node to be visited as u, so $d(u) \leq d(x)$ for every other unvisited node x.

If there is a path P to u that has length less than $d(u)$, then by (ii) P must contain at least one other unvisited node. Let y be the first unvisited node in P. By hypothesis the shortest path to y passing only through visited nodes has length $d(y)$. But this implies that the length of the subpath of P from the start to y must

```python
from queue import PriorityQueue
from math import inf                # Infinity

def dijkstra(graph,s,t):
    """ Find shortest path from s to t in `graph`, where
    `graph` is a dict mapping each node to a dict mapping
    neighbors to weights.
    """

    # Initialize.
    Q = PriorityQueue()
    Q.put((0,s))                    # Start Q with only s
    pred = {}                       # Dictionary of predecessors
    finished = set()                # Set of finished nodes
    d = {u:inf for u in graph.keys()} # Dictionary of distances
    d[s]=0

    # Iterate through the nodes in Q.
    while not Q.empty():
        (_,v) = Q.get()             # Pop the node with smallest d
        if v == t: break            # Success!
        elif v in finished: continue      # v was already done
        else:
            finished.add(v)               # v is now finished
            for u in set(graph[v])-finished: # Unfinished nghbrs
                if d[v] + graph[v][u] < d[u]:
                    d[u] = d[v] + graph[v][u] # Update dist to u
                    pred[u]=v         # Update predecessor of u
                    Q.put((d[u],u))   # Push (d[u],u) onto Q

    # Build the optimal path, from t back to s
    path = [t]                      # Start at t
    while path[-1] != s:
        path.append(pred[path[-1]])  # Add predecessor to path

    return d[t], path[::-1]    # Invert path to return it forward
```

Algorithm 4.3. *Implementation of Dijkstra's algorithm in Python to find the shortest path from vertex* `s` *to vertex* `t` *in a weighted graph* `graph`*. Here* `graph` *is given as a dictionary of dictionaries, where the outer dictionary maps nodes to neighbors and the inner dictionary maps nodes to weights. Example 4.2.6 gives details on applying this code to the graph in Figure 4.4 and Example 4.2.5.*

be at least $d(y)$, and hence the length of P must be at least $d(y)$. Since u was the next unvisited node to visit, and y is also unvisited, we have $d(y) \geq d(u)$, which is a contradiction. Therefore the shortest path to u must have length $d(u)$. This shows that (i) holds.

Finally, after visiting u, and given any node x in the graph, if the shortest path to x containing only visited nodes passes through u, then x is adjacent to u and that path has length $d(u) + w(u, x)$. If that path does not pass through u, then, by the induction hypothesis, it was already found before visiting u and has length $d(x) \leq d(u) + w(u, x)$. Thus (ii) holds. □

The temporal complexity of Dijkstra's algorithm is dominated by the cost of the priority queue operations. Let $|E|$ denote the total number of edges in the graph and $|V|$ the total number of vertices. There are at most $|E| + 1$ inserts of new elements into Q, contributing $O(|E| \log(|E|))$ to the temporal complexity. Moreover, the algorithm requires `pop_min` for each node in Q, including possible duplicates, for a total of up to $|E| + 1$ calls to `pop_min`, each of cost $O(\log(|E|))$, for a total complexity of $O(|E| \log(|E|))$. Note that $|E| \leq \binom{|V|}{2} \leq |V|^2$, and, if the graph is connected, then we also have $|V| - 1 \leq |E|$. Therefore, $\log(|V| - 1) \leq \log(|E|) \leq 2 \log(|V|)$, so we can also write the complexity of Dijkstra's algorithm as $O(|E| \log(|V|))$. One can also implement the priority queue with a specialized data structure called a *Fibonacci heap* which reduces the overall complexity of Dijkstra's algorithm to $O(|E| + |V| \log(|V|))$.

4.3 Minimum Spanning Trees

A *spanning tree* is a subgraph of an undirected graph that is a tree and contains all vertices. For example, if the graph represents all the potential connections that could be made to provide electricity to customers, then a spanning tree represents a way to make a connection to every customer without any loops (cycles) in the network. A *minimum spanning tree (MST)* is a spanning tree in a weighted, undirected graph that minimizes the total weight of the edges in the tree. For example, in a utility network an MST would represent a network that reaches every customer for the least cost. A graph may have many spanning trees (as shown in Figure 4.5) and even many MSTs.

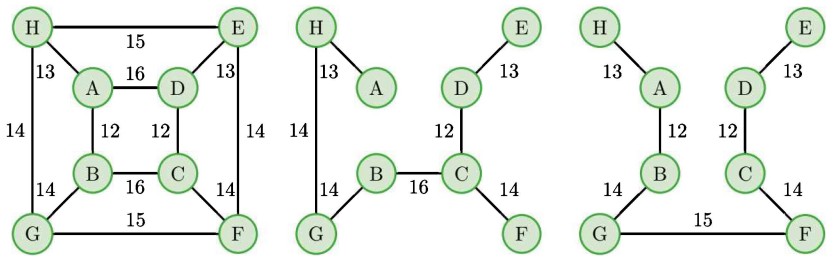


Figure 4.5. *A weighted graph (left) and two spanning trees (center and right). The tree in the center has weight* 96 *and the tree on the right has weight* 93. *Is there a spanning tree for this graph of weight less than* 93*?*

4.3.1 Prim's Algorithm

Prim's algorithm is a greedy algorithm for finding an MST. At each stage it adds the shortest edge that connects a node in the existing tree to a node that is not in the tree. But despite the fact that the algorithm only looks for these locally minimal edges, the tree it produces actually gives a global minimum.

As with Dijkstra's algorithm, the main data structure used in Prim's algorithm is a priority queue Q, consisting of nodes adjacent to, but not contained in, the current tree, ordered by the length (weight) of the edge connecting the node in Q to a node in the tree. We also need to repeatedly identify which nodes have already been processed, so we put each processed node into a set V.

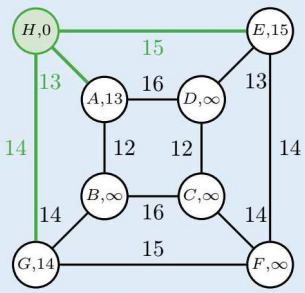
The algorithm is initialized as follows: For each node u set $d(u) = \infty$. Choose an arbitrary starting node s and set $d(s) = 0$. Push node s onto Q with priority 0. Also set V and E to be empty sets.

As long as Q is not empty, repeat the following steps. Pop the minimal element off of Q. If it is in V, discard it and keep popping elements off Q until getting one, call it v, that is not in V. Insert v into V. If the predecessor p of v exists, then add the edge (p, v) into E. For each neighbor u of v that is not in V, let $w(v, u)$ be the weight of the edge from v to u. If $w(v, u) < d(u)$, then set the predecessor of u to v, update $d(u)$ to $w(v, u)$, and push u onto Q with new priority $d(u)$ (or update the priority of u in Q).

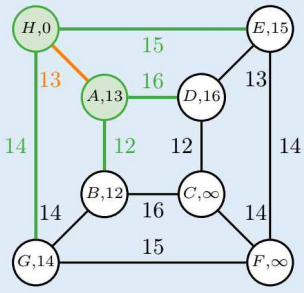
Once Q is empty, the main algorithm is finished. The MST is recovered as the graph whose nodes are in V and whose edges are in E. If V is not all the nodes of G, the graph is not connected. An implementation of the algorithm in Python is given in Algorithm 4.4.

Example 4.3.1. We use Prim's algorithm to find an MST for the leftmost graph in Figure 4.5. Nodes that have been processed are green, while those which have not been processed are white. Edges that are in the current tree E are orange, those that are predecessors of some node but are not yet confirmed members of the tree are green. Nodes that have been processed but are not predecessors of any node are grayed out.

Initialize by choosing an arbitrary starting node (in this case H) and setting $d(H) = 0$ and $d(u) = \infty$ for every other node u. Put H in priority queue Q.

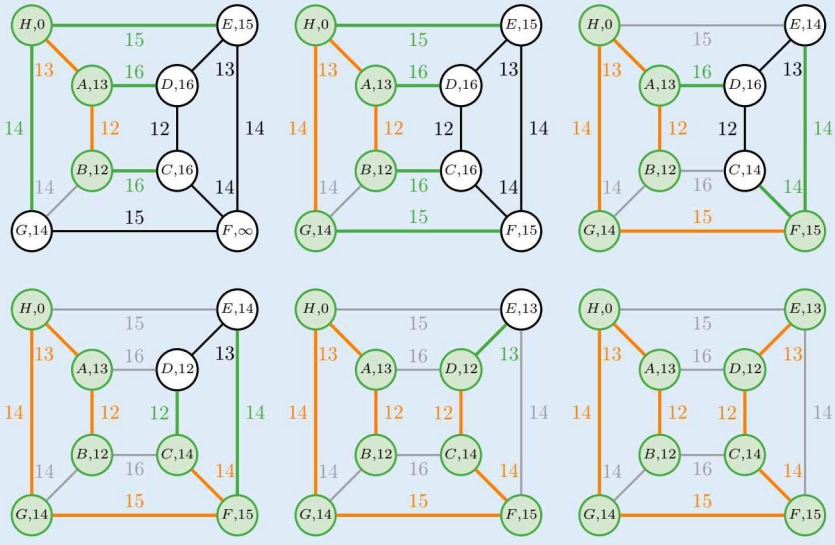


Pop H off Q and update the neighbors to give $d(A) = 13$, $d(G) = 14$ and $d(E) = 15$. Push each of these onto Q, and set the predecessor of each to H, indicated by the green edges.



Pop A off Q and add the edge to its predecessor H to the tree (orange). Update the neighbors of A to give $d(B) = 12$, $d(D) = 16$ and push these onto Q. Set the predecessor of each of these to A.

The rest of the steps are similar and are displayed here in sequence.



Example 4.3.2. An implementation of Prim's algorithm in Python is given in Algorithm 4.4. In that implementation, as in Algorithm 4.3, the graph `graph` is given as a dictionary of dictionaries, where the outer dictionary maps nodes to neighbors and the inner dictionary maps nodes to weights. The graph in Example 4.3.1 is stored as `graph = {'H':{'G':14, 'A':13, 'E':15}, 'G':{'H':14, 'B':14, 'F':15}, 'F':{'G':15, 'C':14, 'E':14}, 'E':{'H':15, 'D':13, 'F':14}, 'D':{'C':12, 'A':16, 'E':13}, 'C':{'D':12, 'B':16, 'F':14}, 'B':{'G':14, 'A':12, 'C':16}, 'A':{'H':13, 'B':12, 'D':16}}`.

Running `prim(graph)` returns the edges (and lengths) of an MST and the total length of the tree: `{('A', 'H'): 13, ('B', 'A'): 12, ('G', 'H'): 14, ('E', 'H'): 15, ('D', 'E'): 13, ('C', 'D'): 12, ('F', 'E'): 14}, 93`

```python
from queue import PriorityQueue
from math import inf

def prim(graph):
    """ Construct a minimum spanning tree of `graph`.
    """

    # Initialize.
    s = next(iter(graph))  # Arbitrary starting node
    Q = PriorityQueue()
    Q.put((0,s))           # Start with only s on Q
    pred = {}              # Dict of predecessors
    V = set()              # Processed vertices
    E = {}                 # Dict of MST edges & lengths
    d = {x:inf for x in graph.keys()} # Dist: node to tree
    d[s] = 0

    #Build the MST by iterating through nodes in Q
    while not Q.empty():
        (_,v) = Q.get()          # Pop the node with smallest d
        if v in V: continue      # v was already done, retry.
        V.add(v)
        if v in pred:            # If v has a predecessor
            E[(v,pred[v])] = graph[pred[v]][v] # Add edge
        for u in set(graph[v])-V:  # Neighbors of v not in V
            if graph[v][u] < d[u]: # If wt(u,v) < old dist
                pred[u] = v      # Update predecessor of u
                d[u] = graph[v][u] # Update dist from u to tree
                Q.put((d[u],u)) # Push (d[u],u) onto Q

    return E, sum(x for x in E.values()) # Edges of MST & length
```

Algorithm 4.4. *Implementation in Python of Prim's algorithm to find an MST in a weighted, undirected graph* `graph`*. As in Algorithm 4.3, the graph* `graph` *is given as a dictionary of dictionaries, where the outer dictionary maps nodes to neighbors and the inner dictionary maps nodes to weights.*

Theorem 4.3.3. *Prim's algorithm applied to a finite, connected, weighted graph G always terminates and returns an MST.*

Proof. We continue with the same notation used in the description of the algorithm, but we denote the queue Q and the sets V and E at the kth stage of the algorithm by Q_k, V_k, and E_k, respectively. Let T_k be the subgraph of G consisting of the edges E_k and the nodes defining those edges. The algorithm always terminates because each vertex v can be added to the queue Q only when an edge into v is processed, and each edge is processed only once, so the maximum number of entries in Q is the total number of edges in G, which is finite.

We first prove by induction that Prim's algorithm returns a tree. The induction hypothesis is that after the kth step, the subgraph T_k forms a subtree of G. The initial hypothesis is trivially true since $E_1 = \emptyset$ contains no edges and hence T_1 has no cycles. At stage $k+1$, the next edge to be added connects the existing tree T_k to a node that is not in E_k, and thus there can be no cycles in the new graph. Therefore the result T_{k+1} returned by Prim's algorithm is a tree. By induction, the terminal tree T contains all the vertices of G and is thus a spanning tree.

We have shown that a spanning tree must exist. Since the graph is finite, there must be an MST. Let $\widetilde{T}$ be an MST of G. Order the vertices of G by when they were visited by Prim's algorithm in the construction of T. Let v be the first vertex such that the edge e of T from the previous vertex u to v is not in $\widetilde{T}$, and assume that v was the kth vertex added to T by the algorithm. Let T_{k-1} be the subtree generated by Prim's algorithm up to step $k-1$, that is, including u but not v.

Let P be a path in $\widetilde{T}$ connecting u to v, and let $\widetilde{e}$ be an edge in P that is not in T_{k-1} but has one vertex in T_{k-1}. Construct a subgraph $\widehat{T}$ of G from $\widetilde{T}$ by replacing edge $\widetilde{e}$ in $\widetilde{T}$ with edge e. We claim that $\widehat{T}$ is also an MST. To see that $\widehat{T}$ is connected, first note that adding edge e to the path P makes a cycle, and removing the edge $\widetilde{e} = (a, b)$ from that cycle gives a path $\widehat{P}$ from a to b that lies in $\widehat{T}$. Now pick any two vertices x and y. Let $\widetilde{P}$ be the path in $\widetilde{T}$ that connects them. If $\widetilde{e} \notin \widetilde{P}$, then $\widetilde{P}$ is also in $\widehat{T}$. If $\widetilde{e} \in \widetilde{P}$, then form a walk W in $\widehat{T}$ by replacing $\widetilde{e}$ in $\widetilde{P}$ with $\widehat{P}$. The walk W lies in $\widehat{T}$ and connects x to y, hence $\widehat{T}$ is connected.

The total weight of $\widehat{T}$ is less than or equal to that of $\widetilde{T}$ because any difference in weight can only be due to the edge e, but e was chosen by Prim's algorithm at stage k instead of $\widetilde{e}$, so $w(e) \leq w(\widetilde{e})$. The tree $\widetilde{T}$ is an MST, so the weight of $\widehat{T}$ cannot be less, therefore it must be the same as that of $\widetilde{T}$. Finally, we note that $\widehat{T}$ is a tree. If it weren't, then it would have an MST which would have fewer edges and, hence, less weight than $\widetilde{T}$. This is a contradiction to the fact that $\widetilde{T}$ is an MST. Thus, $\widehat{T}$ is a tree, and therefore an MST of G.

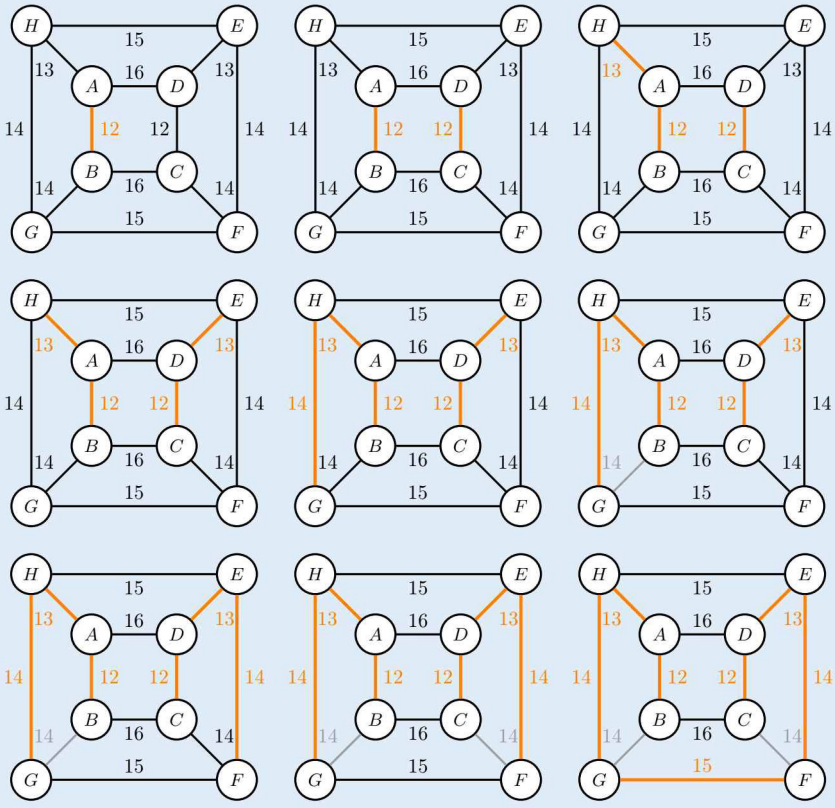
The new MST $\widehat{T}$ has one more edge in common with T than $\widetilde{T}$ did. Repeating this process of constructing a new MST closer to T eventually gives an MST that is equal to T. Therefore T is an MST of G. □

The analysis of Prim's algorithm is similar to Dijkstra's. If a binary heap is used for the priority queue, then the temporal complexity of Prim's algorithm is $O(|E| \log |E|)$. If a Fibonacci heap is used, then temporal complexity of Prim's algorithm is $O(|E| + |V| \log |V|)$.

4.3.2 Kruskal's Algorithm

The other standard algorithm for finding an MST of a undirected weighted graph is called *Kruskal's algorithm*, which we describe here only briefly. It is also greedy, and it works by choosing edges one at a time from lowest weight to highest weight and discarding the edge if the resulting graph has a cycle. The algorithm terminates when all the edges are either added to the subgraph or discarded. The resulting subgraph is connected with no cycles (and is therefore a tree) and by construction has the lowest possible total weight. Therefore, it is an MST.

Example 4.3.4. We use Kruskal's algorithm to find an MST for the leftmost graph in Figure 4.5. Edges that are in the priority queue are black. Those that are added to the tree are orange, and those that have been discarded are grayed out. The algorithm steps through as follows:



The final three black edges can also be grayed out, since the final orange subgraph is already an MST.

Kruskal's algorithm can also be made to run in $O(|E| \log |E|)$ time. It tends to do better than Prim's algorithm when the graph is sparse.

4.4 Huffman Encoding

In the first half of the 19th century, the invention of the telegraph changed the world. Communications that had previously taken days, weeks, or even months through the mail service became nearly instantaneous, with the main bottleneck being the human operators who sent, received, and relayed messages. By the second half of the 19th century, a transatlantic cable connected North America and Europe.

Across the world, telegraph messages were sent and received through Morse code, a system of dots and dashes used to encode letters and numbers. Telegraph

a	•–	j	•–––	s	•••	1	•————
b	–•••	k	–•–	t	–	2	••–––
c	–•–•	l	•–••	u	••–	3	•••––
d	–••	m	––	v	•••–	4	••••–
e	•	n	–•	w	•––	5	•••••
f	••–•	o	–––	x	–••–	6	–••••
g	––•	p	•––•	y	–•––	7	––•••
h	••••	q	––•–	z	––••	8	–––••
i	••	r	•–•	0	—————	9	————•

Table 4.1. *Morse code chart. Note that each codeword ends with a space, and a space may occur only at the end of a codeword.*

operators were able to send and receive messages at rates of 20 to 80 words per minute. This was the primary mode of long-distance and maritime communication for over 100 years, used even into the latter part of the 20th century.

Morse code has a built-in efficiency, some of the most commonly used letters having the shortest code and less used letters having longer codes. For example, the letters E and T are, respectively, a single dot and a single dash, whereas Z is two dashes and then two dots. For a chart of the entire coding scheme, see Table 4.1.

In this section, we discuss how to create efficient encodings, given knowledge of the relative frequencies of the symbols being encoded. This allows more efficient transmission and storage of information. The most efficient of these encoding methods is called *Huffman encoding*.

4.4.1 Introduction to Coding

Encoding is the process of mapping a source alphabet S to a set C of codewords, formed by combining strings from a code alphabet A. For example, in Morse code, the source alphabet is alphanumeric (letters and numbers used in the English language), and the code alphabet is dots, dashes, and pauses (spaces). The codewords are the sequences of dots and dashes used to form the alphanumeric letters, and pauses go between codewords, as described in Example 4.4.1.

Example 4.4.1. Morse code is a map f from the source alphabet

$$S = \{a, b, c, \ldots, z, 0, 1, 2, \ldots, 9\}$$

to a collection of codewords constructed from the code alphabet consisting of three code letters: •, –, and "space." Each codeword ends with a space, and a space may not appear anywhere else in any codeword. For example, the map sends the source letter a to the codeword •– (including the trailing space). The full map is given in Table 4.1.

Example 4.4.2. Let $S = \{a, b, c, d, \ldots, z\}$ and $C = \{01, 02, 03, \ldots, 26\}$, with the obvious mapping $a \mapsto 01$, $b \mapsto 02, \ldots, z \mapsto 26$ as the (bijective) encoding. In this encoding scheme, the word *acme* maps to the code 01031305.

Definition 4.4.3. *An* encoding scheme *(sometimes just called a* code*) is a bijective map $f : S \to C$ from a source alphabet S to a set C of codewords constructed from some* code alphabet *A.*

Remark 4.4.4. The terms *alphabet* and *letters* for the set S may be misleading. The elements of S need not be traditional letters—they could just as well be Chinese characters, Egyptian hieroglyphics, English words, or entire English sentences.

A desirable property of an encoding scheme is that it be *uniquely decipherable*, meaning that we can uniquely reconstruct any source string from its encoded form. Codes that are not bijective cannot be uniquely deciphered, but unless we have a way of distinguishing the end of one encoded letter from the beginning of the next, even bijectivity is not necessarily enough, because two different source strings could still be mapped into the same codeword.

Unexample 4.4.5. Let $S = \{a, b, c, d, \ldots, z\}$ and $C = \{0, 1, 2, \ldots, 25\}$, with the mapping $a \mapsto 0$, $b \mapsto 1, \ldots, z \mapsto 25$. The word "RAT" maps to 17019, and "BHABJ" also maps to 17019; thus this encoding is not uniquely decipherable.

Another common expectation, stronger than unique decipherability, is that a code be *instantaneous*, so that a string can be decoded as soon as a codeword is received, rather than needing to wait for more of the message to be received before decoding it. Clearly if a code is not uniquely decipherable, it cannot be instantaneously decipherable.

Unexample 4.4.6. Let $S = \{x, y, z\}$ and $C_1 = \{0, 01, 011\}$, with the mapping $x \mapsto 0$, $y \mapsto 01$, and $z \mapsto 011$. This encoding scheme is uniquely decipherable, but it is not instantaneous because the string 000101101001 does not decipher to $xxyzyxy$ until you get to the end of the message and then work backward to pick off each codeword.

The next proposition provides a useful characterization of instantaneous codes.

Proposition 4.4.7. *An encoding scheme is instantaneous if and only if no codeword is a prefix of any other codeword.*

Proof. If any codeword $f(s)$ is a prefix of another codeword $f(s')$, then upon receiving $f(s)$ we cannot decode it as s until we receive enough additional code letters to recognize that $f(s')$ was not sent instead. Therefore instantaneous codes must have no codeword that is a prefix of any other codeword.

Conversely, if $f(s)$ is not a prefix of any other codeword, then subsequent codewords will not change how it is decoded, so if we can ever decode it, we must be able to decode it as soon as it is received. □

Remark 4.4.8. Because of the previous proposition, instantaneous codes are also often called *prefix-free codes* or just *prefix codes*.

Example 4.4.9. Let $S = \{x, y, z\}$ and $C_2 = \{00, 01, 11\}$, with the mapping $x \mapsto 00$, $y \mapsto 01$, and $z \mapsto 11$. This encoding scheme is uniquely decipherable and instantaneous. As an example, the code 00000111010001 deciphers to $xxyzyxy$, and you can start deciphering the code instantaneously, as soon as you encounter the first two digits.

The final property that we want from an encoding scheme is that it should be efficient, that is, we want to minimize the total number of symbols used when sending messages. This increases the amount of information that can be sent down a channel in a given amount of time or that can be stored in a given amount of memory. To make sense of the concept of efficiency, we need to know something about how frequently the various source letters occur.

Definition 4.4.10. *An* information scheme *is an ordered pair (S, P) where S is a source alphabet and $P : S \to [0, 1]$ is a probability distribution on S (that is, for each $s \in S$ the value of $P(s)$ is the relative frequency with which the letter $s \in S$ occurs or is expected to occur).*

Example 4.4.11. From the words listed in the main entries of the *Concise Oxford English Dictionary*, the letter e represents 11.1607% of all letters used, and a represents 8.4966%. So in a setting where we expect to encounter words sampled uniformly from those listed in the *Concise Oxford English Dictionary*, we could define an information scheme where S is the set of letters a through z, and the probability distribution P would have $P(e) = 0.111607$, $P(a) = 0.084966$, and so forth.

Of course, words are not usually sampled uniformly from a dictionary—in most settings words like *the* are much more common than words like *avuncular*. So the dictionary-based information scheme would be a poor model for most English text. A better model could be constructed by sampling text that is typical of what you expect to encode.

The standard measure of efficiency of an encoding scheme is the *average word length*.

Definition 4.4.12. *Given an information scheme* (S, P)*, the* average word length *of an encoding scheme* $f : S \to C$ *is*

$$\text{awl}(f) = \sum_{s \in S} \text{len}(f(s))P(s). \tag{4.4}$$

Example 4.4.13. Consider the source alphabet $S = \{a, b, c, d\}$, with frequencies given by the table below:

Symbol	Frequency
a	$2/17$
b	$2/17$
c	$8/17$
d	$5/17$

Given the encoding

$$a \mapsto 11, \quad b \mapsto 0, \quad c \mapsto 100, \quad d \mapsto 1010,$$

we compute the average codeword length as

$$\text{awl} = 2\left(\frac{2}{17}\right) + 1\left(\frac{2}{17}\right) + 3\left(\frac{8}{17}\right) + 4\left(\frac{5}{17}\right) = \frac{50}{17}.$$

Example 4.4.14. Given the encoding

$$a \mapsto 01010, \quad b \mapsto 00, \quad c \mapsto 10, \quad d \mapsto 11$$

for the alphabet and frequency chart in the previous example, we have

$$\text{awl} = 5\left(\frac{2}{17}\right) + 2\left(\frac{2}{17}\right) + 2\left(\frac{8}{17}\right) + 2\left(\frac{5}{17}\right) = \frac{40}{17}.$$

Thus, this scheme is more efficient (has a shorter average word length) than the one in the previous example, despite having one codeword that is much longer than any codeword in the other scheme.

4.4.2 Binary Codes and Trees

One of the most common code alphabets consists of just the two symbols 1 and 0. Any code formed with only two symbols is called *binary*. Any instantaneous binary

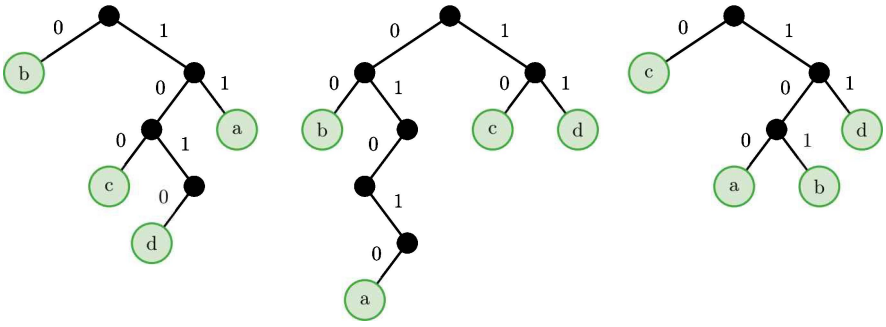


Figure 4.6. *Representation (as binary trees) of the instantaneous binary codes of Examples 4.4.13 and 4.4.14. The tree on the left corresponds to the first encoding scheme and the tree in the center to the second scheme. To decode a codeword, begin at the root of the tree and follow the edges determined by the codeword until reaching a leaf. A more efficient encoding scheme for this information scheme is given by the tree on the right. This last code has average word length of* $\frac{30}{17}$*. Note that c has the highest probability and the shortest codeword, whereas a and b have the lowest probabilities and the longest codewords.*

code can be represented by a binary tree, letting the leaves correspond to source letters and letting edges from the nonleaf nodes correspond to the various code letters of possible codewords. So the left and right edges out of the root correspond to the first letter of a codeword being 0 or 1, respectively. The edges out of the next node correspond to the next letter of the codeword, and so on. We show how this works with the codes of Examples 4.4.13 and 4.4.14 in Figure 4.6.

4.4.3 Huffman Encoding

Huffman encoding is an algorithm for creating an instantaneous code with minimal average codeword length. Like some of the other algorithms we have covered in this chapter, it is an example of dynamic programming, that is, building the desired solution out of the solution of smaller subproblems.

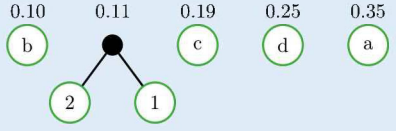
Codes of low average word length must use the shortest codewords possible, but there are limited numbers of codewords of each length. Therefore, the most common source letters must be encoded with the shortest codewords and the least common letters with longer codewords. Additionally, once a short codeword is assigned, it cannot be the prefix for any other codeword, further limiting the remaining available short codewords. In spite of this apparent complexity, the Huffman algorithm is surprisingly simple.

Begin by sorting the source alphabet in ascending order by frequency (or probability). These form the leaves of the tree. Create a parent of the two least frequent leaves and give the new node a weight equal to the sum of the frequencies of the children. Now repeat the process: reorder all parentless nodes by frequency, create a parent for the two least frequent nodes, and set the weight of the parent equal to the sum of its children. Continue until there is a unique root.

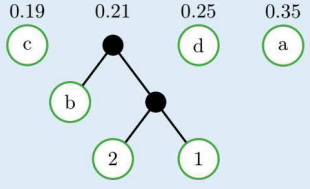
Example 4.4.15. Consider the source alphabet $S = \{a, b, c, d, 1, 2\}$. Assume the following frequencies:

Symbol	Frequency
a	0.35
b	0.10
c	0.19
d	0.25
1	0.06
2	0.05

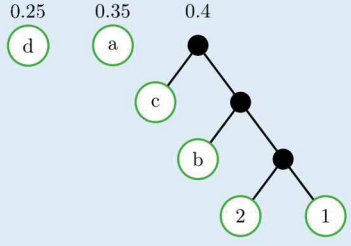
Form a leaf node for each letter of the source alphabet and sort the leaves by increasing weight.



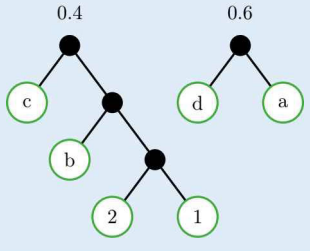
Give the first two leaves a parent with weight equal to the sum of its children. Then reorder by increasing weight.

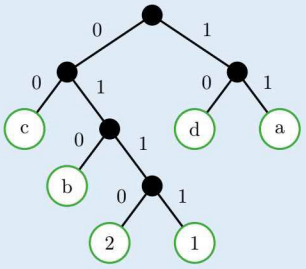


Again, create a parent for the two lowest-weight nodes and give it a weight equal to the sum of its children. Then resort by increasing weight.



Repeat this process for several more iterations.





Finally, only one parentless node remains (the root) and we have the completed tree.

The encoding scheme given by this tree is

$$a \mapsto 11, \quad b \mapsto 010, \quad c \mapsto 00, \quad d \mapsto 10, \quad 1 \mapsto 0111, \quad 2 \mapsto 0110.$$

The average word length of this code is

$$\text{awl} = 2(0.35) + 3(0.10) + 2(0.19) + 2(0.25) + 4(0.06) + 4(0.05) = 2.32.$$

Remark 4.4.16. At each stage of the algorithm, there is a choice about which child to put on the left and which to put on the right. The length of each codeword for each of the resulting codes is independent of these choices, and these could be considered equivalent codes, even if they appear very different.

Implementation

The Huffman algorithm can be implemented with a priority queue consisting of the parentless nodes, ordered by weight (frequency). Each step consists of two `pop_min` operations, creation of a new parent node (constant time) and an insertion into the queue. Thus the temporal complexity of the algorithm is $O(n \log n)$ for a source alphabet of n letters. But the temporal complexity of this algorithm is often not very important because the alphabets involved are typically small.

Once the code is constructed, information can be encoded rapidly by looking up the codeword for each letter in a table, and codewords can be decoded rapidly by traversing the code's binary tree.

Example 4.4.17. A very common way to store text is in the ASCII format, which uses 7 bits per letter. To store the string *MISSISSIPPI* in ASCII requires 77 bits. Huffman encoding for this word produces the code $S \mapsto 0$, $I \mapsto 10$, $P \mapsto 110$, and $M \mapsto 111$. With this code we can rewrite *MISSISSIPPI* as 11110001000101101010, which is only 21 bits. This is roughly 27% of the ASCII 77 bits.

4.4.4 *Huffman Encoding Is Optimal

In this section we prove the following theorem, showing that Huffman encoding is optimal.

Theorem 4.4.18. *The Huffman algorithm produces an instantaneous binary code with minimal average word length.*

The first steps in the proof of Theorem 4.4.18 are given in the following lemmata.

Lemma 4.4.19. *The average word length of an instantaneous binary code with a corresponding binary tree T is given by*

$$\operatorname{awl}(T) = \sum_{\ell} P(\ell) \operatorname{depth}(\ell),$$

where the sum runs over leaves ℓ of T, where $P(\ell)$ denotes the probability (frequency) of the leaf ℓ and where $\operatorname{depth}(\ell)$ *denotes the depth of ℓ in the tree.*

Proof. The leaves of T are exactly the letters of the alphabet of S. The depth of the leaf in the tree is the number of edges lying between the leaf and the root, and this is exactly the length of the corresponding codeword for that leaf. □

Lemma 4.4.20. *If T is a binary tree defining an encoding of average word length a, and if T' is a tree obtained from T by swapping the position of two leaves x and y, then*

$$\operatorname{awl}(T) - \operatorname{awl}(T') = (\operatorname{depth}_T(x) - \operatorname{depth}_T(y))(P(x) - P(y)). \tag{4.5}$$

Proof. First note that we have $\operatorname{depth}_{T'}(x) = \operatorname{depth}_T(y)$ and $\operatorname{depth}_{T'}(y) = \operatorname{depth}_T(x)$. Moreover, the sums $\operatorname{awl}(T)$ and $\operatorname{awl}(T')$ are identical except for the terms involving x and y. Thus we have

$$\begin{aligned}\operatorname{awl}(T) - \operatorname{awl}(T') &= \operatorname{depth}_T(x)P(x) + \operatorname{depth}_T(y)P(y) \\ &\quad - (\operatorname{depth}_{T'}(x)P(x) + \operatorname{depth}_{T'}(y)P(y)) \\ &= (\operatorname{depth}_T(x) - \operatorname{depth}_T(y))(P(x) - P(y)). \quad \square\end{aligned}$$

Lemma 4.4.21. *There exists an optimal tree such that two leaves with lowest frequency are siblings and are at greatest depth among all leaves in T.*

Proof. Given an optimal tree T, we create a new optimal tree having two leaves with lowest frequency that are siblings.

Denote the two leaves with the lowest frequency as a and b. If there are more than two such leaves, let a and b be two of lowest frequency with the greatest depth among all lowest-frequency leaves in T.

If a and b are already siblings, they must be at the greatest depth among all leaves in T because if not, swapping a with a leaf at greater depth would reduce the average word length of T, so T could not be optimal.

If a and b are not siblings, assume (without loss of generality) that $\operatorname{depth}_T(a) \geq \operatorname{depth}_T(b)$. This implies that $P(a) \leq P(b)$, because if not, swapping a with b in the tree would, by (4.5), produce a new tree with strictly smaller average word length, and T would not be optimal. Since a and b have the lowest frequencies, we may assume that $P(a) \leq P(x)$ for all $x \in T$.

We claim that $\text{depth}_T(a) \geq \text{depth}_T(x)$ for every leaf x in T. To see this, assume the contrary. We cannot have $P(a) = P(x)$ because a was assumed to have the greatest depth of all minimal-frequency leaves. Therefore, $P(a) < P(x)$. Let T' be the tree obtained by swapping a and x. By equation (4.5), we have $\text{awl}(T) > \text{awl}(T')$, which contradicts the optimality of T. Thus $\text{depth}_T(a) \geq \text{depth}_T(x)$ for every leaf x in T.

The leaf a cannot be an only child, because if it were, removing its parent from the tree (attaching a to its grandparent) would reduce the average word length of T, and T would not be optimal.

Therefore, a must have a sibling c, and by assumption $P(b) \leq P(c)$. Let T' be the tree created from T by swapping b and c, so that a and b are siblings. Equation (4.5) gives $\text{awl}(T) \geq \text{awl}(T')$, but the optimality of T guarantees that $\text{awl}(T) = \text{awl}(T')$, so T' is also optimal. □

Lemma 4.4.22. *Let T be a tree (not necessarily optimal) for the information scheme (S, P) with minimal-frequency leaves a and b that are siblings at the greatest depth. Let T' be the tree obtained by removing a and b from T and assigning the frequency $P(a) + P(b)$ to their parent c, so that T' is a tree for the information scheme (S', P'), where $S' = (S \cup \{c\}) \setminus \{a, b\}$, and $P'(x) = P(x)$ if $x \neq c$ and $P'(c) = P(a) + P(b)$. We have*

$$\text{awl}(T) = \text{awl}(T') + P(a) + P(b). \tag{4.6}$$

Proof. We have

$$\begin{aligned}
\text{awl}(T) &= \sum_{x \in S} \text{depth}_T(x) P(x) \\
&= \text{depth}_T(a)P(a) + \text{depth}_T(b)P(b) + \sum_{x \in S \setminus \{a,b\}} \text{depth}_T(x)P(x) \\
&= \text{depth}_T(a)(P(a) + P(b)) + \sum_{x \in S \setminus \{a,b\}} \text{depth}_T(x)P(x) \\
&= (\text{depth}_T(c) + 1)(P(a) + P(b)) + \sum_{x \in S' \setminus \{c\}} \text{depth}_T(x)P(x) \\
&= (P(a) + P(b)) + \sum_{x \in S'} \text{depth}_T(x)P(x) \\
&= (P(a) + P(b)) + \text{awl}(T'). \quad \square
\end{aligned}$$

Proof of Theorem 4.4.18. We induct on the size $n = |S|$ of the source alphabet S. The induction hypothesis is that the Huffman algorithm applied to any information scheme with source alphabet of size n produces an optimal code for that information scheme.

This is trivial in the case that $n = 1$. Assume now that the induction hypothesis holds for all information schemes with $|S| \leq k$. Let H be a tree constructed by the Huffman algorithm for an information scheme (S, P) with $|S| = k + 1$. By construction, H has two lowest-frequency leaves a and b that are siblings and are at the greatest depth.

By Lemma 4.4.21, there exists an optimal tree T for this information scheme such that two lowest-frequency leaves a' and b' are siblings and are at the greatest depth. Since $\{a, b\}$ and $\{a', b'\}$ are both lowest frequency in S, we may swap a with a' and b with b' in the optimal tree T without changing its average word length; therefore we may assume that the two lowest-frequency siblings at greatest depth in T are a and b.

Removing a and b from T gives a tree T' for the information scheme (S', P'), as in Lemma 4.4.22, so that

$$\text{awl}(T) = \text{awl}(T') + P(a) + P(b).$$

Let H' be the tree constructed (as in Lemma 4.4.22) by removing a and b from H, so that we have

$$\text{awl}(H) = \text{awl}(H') + P(a) + P(b).$$

Note that H' is constructed by the Huffman algorithm for the information scheme (S', P'), and so, by the induction hypothesis, H' must be optimal. Therefore $\text{awl}(H') \leq \text{awl}(T')$, and hence

$$\text{awl}(H) = \text{awl}(H') + P(a) + P(b) \leq \text{awl}(T') + P(a) + P(b) = \text{awl}(T).$$

Since T is minimal, H must also be minimal. □

Remark 4.4.23. Given a source alphabet and the frequency of each symbol, Huffman encoding provides a scheme that has the smallest average word length. However, there are compression and encoding concepts that go beyond Huffman encoding. For example, the optimality of Huffman encoding assumes that each symbol in the source alphabet is independently drawn from a distribution, when in reality, there's a strong statistical dependence between letters in a given language. For example, in English q is almost always followed by u, and taking advantage of these dependencies opens the door to further efficiencies.

4.5 Hard Problems

So far we have mostly focused on algorithms that run in polynomial time, that is, algorithms that, given an input of size n, have a worst-case temporal complexity of $O(n^k)$ for some fixed k. The class of all problems for which there exists a polynomial-time solution is denoted **P**. But, of course, not all problems can be solved in polynomial time. For some problems we can actually prove there is no polynomial-time algorithm that solves the problem. For other problems, no one knows whether a polynomial-time algorithm exists. In this section we briefly and informally discuss some of the various types of these problems and give some examples.

A large class of interesting and important problems are those for which there exists a polynomial-time algorithm to check any proposed solution to see whether it really is a solution. For such problems there might not be a polynomial-time algorithm to find a solution, but there is a polynomial-time algorithm to check whether a given candidate is a solution. The class of all such problems is denoted

NP, short for *nondeterministic polynomial* time.[22] Of course any problem in **P** also lies in **NP**, because if we can find all the solutions in polynomial time, then given any proposed solution, we can verify it by computing the actual solutions and comparing them to the proposed solution. Thus we have $\mathbf{P} \subset \mathbf{NP}$.

Example 4.5.1. One NP problem that is not known to be polynomial is the problem of finding a *Hamiltonian path* in a graph, that is, finding a path in a graph that visits every vertex exactly once.

Given a path, it is easy to verify whether the path does indeed visit every vertex exactly once. To do this traverse the path and keep track of how many times each vertex is visited. If any vertex is visited more than once, or if any vertex is not visited, reject the candidate path, and otherwise accept it. This verification algorithm takes only $O(n)$ steps in a graph with n vertices, so this problem is in **NP**. But there is currently no known algorithm for finding a Hamiltonian path in polynomial time.

A natural question to ask is whether there are any problems in **NP** that are not in **P**. At the time of this writing, the question is one of the Clay Math Institute's *millennium problems*, and an answer to the question, with a proof of correctness, will win you a prize of one million dollars.

As a first step to attack this problem, we consider the class of *NP-hard* problems. A problem X is NP-hard if any problem in **NP** has a polynomial-time reduction to X. That is to say, given any problem $Y \in \mathbf{NP}$ there is an algorithm f solving Y, where f consists of a polynomial number of standard computational steps and a polynomial number of calls for a solution of X. An NP-hard problem need not be in **NP**, but it is at least as hard as any NP problem.

Nota Bene 4.5.2. Beware that the abbreviation NP stands for *nondeterministic polynomial*. It does *not* mean "not polynomial." In fact, all polynomial problems are in **NP**, and the million-dollar question is whether there are *any* problems in **NP** that are not polynomial.

Nota Bene 4.5.3. The name *NP-hard* is misleading. NP-hard problems are not necessarily hard to solve for moderately sized inputs. The initial growth of the problem may not be very fast, so that for smaller inputs, solutions to an NP-hard problem can sometimes be very computable. The word *hard* in this context means, roughly speaking, at least as hard as any **NP** problem *once the size of the inputs is large enough.*

[22] More formally, **NP** is the class of all problems for which proposed solutions can be verified as correct in polynomial time by a *nondeterministic Turing machine*. Since the goal in this section is just to give a quick overview of the subject, we do not go into careful detail about Turing machines and the formal theory of computation related to these problems.

Conversely, polynomial problems are not necessarily easy. A problem that can be solved in $O(n^{1561600})$ is in **P**, but it is uncomputable in essentially all circumstances. An algorithm in $O(1)$ does not grow in complexity as the input size grows, but its constant complexity could be so large as to make the solution uncomputable for all inputs.

An especially interesting subset of the NP-hard problems is the set of *NP-complete* problems. These are problems in **NP** that are also NP-hard. So a proof that any one of these is in **P** would show that **P** = **NP**. Later in this section we discuss some problems that are known to be NP-complete (but, of course, they might also still be contained in **P**). Two possible Venn diagrams depicting these relations are shown in Figure 4.7.

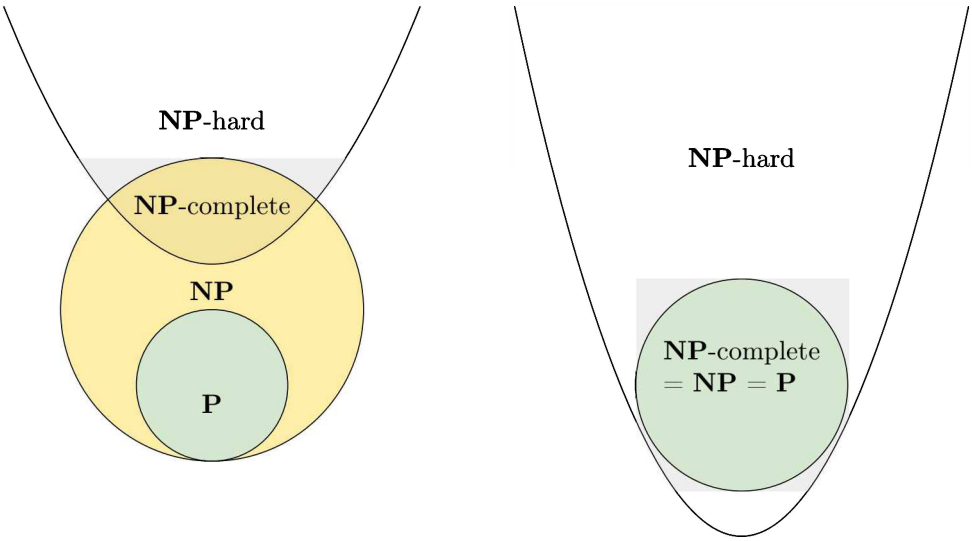


Figure 4.7. *Two possible Venn diagrams depicting the relations among* **P**, **NP**, **NP**-*complete, and* **NP**-*hard. The left diagram depicts the situation if* $\mathbf{P} \neq \mathbf{NP}$. *The right diagram depicts the situation if* $\mathbf{P} = \mathbf{NP}$.

4.5.1 Knapsack Problems

One class of combinatorial optimization problems that are known to be NP-hard is the class of *knapsack problems*. The basic idea is that we are given a collection of objects from which to load a knapsack. The goal is to choose the combination of objects to put in the knapsack that will have the greatest value, given various constraints (for example, the total weight of the objects can't be more than a certain amount, or the total volume can't be more than a certain amount, or we can only take at most a certain number of each item).

In one basic form of the knapsack problem, we are given a list of values $v_1, \ldots, v_n$ and (positive) weights $w_1, \ldots, w_n \in \mathbb{Q}$, and we must choose the numbers $x_1, \ldots, x_n \in$

$\mathbb{N}$ of each item so as to maximize $\sum_{i=1}^{n} v_i x_i$, subject to a budget of $\sum_{i=1}^{n} w_i x_i \leq W$. A common variant of this problem limits x_i to $\{0, 1\}$; in this variant we can take at most one of each item.

Example 4.5.4. Consider a situation with four items, whose weights (in kilograms) and values are listed below.

Item	Weight	Value
1	6	30
2	3	14
3	4	16
4	2	9

If I am not able to carry more than 10 kilos, which items should I take to maximize the total value? If I can only take at most one of each item, the optimal solution is to take item 1 and item 3, for a total value of 46. If multiples are allowed, I can do better by taking one of item 1 and two of item 4, for a total value of 48.

There are a number of ways to try to attack this problem. A naïve way is the exhaustive approach of checking every possible configuration. But this is much too costly if W is much larger than most of the weights w_i.

A better solution is a bottom-up dynamic programming approach; that is, find the solution for certain very small maximum weights, and then assemble the small solutions together into the solution for slightly larger weights, and so on until reaching the desired result.

If $m(w)$ is the maximum value achievable with total weight w, then we want to know $m(W)$. The lowest level of the problem is easy: $m(0) = 0$. The trick is to realize that if we put item i in the knapsack, then we still have weight $w - w_i$ left, and now the best value we can get is $v_i + m(w - w_i)$. Therefore, the maximum value $m(w)$ can be constructed from many smaller solutions as

$$m(w) = \max_{w_i \leq w} (v_i + m(w - w_i)), \tag{4.7}$$

which is another variant of Bellman's optimality principle.

Since there are at most a finite number of weights and they are all rational, we may assume that all the weights w_i are integer multiples of some basic value and reformulate the problem with only (positive) integer values of w_i and W. Now use (4.7) repeatedly to construct an $(n + 1) \times (W + 1)$ array M, where $M(i, w)$ is the maximum value achievable with a weight limit of w using only the first i items. The desired solution is $M(n, W)$.

Some initial values are immediate: $M(0, w) = 0$ for all w, and $M(i, 0) = 0$ for all i. Now compute from the bottom up; that is, starting with lower values of i and w and moving to higher ones, compute

$$M(i, w) = \max(M(i - 1, w), v_i + M(i, w - w_i)).$$

Since each w_i is positive, both of the two terms on the right are already known and can be used to compute $M(i, w)$. Iterating over all i from 0 to n and all w from 0 to

W finally reaches $M(n, W)$, which is the desired solution. Of course the algorithm described here does not give the actual choice of which items to put in the knapsack to achieve the maximum value, but that can easily be added to the algorithm by tracking the items added at each stage.

The temporal complexity of this algorithm is $O(nW)$. An important but subtle point is that complexity of an algorithm is usually measured in terms of the number of inputs, not the value represented by the input. Integers are usually entered as a sequence of binary digits, and there are $d = \log_2 W$ digits of W. Thus the complexity of this algorithm, in terms of the number of inputs (n and d), is $O(n2^d)$—exponential in d. But it is polynomial in terms of the value of the input W. Algorithms like this, that are polynomial in the value of their inputs rather than in the number of their inputs, are called *pseudopolynomial*.

It is known that the knapsack problem is NP-hard (measured in terms of the number of inputs). Of course, this does not mean the problem cannot ever be solved. Many specific instances of the problem can be solved in a reasonable amount of time, despite the fact that the best-known algorithm for the general solution takes exponential time.

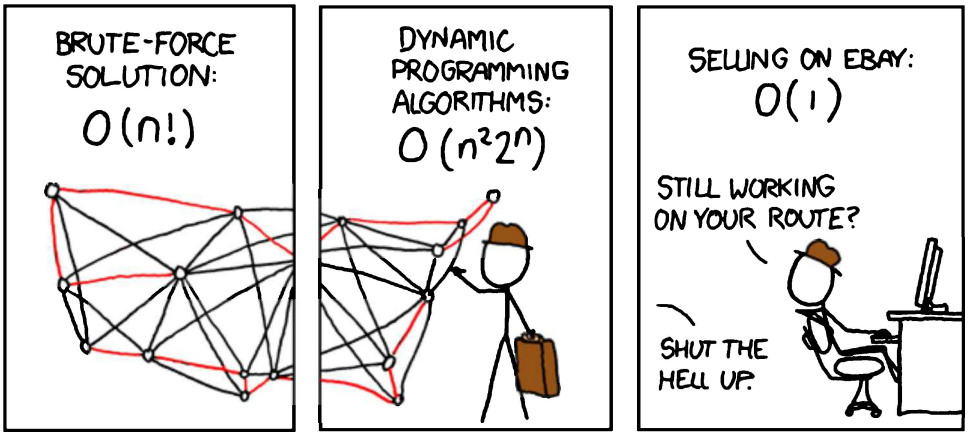


Figure 4.8. *A practical approach to the traveling salesman problem. Source: XKCD, Randall Munroe.* `http://xkcd.com/399/`

4.5.2 Traveling Salesman Problem

Given a list of cities and distances between them (a weighted undirected graph), consider the problem of finding the shortest path that visits all the cities exactly once and that starts and ends at the traveler's home city (such a path is sometimes called a *tour* or a *Hamiltonian cycle*). This problem was introduced in Example 4.0.2 and is known to be NP-complete.

As in the case of the knapsack problem, there is a dynamic optimization algorithm for the TSP that is better than the exhaustive approach. This is due to Held–Karp and (independently) Bellman. The algorithm is as follows: Let the home city be numbered 0 and let the remaining cities be numbered 1 through n. Denote the distance from city i to city j by $d_{i,j}$. For each $S \subset \{1, \ldots, n\}$ and for each $x \in S$

let $d(S, x)$ be the minimum distance it takes to travel from home to all of the cities in S, finishing at x.

The optimal path for the traveler has length

$$\min_{x \in \{1,\ldots,n\}} d(\{1,\ldots,n\}, x) + d_{0,x}.$$

Again we have an optimality principle that allows us to assemble solutions of simpler subproblems into solutions of larger problems.

$$d(S, x) = \min_{y \in S \setminus \{x\}} d(S \setminus \{x\}, y) + d_{y,x}.$$

As with other bottom-up dynamic programming algorithms, we start with the smallest (singleton) subsets: for $S = \{x\}$ we have

$$d(\{x\}, x) = d_{0,x},$$

and we compute $d(S, x)$ for increasing S and all x. Given any S, if $d(S', y)$ has already been computed for each $S' \subset S$, then for any $x \in S$ computing $d(S, x)$ is $O(|S|)$. Thus, for each S the complexity of computing $d(S, x)$ for all $x \in S$ is $O(|S|^2)$, so the overall complexity is

$$O\left(\sum_S |S|^2\right) = O\left(\sum_{k=1}^{n} \sum_{|S|=k} k^2\right) = O\left(\sum_{k=1}^{n} \binom{n}{k} k^2\right).$$

We can compute the sum $O(\sum_{k=1}^{n} \binom{n}{k} k^2) = O(\sum_{k=1}^{n} \binom{n}{k} k(k-1))$ as follows: Let

$$f(t) = \sum_{k=2}^{n-2} \binom{n}{k} k(k-1) t^{k-2} = \frac{d^2}{dt^2} \sum_{k=0}^{n} \binom{n}{k} t^k = n(n-1)(1+t)^{n-2}.$$

This gives

$$\begin{aligned} \sum_{k=1}^{n} \binom{n}{k} k(k-1) &= f(1) + \binom{n}{n-1}(n-1)(n-2) + \binom{n}{n} n(n-1) \\ &= n(n-1)2^{n-2} + n(n-1)^2 \\ &= O(2^n n^2). \end{aligned}$$

So the Held–Karp algorithm has complexity $O(2^n n^2)$. Compare this to the naïve algorithm with complexity $O(n!)$, which by Stirling's approximation is $O\left(\left(\frac{n}{e}\right)^n \sqrt{n}\right)$.

4.5.3 Better Approaches

As a general rule, if you are trying to find the exact solution to an NP-hard problem with large inputs, then you are doing something wrong. In such cases you probably need to stop looking for an algorithm to always find the exact solution and instead think about some alternatives. Here are three options to consider:

(i) Look for a heuristic to solve a reasonable fraction of the most common cases efficiently.

(ii) Solve the problem approximately instead of exactly.

(iii) Solve a similar, but easier, problem.

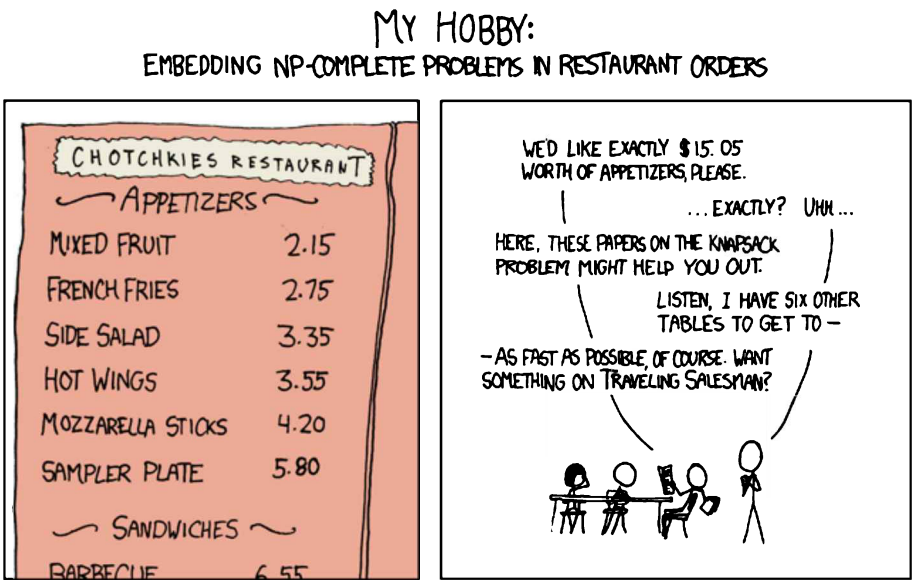


Figure 4.9. *A real-world application of the change-making problem? Source: XKCD, Randall Munroe.* `http://xkcd.com/287/`

Heuristics

Heuristics are simple guidelines or strategies that can be used to try to get a good solution, but without guarantees of the quality of the solution. An obvious and common heuristic is the greedy strategy of choosing the step that looks optimal now.

A greedy approach to the knapsack problem is to always choose, at each stage, to put as much as possible of the most valuable item into your knapsack. This algorithm seems intuitive to many people, and it sometimes works. But sometimes it also gives a very bad solution, and in some cases it can give the worst possible solution. A better greedy algorithm is to sort the items by value per unit weight v_i/w_i and add as much as possible of the highest of these. This often gives good results but still does very poorly in some cases (see Exercise 4.23).

A common greedy approach to the TSP is to form a path by choosing, at each stage, to visit the nearest unvisited city. This is called the *nearest neighbor* heuristic, and it has temporal complexity $O(n^2)$. It normally gives a path that is no more than 25% longer than the best solution, but it is possible to construct cases where the nearest neighbor algorithm gives the worst of all possible paths (see Exercise 4.24).

A different greedy approach to the TSP is to collect the shortest edges without worrying about whether they form a connected path. This algorithm builds a collection of edges that will eventually form a path, and at each stage it sorts the remaining edges by length and adds to the collection the shortest one that does not make a short cycle and does not make any vertex have more than two edges. This algorithm is called *the greedy algorithm* for TSP, and it has temporal complexity

$O(n^2 \log n)$. It typically constructs paths that are within 15% to 20% of optimal, but again, there are some cases where it does very poorly.

There are several other heuristics for the TSP. Some begin with an optimal tour of fewer cities and try to insert additional cities into the existing tour in an optimal manner, while others start with an existing, suboptimal tour of all the cities and try to improve it incrementally. Some of these heuristic improvement algorithms have run times of $O(n^2)$ or better and tend to give answers no worse than 5% longer than optimal, but in some cases they still give answers that are much worse.

Approximation Algorithms

In many circumstances we don't need the absolute best solution—a solution that is near best may be good enough. For many NP-hard problems, allowing a little bit of error lets us rapidly construct solutions that are guaranteed to be within a certain percentage of the optimal solution. Of course, there is usually a trade-off between speed and accuracy—the more accurate the answer must be, the more time it will take to compute.

Here we give one simple example of an approximation algorithm for the TSP in the case that the cities all lie in the plane and that the distances involved satisfy the triangle inequality—an edge between two cities is never longer than the length of any other path connecting them. In this case, we can approximate the optimal solution by constructing an MST. Removing one edge from any tour will produce a spanning tree, so the MST must have weight no more than the length of the shortest tour.

Since the MST lies in the plane, we can put an ordering on the vertices by imagining an ant starting at the home city and walking along the outside of the tree until returning home again. The total distance the ant walks is twice the weight of the MST, since the ant traverses every edge exactly twice.

Listing the vertices in the order they are first encountered on the ant's walk will give a tour. For each edge $e = (v, v')$ of the tour, the ant will have to traverse some edges of the tree to get from v to v', and because the graph lies in the plane, the triangle inequality holds and e must be no longer than the sum of the lengths of the edges walked by the ant. Summing over all the edges in the tour, the length of the tour is no more than the total length of the ant's walk, which is no more than twice the total weight of the tree, which is no more than twice the optimal tour length.

This shows that any tour produced by an MST is no more than twice the optimal length. But Prim's algorithm produces an MST in $O(n^2 \log n)$, so if we can accept a tour guaranteed to be no worse than twice optimal, this algorithm is fairly efficient.

For the TSP there are many other approximation algorithms. One of the best known is Cristofides' algorithm that runs in $O(n^{2.2})$ time and is guaranteed to give an answer no worse than 1.5 times optimal.

For the knapsack problem, there is an approximation algorithm that for any choice of $\varepsilon > 0$ gives an answer that is no worse than $(1 - \varepsilon)$ times the maximum value and does so in better than $O(n + 1/\varepsilon^3)$ time. This is an example of what is called a *fully polynomial-time approximation scheme* (FPTAS). This shows very clearly the trade-off between accuracy and temporal complexity: the more accurate an answer must be (the smaller ε), the greater the run time.

A Similar, but Easier, Problem

Finally, it is important to remember that the mathematical problems we solve in applications are mathematical models of real-world problems. To formulate a model, we typically need to make several assumptions and simplifications. Sometimes these assumptions and simplifications make the problems easier to solve, but in some cases, what we think of as a simplification actually makes the problem harder.

Some types of *scheduling problems* provide an example of this. For these problems one is given a collection of jobs of various (fixed) completion time and a collection of machines with various properties. The problem is to decide which jobs to schedule on which machines in order to minimize total time to completion of all the jobs (subject to various additional requirements). Many of these scheduling problems are known to be NP-hard.

But these simple models are also not completely realistic—for example, slowdowns might occur for many reasons, causing random changes in the production speeds of the machines. Remarkably, in some cases, a more complicated *stochastic*[23] model that accounts for this randomness actually yields a version of the scheduling problem that is more tractable than the "simpler" deterministic model [Leu04, Section 38.4.1].

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

4.1. Code up the naïve, the memoized, and the bottom-up dynamic programming algorithms for computing the Fibonacci number $F(n)$ for $n \in \mathbb{N}$. Time all three methods and compare their performance for $n \in \{1, 2, \ldots, 40\}$. Experiment to find the largest value of n for which each of the three algorithms gives an answer in less than one minute.

4.2. Code up both the naïve recursion and the bottom-up dynamic programming algorithm to compute the optimal number $n(v)$ of coins in the change-making problem for v cents ($v \in \mathbb{N}$) and an arbitrary coinage system C (a set of coin

[23] The word *stochastic* here means that the model involves some element of randomness.

values, in cents). Adapt your code to also return the optimal configuration of coins summing up to v. Time both methods and compare their performance on each of the values $v \in \{1, 2, \ldots, 1999\}$ for the current U.S. coinage system, where $C = \{1, 5, 10, 25, 50, 100\}$. It is acceptable to stop your code for values that take more than one minute to run.

4.3. Code up a greedy version of the change-making problem. For the U.S. coinage system verify that the greedy solution is the same as the optimal solution for all values $v \in \{1, 2, \ldots, 1999\}$. Time your code for the greedy solution on those values and compare your answers with those in Exercise 4.2.

4.4. Consider a coinage system with the following denominations:

$$C = \{1, 5, 7, 10, 20, 25, 40, 50, 100\}.$$

Provide some examples to the change-making problem in this coinage system where the greedy solution is not the optimal solution.

4.5.* Consider a sequence $(x_n)_{n=0}^{\infty}$ defined by the recurrence relation $x_0 = x_1 = 2$ and for $n > 1$,

$$x_n = \sum_{k=0}^{n-2} x_k x_{k+1}.$$

We are interested in computing x_n, given n.

(i) Show that implementing this recursion directly, as written, uses more than 2^n arithmetic operations to compute x_n when $n \geq 6$.

(ii) Use memoization to give an algorithm that uses only $O(n^2)$ arithmetic operations to compute x_n (and prove the bound of $O(n^2)$).

(iii) Use bottom-up dynamic programming to give an algorithm that only uses $O(n)$ arithmetic operations to compute x_n (and prove the bound of $O(n)$).

4.6.* Given two sequences $\mathbf{x} = x_1, x_2, \ldots, x_n$ and $\mathbf{y} = y_1, y_2, \ldots, y_m$, let $s(\mathbf{x}, \mathbf{y})$ be the length of the longest possible sequence that is a subsequence of both $\mathbf{x}$ and $\mathbf{y}$. So if $\mathbf{x} = 6, 0, 1, 2, 3, 4, 5$ and $\mathbf{y} = 5, 6, 2, 3$, then $s(\mathbf{x}, \mathbf{y}) = 3$, since $6, 2, 3$ is the longest sequence appearing as a subsequence in both.

(i) Let $T\mathbf{x} = x_1, \ldots, x_{n-1}$ and $T\mathbf{y} = y_1, \ldots, y_{m-1}$. Show that if $x_n = y_m$, then $s(\mathbf{x}, \mathbf{y}) = s(T\mathbf{x}, T\mathbf{y}) + 1$.

(ii) Find a recursive formula for $s(\mathbf{x}, \mathbf{y})$ if $x_n \neq y_m$.

(iii) Provide an algorithm for computing $s(\mathbf{x}, \mathbf{y})$ with temporal complexity $O(mn)$ (and prove the bound of $O(mn)$).

4.7. In the graph in Figure 4.10, beginning at the node labeled 50:

(i) Show the sequence of nodes visited when using BFS to find the node labeled 90. Assume that neighbors of a node are always added to the queue in numerical order (smallest to largest).

(ii) Repeat the previous problem, but with the neighbors of a node added to the queue in reverse numerical order (largest to smallest).

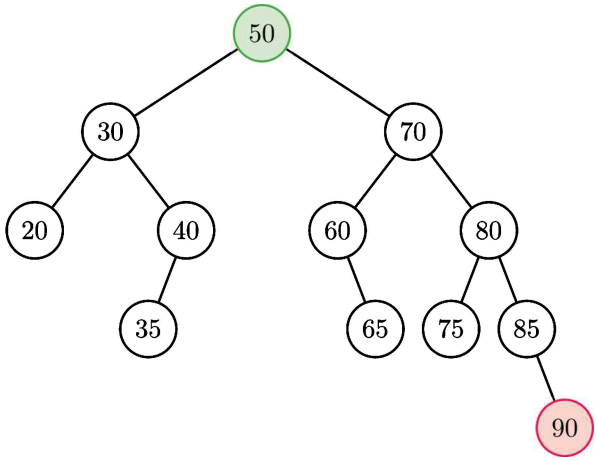


Figure 4.10. *Graph for BFS and DFS of Exercise 4.7.*

(iii) Show the sequence of nodes visited when using DFS to find the node labeled 90. Assume that neighbors of a node are always added to the stack in numerical order (smallest to largest).

(iv) Repeat the previous problem, but with the neighbors of a node added to the stack in reverse numerical order (largest to smallest).

4.8. Given an undirected graph, which graph search method (BFS or DFS) would be most useful for finding a cycle in the graph? Describe, in detail, an algorithm for finding a cycle in any undirected graph, and explain why your algorithm is correct (meaning that it is guaranteed to find a cycle if one exists and will not give a false answer).

4.9. For the graph in Figure 4.11, describe what happens (and the state of all the various variables, queues, etc.) at each stage of Dijkstra's algorithm, beginning at node A. Do not stop until every node has been processed; that is, find the minimum distance from A to *every* node in the graph.

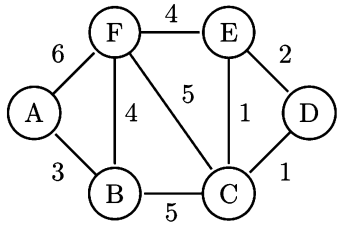


Figure 4.11. *Graph for Exercise 4.13 and Exercise 4.9.*

4.10. For any weighted, undirected graph G with nonnegative edge weights, and for any vertex $v \in G$, let $\delta(s, v)$ denote the (actual) minimum distance from the source s to v.
Prove that if Dijkstra's algorithm processes node u before it processes node v, then $\delta(s, u) \leq \delta(s, v)$.

4.11. Give a careful justification for why Remark 4.2.4 is true.

4.12. Let G be a connected weighted undirected graph. Prove that if all the weights are distinct, then the MST is unique.

4.13. For the graph in Figure 4.11, describe what happens (and the state of all the various variables, tables, dictionaries, etc.) at each stage of Prim's algorithm, beginning at node A.

4.14. In Prim's algorithm, for any vertex v, let $d(v)$ denote the distance from v to the current minimum tree. What is the maximum number of times that $d(v)$ will be updated? What is the minimum number of times it will be updated?

4.15. Let G be a connected weighted undirected graph with at least one cycle. Prove that if e is an edge in the cycle of strictly larger weight than the other edges in the cycle, then it cannot be contained in the MST of G.

4.16. Adapt Prim's algorithm to find the MST.

4.17. Prove that if a coding scheme $f : S \to C$ is bijective and instantaneous, then it is uniquely decipherable.

4.18. Give an example of a uniquely decipherable code that is not instantaneous.

4.19. Compute a (binary) Huffman code for the source alphabet $\{a, b, c, d, e\}$ with the probability distribution $P(a) = 0.07$, $P(b) = 0.05$, $P(c) = 0.70$, $P(d) = 0.08$, $P(e) = 0.10$.

4.20. Use Huffman encoding to compress the string "*The harder I work, the luckier I get.*" How many total bits are required to encode this string using the Huffman code? Compare this to the number of bits required to store the string in ASCII. (Remember to encode the spaces and punctuation.)

4.21. Any code can be transformed into another obviously equivalent code by permuting the letters of the code alphabet. For example, trading 0 and 1 in any binary code will give another code that is essentially equivalent to the first. But even accounting for these permutations, Huffman codes are not necessarily uniquely determined. Give an example of a source alphabet and probability distribution for which there are two different (binary) Huffman trees/codes such that neither one can be obtained from the other by permuting the code letters 0 and 1.

4.22. For each of the following, explain whether proving the result would qualify for the one-million-dollar millennium prize for P versus NP. Justify why or why not.

(i) There is at least one NP-hard problem that is P.

(ii) There is at least one NP-hard problem that is not P.

(iii) There exists a problem that is NP but is neither NP-complete nor P.

(iv) Every problem that is NP is either NP-complete or P.

4.23. Give an example of a $\{0, 1\}$-knapsack problem where the include-the-item-with-the-most-value-per-weight heuristic gives the worst solution.

4.24. Give an example of a graph where the nearest neighbor heuristic (always choosing at each stage to visit the nearest unvisited city) gives the worst solution to the TSP.

4.25. Prove carefully that the naïve, exhaustive approach to the TSP has temporal complexity at least $O((n-1)!)$, where n is the total number of cities. Explain how to do this exhaustive search with *spatial* complexity of only $O(n^2)$. Hint: Think about DFS.

4.26. Implement the dynamic programming algorithm for the $\{0,1\}$-knapsack problem (where no multiples are allowed). Your code should accept as input a maximum weight `W` and a list `Items` of tuples (weight,value). So, for example, the list

$$\texttt{Items} = [(20, 0.5), (100, 1)]$$

would correspond to a collection consisting of item 0 of weight 20 and value 0.50, and item 1 of weight 100 and value 1.00.
Your code should return the maximum value that can be carried in the knapsack for the given `Items` and weight `W`.

4.27. Modify your code in the previous problem to also return a list of which items should be included to achieve the maximum value.

Notes

For more about implementing Dijkstra's and Prim's algorithms with a Fibonacci heap, see [FT87] or [CLRS01]. Prim's algorithm is originally due to Vojtěch Jarník in 1930 [Jar30] and was rediscovered by Prim in 1957 and by Dijkstra in 1959. As a result, some people call Prim's algorithm the Jarník–Prim algorithm.

Our statistics on word frequency in the *Concise Oxford English Dictionary* are taken from [Mat15].

For more details on computability and formal Turing machines, some good references include [CLRS01, KT05], and [Mac18]. For a fun example of an essentially uncomputable polynomial-time algorithm, [DDM+14] provides a polynomial-time algorithm for a picture hanging, but the best-known bound on the size is $O(n^{1561600})$.

Among computer scientists, it is generally believed that $\mathbf{P} \neq \mathbf{NP}$. The main justification for this belief is that lots of smart people have, for a long time, tried and failed to find a polynomial-time algorithm for an NP-complete problem. A problem of similar significance for economists is the efficient market hypothesis (EMH), which is a question of whether markets are (weakly) efficient, meaning that future prices cannot be predicted from analyzing prices in the past. The great majority of economists believe the EMH. So it is interesting to note that, according to Maymin [May11], the EMH holds if and only if $\mathbf{P} = \mathbf{NP}$. This means either Maymin is wrong, or most economists are wrong about the EMH, or most computer scientists are wrong about $\mathbf{P} \neq \mathbf{NP}$.

For more about the TSP, see [GYZ]. General references on approximation algorithms include [DKH12, WS11]. For approximation algorithms specific to the knapsack problem, see [JK15].

Figure 4.7 is modeled after a diagram by Behnam Esfahbod [Wik15], and Exercises 4.5 and 4.6 are modified versions of problems in [Pru17].

5 Probability

Never tell me the odds.
—Han Solo

Probability is the mathematics of uncertainty, and it plays a key role in modeling the world around us. We focus more on modeling with these tools in Volume 3 (where we also treat probability in more depth), but probability is also central to the study of algorithms and optimization for several reasons. Probability is essential to analyzing algorithmic complexity, and it is useful in helping us construct better algorithms. It is also a source of many of the most important optimization problems.

Although probability has been used for centuries in areas such as gambling and insurance, a rigorous and satisfactory development did not exist until after *measure theory* was developed. In fact, it wasn't until the 1930s that a rigorous theory was developed by the Russian mathematician Andrei Kolmogorov.

Although we postpone many of the details of probability theory to Volume 3, in this chapter we provide a lightweight version that allows us to understand and use basic principles of probability in a wide variety of settings. In the first five sections of the chapter we discuss discrete probability and discrete random variables, where the space of possible outcomes is countable. In subsequent sections we discuss how to generalize this to the continuous case, where the possible outcomes span a continuum.

In the subsequent chapter we discuss what is arguably the most important theorem in probability and statistics—the *central limit theorem*. The central limit theorem is the key behind our ability to draw inferences and is truly central to probability and statistics.

5.1 Probability Theory

Probability theory begins with a nonempty *sample space* Ω, consisting of all possible outcomes of an experiment. In this section we begin a discussion of *discrete probability*, which requires the sample space to be countable. We also discuss *continuous probability*, corresponding to more general sets, later in the chapter.

Example 5.1.1. If a coin is flipped three times, the sample space, or set of all possible outcomes, is

$$\Omega = \{\text{HHH}, \text{HHT}, \text{HTH}, \text{HTT}, \text{THH}, \text{THT}, \text{TTH}, \text{TTT}\}.$$

The event E = "heads occurs exactly twice" is represented by the subset $E = \{\text{HHT}, \text{HTH}, \text{THH}\}$.

Definition 5.1.2. *The* power set *of a set S is the set of all subsets of S. We often denote it by the symbol 2^S.*

Remark 5.1.3. In the discrete setting, any subset $E \subset \Omega$ is called an *event*; that is, the collection of all events is 2^Ω. In the continuous setting, most subsets must be excluded from consideration, because allowing every possible subset to be an event constrains the theory so much as to make it uninteresting. In either case, we denote the collection of all events by $\mathscr{F}$. In the discrete setting we have $\mathscr{F} = 2^\Omega$, but in the more general setting we only have $\mathscr{F} \subset 2^\Omega$.

5.1.1 Axioms of Discrete Probability

Example 5.1.4. If $S = \{a, b\}$, then the power set of S is $\{\emptyset, \{a\}, \{b\}, S\}$. If S is finite, of order $|S|$, the power set of S always has $2^{|S|}$ elements in it. This motivates the notation 2^S.

Given two events A and B in Ω, the event E that both A and B occur is the intersection $E = A \cap B$. Similarly, the event F that at least one of A or B occurs is the union $F = A \cup B$.

The probability of an event is a value between 0 and 1 (inclusive), where we think of probability as some measure of plausibility that the event will occur: a probability of 1 means that the event is practically guaranteed to occur and a probability of 0 means it is practically guaranteed not to occur.

Definition 5.1.5. *If a collection of events $\{E_i\}_{i \in I}$ is pairwise disjoint, meaning that $E_i \cap E_j = \emptyset$ whenever $i \neq j$, then we say the sets are* mutually exclusive. *If the union $\bigcup_{i \in I} E_i$ of all the events is the entire sample space Ω, then we say that the events E_i are* collectively exhaustive.

Remark 5.1.6. If the subsets E_i are all nonempty, then saying they are mutually exclusive and collectively exhaustive is another way of saying that they form a partition of Ω.

Definition 5.1.7. *Consider a countable sample space Ω, and let $\mathscr{F} = 2^\Omega$. A function $P : \mathscr{F} \to [0, 1]$ is called a* discrete probability measure *whenever the following conditions hold:*

(i) $P(\Omega) = 1$.

(ii) *Additivity: If* $\{E_i\}_{i\in I} \subset \mathscr{F}$ *is a collection of mutually exclusive events, indexed by a countable set* I*, then*

$$P\left(\bigcup_{i\in I} E_i\right) = \sum_{i\in I} P(E_i). \tag{5.1}$$

In this case, the triple $(\Omega, \mathscr{F}, P)$ *is called a* discrete probability space. *We say that an event* $E \in \mathscr{F}$ occurs with probability $P(E)$. *In the case that* $E = \{\omega\}$ *is a singleton set, it is common to write* $P(\omega)$ *instead of* $P(\{\omega\})$.

The two conditions on a probability measure should coincide with your intuition about how probability behaves. First, the event Ω is the set of all possible outcomes, and since some outcome must occur, the probability of Ω should be 1. Second, if A and B are mutually exclusive events, the probability $P(A \cup B)$ of at least one of A or B occurring should be the same as the probability of A plus the probability of B.

Example 5.1.8. Consider the space of three coin flips from Example 5.1.1, together with its power set $\mathscr{F} = 2^\Omega$. A common probability measure to use in this setting is $P(\omega) = \frac{1}{8}$ for every $\omega \in \Omega$. That means $P(E) = \frac{1}{8}|E|$ for every event $E \in \mathscr{F}$.

Since Ω consists of eight elements, we have $P(\Omega) = 1$. It is easy to see that for a collection of mutually exclusive events $\{E_i\}_{i\in I}$ we have

$$P\left(\bigcup_{i\in I} E_i\right) = \frac{1}{8}\left|\bigcup_{i\in I} E_i\right| = \frac{1}{8}\sum_{i\in I}|E_i| = \sum_{i\in I}\frac{1}{8}|E_i| = \sum_{i\in I} P(E_i).$$

So P really is a probability measure on $(\Omega, \mathscr{F})$.

With this probability measure, we have $P(\text{HHT}) = P(\text{TTT}) = \frac{1}{8}$, and $P(\text{“heads occurs exactly twice”}) = \frac{3}{8}$.

Remark 5.1.9. It is common to write $P(E, F)$ instead of $P(E \cap F)$ to indicate the probability that both E and F occur.

The following proposition gives some simple but useful tools for computing probabilities.

Proposition 5.1.10. *Let* $(\Omega, \mathscr{F}, P)$ *be a discrete probability space. If* $E, F \in \mathscr{F}$ *and* $E^c = \Omega \setminus E$*, then*

(i) $P(E^c) = 1 - P(E)$*, and, in particular,* $P(\emptyset) = 0$*;*

(ii) $E \subset F$ *implies* $P(E) \leq P(F)$*; and*

(iii) $P(E \cup F) = P(E) + P(F) - P(E \cap F)$.

Proof.

(i) Since E and E^c are disjoint and $E \cup E^c = \Omega$, we have

$$1 = P(\Omega) = P(E \cup E^c) = P(E) + P(E^c),$$

and thus $P(E^c) = 1 - P(E)$. Since $\Omega^c = \emptyset$, we have $P(\emptyset) = 0$.

(ii) The events $F \cap E$ and $F \cap E^c$ are disjoint and have F as their union, that is, $(F \cap E) \cup (F \cap E^c) = F$. If $E \subset F$, then $E \cap F = E$, and thus

$$P(E) + P(F \cap E^c) = P(F \cap E) + P(F \cap E^c) = P(F).$$

Since $P(F \cap E^c) \geq 0$ it follows that $P(E) \leq P(F)$.

(iii) We have

$$P(F \cap E) + P(F \cap E^c) = P(F) \quad \text{and} \quad P(E \cup F) = P(E) + P(F \cap E^c).$$

Combining these equations yields $P(E \cup F) = P(E) + P(F) - P(E \cap F)$. □

Example 5.1.11. During a particularly bad flu season, the probability you will have a sore throat is 0.15. The probability you will have a headache is 0.10. If the probability of neither is 0.80, what's the probability that you will have both a sore throat and a headache?

To answer this, let E be the event "you have a sore throat" and F be the event "you have a headache." We are given that $P(E) = 0.15$, $P(F) = 0.10$, and $P((E \cup F)^c) = 0.80$. This implies that $P(E \cup F) = 0.20$. Thus, we have that $P(E \cap F) = P(E) + P(F) - P(E \cup F) = 0.05$.

5.1.2 Equally Likely Outcomes

Definition 5.1.12. *Assume Ω is finite and $(\Omega, \mathcal{F}, P)$ is a discrete probability space. We say that all outcomes of Ω are* equally likely *if $P(\omega) = 1/|\Omega|$ for every $\omega \in \Omega$.*

Example 5.1.13. In the probability space of Example 5.1.8 all outcomes are equally likely.

In the case of equally likely outcomes, probability problems become counting problems.

Example 5.1.14. Two fair (six-sided) dice are rolled and their sum is noted. If the event E is "the sum of the dice is 5," what is $P(E)$? The sample space Ω for this problem is the set of all possible pairs of rolls.

$$\Omega = \left\{ \begin{array}{cccc} (1,1), & (1,2), & \ldots, & (1,6) \\ (2,1), & (2,2), & \ldots, & (2,6) \\ \vdots & & & \vdots \\ (6,1), & (6,2), & \ldots, & (6,6) \end{array} \right\}.$$

The word *fair* implies that all of these outcomes (but not all sums) are equally likely. Thus, we can answer the question by counting the number of outcomes where the sum is 5, that is, counting the elements of

$$E = \{(1,4), (2,3), (3,2), (4,1)\},$$

and comparing this to the total number of possible outcomes, which is 36. Therefore, we have $P(E) = \frac{4}{36} = \frac{1}{9}$. Similarly, if the event F is "the sum of the dice is 7," then

$$F = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\},$$

so $P(F) = \frac{6}{36} = \frac{1}{6}$.

Example 5.1.15. A cooler has three Diet Cokes and seven regular Cokes. If you randomly draw three cans from the cooler, and all the cans are equally likely to be drawn, what is the probability that you draw at least one of each type?

We solve this by realizing that there are two ways you can have one of each. Either you get one diet and two regular or you get two diet and one regular. There are $C(3,1)$ ways to choose one diet and $C(7,2)$ ways to get two regular, so $C(3,1) \cdot C(7,2)$ ways to choose one diet and two regular. Similarly, there are $C(3,2) \cdot C(7,1)$ ways to choose two diet and one regular. Summing these gives the total number of ways to get at least one of each. To get the probability of this event, divide by the total number $C(10,3)$ of ways to draw three cans out of a cooler of 10 cans. Thus, the probability is given by

$$\frac{C(3,1) \cdot C(7,2) + C(3,2) \cdot C(7,1)}{C(10,3)} = \frac{84}{120} = \frac{7}{10}.$$

Example 5.1.16. The cards in a standard 52-card deck of playing cards come in four different suits (clubs, diamonds, hearts, and spades) and 13 different ranks (2, 3, ..., 10, jack, queen, king, ace). What's the probability that five cards, randomly selected from such a deck, will form a full house, that is, a three-of-a-kind and a pair? We solve this by first determining which number is the three-of-a-kind and which is the pair. Notice that while order within the three-of-a-kind or pair does not matter, order of which is the three-of-a-kind and which is the pair does matter. Thus we have $13 \cdot 12$ ordered pairs, where the first number is the rank of the three-of-a-kind and the second number is the rank of the pair. We then multiply by $C(4,3)$ and $C(4,2)$ to account for the number of ways we can choose the three-of-a-kind and pair, from their respective ranks. We find that the probability is given by

$$\frac{13 \cdot 12 \cdot C(4,3) \cdot C(4,2)}{C(52,5)} \approx 0.0014.$$

Example 5.1.17 (The Birthday Problem). In a given group of people, how likely is it that two or more people share the same birthday? To simplify, we assume there are exactly 365 days in a year (that is, no leap years) and that a given person is equally likely to have her or his birthday on any of these 365 days.

In this situation, it is simpler to solve the complementary problem instead, namely, what is the probability that all the birthdays are distinct? In a group of k people, the total number of ways that birthdays might occur is 365^k. If $k > 365$, at least two people must have the same birthday, by the pigeonhole principle,[a] but if $k \leq 365$, the number of ways that all the birthdays could be distinct is $\frac{365!}{(365-k)!}$. Thus, the probability $Q(k)$ that a randomly selected group of k people will have all distinct birthdays is $Q(k) = \frac{365!}{(365-k)!365^k}$. The probability that at least two people share a birthday is

$$P(k) = 1 - Q(k) = 1 - \frac{365!}{(365-k)!365^k}. \tag{5.2}$$

The probability $P(k)$ is plotted in Figure 5.1. A careful look at the plot reveals the surprising result that $P(k) > 50\%$ whenever $k \geq 23$. Thus, if birthdays are uniformly distributed among the days of the year, then in a room of only 23 randomly selected people, there is a greater than 50% probability that at least two of them will share a birthday.

[a] No matter how you put $n+1$ pigeons into n pigeonholes, at least one pigeonhole has at least two pigeons.

Remark 5.1.18. The number 365 is large enough that Stirling's approximation is reasonably accurate for 365!, and if k is not too big, then Stirling also gives a reasonable approximation for $(365-k)!$. Using Stirling for both of these in the

previous example gives

$$Q(k) \approx \frac{365^{365+\frac{1}{2}}}{365^k e^k (365-k)^{365-k+\frac{1}{2}}} = e^{-k}\left(1-\frac{k}{365}\right)^{k-365-\frac{1}{2}}$$

or

$$P(k) \approx 1 - e^{-k}\left(1-\frac{k}{365}\right)^{k-365-\frac{1}{2}}. \tag{5.3}$$

The approximation (5.3) is plotted along with the actual value in Figure 5.1.

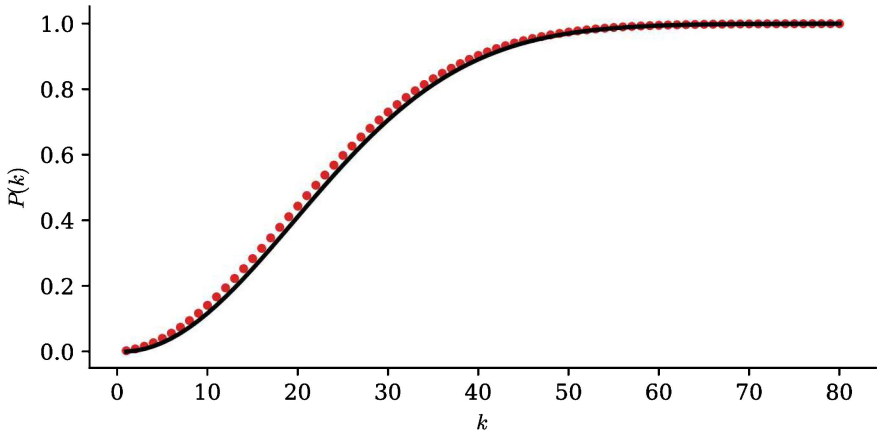


Figure 5.1. *Plot of the probability $P(n)$ (in black) and its approximation* (5.3) *(red) of a birthday collision between two or more individuals in a group of n people, as described in Example 5.1.17. Note that when $n \geq 23$ the probability of two or more people sharing a birthday is more than* 50%.

5.2 Conditional Probability and Bayes' Rule

Knowing that one event has occurred can give useful information about the probability of another event. For example, once your roommate has the flu, the probability that you will also get the flu increases. Conditional probability allows us to account for changes in the world and to update our understanding of the situation to reflect new information. Conditioning is also an extremely powerful tool for solving a wide range of problems, including many problems that may seem at first to have nothing to do with conditioning.

5.2.1 Conditional Probability

Definition 5.2.1. *Let E and F be events in a probability space $(\Omega, \mathscr{F}, P)$, and assume that $P(F) > 0$. The probability of E occurring,* given that *F occurs, denoted $P(E \mid F)$, is written as*

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)}. \tag{5.4}$$

Alternatively we say that the left side is the probability of E conditioned on *F. If $P(F) = 0$, then $P(E \mid F)$ is undefined.*

Remark 5.2.2. One way to think about conditional probability is to think of probability as the percentage of times a given event occurs when the experiment is repeated a large number of times. Under this interpretation, $P(E)$ is the percentage of times that E occurs, and $P(F)$ is the percentage of times that F occurs. So if there are N total experiments, of which E occurs n_E times and F occurs n_F times, then we have

$$P(E) \approx \frac{n_E}{N} \qquad \text{and} \qquad P(F) \approx \frac{n_F}{N}.$$

The conditional probability $P(E \mid F)$ corresponds to throwing out all the trials for which F did not occur, so $P(E \mid F)$ is the percentage of times E occurs among those trials where F occurred. That is, if n_{EF} is the number of times that E and F both occurred, then

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)} \approx \frac{n_{EF}/N}{n_F/N} = \frac{n_{EF}}{n_F}.$$

Remark 5.2.3. Alternatively, you can think of the situation where F is known to occur as giving a new sample space F, replacing the original sample space Ω. For each E in the power set $\mathscr{F}$ of Ω we can construct a new set $E' = E \cap F$ in the power set of F. Each of these sets E' corresponds to an event on the new sample space. The collection of all such new events is the power set $\mathscr{F}' = 2^F$ of the new sample space F. Finally, we define a new probability measure P' on $\mathscr{F}'$ by $P'(E') = P(E \mid F) = \frac{P(E \cap F)}{P(F)}$. It is straightforward to check that P' is indeed a probability measure on $\mathscr{F}'$.

Example 5.2.4. A cooler has four Diet Cokes, three regular Cokes, and three bottled waters. If someone randomly draws two drinks from the cooler and tells us that they are not regular Cokes, but does not show us what they are, what is the probability that the two choices are both Diet Cokes?

Let E be the event "Both are Diet Cokes" and F be the event "Neither is a regular Coke." Note that $E \cap F = E$, and thus

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)} = \frac{P(E)}{P(F)} = \frac{\frac{C(4,2) \cdot C(3,0) \cdot C(3,0)}{C(10,2)}}{\frac{C(7,2) \cdot C(3,0)}{C(10,2)}} = \frac{6}{21} = \frac{2}{7}.$$

Example 5.2.5. A doctor has six patients in the waiting room, two men and four women. Patients are called up in random order and seen in the order in which they were called. What is the probability that the second patient is a female given that the first is a male?

This situation is depicted by the tree in Figure 5.2. Let A be the event "the first patient is male," corresponding to the yellow node and its children, and let B be the event "the second patient is female," corresponding to the two green nodes.

To find the conditional probability $P(B \mid A)$ note that if the first patient is male, the number of patients remaining is 5, and there are still four women but only one man; so the probability that the second patient is also a man is $\frac{1}{5}$, while the probability that the second patient is a woman is $\frac{4}{5}$. A similar computation gives the probabilities for the edges in the lower half of the tree. Note that we did not need to use (5.4) to compute the conditional probability in this case.

5.2.2 The Chain Rule

Many useful results follow easily from the definition of conditional probability. One of these is the *chain rule*. The chain rule gives a way to write the probability of the intersection of several events in terms of conditional probabilities.

Proposition 5.2.6 (Chain Rule). *If $\{E_i\}_{i=1}^n$ are events in a probability space $(\Omega, \mathscr{F}, P)$ with $P(E_1, \ldots, E_{n-1}) > 0$, then*

$$P(E_1, E_2) = P(E_1)P(E_2 \mid E_1),$$

$$P(E_1, E_2, E_3) = P(E_1)P(E_2 \mid E_1)P(E_3 \mid E_2, E_1),$$

and more generally

$$P(E_1, \ldots, E_n) = P(E_1) \prod_{i=2}^{n} P(E_i \mid E_1, \ldots, E_{i-1}).$$

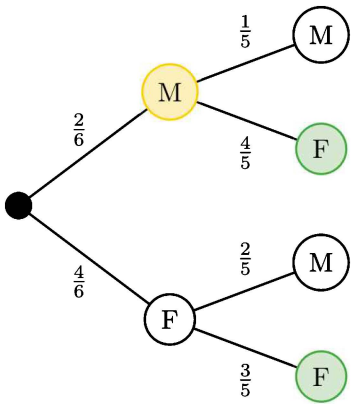


Figure 5.2. *A tree depicting the probabilities that male and female patients are selected first or second in Example 5.2.5. The yellow node and its children correspond to the event "the first patient is male," and the green node corresponds to the event "the second patient is female, and the first patient is male." The leftmost edges are labeled with the probabilities for the first patient (male or female, respectively), and the rightmost edges are the conditional probabilities for the second patient. For example, the edge from the yellow M to the green F is labeled with the conditional probability that the second patient is female, given that the first is male.*

Proof. The proof is Exercise 5.11. □

Example 5.2.7. In the waiting room of Example 5.2.5, we might also ask what is the probability that both the first patient is a man and the second is a woman, or, in the notation of that example, what is $P(A, B)$? The event $A \cap B$ is the upper green node. By the chain rule we have

$$P(A, B) = P(B \mid A)P(A) = \frac{4}{5} \cdot \frac{2}{6} = \frac{8}{30}.$$

Going one more step, what is the probability that the sequence of patients is MFM? Let C be the event that the third patient is male, so we want to find $P(A, B, C)$. To find this we use the chain rule again but we first need the conditional probability $P(C \mid A, B)$. In the event of $A \cap B$, there are four patients remaining—three women and one man—so the conditional probability $P(C \mid A, B)$ is $\frac{1}{4}$. Now the chain rule gives

$$P(A, B, C) = P(C \mid A, B)P(B \mid A)P(A) = \frac{1}{4} \cdot \frac{8}{30} = \frac{1}{15}.$$

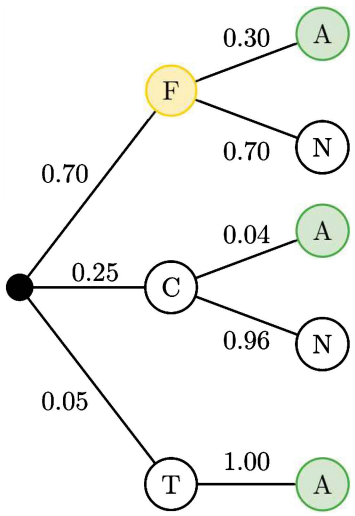


Figure 5.3. *A tree depicting the probabilities of a Ford (F), Chevy (C), or Tesla (T) having autonomous capabilities (A) or not (N), as described in Example 5.2.8. The left edges, from the black node to F, C, and T, are labeled with their corresponding probabilities $P(F)$, $P(C)$, and $P(T)$, respectively. The rightmost nodes depict the case of two events both occurring, so the upper right node (A) depicts the situation where a car is an autonomous Ford. The edge from node F to the upper right node is labeled with the conditional probability $P(A \mid F) = 0.30$, and the other right-hand edges are similarly labeled with their corresponding conditional probabilities.*

Example 5.2.8. In a large fleet of cars, 70% are Fords, 25% are Chevys, and 5% are Teslas. Thirty percent of the Fords are autonomous (self-driving), while only 4% of the Chevys are. All Teslas are autonomous, obviously. If you insist on an autonomous car and are given one at random, what is the probability that it will be a Ford?

We begin to solve this by drawing a tree as in Figure 5.3, where the edges from the black node to the nodes F, C, and T denote the probability of each brand (without the constraint that the car be autonomous), and the right-hand edges from F, C, and T to the nodes labeled A (autonomous) or N (not autonomous) correspond to the conditional probabilities; for example, the edge from F to the uppermost A is labeled with the conditional probability $P(A \mid F)$.

By the chain rule, the probability that a car is an autonomous Ford is

$$P(F, A) = P(A \mid F)P(F) = 0.30 \cdot 0.70 = 0.21.$$

Similarly, the probability of an autonomous Chevy is

$$P(C, A) = P(A \mid C)P(C) = (0.04)(0.25) = 0.01,$$

and finally

$$P(T, A) = P(A \mid T)P(T) = (1)(0.05) = 0.05.$$

We can now solve the problem because the three makes of car correspond to mutually exclusive events, so by the additivity property of probability we have

$$P(A) = P(F, A) + P(C, A) + P(T, A) = 0.21 + 0.01 + 0.05 = 0.27,$$

and thus

$$P(F \mid A) = \frac{P(F, A)}{P(A)} = \frac{0.21}{0.27} = \frac{7}{9}.$$

Example 5.2.9. Assume in the previous example that 70% of the autonomous Chevys are also electric, and the rest use fossil fuels. By the chain rule, the probability that a randomly selected car is an electric, autonomous Chevy is

$$P(E, A, C) = P(E \mid A, C)P(A \mid C)P(C) = (0.70)(0.04)(0.25) = 0.007.$$

5.2.3 Law of Total Probability

The additivity property of probability, combined with the definition of conditional probability, gives another useful tool, the *law of total probability*, which allows us to condition on a partition in order to compute the probability of an event.

Proposition 5.2.10 (Law of Total Probability). *If $\{E_i\}_{i\in I}$ is a countable collection of mutually exclusive and collectively exhaustive events in a probability space $(\Omega, \mathscr{F}, P)$, then for any event $F \in \mathscr{F}$, we have*

$$P(F) = \sum_{i\in I} P(F \mid E_i)P(E_i).$$

Here we use the convention that $P(F \mid E_i)P(E_i) = 0$ whenever $P(E_i) = 0$, even though $P(F \mid E_i)$ is undefined in that case.

Proof. The proof is Exercise 5.12. □

The law of total probability provides a powerful problem-solving strategy for computing probabilities. The idea is to identify a collection of mutually exclusive and collectively exhaustive events (or just an event and its complement) having the property that if we knew which one occurred, the problem would be easy. We can get the desired probability by conditioning on each of these events and assembling the results using the law of total probability. This is illustrated in the following two examples.

Example 5.2.11. Every day one of my three daughters, Adriana, Bhavana, or Ciara, borrows my car for the entire day, and when she does, she sometimes leaves a note (event N) to say thank you. Of the three, Adriana takes the car (event A) 50% of the time, and when she does, the probability she'll leave a note is $P(N \mid A) = 25\%$. Bhavana takes the car (event B) 30% of the time, and when she does, she leaves a note $P(N \mid B) = 10\%$ of the time. Finally, Ciara takes the car (event C) 20% of the time, and she always leaves a note. I don't know who borrowed my car today. What is the probability that I'll get a note?

Although I don't know who borrowed the car, if I did know that, the problem would be easy. So I can compute the conditional probabilities, conditioned on who borrowed the car, and then assemble the results using the law of total probability:

$$\begin{aligned} P(N) &= P(N \mid A)P(A) + P(N \mid B)P(B) + P(N \mid C)P(C) \\ &= (0.25)(0.5) + (0.1)(0.3) + (1.0)(0.2) \\ &= 0.355. \end{aligned}$$

Example 5.2.12. In the waiting room example (Example 5.2.5) what is the probability that the second patient is a woman? Once again, if we knew the gender of the first patient, the problem would be easy—it would be one of the conditional probabilities we already computed (listed on the edges of the graph in Figure 5.2). So, we use the law of total probability to assemble these conditional probabilities into the total probability.

Let A be the event that the first patient is male, B be the event that the second patient is female, and C be the event that the first patient is female. Since A and C are mutually exclusive and collectively exhaustive, we have

$$P(B) = P(B \mid A)P(A) + P(B \mid C)P(C) = \frac{4}{5} \cdot \frac{2}{6} + \frac{3}{5} \cdot \frac{4}{6} = \frac{2}{3}.$$

5.2.4 Bayes' Rule

In Example 5.2.8 we solved $P(F \mid A)$ using information about $P(A \mid F)$. This is a very useful method, and it generalizes to give a fundamental tool of conditional probability called *Bayes' rule* or *Bayes' formula*.

Theorem 5.2.13 (Bayes' Rule). *Let $(\Omega, \mathscr{F}, P)$ be a probability space, and let $E, F \in \mathscr{F}$ with $P(E) > 0$ and $P(F) > 0$. We have*

$$P(E \mid F) = \frac{P(F \mid E)P(E)}{P(F)}. \tag{5.5}$$

Moreover if $\{E_i\}_{i=1}^n \subset \mathscr{F}$ is a collection of mutually exclusive and collectively exhaustive subsets of Ω with $P(E_j) > 0$ for each j, then for any choice of i we have

$$P(E_i \mid F) = \frac{P(F \mid E_i)P(E_i)}{\sum_{j=1}^n P(F \mid E_j)P(E_j)}. \tag{5.6}$$

Proof. Equation (5.4) gives

$$P(E_i \cap F) = P(E_i \mid F)P(F) = P(F \mid E_i)P(E_i).$$

Thus

$$P(E_i \mid F) = \frac{P(F \mid E_i)P(E_i)}{P(F)} = \frac{P(F \mid E_i)P(E_i)}{\sum_{j=1}^n P(F \mid E_j)P(E_j)}.$$

The last equality follows by the law of total probability. □

Example 5.2.14. Revisiting the problem in Example 5.2.8, we see that Bayes' rule immediately gives

$$\begin{aligned} P(F \mid A) &= \frac{P(A \mid F)P(F)}{P(A \mid F)P(F) + P(A \mid C)P(C) + P(A \mid T)P(T)} \\ &= \frac{(0.3)(0.7)}{(0.3)(0.7) + (0.04)(0.25) + (1.0)(0.05)} = \frac{0.21}{0.27} = \frac{7}{9}. \end{aligned}$$

5.3 Independence, Paradoxes, and Pitfalls

Although knowing one event often gives new information about the probability of another, there are also times when two events are completely independent, and knowing about one tells us nothing about the other. For example, the outcome of one coin flip generally has no impact on the outcome of another. In this section we treat the idea of independence and then discuss a number of paradoxes and pitfalls in probability theory.

5.3.1 Independence

Informally, we say two events are independent if knowing the outcome of one event gives no information about the probability of the other. Said more carefully, events E and F are independent if

$$P(E \mid F) = P(E) \qquad \text{and} \qquad P(F \mid E) = P(F). \tag{5.7}$$

Combining (5.4) and (5.7) gives $P(E)P(F) = P(E \cap F)$. This is our definition of independence.

Definition 5.3.1. *Two events E, F in a probability space are* independent *if*

$$P(E \cap F) = P(E)P(F). \tag{5.8}$$

Remark 5.3.2. Although they are almost equivalent, (5.8) is more informative than (5.7) because it does not require $P(E)$ or $P(F)$ to be nonzero.

Example 5.3.3. A card is selected at random from an ordinary deck of cards. Let E be the event "the card is an ace" and F be the event "the card is a spade." Since $P(E) = \frac{1}{13}$, $P(F) = \frac{1}{4}$, and $P(E \cap F) = \frac{1}{52} = \frac{1}{4} \cdot \frac{1}{13}$, the events E and F are independent.

Unexample 5.3.4. If A and B are disjoint events, then $P(A \cap B) = P(\emptyset) = 0$. If $P(A) > 0$ and $P(B) > 0$, then these events cannot be independent.

If the outcome of E has no impact on the probability of F, then it also has no impact on the probability of the complement of F.

Proposition 5.3.5. *If E and F are independent events, then E^c and F are also independent.*

Proof. The proof is Exercise 5.13. □

We also need to consider collections of independent events. This is a little more subtle than just requiring pairwise independence.

Definition 5.3.6. *Let $(\Omega, \mathscr{F}, P)$ be a probability space. A collection $\mathscr{C} = \{E_i\}_{i\in I}$ of events is* independent *if for every finite subcollection $\{E_{i_k}\}_{k=1}^m$ of $\mathscr{C}$, we have*

$$P\left(\bigcap_{k=1}^{m} E_{i_k}\right) = \prod_{k=1}^{m} P(E_{i_k}). \tag{5.9}$$

Unexample 5.3.7. Consider the situation where two fair dice are rolled, as in Example 5.1.14. Let A be the event that 1 shows on the first die. Let B be the event that 1 shows on the second die, and let C be the event that the sum of the numbers showing is 7. We have

$$P(A) = P(B) = P(C) = \frac{1}{6} \quad \text{and} \quad P(A\cap B) = P(B\cap C) = P(C\cap A) = \frac{1}{36}$$

so any two of these events are independent (we call this *pairwise independence*). But

$$P(A\cap B\cap C) = 0 \neq P(A)P(B)P(C) = \left(\frac{1}{6}\right)^3,$$

so (5.8) fails to hold for the subcollection $\{A, B, C\}$ even though it holds for each of the smaller subcollections $\{A, B\}$, $\{A, C\}$, and $\{B, C\}$. Thus, A, B, and C are not independent.

Unexample 5.3.8. Consider three outcomes $x, y, z \in \Omega$ in a discrete probability space Ω with $P(x) = P(y) = P(z) = \frac{1}{4}$. The three events $A = \{x, y\}$, $B = \{y, z\}$ and $C = \{x, z\}$ are pairwise independent because

$$P(A\cap B) = P(B\cap C) = P(C\cap A) = \frac{1}{4} = P(A)P(B) = P(B)P(C) = P(C)P(A),$$

but the three events are not independent because

$$P(A\cap B\cap C) = P(\emptyset) = 0 \neq \frac{1}{8} = P(A)P(B)P(C).$$

Example 5.3.9. A sequence of n independent trials is to be performed. Each trial results in a success S with probability $0 \leq p \leq 1$ and failure F with probability $1 - p$ (these are called *Bernoulli trials*).

(i) If $n = 3$, the set of all possible outcomes is

$$\Omega = \{\text{SSS}, \text{FSS}, \text{SFS}, \text{SSF}, \text{FFS}, \text{FSF}, \text{SFF}, \text{FFF}\}.$$

The event "success on the ith trial" corresponds to the subset E_i of all outcomes that have S in the ith position, so $E_2 = \{\text{SSS}, \text{FSS}, \text{SSF}, \text{FSF}\}$, and $P(E_i) = p$. The fact that the trials are independent implies that the events E_i and E_j are independent for all $i \neq j$. Thus

$$P(E_1 \cap E_2) = P(\{\text{SSS}, \text{SSF}\}) = P(E_1)P(E_2) = p^2$$

and

$$\begin{aligned} P(E_1 \cap E_2^c) &= P(E_1 \setminus (E_1 \cap E_2)) = P(E_1) - P(E_1 \cap E_2) \\ &= p - p^2 = p(1-p) = P(E_1)P(E_2^c), \end{aligned}$$

so E_1 and E_2^c are also independent.

(ii) The probability that the first trial will be a success and the others will be a failure is $P(E_1 \cap E_2^c \cap \cdots \cap E_n^c) = p(1-p)^{n-1}$.

5.3.2 Some Pitfalls in Conditioning

Conditional probability can be tricky, especially if you try to skip the calculations and just estimate the values. In this section we give a few examples of the types of pitfalls that lie in wait for those who are careless.

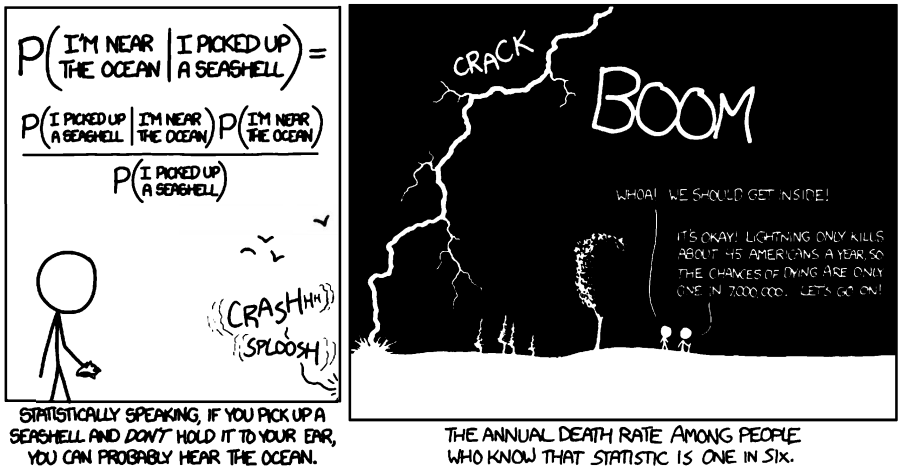


Figure 5.4. *Some pitfalls in conditional probability. Source: XKCD, Randall Munroe.* `http://xkcd.com/1236/` *and* `http://xkcd.com/795/`

Prosecutor's Fallacy

A common error in conditional probability is to use $P(A \mid B)$ when we really want $P(B \mid A)$. This error is sometimes called the *prosecutor's fallacy*, corresponding to

the situation when A is the evidence of guilt and B is the event that the defendant is actually guilty. When written out carefully, it seems clear that we cannot expect $P(A \mid B)$ to be equal to $P(B \mid A)$, yet when encountered in the wild, it is easy to forget that these are not the same.

Example 5.3.10. When a large national database of DNA samples (selected randomly) becomes available, investigators reopen an old, unsolved murder case and search the database for a match with DNA found at the crime scene. One match is found with a person who is not too young to have committed the crime. DNA experts agree that the probability of a random match using this particular DNA test is 1 in 3 million. We write this as

$$P(M \mid I) = \frac{1}{3} \times 10^{-6},$$

where M indicates the event of a match and I indicates that the person is innocent. The prosecutor's fallacy is to claim this means the person who matched has only a 1 in 3 million chance of being innocent, but the prosecutor has mistaken $P(M \mid I)$ for $P(I \mid M)$.

In the absence of any other evidence for or against this person's guilt, we can use Bayes' rule to compute $P(I \mid M)$:

$$P(I \mid M) = \frac{P(M \mid I)P(I)}{P(M)} = \frac{P(M \mid I)P(I)}{P(M \mid I)P(I) + P(M \mid I^c)P(I^c)}.$$

To compute $P(I)$ we need to know the total population of people in the country who could have committed the crime. Assume this is 250 million, so $P(I) = 1 - \frac{1}{250} \times 10^{-6}$ and $P(I^c) = \frac{1}{250} \times 10^{-6}$. Assume also that the DNA will always match if a person is guilty, so $P(M \mid I^c) = 1$. Putting this all together, we have

$$\begin{aligned} P(I \mid M) &= \frac{(\frac{1}{3} \times 10^{-6})(1 - \frac{1}{250} \times 10^{-6})}{(\frac{1}{3} \times 10^{-6})(1 - \frac{1}{250} \times 10^{-6}) + (1)(\frac{1}{250} \times 10^{-6})} \\ &= \frac{250 - 10^{-6}}{253 - 10^{-6}} \approx 0.988. \end{aligned}$$

This seems counterintuitive to many people—probably because most of us tend to commit the prosecutor's fallacy. But you can see it is approximately right by using the following argument. If we had a database of DNA for all 250 million people, then a 1 in 3 million chance of random matching means that there will be about $\frac{250}{3} \approx 83$ false matches and one real match. Thus, the probability that a specific one of those 84 matches is not guilty is close to $\frac{83}{84} \approx 0.988$, matching our previous calculation with Bayes' rule.

Example 5.3.11. At the time of this writing (2019), the *incidence* of breast cancer among women ages 45 to 54 is about 0.3%. If a woman in this age group with no other symptoms, risks, or evidence of cancer has a mammogram that is positive, we want to know the probability that she actually has breast cancer. Assume the following:

(i) The probability of a positive test, given the disease is present, is 0.90.

(ii) The probability of a negative test, given there is no disease, is 0.95.

Denote the event "disease present" by D and the event "disease absent" by H (healthy). Let T^+ denote a positive test and T^- a negative test. Thus $P(T^+ \mid D) = 0.90$ and $P(T^- \mid H) = 0.95$.

In this situation, the prosecutor's fallacy is to look at $P(T^+ \mid D) = 90\%$ and think that $P(D \mid T^+)$ would also be 90%. To calculate the correct value of $P(D \mid T^+)$, we can use Bayes' rule. We assume that D and H are mutually exclusive and collectively exhaustive—everyone is either healthy or diseased, but not both. We compute

$$\begin{aligned} P(D \mid T^+) &= \frac{P(T^+ \mid D)P(D)}{P(T^+ \mid D)P(D) + P(T^+ \mid H)P(H)} \\ &= \frac{(0.90)(0.003)}{(0.90)(0.003) + (1 - 0.95)(1 - 0.003)} \\ &\approx 0.051 = 5.1\%. \end{aligned}$$

This result is counterintuitive for many people. A positive result on this test that is supposed to be 90% to 95% correct only means that you have a roughly 5% probability of actually having the disease.[a] Because of this, it is common to refer to a positive mammogram as simply *abnormal*.

You can make a rough estimate to see that this result is reasonable. If a group of 1000 people in this age group had mammograms, we would expect roughly 50 of them to have false positives and 3 to actually have the disease. So, roughly 3 in 53 people who test positive for breast cancer are expected to have the disease, and indeed, 3 in 53 is 5.7%—not far from the correct answer of 5.1%.

This does *not* mean that you should skip your mammogram. It only means that if your mammogram is abnormal, then you shouldn't be too discouraged and should undergo further testing under the care of a physician.

[a] In fact, the true-positive and true-negative rates of most mammograms are worse than the numbers we have used here [OSS+16], so the probability of disease is less likely than we have computed here. However, these computations are only valid assuming the absence of any other information about the presence of cancer—additional follow-up tests can and should be used to confirm or exclude the possibility of any actual disease.

Nota Bene 5.3.12. Faculty and students at Harvard Medical School were asked a problem similar to that of Example 5.3.11, and fewer than 20% of them got it right. Almost half of them said 95%—about as far away from the correct answer as you could get [HLHG00]. One lesson to take from this is that most human beings are really bad at estimating conditional probabilities. In this particular case, it looks like the half that chose 95% were committing the prosecutor's fallacy.

Neglecting to Condition

In most real-life situations, the probabilities we know and the probabilities we must think about are conditional probabilities. Using absolute probabilities, or forgetting to condition on all the data, gives bad results.

Example 5.3.13. A famous example of forgetting to condition on all the evidence is when the Center for Naval Analysis tried to minimize bomber losses in World War II. After looking at the bullet hole locations in returning bombers, they recommended putting armor in the places that showed the most bullet holes. The mathematician Abraham Wald pointed out that they had forgotten to account for the fact that the only bombers observed were those that had survived in combat. The holes in the bombers that did not survive were not observed.

For each location on the plane, the Center for Naval Analysis had not computed the absolute probability of being hit by a bullet in that location but, rather, the conditional probability of being hit by a bullet in that location, *given that the plane survived.* Moreover, what was really needed was the conditional probability of surviving, given that the plane is struck in that location, and that was clearly higher for the locations where many bullet holes were observed. That is, the best place to put armor was exactly the locations where the fewest bullet holes were observed—not where the most bullet holes were observed.

Written out mathematically, it is reasonable to assume bullets will hit almost all locations with equal probability; that is, $P(\text{hit here})$ is roughly the same for all locations. For a given location on the plane, Bayes' rule gives

$$P(\text{survive} \,|\, \text{hit here}) = \frac{P(\text{hit here} \,|\, \text{survive})P(\text{survive})}{P(\text{hit here})},$$

and since $P(\text{hit here})$ and $P(\text{survive})$ are independent of location, the desired number $P(\text{survive} \,|\, \text{hit here})$ is higher at exactly those locations where $P(\text{hit here} \,|\, \text{survive})$ is highest.

Example 5.3.14. A poll conducted by a cable TV station found that a certain political figure had an approval rating of 35%—much higher than expected. Phrased in terms of probability, the claim was that $P(\text{approve}) = 0.35$. However, this claim failed to account for some important factors, like the fact that the only people who responded to this poll were people who happened to be watching this station at 10 a.m. on Tuesday, when the poll was conducted. People who did not watch this particular TV station at 10 a.m. on a Tuesday morning, and people who, even if they were watching, were not willing to answer a survey, were not included in the results. So it would be more accurate to say that the poll really gave the conditional probability

$$P(\text{approve} \mid \text{watches this station \& answers a survey at 10 a.m. on Tues}).$$

Since the condition *watches this station and willing to answer a survey at 10 a.m. on Tuesday* is not very representative of the general population, the result is unlikely to be very similar to the unconditional probability $P(\text{approve})$ that we really want to know. When we want an unconditional result, but our data supports only a conditional result, the difference between the unconditional and conditional results is called *selection bias.*

5.3.3 *Pitfalls of Assuming Independence

When dealing with apparently independent events, it is important to have a clear understanding of the problem being solved and the questions being asked. Here we consider an example that teaches us to be careful in our thinking when it comes to independence.

Suppose that a fair die has both red and green dots on each side. Assume that if you shine a red flashlight on the die in a dark room, you see only the green dots, and if you shine a green flashlight on the die, you see only the red dots. In other words, the flashlight drowns out its own color so that you can see only the other color. When illuminated with the green light, the die has three odd-numbered red sides and three even-numbered red sides; when illuminated with the red light, the die has two odd-numbered green sides and four even-numbered green sides.

Consider an experiment where the outcome of a fair coin determines the color of flashlight used to illuminate the die when it is rolled: heads means the flashlight is green and tails means the flashlight is red. In each trial, the outcome of the coin flip and the observed die roll are noted; see Figure 5.5 for a description of the outcomes and probabilities.

Are the coin and the die roll independent? Of course they are—the coin and the die do not influence each other. However, the coin does influence the choice of flashlight, which affects the observation that is recorded. Therefore the coin and the *observed* die roll are dependent, even though the coin and the die roll are independent processes. We see this mathematically by noting that $P(H) = \frac{1}{2}$, $P(\text{even}) = \frac{7}{12}$, and $P(H, \text{even}) = \frac{3}{12}$, that is, $P(H, \text{even}) \neq P(H)P(\text{even})$. It is important to recognize that independent processes can be observed in a way that makes the final results dependent.

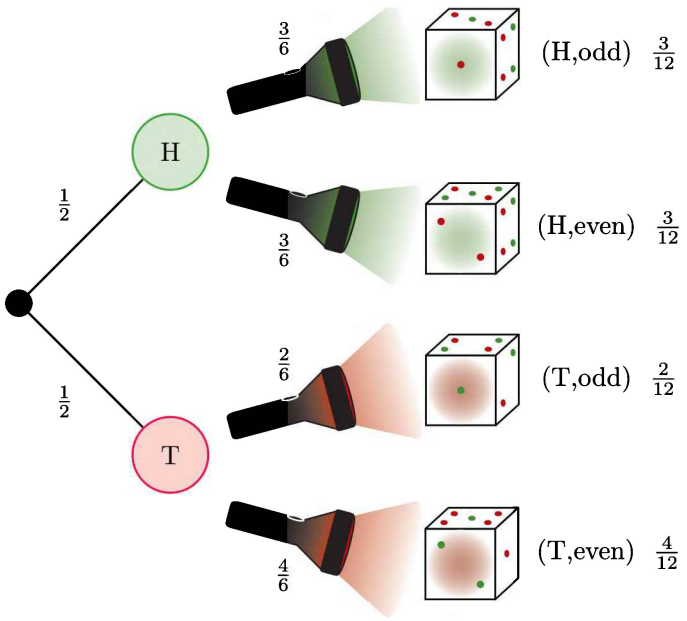


Figure 5.5. *A tree diagram of the experiment described in Section 5.3.3.*

5.4 Discrete Random Variables

Outcomes in a probability space don't necessarily have to be numbers—they could be nearly anything, including colors, textures, or flavors. A *random variable* is a rule (function) that assigns a number (or a vector) to each possible outcome. For example, for a coin flip, we could define a random variable that takes the values 0 for tails and 1 for heads. Or for the roll of a die, we could define a random variable that takes the value (1 through 6) of the face showing upward. But a random variable can also represent many other things, like the amount of money you win in a game of chance, or the number of votes a candidate will receive in an election.

Random variables are a fundamental tool of probability theory. In this section, we define discrete random variables and their properties. We sketch how to extend this to more general settings (uncountable probability spaces) in Section 5.6.

5.4.1 Definition and Examples

A discrete random variable is a function $X : \Omega \to \mathbb{R}$ on a discrete probability space $(\Omega, \mathscr{F}, P)$.

Definition 5.4.1. *Let $(\Omega, \mathscr{F}, P)$ be a discrete probability space. Any function $X : \Omega \to \mathbb{R}$ is called a* discrete random variable *or a* random variable on $(\Omega, \mathscr{F}, P)$. *It is common to denote random variables by capital letters.*

Example 5.4.2. Let Ω be the set of possible outcomes of n flips of a fair coin:

$$\Omega = \{\text{HHH}\ldots\text{HH}, \text{THH}\ldots\text{HH}, \text{HTH}\ldots\text{HH}, \ldots, \text{TTT}\ldots\text{TH}, \text{TTT}\ldots\text{TT}\}$$

All outcomes are equally likely, so $P(\omega) = 2^{-n}$ for all $\omega \in \Omega$, and $P(E) = 2^{-n}|E|$ for all $E \in \mathscr{F}$. We define random variables

$$X_k = \begin{cases} 1 & \text{if } k\text{th coin flip is H,} \\ 0 & \text{if } k\text{th coin flip is T.} \end{cases}$$

For example, $X_1(HHH) = X_1(HTH) = 1$, while $X_1(THT) = 0$. The sum $X = \sum_{k=1}^{n} X_k$ is also a random variable, which counts the total number of heads.

Remark 5.4.3. We remind the reader that for any function $f : A \to B$ and for any subset $S \subset B$, the *preimage of* S is the set $f^{-1}(S) = \{a \in A \mid f(a) \in S\}$. For a single element $b \in B$, we often abuse notation and write $f^{-1}(b)$ when we mean $f^{-1}(\{b\})$.

Nota Bene 5.4.4. Beware that the notation $f^{-1}(S)$ makes sense even if f has no inverse. If f does have an inverse g, then the preimage $f^{-1}(S)$ is what you might expect it to be, that is, the set $f^{-1}(S) = \{g(s) \mid s \in S\}$. But $f^{-1}(S)$ exists for any function f, whether f has an inverse or not.

Example 5.4.5. In Example 5.4.2, the set $X_1^{-1}(1)$ consists of all elements of Ω which begin with H. The set $X_3^{-1}(0)$ consists of all elements of Ω whose third term is T. And if $n = 4$, we have

$$X^{-1}(\{3,4\}) = \{\text{HHHH}, \text{HHHT}, \text{HHTH}, \text{HTHH}, \text{THHH}\}.$$

Definition 5.4.6. *Let X be a random variable on a discrete probability space. Given $a \in \mathbb{R}$, define the event "$X = a$" to be the set $X^{-1}(a) = \{\omega \in \Omega \mid X(\omega) = a\}$. This set is an element of $\mathscr{F}$, so it makes sense to define the probability of this event:*

$$P(X = a) = P(X^{-1}(a)).$$

The probability mass function *(p.m.f.) of X is the function $g_X : \mathbb{R} \to [0,1]$ given by*

$$g_X(a) = P(X = a).$$

Remark 5.4.7. The domain of a p.m.f., as we have defined it here, is all of $\mathbb{R}$, but it is often convenient to restrict the domain to be the range of the corresponding random variable. This often simplifies the formulas for the p.m.f.

Example 5.4.8. Let Ω be the set of possible outcomes for n Bernoulli trials with probability p of success. Let

$$Y_k = \begin{cases} 1 & \text{if the } k\text{th trial is successful,} \\ 0 & \text{if the } k\text{th trial is a failure} \end{cases} \qquad \text{and} \qquad Y = \sum_{k=1}^{n} Y_k.$$

Example 5.3.9 shows that

$$g_Y(r) = P(Y = r) = \begin{cases} \binom{n}{r} p^r (1-p)^{n-r} & \text{if } r \in \{0, 1, \ldots, n\}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

A random variable X whose p.m.f. is equal to g_Y is said to be *binomially distributed* with parameters n and p.

5.4.2 Expectation

The *expectation* of a random variable is the sum of the values of the random variable weighted by probability. This is often called the *mean* of the random variable.

Definition 5.4.9. *Let X be a random variable on a discrete probability space. The* expectation *(or* expected value*) of X is given by*

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) P(\omega),$$

provided this sum converges absolutely. If the sum does not converge absolutely, the expected value does not exist.

It is immediate from the definition that

$$\mathbb{E}[X] = \sum_i i\, P(X = i) \; = \sum_i i\, g_X(i), \tag{5.11}$$

where the sums run over all values i in the image of X.

Thinking of probability in terms of mass is a very useful analogy here. Each x in the image of X corresponds to a point on the real line at position x with mass $g_X(x) = P(X = x)$. Under this analogy, the expected value of X is the location of the center of mass of the collection of all these points on the real line.

Example 5.4.10. A random variable X with range $\{0, 1\}$ is a *Bernoulli* random variable. Note that since X can take on only the values 0 and 1, we must have $P(X = 0) + P(X = 1) = 1$. Thus, if $P(X = 1) = p$, then we must have $P(X = 0) = 1 - p$. This gives

$$\mathbb{E}[X] = 0 \cdot P(X = 0) + 1 \cdot P(X = 1) = 0 \cdot (1 - p) + 1 \cdot (p) = p.$$

Example 5.4.11. You flip a coin until it comes up heads. Let X be the random variable corresponding to the number of flips it takes to get heads. If it comes up heads on the first flip, then $X = 1$. If it takes two flips to come up heads, then $X = 2$, and so forth. The sample space is $\Omega = \{\text{H}, \text{TH}, \text{TTH}, \text{TTTH}, \dots\}$. The probability $P(X = n)$ is the probability of getting tails $n - 1$ times in a row, followed by heads. Since each coin flip is independent, we have

$$P(X = n) = \left(\frac{1}{2}\right)^{n-1} \cdot \frac{1}{2} = 2^{-n}.$$

The expected value of X is

$$\mathbb{E}[X] = \sum_{n=1}^{\infty} nP(X = n) = \sum_{n=1}^{\infty} n2^{-n} = \frac{1/2}{(1 - 1/2)^2} = 2,$$

where the third equality follows from Exercise 1.20. Notice that the expected value of X is well defined and finite, even though the sample space is infinite.

Nota Bene 5.4.12. To calculate the expected value of a random variable, we need only the p.m.f. g_X. We do not need to know the particulars of the sample space Ω, nor do we need to know all the values of the probability distribution $P(\omega)$ for every $\omega \in \Omega$. This is a big deal, because many different random variables defined on many different sample spaces end up having the same p.m.f. This means that many different situations can be modeled and understood with the same p.m.f. We discuss many of the most common p.m.f.s in the next section. These few examples describe a surprisingly large number of the most important situations you will encounter in discrete probability.

Proposition 5.4.13. *For any constant $\alpha \in \mathbb{R}$, we have $\mathbb{E}[\alpha] = \alpha$.*

Proof. Since $X = \alpha$ is constant, we have $P(X = \alpha) = 1$ and $P(X \neq \alpha) = 0$, so we have

$$\mathbb{E}[\alpha] = \alpha\, P(X = \alpha) = \alpha. \quad \square$$

The next theorem shows that expectation is a linear operator on the space of random variables.

Theorem 5.4.14. *For any constants $\alpha, \beta \in \mathbb{R}$ and any two random variables X and Y on the same probability space Ω, we have*

$$\mathbb{E}[\alpha X + \beta Y] = \alpha\mathbb{E}[X] + \beta\mathbb{E}[Y].$$

Proof.

$$\begin{aligned}\mathbb{E}[\alpha X + \beta Y] &= \sum_{\omega \in \Omega} (\alpha X(\omega) + \beta Y(\omega)) P(\omega) \\ &= \alpha \sum_{\omega \in \Omega} X(\omega) P(\omega) + \beta \sum_{\omega \in \Omega} Y(\omega) P(\omega) \\ &= \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y]. \quad \square\end{aligned}$$

Given a function $h : \mathbb{R} \to \mathbb{R}$ and a discrete random variable $X : \Omega \to \mathbb{R}$, the composition $h \circ X : \Omega \to \mathbb{R}$ is also a discrete random variable. By definition, the expected value of $h \circ X$ is

$$\mathbb{E}[h \circ X] = \sum_j j \, P(h \circ X = j),$$

where j runs over all the values in the image of $h \circ X$. If we weren't thinking carefully (if we were acting "unconsciously"), we might instead write

$$\mathbb{E}[h \circ X] = \sum_i h(i) P(X = i),$$

where i runs over all values in the image of X. Surprisingly, this unconscious computation still gives the right answer. This fact is not very hard to prove, but it is very useful. It is sometimes called the *law of the unconscious statistician.*

Theorem 5.4.15 (The Law of the Unconscious Statistician). *If X is a discrete random variable, and $h : \mathbb{R} \to \mathbb{R}$ is any function, then $h \circ X$ is a discrete random variable, and the expected value of $h(X)$ is given by*

$$\mathbb{E}[h(X)] = \sum_i h(i) P(X = i) = \sum_i h(i) g_X(i).$$

Proof. The proof is Exercise 5.20. $\square$

Example 5.4.16. You are flipping coins to get heads, as in Example 5.4.11, and someone offers you a bet, based on the outcome of the coin flip. If $X = n$, then she will pay you $\frac{1}{n!}$ dollars. Let Y be the random variable corresponding to the amount you win. Note that $Y = \frac{1}{X!}$, so the expected value of Y is

$$\mathbb{E}[Y] = \sum_{n=1}^{\infty} \frac{1}{n!} P(X = n) = \sum_{n=1}^{\infty} \frac{2^{-n}}{n!} = e^{1/2} - 1$$

by the law of the unconscious statistician.

A random variable X defines many events of the form $X^{-1}(a) = \{\omega \mid X(\omega) = a\}$. Just as events can be independent, random variables can also be independent, if the corresponding events they define are independent, as follows.

Definition 5.4.17. *Two discrete random variables X and Y are* independent *if the events $X = a$ and $Y = b$ are independent for all $a, b \in \mathbb{R}$, that is,*

$$P\left((X = a) \cap (Y = b)\right) = P(X = a)P(Y = b). \tag{5.12}$$

Proposition 5.4.18. *For any independent random variables X and Y on a discrete probability space $(\Omega, \mathscr{F}, P)$, the product XY is also a random variable, defined by $XY(\omega) = X(\omega)Y(\omega)$, with expectation*

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y].$$

Proof. The fact that XY is a random variable is immediate. We compute its expectation as follows:

$$\begin{aligned}
\mathbb{E}[XY] &= \sum_{\omega \in \Omega} XY(\omega)P(\omega) = \sum_n nP(XY = n) \\
&= \sum_n \sum_{x,y:xy=n} nP(X = x, Y = y) = \sum_x \sum_y xyP(X = x, Y = y) \\
&= \sum_x \sum_y xyP(X = x)P(Y = y) \qquad \text{(by independence)} \\
&= \left(\sum_x x\,P(X = x)\right)\left(\sum_y y\,P(Y = y)\right) = \mathbb{E}[X]\mathbb{E}[Y]. \qquad \square
\end{aligned}$$

5.4.3 Variance

The *variance* of a random variable is a measure of how far the random variable typically differs from its mean. Some random variables are spread out from the mean (high variance), and others are bunched up near the mean (low variance). For example, if all the darts on a dartboard are close to each other (but not necessarily close to the bull's-eye), this is an example of a random variable (the location of a dart) with low variance. However, if the darts are spread all over the board, and maybe even on the wall surrounding the dartboard, this is an example of a random variable with high variance.

Definition 5.4.19. *Let X be a discrete random variable with $\mathbb{E}[X] = \mu$. The* variance *of X is the quantity*

$$\mathrm{Var}(X) = \mathbb{E}\left[(X - \mu)^2\right], \tag{5.13}$$

provided this expectation is defined (absolutely convergent). The standard deviation *is the square root of the variance.*

Here $|X - \mu|$ is the distance from X to its mean—how spread out it is. The square $|X - \mu|^2 = (X - \mu)^2$ is even greater than that distance when $|X - \mu| > 1$, but it is smaller when $|X - \mu| < 1$. Therefore, the expected value of $(X - \mu)^2$ is big when X is usually far from μ and is small if X is usually close to μ.

Theorem 5.4.20. *Let X be a random variable with $\mathbb{E}[X] = \mu$. Denote by X^2 the random variable given by $X^2(\omega) = (X(\omega))^2$. We have*

$$\operatorname{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathbb{E}[X^2] - \mu^2. \tag{5.14}$$

Proof. The proof is Exercise 5.22. □

Example 5.4.21.

(i) For a Bernoulli random variable X with parameter p, we have

$$\mathbb{E}[X] = p \qquad \text{and} \qquad \mathbb{E}[X^2] = 1^2 p + 0^2(1-p) = p.$$

It follows that

$$\operatorname{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = p - p^2 = p(1-p).$$

(ii) For a binomial random variable X with parameters n and p, we have $\mathbb{E}[X] = np$ and

$$\mathbb{E}[X^2] = \sum_{i=0}^{n} i^2 P(X = i) = \sum_{i=1}^{n} i^2 \binom{n}{i} p^i (1-p)^{n-i}.$$

It is not hard to show that $i\binom{n}{i} = n\binom{n-1}{i-1}$, which is used to prove that

$$\mathbb{E}[X^2] = (np)^2 - np^2 + np.$$

It follows that

$$\operatorname{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = -np^2 + np = np(1-p).$$

Proposition 5.4.22. *For any random variable X and constants $\alpha, \beta \in \mathbb{R}$, we have*

$$\operatorname{Var}(\alpha X + \beta) = \alpha^2 \operatorname{Var}(X).$$

Proof. Expanding the definition and collecting like terms gives

$$\begin{aligned}
\operatorname{Var}(\alpha X + \beta) &= \mathbb{E}[(\alpha X + \beta)^2] - (\mathbb{E}[\alpha X + \beta])^2 \\
&= \mathbb{E}[\alpha^2 X^2 + 2\alpha\beta X + \beta^2] - (\alpha \mathbb{E}[X] + \beta)^2 \\
&= \alpha^2 \mathbb{E}[X^2] - \alpha^2 \mathbb{E}[X]^2 \\
&= \alpha^2 \left(\mathbb{E}[X^2] - \mathbb{E}[X]^2\right) \\
&= \alpha^2 \operatorname{Var}(X). \quad \square
\end{aligned}$$

Proposition 5.4.23. *If X and Y are random variables and $\alpha, \beta \in \mathbb{R}$ are constants, then*

$$\operatorname{Var}(\alpha X + \beta Y) = \alpha^2 \operatorname{Var}(X) + 2\alpha\beta(\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]) + \beta^2 \operatorname{Var}(Y).$$

If X and Y are independent, then variance behaves like the square of a norm:

$$\operatorname{Var}(\alpha X + \beta Y) = \alpha^2 \operatorname{Var}(X) + \beta^2 \operatorname{Var}(Y). \tag{5.15}$$

Proof. The proof is Exercise 5.23. □

5.5 Discrete Distributions

Most of the important properties of a discrete random variable X are determined by its p.m.f. As described in Nota Bene 5.4.12, this means we rarely need to think about the sample space Ω or the probability distribution on Ω—it is enough just to understand the p.m.f. When we talk about a discrete random variable having a particular *distribution*, we mean that it has a particular p.m.f.

In this section we give some important examples of probability distributions for discrete random variables. Many different random variables defined on many different probability spaces have the same p.m.f.; that is, they all have the same distribution. Therefore, understanding just a few distributions gives us the power to model and understand a large number of different probabilistic situations. Many more random variables can be described as a composition of some function with one of the basic random variables, and so by the law of the unconscious statistician, the most important properties of these other random variables can also be described using these basic distributions.

The set of values of x for which $g_X(x)$ is nonzero is usually called the *support*[24] of the discrete distribution. It is a subset of the domain of g_X and a subset of the range of the random variable X. Throughout this section, the values of the p.m.f. $g_X(x)$ are given only for x in the support of the distribution. The p.m.f. is always 0 for values of x that do not lie in the support.

5.5.1 Bernoulli Distribution

The Bernoulli distribution is among the most fundamental of discrete distributions. It typically represents the results of a Bernoulli trial—like a coin flip or a free throw—where the results are always in exactly one of two categories.

A random variable X has *Bernoulli distribution* if its support is equal to $\{0, 1\}$. As described in Example 5.4.10, if $P(X = 1) = p$, then $P(X = 0) = 1 - p$ and the p.m.f. of X is

$$g_X(x) = p^x(1-p)^{1-x} = \begin{cases} p & \text{if } x = 1, \\ 1-p & \text{if } x = 0. \end{cases} \tag{5.16}$$

[24] More generally, the *support* of a function $f : X \to Y$ is the closure of the set of all $x \in X$ such that $f(x) \neq 0$. The support of a discrete distribution is just the support of the p.m.f. of the distribution.

In this case we write $X \sim \text{Bernoulli}(p)$ and say "X has a Bernoulli distribution with parameter p." Example 5.4.21 shows that $\mathbb{E}[X] = p$ and $\text{Var}(X) = p(1-p)$.

Example 5.5.1. Assume the probability that a basketball player will make a free throw is 88.08%. We can define a random variable X that is 1 with a successful throw and 0 with a failed throw. This is Bernoulli distributed, and $P(X = 1) = 88.08\%$, while $P(X = 0) = 11.92\%$, so

$$\begin{aligned} g_X(x) &= \begin{cases} 0.8808, & x = 1, \\ 0.1192, & x = 0, \end{cases} \\ &= (0.8808)^x (0.1192)^{(1-x)}. \end{aligned}$$

If you enter a bet where you win \$5 if she misses and \$2 if she is successful, this defines a new random variable $G(\text{miss}) = 5$ and $G(\text{success}) = 2$, which we can rewrite as $G = h \circ X$ with $h(0) = 5$ and $h(1) = 2$. By the law of the unconscious statistician, the expected return for this bet is

$$\mathbb{E}[G] = \mathbb{E}[h \circ X] = h(0)g_X(0) + h(1)g_X(1) = \$5 \cdot 0.1192 + \$2 \cdot 0.8808 = \$2.36.$$

Definition 5.5.2 (Indicator Random Variable). *Let E be any event in a probability space $(\Omega, \mathscr{F}, P)$. The function $\mathbb{1}_E : \Omega \to \{0, 1\}$, given by*

$$\mathbb{1}_E(\omega) = \begin{cases} 1 & \text{if } \omega \in E, \\ 0 & \text{if } \omega \notin E, \end{cases}$$

is called the indicator random variable *of E.*

The indicator $\mathbb{1}_E$ is Bernoulli distributed with parameter $p = P(E)$ (unless $E = \Omega$ or $E = \emptyset$). Conversely, given any Bernoulli-distributed random variable X with parameter p, letting $E = X^{-1}(1)$ gives $E^c = X^{-1}(0)$. For any $\omega \in \Omega$ we have

$$\begin{aligned} \mathbb{1}_E(\omega) &= \begin{cases} 1 & \text{if } \omega \in E, \\ 0 & \text{if } \omega \in E^c \end{cases} \\ &= \begin{cases} 1 & \text{if } X(\omega) = 1, \\ 0 & \text{if } X(\omega) = 0 \end{cases} \\ &= X(\omega), \end{aligned}$$

and thus $\mathbb{1}_E = X$. So any Bernoulli random variable X is the indicator function of the set $E = X^{-1}(1)$.

5.5.2 Binomial Distribution

The binomial distribution describes the number of successes of a sequence of n *repeated* Bernoulli trials (by repeated, we mean independent and with the same

parameter p). For example, the binomial distribution describes the number of heads that occur when a coin is flipped $n = 100$ times.

We say that a random variable X has *binomial distribution* with parameters n and p if the support is $\{0, 1, 2, \ldots, n\}$ and the p.m.f. of X is

$$g_X(x) = \binom{n}{x} p^x (1-p)^{n-x}. \tag{5.17}$$

In this case we write $X \sim \text{Binomial}(n, p)$.

We will show below that

$$\mathbb{E}[X] = np \qquad \text{and} \qquad \text{Var}(X) = np(1-p).$$

As the next two examples show, the sum of n independent Bernoulli random variables $X_1, \ldots, X_n$, all with parameter p, is a binomially distributed random variable with parameters n and p.

Example 5.5.3. Continuing with the situation of Example 5.5.1, assume that all of Kevin Durant's free throws are independent, so his ability to make a shot is not affected—positively or negatively—by any previous failures or successes.

If he takes 10 free throws, let T be the total number of successes. The probability that $T = 7$ can be computed by listing each of the ways that he could make 7 and miss 3 and then summing the probability of all of those. If F denotes failure and S denotes success, we have the following possibilities and probabilities:

$$\begin{array}{ll} \text{FFFSSSSSSS} & (1-p)^3 p^7, \\ \text{FFSFSSSSSS} & (1-p)^2 p (1-p) p^6 = (1-p)^3 p^7, \\ \text{FFSSFSSSSS} & (1-p)^2 p^2 (1-p) p^5 = (1-p)^3 p^7, \\ \text{FFSSSFSSSS} & (1-p)^2 p^3 (1-p) p^4 = (1-p)^3 p^7, \\ \vdots & \vdots \qquad\qquad \vdots \end{array}$$

There are $\binom{10}{7} = 120$ of these possibilities, all with the same probability, so summing them all up gives

$$P(T = 7) = g_T(7) = \binom{10}{7} (1-p)^3 p^7 = 120 \cdot (0.8808)^7 (0.1192)^3.$$

A similar argument for any $t \in \{0, 1, \ldots, 10\}$ shows that

$$P(T = t) = \binom{10}{t} p^t (1-p)^{10-t},$$

and hence T is binomially distributed with $n = 10$ and $p = 0.8808$.

Example 5.5.4. Generalizing the previous example, for any n repeated (independent) Bernoulli trials all with the same parameter p, let X_i be 1 if the ith trial is successful and 0 otherwise. Each X_i is Bernoulli distributed with parameter p, and the sum $X = \sum_{i=1}^{n} X_i$ is also a random variable.

Making the same sort of argument as in Example 5.5.3, we find the probability that $X = x$, by listing each of the possibilities that sum to x and summing those probabilities:

$$\begin{array}{ll} \underbrace{1,1,\ldots,1}_{x},\underbrace{0,0,\ldots,0}_{n-x} & p^x(1-p)^{n-x} \\ 0,\underbrace{1,\ldots,1,1}_{x},\underbrace{0,\ldots,0}_{n-x-1} & p^x(1-p)^{n-x} \\ \vdots & \vdots \end{array}$$

There are $\binom{n}{x}$ of these possibilities, all with the same probability, so summing them all up gives

$$P(X = x) = g_X(x) = \binom{n}{x} p^x (1-p)^{n-x}.$$

This shows that $X = (\sum_{i=1}^{n} X_i) \sim \text{Binomial}(n, p)$.

Proposition 5.5.5. *If* $X \sim \text{Binomial}(n, p)$, *then*

$$\mathbb{E}[X] = np \qquad \textit{and} \qquad \text{Var}(X) = np(1-p).$$

Proof. Note that the expected value and variance of a discrete random variable are completely determined by its p.m.f., and the discussion in Example 5.5.4 shows that X has the same p.m.f. as $\sum_{i=1}^{n} X_i$, where each $X_i \sim \text{Bernoulli}(p)$. Therefore $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i]$ and $\text{Var}(X) = \text{Var}(\sum_{i=1}^{n} X_i)$. Since expected value is linear, we have

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = np.$$

Also, by (5.15) we have

$$\text{Var}(X) = \sum_{i=1}^{n} \text{Var}(X_i) = np(1-p). \quad \square$$

Figure 5.6 gives a plot of the p.m.f. of the binomial distribution for several values of p.

5.5.3 Poisson Distribution

The Poisson distribution is used to describe the number X of occurrences of an event in a given interval of time or space, where the interval is made up of many small subintervals in which the probability of an occurrence is low, and the occurrence of

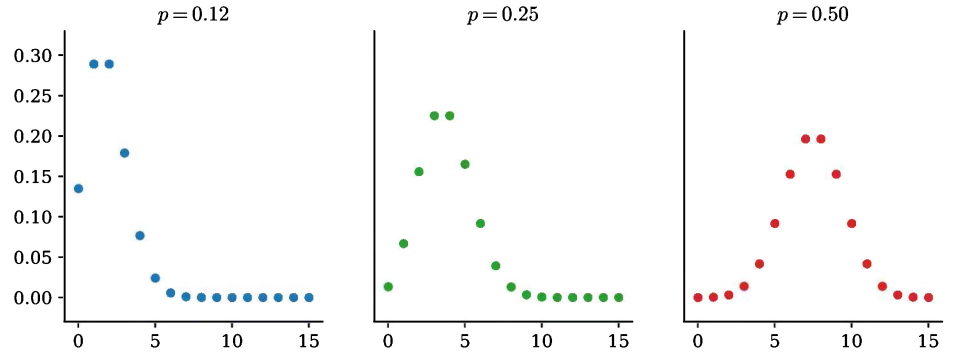


Figure 5.6. *Graphs of the p.m.f. $g_X(x)$ for the binomial distribution with $n = 15$ and with $p = \frac{1}{8}$, $p = \frac{1}{4}$, and $p = \frac{1}{2}$, respectively. For each value of x, the height of the point above x is the probability $P(X = x)$. In the leftmost graph, with $p = \frac{1}{8}$, the probability of 7 or more successes in 15 trials is essentially zero, but the probability of 1 success is almost 30%. The expected value is $\mathbb{E}[X] = 15p = \frac{15}{8} = 1.875$, which happens to lie between the two most likely values of $x = 1$ and $x = 2$. The variance is $15p(1-p) = \frac{105}{64} \approx 1.65$, corresponding to the fact that most of the probability (or mass) is concentrated within one or two units of the mean. In the rightmost graph, with $p = \frac{1}{2}$, the mass is more spread out, corresponding to the fact that $\mathrm{Var}(X) = \frac{15}{4} = 3.75$.*

an event in a given subinterval is essentially independent of the occurrence of any event in any other subinterval.

It is often used to describe situations like the number of radioactive particles that hit a detector in a second, the number of automobiles that arrive at an intersection in a minute, or the number of customers that come into a store in an hour.

We say that X has *Poisson distribution* with rate λ (denoted $X \sim \mathrm{Poisson}(\lambda)$) if the support is $\mathbb{N}$ and the p.m.f. is

$$g_X(x) = \frac{e^{-\lambda}\lambda^x}{x!}. \tag{5.18}$$

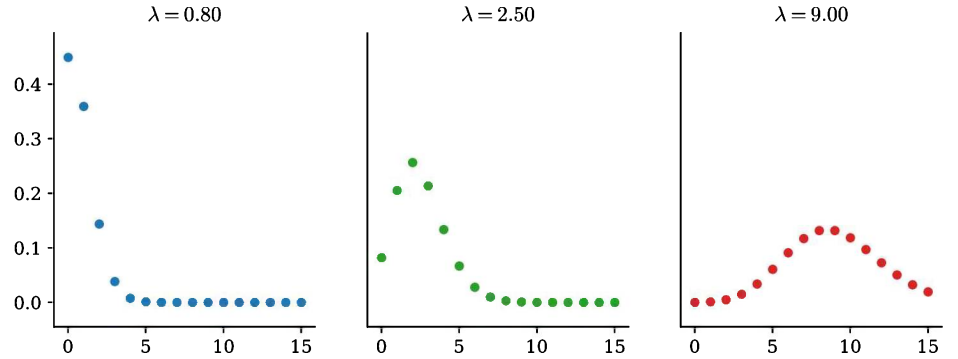


Figure 5.7. *Graphs of the p.m.f. for the Poisson distribution with $\lambda = 0.8$, $\lambda = 2.5$, and $\lambda = 9$, respectively.*

The expectation is

$$\mathbb{E}[X] = \sum_{k=0}^{\infty} \frac{ke^{-\lambda}\lambda^k}{k!} = \lambda e^{-\lambda} \sum_{k=1}^{\infty} \frac{\lambda^{k-1}}{(k-1)!} = \lambda e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^k}{k!} = \lambda e^{-\lambda} e^{\lambda} = \lambda,$$

and the variance is also equal to λ (see Exercise 5.32). Figure 5.7 gives a plot of the Poisson p.m.f. for several values of λ.

Example 5.5.6. Assume that the number X of customers buying a certain product at a website averages two per hour and that the customers buy independently of each other. In this situation we assume that X has a Poisson distribution. If the basic unit of time is an hour, then λ is 2, and the p.m.f. is

$$g_X(x) = \frac{e^{-2}2^x}{x!}.$$

Thus the probability that exactly four customers will buy the product in a given hour is $g_X(4) = \frac{e^{-2}2^4}{4!} = \frac{2}{3}e^{-2} \approx 0.0902$.

Example 5.5.7. We can also find the probability of a given number of events in a different time period. In a period of t units, the expected number of events is λt, so the probability of x events in a period of t units is

$$\frac{e^{-t\lambda}(t\lambda)^x}{x!}.$$

Thus in the previous example, with $\lambda = 2$ per hour, the probability of 10 customers buying the product in a 3-hour period is $e^{-6}(6)^{10}/10! \approx 0.0413$.

Example 5.5.8. The usefulness of the Poisson distribution is not limited to time intervals. For example, the number of chocolate chips in a chocolate chip cookie is a random variable that is approximately Poisson distributed. Here the intervals are intervals of volume rather than of time. The average number of chocolate chips in one cookie is proportional to the volume of the cookie, and the number in one cookie is essentially independent of the number in another cookie. Of course this is not precisely true, because the total number of chips used to make one batch of cookies is probably fixed, but if there are lots of cookies from one batch, having a few more chips in one cookie doesn't significantly affect the number in the next cookie.

If λ is the average number of chips per cubic centimeter of cookie, the number X of chips in a given cookie of t cubic centimeters is approximately Poisson distributed with

$$P(X = x) = \frac{e^{-t\lambda}(t\lambda)^x}{x!}.$$

The expected number of chips in a cookie of size t is $\mathbb{E}[X] = \lambda t$, and the variance is also $\mathrm{Var}(X) = \lambda t$.

5.5.4 *Negative Binomial Distribution

The negative binomial distribution with parameters $k \in \mathbb{Z}^+$ and $p \in [0, 1]$ describes the number of successes that occur in a sequence of repeated Bernoulli trials with parameter p before k failures occur. For example, in a sequence of independent games played against the same opponent, this distribution describes the number of games you win before you lose k times (see Examples 5.5.10 and 5.5.11).

We say that X has negative binomial distribution with parameters k and p (written $X \sim \mathrm{NegBin}(k, p)$) if the support is $\mathbb{N}$ and the p.m.f. is

$$g_X(x) = \binom{x+k-1}{x} p^x (1-p)^k. \tag{5.19}$$

Exercise 5.35 shows that the expected value and variance are

$$\mathbb{E}[X] = \frac{pk}{1-p} \qquad \text{and} \qquad \mathrm{Var}(X) = \frac{pk}{(1-p)^2}. \tag{5.20}$$

Figure 5.8 gives a graph of the negative binomial p.m.f. for several values of p.

Remark 5.5.9. The special case of $k = 1$ is usually called the *geometric distribution*.

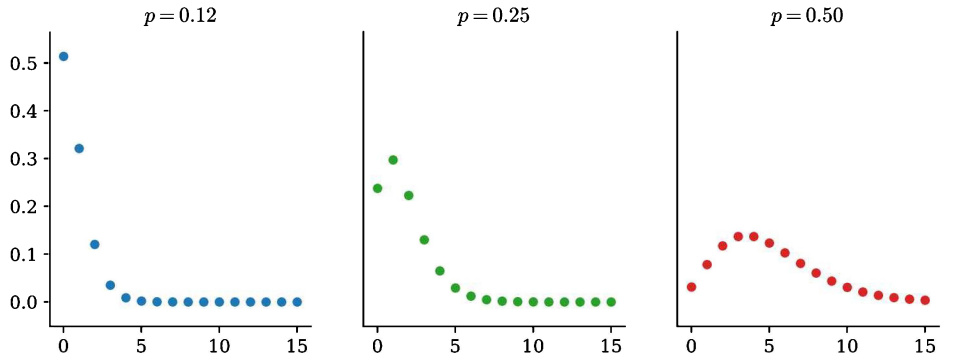


Figure 5.8. *Graphs of the p.m.f. $g_X(x)$ for the negative binomial distribution with $n = 15$, $k = 5$, and $p = \frac{1}{8}$, $p = \frac{1}{4}$, and $p = \frac{1}{2}$, respectively.*

Example 5.5.10. Blaise and Pierre play a game where they flip a coin repeatedly. Every time the coin comes up heads, Blaise gets a point, and every time the coin comes up tails, Pierre gets a point. They play the game until Pierre has k points. Assuming the probability of heads is p, what is the probability $g_X(x)$ that Blaise will have exactly x points when the game ends?

We begin with the case of $k = 1$. The only way for Blaise to have exactly x points when Pierre gets his first point is if the sequence $\underbrace{HH\ldots H}_{x}T$ occurs, and this has probability $p^x q$, where $q = 1 - p$. If $k = 2$, things are a little trickier. If the game ends with Pierre having x points, then the second T occurred on coin flip number $x+2$. Thus, the first $x+1$ flips consist of exactly x heads and one tail, taken in any order. This implies $g_X(x) = \binom{x+1}{x} p^x (1-p)^2$.

More generally, for arbitrary k, if Pierre ends the game with x points, then flip number $x + k$ must be T, and that means the remaining flips consist of x heads and $k - 1$ tails, taken in any order. Thus we have

$$g_X(x) = \binom{x+k-1}{x} p^x (1-p)^k.$$

Example 5.5.11. Experience shows that when you play tennis with your friend your probability of winning a set is $p = 0.55$. Your friend challenges you to a best-of-five tennis match, so you win the match if you win three sets before she wins three sets. Assuming that the outcome of each set is independent of the other sets, what is the probability that you will win the match?

One way to model this situation is with a negative binomial distribution with $k = 3$ (the number of sets your friend needs to win in order to beat you). You lose the match if the number x of your wins is two or fewer when she reaches $k = 3$. So we have

$$\begin{aligned} P(\text{you win}) &= 1 - \sum_{x=0}^{2} g_X(x) \\ &= 1 - \sum_{x=0}^{2} \binom{x+2}{x} p^x (1-p)^3 = 1 - (1-p)^3 \sum_{x=0}^{2} \frac{(x+2)(x+1)}{2} p^x \\ &= 1 - (1-p)^3(1 + 3p + 6p^2) \approx 0.59. \end{aligned}$$

Example 5.5.12. A marketer is having a promotion where shoppers get a special, randomly selected card every time they spend more than \$10. Cards come in 20 different types, and the probability of receiving any specific type of card is the same for every type. Anyone who gets a complete set, consisting

of one of each of the 20 different types, wins a big prize. You have already collected 19 different types. What is the expected number of \$10 purchases you need to make in order to get the last type of card and complete your set?

This is essentially a situation where you want to know how many times you will lose before you win once. Thus, if X is the number of purchases you must make, then $X = Y + 1$, where $Y \sim \text{NegBin}(1, \frac{19}{20})$. Here the parameter p is $(1 - \frac{1}{20}) = \frac{19}{20}$ because the roles of success and failure have been swapped in the story of the negative binomial distribution. Thus we have

$$P(X = x) = P(Y = x - 1) = \left(\frac{19}{20}\right)^{x-1} \frac{1}{20},$$

and the expected number of additional cards you need to acquire is

$$\mathbb{E}[X] = \sum_{x=1}^{\infty} x \left(\frac{19}{20}\right)^{x-1} \frac{1}{20} = \frac{1}{20}\frac{20}{19}\sum_{x=0}^{\infty} x \left(\frac{19}{20}\right)^{x} = \frac{1}{19}\frac{19/20}{(1/20)^2} = 20.$$

Here the penultimate equality follows from Exercise 1.20.

5.6 Continuous Random Variables

Although discrete probability spaces and random variables cover many situations, it is important to generalize the ideas of probability to the so-called *continuous* case—where the probability space and the images of random variables are not countable. We do this carefully and in full detail in Volume 3, but in this section we give a quick sketch of the ideas and how to work with continuous random variables.

An important difference between the discrete and continuous cases is that with discrete distributions every subset of the sample space has a well-defined probability, and for any subset E, the probability of E is the sum of the probabilities of the individual elements in E. But in the continuous case, the uncountable sums don't make sense and must be replaced with integrals, the probability of an individual point $\omega \in \Omega$ is usually zero, and not every subset has a well-defined probability.

5.6.1 Continuous Random Variables

If Ω is uncountable, then insisting that every subset of Ω has a well-defined probability is too restrictive. Instead we only require that probability be defined on a subcollection $\mathscr{F} \subset 2^{\Omega}$ such that $\mathscr{F}$ contains Ω and is closed under complements, countable unions, and countable intersections.[25]

For such a collection $\mathscr{F} \subset 2^{\Omega}$, the definitions of a probability measure and probability space (Definition 5.1.7) still make sense, and all of the basic properties still hold in this more general case. These include Proposition 5.1.10, the definition of independence, conditional probability, and Bayes' rule (Theorem 5.2.13).

[25] Such a collection is called a *σ-algebra* of sets.

Definition 5.6.1. *Let Ω be a set, and let $\mathscr{F} \subset 2^\Omega$ be a collection containing Ω that is closed under complements and countable unions.*[26] *A function $P : \mathscr{F} \to [0,1]$ is a* probability measure *if $P(\Omega) = 1$ and countable additivity (5.1) holds. In this case, the triple $(\Omega, \mathscr{F}, P)$ is called a* probability space.

We want to extend what we have done with discrete random variables to this more general setting. One problem is that the p.m.f. $g_X(x) = P(X^{-1}(x))$ is no longer very useful, because in the continuous case we usually have $P(X = x) = 0$ for all $x \in \mathbb{R}$. Instead we use something called the *cumulative distribution function* (c.d.f.). The c.d.f. of X gives the probability $P(X \leq x)$ that X will be no greater than a given amount x. But for an arbitrary function $X : \Omega \to \mathbb{R}$, talking about $P(X \leq x)$ might not make sense because the sets $X^{-1}((-\infty, x])$ might not have a well-defined probability if they aren't all in $\mathscr{F}$. This motivates the definition of a random variable to be a function for which these probabilities are always defined.

Definition 5.6.2. *A function $X : \Omega \to \mathbb{R}$ on a probability space $(\Omega, \mathscr{F}, P)$ is a* random variable *if $X^{-1}((-\infty, x]) \in \mathscr{F}$ for every $x \in \mathbb{R}$. The* cumulative distribution function *(c.d.f.) of X is the function $F_X : \mathbb{R} \to [0,1]$ given by*

$$F_X(a) = P(X \leq a) = P(X^{-1}(-\infty, a]).$$

In the case of a discrete probability space, we have $\mathscr{F} = 2^\Omega$, so every subset of Ω lies in $\mathscr{F}$ and every function $X : \Omega \to \mathbb{R}$ is a random variable. We discuss the details of the general case in Volume 3, but essentially every function that you are likely to encounter in applications will satisfy the conditions of Definition 5.6.2 and have a well-defined c.d.f.

Example 5.6.3. For a discrete distribution, the c.d.f. can be written as the sum

$$F_X(a) = P(X \leq a) = \sum_{x \leq a} g_X(x),$$

where the sum runs over all values of x in the range of X that are less than or equal to a. For example, if $X \sim \text{Binomial}(n, p)$, then

$$F_X(a) = \sum_{k=0}^{\lfloor a \rfloor} \binom{n}{k} p^k (1-p)^{n-k}.$$

If the range of X lies in $\mathbb{Z}$, then we can reconstruct the p.m.f. of X from the c.d.f. as

$$g_X(a) = F_X(a) - F_X(a-1) = \Delta F_X(a-1),$$

where Δ is the difference operator (see Definition 1.3.6 and Section 1.3.3). Thus, for distributions with range in $\mathbb{Z}$, the p.m.f. of X is a sort of discrete derivative of the c.d.f.

[26] If $\mathscr{F}$ is closed under complements and countable unions, then it is automatically also closed under countable intersections.

Proposition 5.6.4. *The c.d.f. $F_X(x)$ of any random variable $X : \Omega \to \mathbb{R}$ is nondecreasing and satisfies*

$$\lim_{x\to\infty} F_X(x) = 1 \qquad \textit{and} \qquad \lim_{x\to-\infty} F_X(x) = 0.$$

Proof. If $a < b$, then

$$F_X(a) = P(X \le a) \le P(X \le a) + P(X \in (a,b]) = P(X \le b) = F_X(b),$$

and therefore $F_X(x)$ is nondecreasing. Since $0 \le F_X(x) = P(X \le x) \le 1$ for all x, the function $F_X(x)$ must converge to a limit as $x \to \infty$, and it also must converge to a limit as $x \to -\infty$.

Let $B_0 = (-\infty, 0]$, and for every $n \in \mathbb{Z}^+$ let $B_n = (n-1, n]$. The sets B_n are pairwise disjoint and their union is $\mathbb{R}$. It is straightforward to check that the sets $X^{-1}(B_n)$ are mutually exclusive and collectively exhaustive. Therefore, we have

$$\lim_{n\to\infty} F_X(n) = \sum_{n=0}^{\infty} P(X \in B_n) = P\left(\bigcup_n B_n\right) = 1.$$

Finally, we have

$$\begin{aligned}\lim_{n\to-\infty} F_X(n) &= \lim_{k\to\infty} 1 - P(X > -k) \\ &= 1 - \lim_{k\to\infty} P(-X < k) \le 1 - \lim_{k\to\infty} F_{-X}(k-1) = 0. \quad \square\end{aligned}$$

Example 5.6.3 shows that the c.d.f. of a discrete distribution is discontinuous, skipping upward every time x becomes larger than the next point in the image of X. A *continuous distribution* is one whose c.d.f. is not only continuous but also continuously differentiable.

Definition 5.6.5. *A random variable X has a* continuous distribution *if its c.d.f. $F_X(x)$ is a continuously differentiable function of x, when restricted to the range of X. The derivative $f_X(x) = \frac{d}{dx}F_X(x) = F_X'(x)$ is called the* probability density function (p.d.f.) *of X.*

By the fundamental theorem of calculus, we have

$$F_X(b) = \int_{-\infty}^{b} f_X(x)\,dx$$

for any continuously distributed random variable X and for any $b \in \mathbb{R}$. Thus the area of the region bounded on the right by $x = b$ and lying under the curve $y = f_X(x)$ is the probability $P(X \le b)$; see Figure 5.9. Exercise 5.36 shows that

$$\int_{-\infty}^{\infty} f_X(x)\,dx = 1.$$

More generally, for any $a < b$ the probability $P(a < X \le b)$ is given by the integral

$$P(a < X \le b) = \int_a^b f_X(x)\,dx.$$

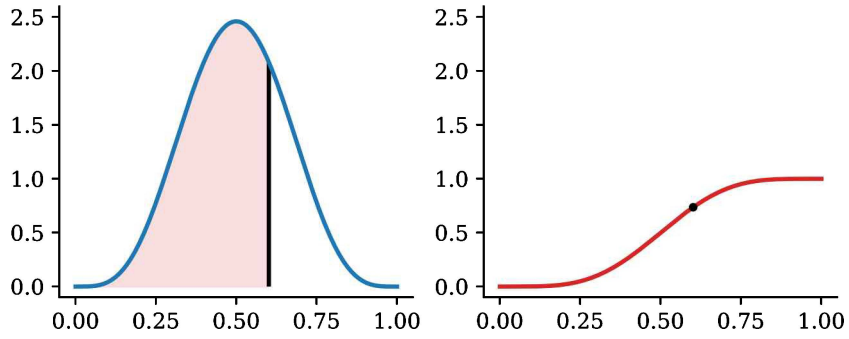


Figure 5.9. *Graphs of the p.d.f. f_X (left panel) and c.d.f. F_X (right panel) of a continuous distribution on $[0,1]$. The value $F_X(x)$ (the height of the black dot) at a point x (here $x = 0.6$) is the probability $P(X \leq x)$. This is the area (pink shaded) under the graph of f_X to the left of x. The value $f_X(x)$ is* not *the probability of $X = x$. That probability is always 0 for a continuous distribution. For more on this, see Definition 5.6.5.*

A useful analogy is the problem of computing the mass of a discrete collection of objects on a line, versus computing the mass of a solid rod of varying density. In either case the c.d.f. $F_X(b)$ gives the mass (probability) of everything to the left of the point $x = b$. In the discrete case, this amounts to adding up the mass (the p.m.f.) at the discrete points along the line to the left of b, but in the continuous case, the mass varies continuously and is the integral of the density function (the p.d.f.) up to b.

Remark 5.6.6. Just as with discrete distributions (see Remark 5.4.7), although the domain of a p.d.f. is actually $\mathbb{R}$, it often simplifies our formulas if we restrict the domain of f_X to be the range of X.

For a continuous random variable, the main change we must make from discrete random variables is the definition of expected value, where we replace the sum with an integral and the p.m.f. with the p.d.f.

Definition 5.6.7. *For a continuous random variable X with p.d.f. f_X, the* expectation *of X is*

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\, dx,$$

provided $\int_{-\infty}^{\infty} |x| f_X(x)\, dx$ is finite.

For a continuous random variable, all the other definitions and results we have proved for discrete random variables still hold, once we make the obvious changes, like replacing sums by integrals:

(i) Linearity of expectation:

$$\mathbb{E}[\alpha X + \beta Y] = \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y].$$

(ii) Law of the unconscious statistician: if h is continuous, then $h(X)$ is a continuous random variable and

$$\mathbb{E}[h(X)] = \int_{-\infty}^{\infty} h(x) f_X(x)\, dx.$$

(iii) Expectation of a product of independent random variables is a product:

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] \qquad \text{if } X \text{ and } Y \text{ are independent.}$$

Note the definition of independence in the continuous case also requires a substitution of p.d.f.s for probabilities in (5.12). See Definition 5.7.8 for details.

(iv) Variance:

$$\operatorname{Var}(X) = \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

(v) If X and Y are independent, then variance satisfies

$$\operatorname{Var}(\alpha X + \beta Y) = \alpha^2 \operatorname{Var}(X) + \beta^2 \operatorname{Var}(Y).$$

5.6.2 Some Important Continuous Distributions

In this subsection we describe a few important continuous distributions. The simplest is the *uniform distribution*, which describes a continuous version of equally likely outcomes. Possibly the most important distribution is the *normal* (or *Gaussian*) distribution. The last two distributions are the *gamma* and the *beta* distributions. These are especially important for Bayesian statistics (see Section 6.5), but they also occur in many other probability models.

As in the discrete case, we describe the p.d.f. of each distribution only on its support.[27]

Uniform Distribution

The uniform distribution is the continuous analogue of equally likely outcomes (see Section 5.1.2). A random variable X has *continuous uniform distribution on* $[a,b]$ if it has support $[a,b]$ and p.d.f.

$$f_X(x) = \frac{1}{b-a}\mathbb{1}_{[a,b]}(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a,b], \\ 0 & \text{otherwise.} \end{cases} \tag{5.21}$$

This gives

$$F_X(x) = P(X \le x) = \begin{cases} 0 & \text{if } x \le a, \\ \frac{x-a}{b-a} & \text{if } x \in [a,b], \\ 1 & \text{if } x \ge b. \end{cases}$$

We denote this by $X \sim \text{Uniform}([a,b])$. Exercise 5.37 shows that the mean and variance are

$$\mu = \frac{a+b}{2} \qquad \text{and} \qquad \sigma^2 = \frac{(b-a)^2}{12}. \tag{5.22}$$

[27] The *support* of a continuous distribution X is the support of the p.d.f. f_X of the distribution. As in the discrete case, this is a subset of the domain of f_X and a subset of the range of X.

Normal Distribution

The normal distribution is among the most important of all distributions. This is primarily due to its appearance in the central limit theorem (Theorem 6.3.1) and its appearance in nature. The normal distribution is typically used to describe a value that depends on a number of other random factors, like measurement error, or height or weight of a randomly chosen person.

A random variable X is *normally distributed* if it has support $(-\infty, \infty)$ and there exist $\mu \in \mathbb{R}$ and $\sigma > 0$ such that the p.d.f. of X is

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{5.23}$$

We denote this by $X \sim \mathcal{N}(\mu, \sigma^2)$. This gives

$$F_X(x) = P(X \leq x) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt.$$

The mean is μ and the variance is σ^2. If $\mu = 0$ and $\sigma = 1$, then X has a *standard* normal distribution.

Figure 5.10 shows graphs of the p.d.f. for the normal distribution with $\mu = 0$ and several values of σ.

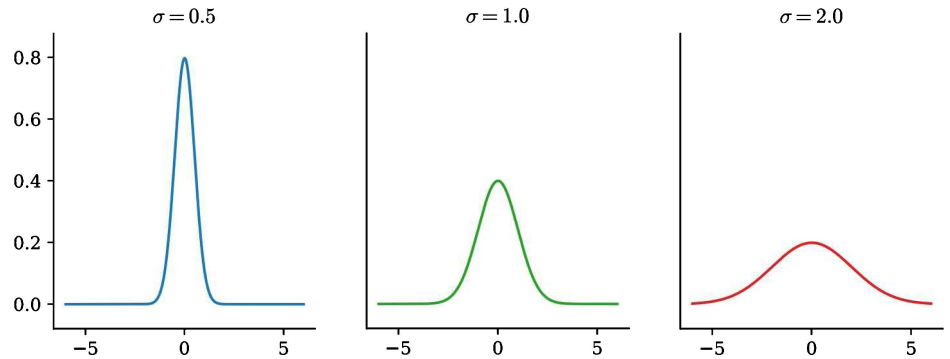


Figure 5.10. *Graphs of the p.d.f. $f_X(x)$ for the normal distribution with $\mu = 0$ and $\sigma = 0.5$, $\sigma = 1.0$, and $\sigma = 2.0$, respectively. The total probability (area under the curve, or mass) is always 1, but as the variance σ^2 increases, the probability becomes more spread out, and as the variance goes toward zero, the probability is increasingly concentrated near the mean.*

Gamma Distribution

The gamma distribution describes the waiting time for at least $a > 0$ events to occur in a homogeneous Poisson process of rate $b > 0$. The support is $[0, \infty)$ and the p.d.f. is

$$f_X(t) = \frac{b^a t^{a-1} e^{-tb}}{\Gamma(a)} \tag{5.24}$$

for all $t \in [0, \infty)$. We denote this by $X \sim \text{Gamma}(a, b)$. The parameter a is called the *shape*, and the parameter b is called the *rate*. It is also common to specify the gamma distribution in terms of a *scale* parameter $\theta = 1/b$. The scale θ corresponds to the average wait time for a single event. The mean and variance of the gamma distribution are

$$\mu = \frac{a}{b} \qquad \text{and} \qquad \sigma^2 = \frac{a}{b^2}.$$

Remark 5.6.8. In the special case that $a = 1$, the gamma distribution is usually called the *exponential distribution*. When $a = \frac{n}{2}$ and $b = \frac{1}{2}$ for an integer n, the gamma distribution is called the *chi-squared distribution* with n *degrees of freedom*. The chi-squared distribution is important for estimating the variance of a sample because it is the distribution of the sum of the squares of n independent, standard normal random variables.

Figure 5.11 shows graphs of the p.d.f. for the gamma distribution with several values of (a, b).

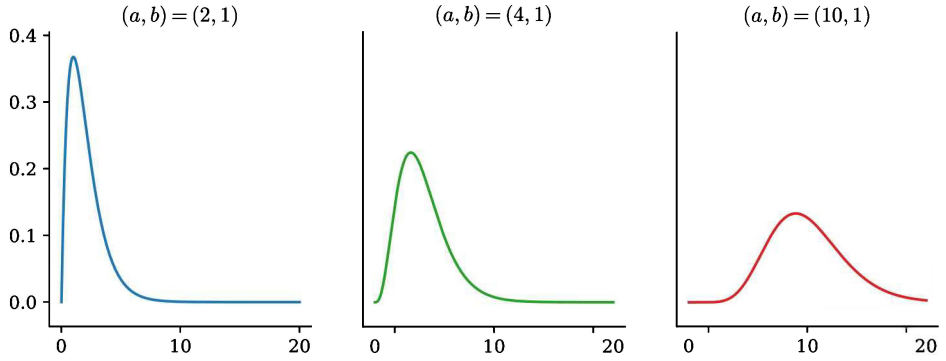


Figure 5.11. *Graphs of the p.d.f. $f_X(t)$ for the gamma distribution with $b = 1$ and $a = 2$, 4, and 10, respectively.*

Beta Distribution

The beta distribution is an important family of distributions defined on the unit interval $[0, 1]$ (the uniform distribution is a special case). It plays an important role in Bayesian statistics, especially as a way of describing the distribution of the parameter p for a Bernoulli or binomial random variable; see Section 6.5 for more on this.

A random variable X taking all its values in the interval $[0, 1]$ has distribution $\text{Beta}(a, b)$ if its p.d.f. is

$$f_X(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}, \qquad a, b > 0, \tag{5.25}$$

defined on the interval $0 \leq x \leq 1$. This is denoted by $X \sim \text{Beta}(a, b)$. The mean and variance of $\text{Beta}(a, b)$ are

$$\mu = \frac{a}{a+b} \qquad \text{and} \qquad \sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}. \tag{5.26}$$

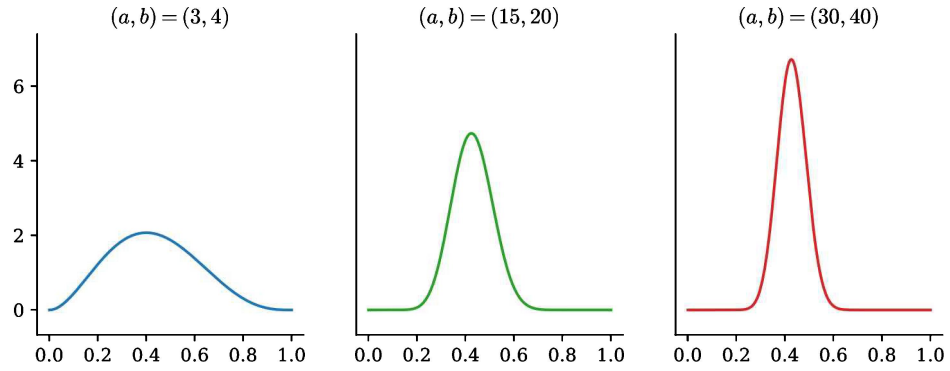


Figure 5.12. *Graphs of the p.d.f. $f_X(x)$ for the beta distribution with $(a,b) = (3,4)$, $(a,b) = (15,20)$, and $(a,b) = (30,40)$, respectively. The mean is always $\mu = \frac{a}{a+b} = \frac{3}{7}$, but as a and b get larger, the variance gets smaller, and the peak gets taller and narrower.*

Given independent random variables A and B having gamma distributions with parameters α, θ and β, θ, respectively, the random variable

$$Z = \frac{A}{A+B}$$

has distribution Beta(α, β). Alternatively, a draw from a beta distribution Beta$(k, n+1-k)$ comes by drawing n numbers from the uniform distribution on $[0,1]$ and then ordering them and taking the kth smallest number.

Figures 5.12 and 5.13 show the graphs of the p.d.f. for the beta distribution with various values of a and b.

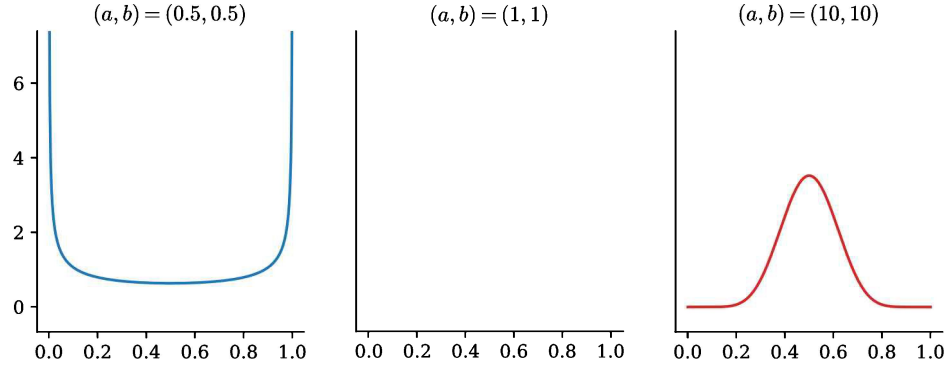


Figure 5.13. *Graphs of the p.d.f. $f_X(x)$ for the beta distribution with $a = b$ for various values of a. The mean for all of these is $\mu = \frac{a}{a+b} = 0.5$, but as a and b get larger, the variance gets smaller. When $a = b = 0.5$ (blue), the distribution has a peak at either end of infinite height. When $a = b = 1$ (green), this is the uniform distribution on $[0,1]$. When $a = b = 10$ (red), this has a single maximum at 0.5.*

5.7 Multivariate Random Variables

We often need to think about collections of several random variables $X_1, \ldots, X_n$ at once. We can think of such a collection as a single function from Ω to $\mathbb{R}^n$, that is, as a *multivariate random variable*. In this section we discuss some basic properties of multivariate random variables and some important examples of multivariate distributions.

5.7.1 Multivariate Random Variables

Definition 5.7.1. *A function $X : \Omega \to \mathbb{R}^n$ on a probability space $(\Omega, \mathscr{F}, P)$ with $X = (X_1, \ldots, X_n)$ is a* multivariate random variable *if every coordinate function $X_i : \Omega \to \mathbb{R}$ is a random variable. If $n = 2$, we call the random variable* bivariate*, and when $n = 1$ (that is, when X is a random variable in the sense of Definitions 5.4.1 and 5.6.2), we call the random variable* univariate.

Example 5.7.2. Consider a bag filled with blue, white, and red balls that are all identical except for their color. Assume there are 30 blue balls, 40 white balls, and 15 red balls. If we choose a ball at random from the bag, then the probability of choosing blue is $p_B = \frac{30}{85}$, choosing white $p_W = \frac{40}{85}$, and choosing red $p_R = \frac{15}{85}$.

If a ball is drawn three times with replacement (with the ball being returned to the bag after each draw), we can model this with the discrete probability space

$$\Omega = \{\text{BBB}, \text{BBW}, \ldots, \text{RRB}, \text{RRR}\},$$

where $P(\text{BBB}) = p_B^3$, $P(\text{BBW}) = p_B^2 p_W$, ... Define a multivariate random variable $X : \Omega \to \mathbb{R}^3$ by $X = (X_B, X_W, X_R)$ with X_B equal to the number of times that blue is drawn, X_W the number of times that white is drawn, and X_R the number of times that red is drawn. Therefore $X(\text{BBB}) = (3, 0, 0)$, $X(\text{BWB}) = (2, 1, 0)$, and so forth. The range of X is the set of all triples $\mathbf{x} = (x_B, x_W, x_R) \in \mathbb{N}^3$ with $x_B + x_W + x_R = 3$. This is an example of what is called the *multinomial distribution*.

Each of the individual coordinates X_B, X_W, and X_R is itself a random variable. We have $X_B \sim \text{Binomial}(3, p_B)$, since X_B counts the number of times that blue occurs (success) versus any other color (failure). The other coordinates are also binomially distributed.

5.7.2 Density, Mass, and Distribution Functions

As in the single-variable case, the ideas of probability mass functions, cumulative distribution functions, and probability density functions are very useful.

Definition 5.7.3. *If $(\Omega, \mathscr{F}, P)$ is a discrete probability space and $X : \Omega \to \mathbb{R}^n$ is a multivariate random variable, then the function $g_X(\mathbf{x}) = P(X = \mathbf{x})$ is called the* joint probability mass function *of the univariate random variables $X_1, \ldots, X_n$*

or just the probability mass function (p.m.f.) *of the multivariate random variable X. For a multivariate random variable $X = (X_1, \ldots, X_n)$ on a general probability space $(\Omega, \mathscr{F}, P)$, the function*

$$F_X(x_1, \ldots, x_n) = P(X_1 \leq x_1, \ldots, X_n \leq x_n)$$

is called the joint cumulative distribution function *of the univariate random variables $X_1, \ldots, X_n$ or the* cumulative distribution function *of the multivariate random variable X. An integrable function $f_X : \mathbb{R}^n \to \mathbb{R}$ is a* joint probability density function *for the random variables $X_1, \ldots, X_n$ and f_X is a* probability density function (p.d.f.) *for the multivariate random variable $X = (X_1, \ldots, X_n)$ if*

$$F_X(x_1, \ldots, x_n) = \int_{-\infty}^{x_n} \cdots \int_{-\infty}^{x_1} f_X(t_1, \ldots, t_n)\, dt_1 \cdots dt_n.$$

Example 5.7.4. The random variable X in Example 5.7.2 has p.m.f.

$$g_X(\mathbf{x}) = \frac{3!}{x_B! x_W! x_R!} p_B^{x_B} p_W^{x_W} p_R^{x_R}$$

for any triple $\mathbf{x} = (x_B, x_W, x_R) \in \mathbb{N}^3$ with $x_B + x_W + x_R = 3$. This is an example of the *multinomial distribution* (see Section 5.7.5).

The value of the joint p.m.f. $g_{X,Y}(x, y)$ of two random variables X and Y corresponds to the probability $P(X = x, Y = y)$ that $Y = y$ and $X = x$. But we can use this to find the probability that $X = x$, with no constraints on Y:

$$P(X = x) = \sum_y P(X = x, Y = y) = \sum_y g_{X,Y}(x, y).$$

This motivates the following definition.

Definition 5.7.5. *If X is a discrete multivariate random variable with p.m.f. $g_X(\mathbf{x})$, then the* marginal p.m.f. *$g_i(a)$ at a is the sum of the joint p.m.f. over all values of $\mathbf{x}$ with the ith coordinate equal to a:*

$$g_i(a) = \sum_{\mathbf{x}: x_i = a} g_X(\mathbf{x}).$$

Similarly, if X is a continuous multivariate random variable with probability density function (p.d.f.) f_X, then for each $i \in \{1, \ldots, n\}$ the marginal p.d.f. *f_i is the integral of the joint p.d.f. $f_X(t_1, \ldots, t_{i-1}, a, t_{i+1}, \ldots, t_n)$ over all values of $\mathbf{t}$ with the ith coordinate equal to a:*

$$\begin{aligned} f_i(a) &= \int_{\mathbb{R}^{i-1} \times \{a\} \times \mathbb{R}^{n-i}} f_X(t_1, \ldots, t_{i-1}, a, t_{i+1}, \ldots, t_n)\, d\mathbf{t} \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_X(t_1, \ldots, t_{i-1}, a, t_{i+1}, \ldots, t_n)\, dt_1 \cdots dt_{i-1} dt_{i+1} \cdots dt_n. \end{aligned}$$

Remark 5.7.6. The name *marginal* comes from the fact that in early statistics books the p.m.f. of a bivariate distribution X was often written out as a table. Summing all entries in each column and writing the sum in the top margin of the table gives a row of values corresponding to the p.m.f. of the coordinate X_1. Similarly, summing all entries in each row and writing the sum in the right margin gives a column of values corresponding to the p.m.f. of the coordinate X_2.

Proposition 5.7.7. *If $X = (X_1, \ldots, X_n)$ is a random variable with p.d.f. f_X, then for each $i \in \{1, \ldots, n\}$ the marginal p.d.f. f_i is the p.d.f. of X_i:*

$$f_i = f_{X_i}.$$

Proof. The proof in the discrete case is Exercise 5.46. The continuous case follows from standard properties of multivariable integration (see Volume 1, Chapter 8). □

As noted earlier, the definition of independence used for discrete random variables (Definition 5.4.17) is not quite correct in the continuous case, but the definition is analogous, with appropriate p.d.f.s substituted for the probabilities in (5.12). We now have everything we need to give the correct definition.

Definition 5.7.8. *Two continuous random variables X and Y are* independent *if their joint p.d.f. factors as the product of the marginals:*

$$f_{X,Y}(x, y) = f_X(x) f_Y(y) \qquad \forall x \text{ and } y. \tag{5.27}$$

5.7.3 Expected Value

The expected value of a multivariate random variable is a straightforward generalization of the univariate case.

Definition 5.7.9. *The expected value $\mathbb{E}[X]$ of a discrete random variable $X : \Omega \to \mathbb{R}^n$ with p.m.f. $g_X(\mathbf{x})$ is*

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) P(\omega) = \sum_{\mathbf{x} \in \mathbb{R}^n} \mathbf{x} g_X(\mathbf{x}) = \begin{bmatrix} \sum_{\mathbf{x} \in \mathbb{R}^n} x_1 g_X(\mathbf{x}) \\ \sum_{\mathbf{x} \in \mathbb{R}^n} x_2 g_X(\mathbf{x}) \\ \vdots \\ \sum_{\mathbf{x} \in \mathbb{R}^n} x_n g_X(\mathbf{x}) \end{bmatrix}.$$

The expected value of a continuous random variable $X : \Omega \to \mathbb{R}^n$ with p.d.f. $f_X(\mathbf{x})$ is

$$\mathbb{E}[X] = \int_{\mathbb{R}^n} \mathbf{x} f_X(\mathbf{x})\, d\mathbf{x} = \begin{bmatrix} \int_{\mathbb{R}^n} x_1 f_X(\mathbf{x})\, d\mathbf{x} \\ \int_{\mathbb{R}^n} x_2 f_X(\mathbf{x})\, d\mathbf{x} \\ \vdots \\ \int_{\mathbb{R}^n} x_n f_X(\mathbf{x})\, d\mathbf{x} \end{bmatrix}.$$

Linearity of expectation follows immediately from linearity of summation and integration.

Proposition 5.7.10. *Expected value for functions of multivariate random variables is linear; that is, if $(\Omega, \mathscr{F}, P)$ is a probability space and if $X, Y : \Omega \to \mathbb{R}^n$ are two multivariate random variables, then for any constants $a, b \in \mathbb{R}$, the expected value of $aX + bY$ is*

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y].$$

In addition to the expected value $\mathbb{E}[X]$ of a random variable, we can also compute the expected value $\mathbb{E}[X_i]$ of each coordinate X_i. These are related in the most natural way.

Proposition 5.7.11. *For any multivariate random variable $X = (X_1, \ldots, X_n)$, the expected value satisfies $\mathbb{E}[X] = (\mathbb{E}[X_1], \ldots, \mathbb{E}[X_n])$.*

Proof. The proof is Exercise 5.48. □

Example 5.7.12. The colored-ball random variable X of Example 5.7.2 has expected value

$$\mathbb{E}[X] = \sum_{\mathbf{x}} \begin{bmatrix} x_B \\ x_W \\ x_R \end{bmatrix} P(\mathbf{x}),$$

which may seem a little painful to compute. But by Proposition 5.7.11 we have

$$\mathbb{E}[X] = \begin{bmatrix} \mathbb{E}[X_B] \\ \mathbb{E}[X_W] \\ \mathbb{E}[X_R] \end{bmatrix}.$$

Each of the coordinate random variables X_B, X_W, and X_R is binomially distributed with probability p_B, p_W, and p_R, respectively, from which we conclude that

$$\mathbb{E}[X] = \begin{bmatrix} 3p_B \\ 3p_W \\ 3p_R \end{bmatrix} = \begin{bmatrix} 90/85 \\ 120/85 \\ 45/85 \end{bmatrix}.$$

5.7.4 Covariance

Variance of a single random variable tells us about how much the variable fluctuates around the mean. When comparing two random variables, it is important to get an idea of how much or how little they are constrained to move together. The main tool for measuring this is the *covariance*.

Definition 5.7.13. *If X and Y are univariate random variables on a probability space Ω with expected value μ_X and μ_Y, respectively, then the* covariance *of X and Y is the quantity*

$$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)].$$

More generally, if $Z : \Omega \to \mathbb{R}^n$ is a multivariate random variable, then the covariance matrix *of Z is the following symmetric matrix:*

$$\Sigma = \begin{bmatrix} \operatorname{Cov}(Z_1, Z_1) & \operatorname{Cov}(Z_1, Z_2) & \dots & \operatorname{Cov}(Z_1, Z_n) \\ \operatorname{Cov}(Z_2, Z_1) & \operatorname{Cov}(Z_2, Z_2) & \dots & \operatorname{Cov}(Z_2, Z_n) \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{Cov}(Z_n, Z_1) & \operatorname{Cov}(Z_n, Z_2) & \dots & \operatorname{Cov}(Z_n, Z_n) \end{bmatrix}.$$

It is convenient to write $\boldsymbol{\mu}_Z$ as a column vector $\boldsymbol{\mu}_Z = \begin{bmatrix} \mu_1 & \dots & \mu_n \end{bmatrix}^\mathsf{T}$ and write

$$\Sigma = \mathbb{E}[(Z - \boldsymbol{\mu}_Z)(Z - \boldsymbol{\mu}_Z)^\mathsf{T}],$$

where the expected value of the $n \times n$ matrix $(Z - \mu_Z)(Z - \mu_Z)^\mathsf{T}$ is taken entry by entry.

Example 5.7.14. If Y and Z are independent univariate random variables on Ω, then we can show that the covariance $\operatorname{Cov}(Y, Z)$ is zero as follows. We define new random variables $Y' = Y - \mu_Y$ and $Z' = Z - \mu_Z$. These are independent because X and Y are independent. Moreover, we have $\mathbb{E}[Y'] = \mathbb{E}[Z'] = 0$. We compute

$$\operatorname{Cov}(Y, Z) = \mathbb{E}[(Y - \mu_Y)(Z - \mu_Z)] = \mathbb{E}[Y'Z'] = \mathbb{E}[Y']\mathbb{E}[Z'] = 0.$$

Nota Bene 5.7.15. The previous example shows that independent random variables X and Y have $\operatorname{Cov}(X, Y) = 0$, but beware that the converse is false. For example, if X is uniformly distributed on $[-1, 1]$ and $Y = X^2$, then X and Y are not independent, but $\operatorname{Cov}(X, Y) = \mathbb{E}[X^3] - \mathbb{E}[X]\mathbb{E}[Y] = 0$.

Proposition 5.7.16. *For any multivariate random variable X and for any $i \neq j$ the covariance satisfies*

$$\operatorname{Cov}(X_i, X_j) = \mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j].$$

Proof. The proof is Exercise 5.49. □

Remark 5.7.17. The covariance of a univariate random variable Y with itself is the variance of Y. Therefore the covariance matrix of a multivariate random variable X can be written as

$$\Sigma = \begin{bmatrix} \operatorname{Var}(X_1) & \operatorname{Cov}(X_1, X_2) & \dots & \operatorname{Cov}(X_1, X_n) \\ \operatorname{Cov}(X_2, X_1) & \operatorname{Var}(X_2) & \dots & \operatorname{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{Cov}(X_n, X_1) & \operatorname{Cov}(X_n, X_2) & \dots & \operatorname{Var}(X_n) \end{bmatrix}.$$

Example 5.7.18. To compute the covariance matrix for the three-colored-ball random variable X of Example 5.7.2, we first use the fact that the coordinates are binomially distributed (see Example 5.7.12) to find the variance for each coordinate; see Proposition 5.5.5.

$$\mathrm{Var}(X_B) = 3p_B(1-p_B), \quad \mathrm{Var}(X_W) = 3p_W(1-p_W), \quad \mathrm{Var}(X_R) = 3p_R(1-p_R).$$

To compute the covariance $\mathrm{Cov}(X_B, X_W) = \mathbb{E}[X_B X_W] - \mathbb{E}[X_B]\mathbb{E}[X_W]$, we need

$$\mathbb{E}[X_B X_W] = \sum_{x_B+x_W+x_R=3} x_B x_W \frac{3!}{x_B! x_W! x_R!} p_B^{x_B} p_W^{x_W} p_R^{x_R}.$$

Most of the terms are zero (whenever $x_B = 0$ or $x_W = 0$), so this becomes

$$\begin{aligned}\mathbb{E}[X_B X_W] &= 6p_B^2 p_W^1 p_R^0 + 6p_B^1 p_W^2 p_R^0 + 6p_B p_W p_R \\ &= 6p_B p_W (p_B + p_W + p_R) = 6p_B p_W.\end{aligned}$$

Assembling all these pieces gives

$$\mathrm{Cov}(X_B, X_W) = -3p_B p_W.$$

The values of $\mathrm{Cov}(X_B, X_R)$ and $\mathrm{Cov}(X_W, X_R)$ are computed similarly. As described in Remark 5.7.17, the covariance matrix Σ can now be assembled from these values and the variances computed at the beginning of this example.

Proposition 5.7.19. *Let $X = (X_1, \ldots, X_n)$ be a multivariate random variable $X : \Omega \to \mathbb{R}^n$ with covariance matrix Σ. Given $\mathbf{a} \in \mathbb{R}^n$, let Y be the univariate random variable $Y = \mathbf{a}^\mathsf{T} X$. We have $\mathrm{Var}(Y) = \mathbf{a}^\mathsf{T}\Sigma\mathbf{a}$.*

Proof. The proof is Exercise 5.50. □

Since $\mathrm{Var}(Y)$ is always nonnegative for any random variable Y, we see that the covariance matrix is always positive semidefinite.

Corollary 5.7.20. *If $X : \Omega \to \mathbb{R}^n$ is a random variable with covariance matrix Σ, then $\mathbf{a}^\mathsf{T}\Sigma\mathbf{a} \geq 0$ for all $\mathbf{a}$.*

Since Σ is symmetric, $\mathbb{R}^n$ has an orthonormal basis of eigenvectors of Σ that diagonalize it, that is, there exists an orthonormal matrix U whose columns are eigenvectors of Σ such that if $Y = \begin{bmatrix} Y_1 & \ldots & Y_n \end{bmatrix}^\mathsf{T} = UX$, then $U^\mathsf{T}\Sigma U = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2)$ is diagonal (see Volume 1, Section 4.3). By Proposition 5.7.19, this means that the new random variables $Y_1, \ldots, Y_n$ have the property that σ_i^2 is the variance of Y_i, for each $i \in \{1, \ldots, n\}$, while $\mathrm{Cov}(Y_i, Y_j) = 0$ for all $i \neq j$. But recall from Nota Bene 5.7.15 that this does not necessarily imply that the new variables are independent.

5.7.5 Common Multivariate Distributions

Multinomial Distribution

The multinomial distribution is a generalization of the binomial distribution that counts the outcomes of a sequence of n repeated (independent, identically distributed) trials of an experiment where exactly one of k outcomes can occur,[28] and for each j, the jth outcome has probability p_j, with $\sum_{j=1}^{k} p_j = 1$. The ith coordinate X_i represents the number of experiments that had result i. The three-colored-balls examples discussed throughout this section has a multinomial distribution with $n = k = 3$.

The range of a random variable with multinomial distribution consists of k-tuples of nonnegative integers $(x_1, x_2, \ldots, x_k)$, satisfying $\sum_{i=1}^{k} x_i = n$. The p.m.f. is

$$g_X(x_1, x_2, \ldots, x_k) = \binom{n}{x_1, x_2, \ldots, x_k} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k}. \tag{5.28}$$

The coordinate X_i is binomially distributed, with probability of success p_i and probability of failure $1 - p_i = \sum_{j \neq i} p_j$, so the marginal p.m.f. of X_i is

$$g_{X_i}(x) = \binom{n}{x} p_i^x (1 - p_i)^{n-x}.$$

The expected value of X is

$$\mathbb{E}[X] = (\mathbb{E}[X]_1, \ldots, \mathbb{E}[X]_k) = (np_1, \ldots, np_k)$$

as calculated in Proposition 5.5.5.

To calculate the covariance matrix of X, first note that since X_i is binomially distributed, we must have $\operatorname{Var}(X_i) = np_i(1 - p_i)$; see Proposition 5.5.5. To find the covariance $\operatorname{Cov}(X_i, X_j)$ for $i \neq j$, write each X_i as a sum $X_i = \sum_{\ell=1}^{n} Y_{i,\ell}$, where $Y_{i,\ell}$ is 1 if the ℓth trial results in outcome i and 0 otherwise. Since covariance is linear in each coordinate, we have

$$\operatorname{Cov}(X_i, X_j) = \sum_{\ell=1}^{n} \sum_{m=1}^{n} \operatorname{Cov}(Y_{i,\ell}, Y_{j,m}). \tag{5.29}$$

If $\ell \neq m$, then $Y_{i,\ell}$ and $Y_{j,m}$ are independent and hence have zero covariance. If $\ell = m$, then $\mathbb{E}[Y_{i,m} Y_{j,m}] = 0$, because the outcome of the mth trial cannot be simultaneously both i and j. This gives

$$\operatorname{Cov}(Y_{i,m}, Y_{j,m}) = \mathbb{E}[Y_{i,m} Y_{j,m}] - \mathbb{E}[Y_{i,m}]\mathbb{E}[Y_{j,m}] = -p_i p_j.$$

Combining this with (5.29) gives $\operatorname{Cov}(X_i, X_j) = \sum_{m=1}^{n} -p_i p_j = -np_i p_j$, and hence the covariance matrix of X is

$$\Sigma = n \begin{bmatrix} p_1(1-p_1) & -p_1 p_2 & \cdots & -p_1 p_n \\ -p_2 p_1 & p_2(1-p_2) & \cdots & -p_2 p_n \\ \vdots & & \ddots & \vdots \\ -p_n p_1 & -p_n p_2 & \cdots & p_n(1-p_n) \end{bmatrix}.$$

[28] A single one of these trials has a *categorical distribution* with parameters $(p_1, \ldots, p_k)$.

Multivariate Normal Distribution

The multivariate normal distribution is a generalization of the single-variable normal distribution and has p.d.f.

$$f_X(x_1, \ldots, x_n) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where Σ is a positive definite $n \times n$ matrix, and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n) \in \mathbb{R}^n$. If a random variable X has this distribution, we denote this by $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. A straightforward, but tedious, calculation shows that $\mathbb{E}[X] = \boldsymbol{\mu}$ and the covariance matrix of X is Σ.

Any coordinate X_i of a multivariate normal random variable X is itself a normally distributed random variable. Moreover, a linear combination $\mathbf{a}^{\mathsf{T}}X$ of the coordinates is normally distributed for any $\mathbf{a} \in \mathbb{R}^n$.

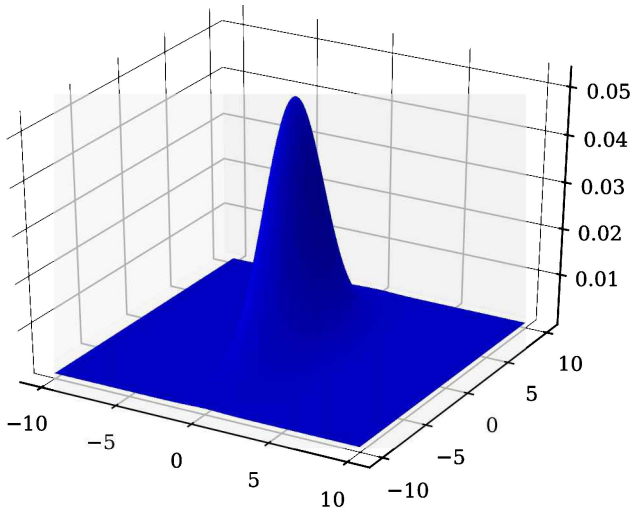


Figure 5.14. *Graph of the p.d.f.* $f_X(x)$ *for the bivariate normal distribution with* $\boldsymbol{\mu} = \mathbf{0}$ *and* $\Sigma = \mathrm{diag}(3, 1)$.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text.

Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

5.1. Consider an experiment where a six-sided die is rolled twice, and for each of the two rolls we record the number showing, modulo 3, that is, whether the number is congruent to 0, 1, or 2 (mod 3).

(i) Describe the sample space Ω of all possible outcomes of the experiment.

(ii) Describe the event "neither roll is congruent to 0 mod 3" as an element of the power set of Ω.

(iii) Describe the discrete probability measure on the power set of Ω if the die is fair. What is the probability of the event "neither roll is congruent to 0 mod 3" in this case?

(iv) Describe the discrete probability measure on the power set of Ω if the die is weighted so that the probability of rolling a 1 is $\frac{2}{7}$, while the probability of rolling any other number is $\frac{1}{7}$. What is the probability of the event "neither roll is congruent to 0 (mod 3)" in this case?

5.2. If you draw 5 cards from a standard deck of 52 cards:

(i) What is the probability of getting at least three of a kind (so a full house or four of a kind might also occur)?

(ii) What is the probability of getting exactly two distinct pairs (not a full house nor four of a kind)?

(iii) What is the probability of getting a flush (all cards are of the same suit)?

5.3. A box contains 10 different pairs of shoes. If 8 shoes are drawn at random, what is the probability that there is at least one matching pair of shoes? What is the probability that there is exactly one pair of shoes?

5.4. Assume you're in a room of n randomly selected people. Assume that birth dates are uniformly distributed among days of the year.

(i) What is the probability that exactly two people have the same birthday and everyone else has a distinct birthday?

(ii) What is the probability that exactly three people have the same birthday and everyone else has a distinct birthday?

(iii) What is the probability that there are exactly two pairs of people who have the same birthday?

5.5. Suppose that the probability of a married couple having $n \geq 1$ children is ap^n, where $a \leq \frac{1-p}{p}$. What is the probability that a couple has no children?

5.6. ⚠ Let $(\Omega, \mathscr{F}, P)$ be a discrete probability space and let $\{B_i\}_{i \in I}$ be a collection of elements of $\mathscr{F}$ indexed by a finite or countable set I, such that

$\{B_i\}_{i\in I}$ is a partition of Ω (so $\Omega = \bigcup_{i\in I} B_i$ and $B_i \cap B_j = \emptyset$ for all $i \neq j$). Prove that for any $A \in \mathscr{F}$, we have

$$P(A) = \sum_{i\in I} P(A \cap B_i).$$

5.7. Five of my friends come to dinner and take their coats off at the door when they arrive, and my ever-helpful son puts the coats away in his room. The guests leave one at a time, and when each one leaves, my son brings back a random coat (selected uniformly) and gives it to them. Since the guests are in a hurry, they each put on the coat given to them without noticing whether it is correct, and then they leave.

(i) What is the probability that the first guest to leave gets the right coat?

(ii) If the first to leave gets the right coat, what is the probability that the second to leave will get the right coat?

(iii) What is the probability that every guest will get the right coat?

(iv) If the first guest to leave gets the coat belonging to the second guest, what is the probability that the second guest will get the right coat?

(v) If the first coat is wrong, but it is also not the second guest's coat, what is the probability that the second guest will get the right coat?

(vi) Without knowing the outcome of the first coat, what is the probability that the second coat will be right? Hint: Consider using the law of total probability.

5.8. Yann and Zoë like to play racquetball, and they both have killer serves. These serves are so hard to return that whoever serves first is much more likely to win than the person who serves second. Let F be the event that Zoë serves first and W be the event that Zoë wins the game. Assume that $P(W \mid F) = 0.65$ and $P(W \mid F^c) = 0.45$. Because of the advantage of serving first, they agree to flip a fair coin to decide who will serve first. Assuming that Zoë wins the game, what is the probability that she won the coin toss for that game?

5.9. In the popular 1970s TV game show *Let's Make a Deal*, a contestant would choose one of three doors to open for a prize. Behind one of the doors was a car and behind the other two were goats. After the contestant picked a door (without opening it), the host, Monty Hall, would open one of the remaining two doors, revealing a goat, and then ask whether the contestant wanted to stay with the first choice or change doors. Show that the probability of winning is much better if the contestant changes to the other unopened door instead of sticking with the original choice. What are these probabilities? Hint: The problem would be much easier if you knew which door the car was behind. Assuming the contestant doesn't switch doors, condition on each of the three possibilities and use the law of total probability to assemble them to get the probability of winning the car. Do the same assuming the contestant does switch doors.

What would the probabilities be if there were 10 doors and Monty opened 8 with goats after the contestant's first choice?

5.10. Prove the claim made in Remark 5.2.3 that P' is a probability measure on $\mathscr{F}'$ (see Definition 5.1.7).

5.11. Prove the chain rule (Proposition 5.2.6).

5.12. Prove the law of total probability (Proposition 5.2.10).

5.13. Prove Proposition 5.3.5.

5.14. Let $\{E_1, E_2, \ldots, E_n\}$ be a collection of independent events. Prove that

$$P\left(\bigcup_{k=1}^{n} E_k\right) = 1 - \prod_{k=1}^{n}(1 - P(E_k)).$$

5.15. A family has three children, named Alice, Bob, and Caroline. Find the conditional probability that Alice is older than Bob, given that Alice is older than Caroline. Hint: It is not $\frac{1}{2}$.

5.16. If a certain type of cancer has a 0.4% incidence in the population, and a certain test for this cancer has a 95% accuracy (meaning 5% false positive rate and 5% false negative rate), what is the probability that a person has this type of cancer given that they tested positive for it? Graph this probability as a function of the false positive rate, as the false positive rate ranges from 10% down to 0.1%. Do the same for the false negative rate, ranging from 10% down to 0.1%, and again for incidence varying from 0.1% up to 5%.

5.17. In a certain town a car was involved in a hit-and-run accident one evening, and a witness claimed the car was blue. In that town 90% of all cars are red and 10% are blue. Some tests of the witness's ability to identify cars under these conditions showed that he identifies car color correctly 80% of the time; so the probability he will identify red when the car is actually red is 0.8, and similarly for blue. Under these assumptions, what is the probability that the perpetrator's car was actually blue?

5.18. Let $\Omega = \{a, b, c, d, e, f\}$ be the sample space of a random experiment and assume that each outcome is equally likely. Define a random variable $X : \Omega \to \mathbb{R}$ as follows:

outcome	a	b	c	d	e	f
X	1	1	1	2	3.5	3.5

(i) What is the p.m.f. of X?

(ii) Find $\mathbb{E}[X]$ and $\mathrm{Var}[X]$.

5.19. Let X be the outcome of the roll of a fair six-sided die. Find the expectation and the variance of X.

5.20. Prove the law of the unconscious statistician (Theorem 5.4.15) in the following steps:

(i) Prove that for each j we have $P(h \circ X = j) = \sum_{\{i:h(i)=j\}} P(X = i)$.

(ii) Prove that for each j we have $jP(h \circ X = j) = \sum_{\{i:h(i)=j\}} h(i)P(X = i)$.

(iii) Prove that $\sum_j jP(h \circ X = j) = \sum_j \sum_{\{i:h(i)=j\}} h(i)P(X = i)$.

(iv) Prove that $\sum_j \sum_{i:h(i)=j} h(i)P(X = i) = \sum_i h(i)P(X = i)$.

5.21. You are flipping coins to get heads, as in Example 5.4.11, and again someone offers you a bet, based on the outcome of the coin flip.

(i) If $X = n$ and n is even, then she will pay you 2^n dollars. But if n is odd, you pay her 2^n dollars. Let Y be the random variable corresponding to the amount you win, so that $Y = (-2)^X$. Prove that $\mathbb{E}[Y] = \sum_{n=1}^{\infty}(-1)^n$ and that this sum does not converge, and therefore the expected value is not defined.

(ii) Now she modifies the game so that you never lose—you win 2^n if it takes n flips to get heads. Let Z be the random variable corresponding to the amount you win, so $Z = 2^X$. Prove that the sum that defines the expected value $\mathbb{E}[Z]$ has only nonnegative terms and the limit of its partial sums is ∞.

5.22. Prove Theorem 5.4.20.

5.23. Prove Proposition 5.4.23.

5.24.* Let X be a binomial random variable with parameters n and p. Prove that

$$\mathbb{E}\left(\frac{1}{X+1}\right) = \frac{1-(1-p)^{n+1}}{(n+1)p}.$$

5.25.* Let $(\Omega, \mathscr{F}, P)$ be a discrete probability space and let $X : \Omega \to \mathbb{R}$ be a discrete random variable. Let $B \in \mathscr{F}$ be an event. Define the *conditional expectation* $\mathbb{E}[X \mid B]$ *of* X *given* B to be

$$\mathbb{E}[X \mid B] = \frac{\sum_{\omega \in B} X(\omega)P(\omega)}{P(B)} = \sum_x xP(X^{-1}(x) \mid B),$$

where the last sum runs over all x in the image of X.
Prove that if $\{B_i\}_{i\in I} \subset \mathscr{F}$ is a collection of events indexed by a countable set I, such that $\{B_i\}_{i\in I}$ is a partition of Ω (mutually exclusive and collectively exhaustive), then

$$\mathbb{E}[X] = \sum_{i\in I} \mathbb{E}[X \mid B_i]\, P(B_i). \tag{5.30}$$

5.26. Make a table of the essential information about the discrete distributions Bernoulli, binomial, and Poisson. Your table should include the support of the random variable X, the p.m.f., $\mathbb{E}[X]$, $\mathrm{Var}(X)$, and a typical situation where the distribution is used.

5.27.* Add all the important information about the negative binomial distribution to the previous table.

5.28. A student guesses randomly and independently on a true-false exam. If he answers 20 questions this way, what distribution describes the probability that he will get exactly five answers correct? What is that probability?

5.29. A call-in service center receives an average of 200 calls per hour.

(i) Which distribution could you use to describe the probability that the call center will receive exactly 100 calls in the next 20 minutes?

(ii) What assumptions should hold for that distribution to be a good model?

(iii) Describe circumstances under which those assumptions would not hold.

(iv) Assuming the assumptions hold and the distribution can be used, what is the probability that the call center receives exactly 100 calls in the next 20 minutes? What is the probability that the call center receives more than 100 calls in the next 20 minutes?

5.30. A biologist is collecting kangaroo rats in the desert and she is hoping to find some with a certain trait that occurs in 10% of the general population of kangaroo rats. She collects 100 rats in total. Assuming that each sample is independent of the others:

(i) Which distribution describes the exact number of rats that have the trait?

(ii) What is the expected number of rats that will have the trait?

(iii) What is the probability that she will find exactly 30 rats with the trait?

5.31. My grandfather sits in a rocker on his front porch and counts the number of cars that go by. He finds that the average number of cars passing is $\lambda = 7$ per hour (he lives in a rural area). Assuming at most one car passes in any given minute, we can think of a car going by in that minute as a Bernoulli random variable (either one car passes or no car passes) with probability $\frac{7}{60}$. Assuming a car in one minute has no impact on the presence of a car in another minute, then the number of cars passing in an hour (call this X) is binomially distributed with parameters $n = 60$ and $p = \lambda/60 = 7/60$. Thus the probability of x cars passing in an hour is $\binom{60}{x}\left(\frac{7}{60}\right)^x\left(1-\frac{7}{60}\right)^{60-x}$.
But, of course, it is possible for two cars to pass by in the same minute, so this is not a perfect model. If we look at a smaller time interval, like a second, it seems more reasonable to assume that two cars would not pass in the same second, so the appearance of a car in a given second is Bernoulli distributed with probability $\frac{7}{3600}$ and the probability of x cars in an hour is $\binom{3600}{x}\left(\frac{7}{3600}\right)^x\left(1-\frac{7}{3600}\right)^{3600-x}$. Yet, it could happen that two cars could pass in the same second, so it is better to divide the hour into k intervals and take the limit as $k \to \infty$. This gives

$$p(X = x) = \lim_{k\to\infty} \binom{k}{x}\left(\frac{\lambda}{k}\right)^x\left(1-\frac{\lambda}{k}\right)^{k-x}.$$

Prove that this limit is exactly $\frac{e^{-\lambda}\lambda^x}{x!}$, corresponding to the Poisson distribution. This (partly) justifies the form of the Poisson p.m.f. It is also useful because, when n is large, the binomial distribution can be approximated with a Poisson distribution, which is sometimes much easier to work with.

5.32. Prove that the variance of a random variable with Poisson distribution of rate λ is λ. Hint: When you encounter the sum

$$\sum_{k=1}^{\infty} \frac{k\lambda^{k-1}}{(k-1)!},$$

it may be helpful to rewrite the summands as

$$\frac{((k-1)+1)\lambda^{k-1}}{(k-1)!} = \frac{\lambda^{k-1}}{(k-2)!} + \frac{\lambda^{k-1}}{(k-1)!}.$$

5.33.* Assume that each ticket in a scratch-off lottery has a one-in-a-hundred chance of winning some prize.

(i) Which distribution describes the number of tickets that must be bought in order to win one prize?

(ii) What is the expected number of tickets that must be bought in order to win one prize? Two prizes?

5.34.* In Exercise 5.30, which distribution describes the number of rats she'll have to check in order to find a certain number with the trait? What is the probability that she'll have to check exactly 20 rats in order to find three with this trait?

5.35.* Prove that the mean and variance of the negative binomial distribution are as given in (5.20). Hint: First do the case of $k = 1$ and then show that if $X \sim \text{NegBin}(k, p)$, then $X = \sum_{i=1}^{k} X_i$, where $X_i \sim \text{NegBin}(1, p)$.

5.36. Prove that the p.d.f. f_X of any continuous distribution satisfies the following properties:

(i) $f_X(x) \geq 0$ for all $x \in \mathbb{R}$.

(ii) $\int_{-\infty}^{\infty} f_X(x)\, dx = 1$.

5.37. Prove that the mean and variance of the uniform distribution are as given in equation (5.22).

5.38. Assume a call-in service center receives calls according to a homogeneous Poisson process on the average of 2 every 15 minutes.

(i) What is the probability that the first call of the day will arrive no more than 5 minutes after opening?

(ii) If the operator steps away to take a lunch break for 15 minutes, what is the probability that there will be no calls during his absence? This could be answered either with the Poisson distribution or the gamma distribution. Compute the answer both ways (and ensure your answers match).

(iii) What is the probability that there will be 3 or more calls during the operator's 15 minute absence? Again, this can be computed with either the Poisson distribution or the gamma distribution. Compute the answer both ways.

5.39. If $x \in \mathbb{R}$ maximizes the p.d.f. of a random variable X, then x is called a *mode* of the distribution of X.

(i) For a normal distribution with mean μ and variance σ^2, show that the mode is μ.

(ii) Find the mode of the gamma distribution when $a > 1$, and prove that it is always less than the mean.

(iii) Find the mode of the beta distribution for $a, b > 1$. Give an example of parameters a and b where the mode of $\text{Beta}(a, b)$ is greater than the mean and an example where the mode is less than the mean.

5.40. Assume X is normally distributed with mean μ and variance σ^2. For each $k = 1, 2, 3, 4, 5, 6$ compute the probability that X lies within k standard deviations (that is, $X \in [\mu - k\sigma, \mu + k\sigma]$), as follows:

(i) Prove that the probability that X lies in the interval $[\mu - k\sigma, \mu + k\sigma]$ is the same as the probability that a random variable with standard normal distribution ($\mu = 0$ and $\sigma = 1$) lies in the interval $[-k, k]$.

(ii) For the standard normal distribution, and for each k, compute the probability that X lies in the interval $[-k, k]$. Hint: Many systems like Python and R have easily accessible modules for computing the c.d.f. of the normal distribution.

Your first three answers should be close to 68%, 95%, and 99.7%, respectively. This is sometimes called the *68-95-99.7 rule.*

5.41. Suppose five numbers are drawn from a uniform distribution on $[0, 1]$ and placed in ascending order. What is the probability that the third largest of these will be less than $\frac{1}{3}$? Hint: Many systems like Python and R have easily accessible modules for computing the c.d.f. of many distributions.

5.42. Show that the p.d.f. (5.24) for the gamma distribution has integral equal to one.

5.43. Let $X = (X_1, X_2, X_3)$ be a multivariate random variable taking values in $\mathbb{R}^3$ with p.d.f. defined for $\mathbf{x} = (x_1, x_2, x_3)$ as

$$f_X(\mathbf{x}) = \begin{cases} \|\mathbf{x}\|^2 & \text{if } x_i \in [0,1] \ \forall i \in \{1,2,3\} \\ 0 & \text{otherwise.} \end{cases}$$

(i) Verify that $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_X(\mathbf{x})\, d\mathbf{x} = 1$.

(ii) Find $P(X_1 \leq \frac{1}{5}, X_2 \leq \frac{1}{3}, X_3 \leq \frac{1}{2})$.

(iii) Find $P(X_1 \geq \frac{1}{5}, X_2 \leq \frac{1}{3}, X_3 \leq \frac{1}{2})$.

(iv) Find $\mathbb{E}[X]$.

(v) Find the marginal p.d.f. $f_1(x)$ of X_1.

(vi) Find the covariance matrix of X.

5.44. Consider a bag containing 10 green, 20 red, 30 blue, and 40 white balls that are all identical except for color. A ball is drawn, its color recorded, and then it is replaced; this experiment is repeated 20 times. Let $X = (X_G, X_R, X_B, X_W)$ be the number of times each color was drawn.

(i) Find $P(X = (2, 4, 6, 8))$.

(ii) Find $\mathbb{E}[X]$.

(iii) Find the covariance matrix of X.

5.45. Let $X : \Omega \to \mathbb{R}^n$ be a multivariate random variable with covariance matrix Σ. Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be an orthonormal basis of eigenvectors of Σ such that the corresponding eigenvalues are $\lambda_1 \geq \cdots \geq \lambda_n$. Prove that $\mathbf{u}_1$ is the direction of greatest variance of X and $\mathbf{u}_n$ is the direction of least variance of X. That is, prove that for all $\mathbf{a} \in \mathbb{R}^n$ with $\|\mathbf{a}\|_2 = 1$, we have $\mathrm{Var}(\mathbf{u}_n^\mathsf{T} X) \leq \mathrm{Var}(\mathbf{a}^\mathsf{T} X) \leq \mathrm{Var}(\mathbf{u}_1^\mathsf{T} X)$.

5.46. Prove Proposition 5.7.7 in the case that X is discrete.

5.47. Prove Proposition 5.7.10.

5.48. Prove Proposition 5.7.11.

5.49.* Prove Proposition 5.7.16.

5.50.* Prove Proposition 5.7.19.

Notes

Remark 5.2.2 is based on [BH15, Chapter 2], which has a very nice treatment of conditional probability in more depth than we have given here. We first learned of Wald's airplane analysis (Example 5.3.13) from [Bli13] (see also [Wik17]). The dice-flashlight thought experiment in section 5.3.3 was taught to us by our colleague Dennis Tolley. The idea of Example 5.5.8 that chocolate chip cookies are Poisson distributed comes from [Alb16]. The derivation of the Poisson p.m.f. in Exercise 5.31, using the story of car counting, is based on [Tan17].

6 Probabilistic Sampling and Estimation

If your experiment needs statistics, you ought to have done a better experiment.
—Ernest Rutherford

Probabilistic sampling is the process of observing or experimenting on a random subset (or sample) of a target population. Statistical estimation is about inferring information about that target population based on the results of the random sample. For example, pollsters will sample the opinions of a random subset of likely voters and then estimate the averages of the overall public opinion. Mathematically, we view the sampling process as draws from a random variable and inference as an estimation of the parameters of the underlying probability distribution.

In this chapter we discuss some of the key tools of probabilistic sampling and estimation theory. We begin by discussing what it means to estimate parameters. We then prove some important inequalities that give useful bounds on certain probabilities and expected values. These inequalities are particularly useful in the study of sums of independent, identically distributed random variables. The two main tools for understanding these sums are the law of large numbers and the central limit theorem. These results provide the mathematical framework for statistical inference.

The estimates performed in the first section of this chapter are called *point estimates* because they estimate specific values like the mean or the variance. By contrast, in Bayesian statistics, which we treat in Section 6.5, the estimates are distributions instead of values. This is a powerful idea that often requires a lot of computational power to use. As computers have become faster and cheaper, Bayesian statistics has become more popular for understanding the world in all its uncertainty.

6.1 Estimation

In a given experiment, we make observations $x_1, x_2, \ldots, x_n$, which result in a body of data. We think of the data as *draws* from, or *realizations* of, the random variable X. Often, the parameters of the distribution are unknown—we only have the data from the sample. To estimate the parameters of the distribution, we apply a function

or formula to this collection of data. A function of data that is used to provide information about the distribution that generates the data is called a *statistic*.

More precisely, we assume that the data are generated by taking draws from an independent and identically distributed (i.i.d.) sequence $X_1, X_2, \ldots, X_n$ of random variables having the same distribution as X; the i.i.d. sequence of random variables is called a *sample* of the distribution. Any function $T(X_1, \ldots, X_n)$ of the sample is called a *statistic* and (assuming it is a sufficiently well-behaved function)[29] is itself a random variable. Statistics are often used to estimate an unknown parameter of a distribution. Statistics that are used to estimate some quantity or parameter are called *estimators*. An *estimate* is the result we get when we replace each random variable X_i in an estimator by the given data x_i, that is, when we evaluate the function $T(X_1, \ldots, X_n)$ at the point $X_1 = x_1, \ldots, X_n = x_n$.

For example, we may want to estimate the mean and variance of a distribution to get an idea of its central tendency and variability.

Definition 6.1.1. *The* sample mean *estimator is given by*

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i. \tag{6.1}$$

The biased sample variance *estimator is given by*

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{\mu})^2. \tag{6.2}$$

We sometimes write $\hat{\mu}_n$ and $\hat{\sigma}_n$ when the length n of the sample might otherwise be unclear.

Example 6.1.2. An exam is an experiment where the data are the students' scores $x_1, \ldots, x_n$ on the exam. It is common to assume that these are drawn (approximately) from a normal distribution $\mathcal{N}(\mu, \sigma^2)$. An estimate for the mean μ is given by the average score

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{6.3}$$

which is the evaluation at $X_1 = x_1, \ldots, X_n = x_n$ of the sample mean estimator $\hat{\mu}$. Similarly, an estimate for σ^2 is given by evaluating the biased sample variance estimator (6.2) at $X_1 = x_1, \ldots, X_n = x_n$ to get

$$\hat{\sigma}^2_{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2. \tag{6.4}$$

[29] Continuous functions are all sufficiently well behaved.

Remark 6.1.3. Estimators such as the sample mean and biased sample variance are random variables because they are functions of the random variables $X_1, \ldots, X_n$, not functions of the data $x_1, \ldots, x_n$. Drawing from the random variables (that is, evaluating the X_i at various points in the sample space Ω), gives data $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, which we can insert into the corresponding estimator to get an estimate. In the previous example, a draw $\mathbf{x}$ from $X_1, \ldots, X_n$ gave an estimate $\bar{x}$ of μ, defined by (6.3), and it gave an estimate $\hat{\sigma}^2_{\mathbf{x}}$ defined by (6.4). A different draw produces a different estimate.

6.1.1 Biased and Unbiased Estimators

Some estimators are better than others for estimating a given quantity. *Bias* is a measure of the average error of an estimator.

Definition 6.1.4. *Consider an estimator $\hat{\theta} = \hat{\theta}(X_1, X_2, \ldots, X_n)$ of a parameter θ. The* bias *of the estimator is given by* $\text{bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta}] - \theta$. *If* $\text{bias}(\hat{\theta}) = 0$, *then the estimator $\hat{\theta}$ is* unbiased*; otherwise, it is* biased.

Example 6.1.5. Let X be a random variable with $\mathbb{E}[X] = \mu$. The sample mean estimator (6.1) is an unbiased estimator of μ since

$$\mathbb{E}[\hat{\mu}] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\, n\mu = \mu.$$

Example 6.1.6. Let X be a random variable with $\mathbb{E}[X] = \mu$ and $\text{Var}(X) = \sigma^2$. We have called the estimator (6.2) the *biased* sample variance. To see that it really is a biased estimator of σ^2, compute

$$\begin{aligned}
\mathbb{E}[\hat{\sigma}^2] - \sigma^2 &= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left((X_i - \hat{\mu})^2 - (X_i - \mu)^2\right)\right] \\
&= \mathbb{E}\left[\hat{\mu}^2 - \mu^2 - \frac{2}{n}\sum_{i=1}^{n} X_i(\hat{\mu} - \mu)\right] \\
&= \mathbb{E}\left[\hat{\mu}^2 - \mu^2 - 2\hat{\mu}(\hat{\mu} - \mu)\right] \\
&= -\mathbb{E}[(\hat{\mu} - \mu)^2] = -\,\text{Var}(\hat{\mu}) \\
&= -\,\text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = -\frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = -\frac{\sigma^2}{n}.
\end{aligned}$$

This shows that $\hat{\sigma}^2$ is indeed a biased estimator of σ^2, and $\mathbb{E}[\hat{\sigma}^2] = \frac{n-1}{n}\sigma^2$. Note that as $n \to \infty$ we have $\mathbb{E}[\hat{\sigma}^2] \to \sigma^2$. Because of this, we say that $\hat{\sigma}^2$ is *asymptotically unbiased.*

Proposition 6.1.7. *Let X be a random variable with $\mathbb{E}[X] = \mu$ and $\mathrm{Var}(X) = \sigma^2$. The estimator*

$$\hat{s}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \hat{\mu})^2 \tag{6.5}$$

is an unbiased estimator of the variance σ^2. We call this the unbiased sample variance *of the random variable X.*

Proof. The proof is Exercise 6.1. □

6.1.2 Maximum Likelihood Estimation

Perhaps the most widely used estimation method is *maximum likelihood estimation*, which chooses the parameter for which the observations are most likely to have occurred.

Definition 6.1.8. *Let $X_1, X_2, \ldots, X_n$ be a sample of a discrete distribution X with p.m.f. $g(x, \theta)$ depending on some parameter θ. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a draw from the sample. The joint probability*

$$L(\theta) = P(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i) = \prod_{i=1}^{n} g(x_i, \theta)$$

is called the likelihood of θ. *Similarly, if X is continuous with p.d.f. $f(x, \theta)$, depending on some parameter θ, then the* likelihood of θ *is the joint p.d.f.*

$$L(\theta) = \prod_{i=1}^{n} f(x_i, \theta).$$

A maximum likelihood estimate *(MLE) of θ is a point $\hat{\theta}$ that maximizes the likelihood. In the case that there is an estimator $\hat{\theta}(X_1, \ldots, X_n)$ whose corresponding estimate $\hat{\theta}(x_1, \ldots, x_n)$ is always the MLE for θ corresponding to $x_1, \ldots, x_n$, we say that $\hat{\theta}(X_1, \ldots, X_n)$ is the* maximum likelihood estimator *of θ.*

Remark 6.1.9. As a notational convention, it is common to write the likelihood function as $L(\theta) = f(\mathbf{x} \mid \theta)$. We sometimes say that $L(\theta)$ is *the density of $\mathbf{x}$ given θ*. This is justified because the X_i are i.i.d. with p.d.f. $f(x, \theta)$ depending on θ, and thus the joint probability density function $f(x_1, \ldots, x_n \mid \theta)$ for $X_1, \ldots, X_n$ factors as $f(x_1, \ldots, x_n \mid \theta) = \prod_{i=1}^{n} f(x_i, \theta)$.

Example 6.1.10. Consider the Bernoulli distribution $g(x, p) = p^x(1-p)^{1-x}$ with unknown value of p. The likelihood of p given a draw $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is

$$L(p) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} = p^{\sum_{i=1}^{n} x_i}(1-p)^{n-\sum_{i=1}^{n} x_i} = p^{n\bar{x}}(1-p)^{n(1-\bar{x})}, \tag{6.6}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is shorthand for the average of the data (the sample mean $\hat{\mu}$ evaluated at the specific value $\mathbf{x}$). If the maximum occurs at $\hat{p}$, then derivative of L is zero at $\hat{p}$. Thus, we solve

$$\begin{aligned} 0 = L'(\hat{p}) &= n\bar{x}\hat{p}^{n\bar{x}-1}(1-\hat{p})^{n(1-\bar{x})} - n(1-\bar{x})\hat{p}^{n\bar{x}}(1-\hat{p})^{n(1-\bar{x})-1} \\ &= \left(n\bar{x}(1-\hat{p}) - n(1-\bar{x})\hat{p}\right)\hat{p}^{n\bar{x}-1}(1-\hat{p})^{n(1-\bar{x})-1}. \end{aligned}$$

This implies that

$$\bar{x}(1-\hat{p}) = (1-\bar{x})\hat{p},$$

which simplifies to $\hat{p} = \bar{x}$. Thus, the MLE of p is $\hat{p} = \bar{x}$ and the maximum likelihood estimator is $\hat{p}(X_1, \ldots, X_n) = \frac{1}{n}\sum_{i=1}^{n} X_i$.

Remark 6.1.11. Since the likelihood function is nonnegative on the domain of the distribution and the logarithm function is strictly increasing, the log of that likelihood, that is, $\ell(\theta) = \log L(\theta)$, achieves its maximum at the same values $\hat{\theta}$ as the likelihood $L(\theta)$ does. Thus, we can solve for the maximum likelihood by solving for the maximum of the log-likelihood. This usually makes calculations much easier. Applying this idea to Example 6.1.10, we have

$$\ell(p) = n\bar{x}\log p + n(1-\bar{x})\log(1-p),$$

which reduces to

$$0 = \ell'(\hat{p}) = \frac{n\bar{x}}{\hat{p}} - \frac{n(1-\bar{x})}{1-\hat{p}}.$$

Solving for $\hat{p}$ gives the same result.

Example 6.1.12. Consider the normal distribution $\mathcal{N}(\mu, \sigma^2)$ for an unknown value of μ but a fixed value of σ^2. The likelihood of μ given a draw $\mathbf{x} = (x_1, \ldots, x_n)$ is

$$L(\mu) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right) = (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}\right).$$

The log-likelihood is

$$\ell(\mu) = -\frac{n}{2}(\log(2\pi) + \log(\sigma^2)) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2. \tag{6.7}$$

If $\hat{\mu}$ is a maximizer of L, then we must have $\frac{d\ell}{d\mu}\big|_{\hat{\mu}} = 0$. This gives

$$-\frac{1}{2\sigma^2}\sum_{i=1}^{n} 2(x_i-\hat{\mu}) = 0, \tag{6.8}$$

and hence $\hat{\mu} = \bar{x}$. Thus, for a given draw $\mathbf{x}$, the MLE for μ is

$$\hat{\mu} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

and the corresponding estimator is the sample mean (6.1).

Example 6.1.13. As in Example 6.1.12, consider the normal distribution $\mathcal{N}(\mu, \sigma^2)$, but now assume that μ is fixed and known. We wish to estimate σ^2 by maximizing the log-likelihood

$$\ell(\sigma^2) = -\frac{n}{2}(\log(2\pi) + \log(\sigma^2)) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2.$$

If $\hat{\sigma}^2$ is a maximizer of ℓ, then we must have $\frac{d\ell}{d(\sigma^2)}\big|_{\hat{\sigma}^2} = 0$. Note that here we are differentiating with respect to σ^2. We treat this not as the square of σ but rather as an awkwardly named variable in its own right. This gives

$$-\frac{n}{2\hat{\sigma}^2} + \frac{1}{2(\hat{\sigma}^2)^2}\sum_{i=1}^{n}(x_i - \mu)^2 = 0. \tag{6.9}$$

Solving for $\hat{\sigma}^2$ gives

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2. \tag{6.10}$$

Thus, for a given draw $\mathbf{x}$, the MLE for σ^2 is (6.10) and the corresponding estimator is the biased sample variance (6.2).

Vista 6.1.14. In simple problems like those of Examples 6.1.10–6.1.13, we can find the maximum likelihood estimator using the standard techniques of calculus, but for more complex problems with lots of data, this analytic method becomes prohibitively difficult to implement. Usually all we can do for such problems is use numerical optimization methods to find the maximizer (MLE) for a given draw. In the second half of this text we cover many techniques and algorithms for optimization. These methods are essential not only for maximum likelihood estimation but also for many other important problems, including linear and logistic regression, and most methods of machine learning.

6.1.3 *Comparing Estimators

There are many different kinds of estimators. One may ask, when is one estimator better than another? To answer this question, we must first recognize that there are different ways to measure "better."

As a motivating example, consider throwing darts at a target, as in Figure 6.1. If all the darts are clustered tightly together but off center, then that is good, in the sense that the pattern is tight and the thrower is consistent, but bad, in the sense that the thrower isn't hitting near the bull's-eye. In this case we say the thrower is *precise* but not *accurate*. By contrast, suppose that the darts are symmetrically scattered on the target where the average is right on the bull's-eye. In this case, we say that the thrower is *accurate*, but not *precise*.

Estimators are like dart throwers in the sense that some can be precise, but not accurate, while others can be accurate, but not precise. Accuracy is described by the bias of the estimator $\mathbb{E}[\hat{\theta}] - \theta$. Precision is described by the variance of the estimator, that is by $\operatorname{Var}\hat{\theta} = \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$.

Example 6.1.15. Consider a sample of i.i.d. random variables X_1, X_2, X_3, each with mean μ and variance σ^2. The statistic

$$Y = \frac{X_1 + 2X_2 + 3X_3}{6}$$

is an unbiased estimator for μ, that is, $\mathbb{E}[Y] = \mu$. Its variance is

$$\operatorname{Var}(Y) = \frac{1}{36}\left(1 + 2^2 + 3^2\right)\sigma^2 = \frac{7}{18}\sigma^2.$$

This is a larger variance than the sample mean

$$\hat{\mu} = \frac{X_1 + X_2 + X_3}{3},$$

which has $\operatorname{Var}(\hat{\mu}) = \frac{1}{3}\sigma^2$. Both estimators are accurate because they are unbiased; however, the sample mean is more precise.

One important way to measure the overall quality of an estimator is to compute its mean squared error.

Definition 6.1.16. *Given an estimator $Y = Y(X_1, X_2, \ldots, X_n)$ of the parameter θ, the* mean squared error (MSE) *of the estimator Y is*

$$\operatorname{MSE}(Y) = \mathbb{E}[(Y - \theta)^2].$$

Proposition 6.1.17. *Given an estimator $Y = Y(X_1, X_2, \ldots, X_n)$ of the parameter θ, the MSE of the estimator Y satisfies the relation*

$$\operatorname{MSE}(Y) = \operatorname{bias}(Y)^2 + \operatorname{Var}(Y).$$

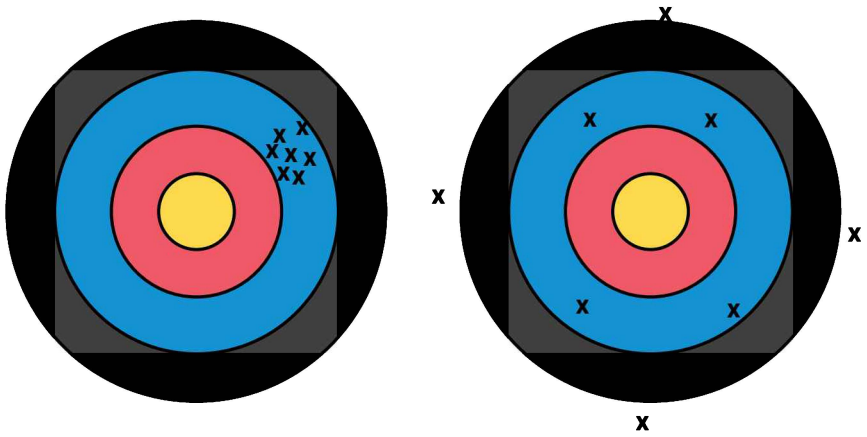


Figure 6.1. *The target on the left shows a pattern that is precise (it has low variance) but not accurate (it is biased) because it is clustered away from the center. In the target on the right, the pattern is not precise (it has high variance), but it is accurate (unbiased) because the sample mean is centered near the bull's-eye.*

Proof.

$$\begin{aligned}
\mathrm{MSE}(Y) &= \mathbb{E}[(\theta - Y)^2] = \mathbb{E}\left[(\theta - \mathbb{E}[Y] + \mathbb{E}[Y] - Y)^2\right] \\
&= \mathbb{E}\left[(\theta - \mathbb{E}[Y])^2 - 2(\theta - \mathbb{E}[Y])(Y - \mathbb{E}[Y]) + (Y - \mathbb{E}[Y])^2\right] \\
&= (\theta - \mathbb{E}[Y])^2 - 2(\theta - \mathbb{E}[Y])\mathbb{E}[(Y - \mathbb{E}[Y])] + \mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right] \\
&= (\theta - \mathbb{E}[Y])^2 + \mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right] \\
&= \mathrm{bias}(Y)^2 + \mathrm{Var}(Y). \qquad \square
\end{aligned}$$

The estimator that minimizes the MSE is called the *minimum MSE* estimator. If we restrict to unbiased estimators, then the previous proposition shows that $\mathrm{MSE}(Y) = \mathrm{Var}(Y)$. In this case, the estimator Y minimizing $\mathrm{Var}(Y)$ is called the *minimum-variance unbiased estimator* of θ. But it is not always optimal to restrict to unbiased estimators. There can be biased estimators with a smaller MSE than the minimum-variance unbiased estimator, so if we wish to minimize the MSE, we may need to accept a biased estimator.

6.2 The Law of Large Numbers

In the previous section we showed that the sample mean

$$\hat{\mu}_n = \frac{X_1 + X_2 + \cdots + X_n}{n} \tag{6.11}$$

of a sample $(X_1, \ldots, X_n)$ of a distribution X is an unbiased estimator of the mean $\mu = \mathbb{E}[X]$. But for any given draw $\mathbf{x} = (x_1, \ldots, x_n)$ the resulting estimate may not be very close to the actual mean. In general one usually expects the estimate to be better as n gets larger. In this section we quantify this and discuss many properties of sums of i.i.d. random variables: how they are distributed, their mean

and variance, and how well they approximate the main parameters of the random variable X. We conclude this section with the *law of large numbers*, which gives information about how rapidly $\hat{\mu}_n$ approaches the mean μ.

6.2.1 Important Inequalities

In order to prove the law of large numbers we need some simple inequalities that are important in their own right.

Lemma 6.2.1. *If X, Y are random variables on a probability space $(\Omega, \mathscr{F}, P)$ with $X(\omega) \leq Y(\omega)$ for all $\omega \in \Omega$, then $\mathbb{E}[X] \leq \mathbb{E}[Y]$.*

Proof. We prove this in the discrete case; the continuous case is similar. Since $X \leq Y$, the difference $Y - X$ is always nonnegative, and so

$$\mathbb{E}[Y - X] = \sum_r rP(Y - X = r) = \sum_{r \geq 0} rP(Y - X = r) \geq 0.$$

This gives $0 \leq \mathbb{E}[Y - X] = \mathbb{E}[Y] - \mathbb{E}[X]$ by Theorem 5.4.14, from which we have the desired inequality. □

Theorem 6.2.2 (Markov's Inequality). *If X is a nonnegative random variable, then for any $a > 0$, we have*

$$P(X \geq a) \leq \frac{\mathbb{E}[X]}{a}. \tag{6.12}$$

Proof. For $a > 0$, let $\mathbb{1}_{x \geq a} : \Omega \to \mathbb{R}$ be the composition

$$\mathbb{1}_{x \geq a} = \mathbb{1}_{[a, \infty)} \circ X = \begin{cases} 1 & \text{if } X \geq a, \\ 0 & \text{if } X < a. \end{cases}$$

We have $\mathbb{1}_{x \geq a} \leq \frac{X}{a}$. Taking the expected value of both sides of this inequality gives

$$\mathbb{E}[\mathbb{1}_{x \geq a}] \leq \mathbb{E}\left[\frac{X}{a}\right] = \frac{\mathbb{E}[X]}{a},$$

and since $\mathbb{E}[\mathbb{1}_{x \geq a}] = P(X \geq a)$, we have (6.12). □

Remark 6.2.3. The last step of the previous proof involves a simple idea that is a fundamental tool of probability, namely, that for any $E \subset \Omega$ the indicator random variable $\mathbb{1}_E$ has expected value equal to the probability of E:

$$\mathbb{E}[\mathbb{1}_E] = P(E). \tag{6.13}$$

This fact is straightforward to prove but extremely powerful. It is sometimes called the *fundamental bridge* between expectation and probability. Similarly, for any set $E \subset \mathbb{R}$ and any random variable X, the expected value of $\mathbb{1}_E \circ X$ is the probability that X lies in E:

$$\mathbb{E}[\mathbb{1}_E \circ X] = P(X \in E). \tag{6.14}$$

Example 6.2.4. Markov's inequality isn't useful when $a \leq \mathbb{E}[X]$ because in that case $1 \leq \frac{\mathbb{E}[X]}{a}$. But when a is larger, it can give a useful bound. For example, if $a = 2\mathbb{E}[X]$, it gives

$$P(X \geq 2\mathbb{E}[X]) \leq \frac{1}{2}.$$

For many distributions this is not a tight bound. Its utility lies in the fact that it holds for all nonnegative distributions. For example, if $X \sim \text{Binomial}(n, \frac{1}{2})$ and $a = 2\mathbb{E}[X] = n$, then Markov's inequality gives $P(X \geq n) \leq \frac{1}{2}$. But we can compute this probability exactly to get $P(X \geq n) = P(X = n) = 2^{-n}$, which is much smaller than $\frac{1}{2}$ when $n \geq 2$.

It should not be especially surprising that an inequality that holds for all nonnegative random variables is not very tight for some specific random variables. To get a very close bound for any particular random variable, we should expect to have to use specific properties of the particular distribution.

Corollary 6.2.5 (Chebyshev's Inequality). *If X is a random variable with finite mean μ and variance σ^2, then for any $\varepsilon > 0$ we have*

$$P(|X - \mu| \geq \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2}.$$

Proof. Since the quantity $(X-\mu)^2$ is a nonnegative random variable, we can apply Markov's inequality:

$$P(|X - \mu| \geq \varepsilon) = P((X-\mu)^2 \geq \varepsilon^2) \leq \frac{\mathbb{E}[(X-\mu)^2]}{\varepsilon^2} = \frac{\sigma^2}{\varepsilon^2}. \quad \square$$

Example 6.2.6. Let X be a random variable with $\mathbb{E}[X] = 50$ and $\text{Var}(X) = 25$. What is the probability that X lies between 40 and 60? By Chebyshev's inequality, we have

$$P(|X - \mu| \leq 10) = 1 - P(|X - \mu| \geq 10) \geq 1 - \frac{25}{100} = \frac{3}{4}.$$

Example 6.2.7. For any X with mean μ and variance σ^2, Chebyshev's inequality gives no information for $\varepsilon \leq \sigma$, because $1 \leq \frac{\sigma^2}{\varepsilon^2}$. If $\varepsilon = 2\sigma$, then we have

$$P(|X - \mu| \geq 2\sigma) \leq \frac{1}{4}.$$

Again, this holds for all distributions, but it is not a very tight bound for many specific distributions.

For example, if $X \sim \mathcal{N}(\mu, \sigma^2)$, then by the 68-95-99.7 rule (see Exercise 5.40) we have

$$P(|X - \mu| \geq \sigma) = 1 - P(|X - \mu| < \sigma) \approx 0.32$$

and

$$P(|X - \mu| \geq 2\sigma) = 1 - P(|X - \mu| < 2\sigma) \approx 0.05.$$

Both of these are much smaller than the bounds we get from Chebyshev.

6.2.2 Law of Large Numbers

We now come to the important result called the *law of large numbers*. Informally, this is often called the *law of averages*. It says that as n gets large, the sample mean $\hat{\mu}_n$ is increasingly likely to be close to the mean μ of the X_i, assuming the X_i are i.i.d. random variables. Figure 6.2 shows several plots of the estimates

$$\bar{x} = \hat{\mu}_n(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

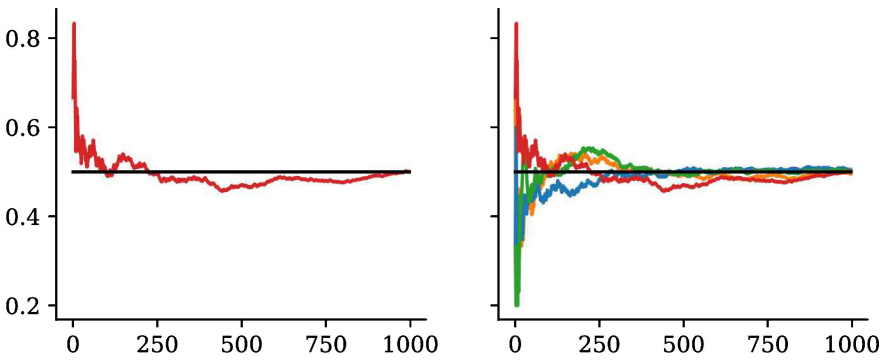


Figure 6.2. *Plot of consecutive values of $\bar{x} = \hat{\mu}_n(\mathbf{x})$ for a sequence of coin flips (draws from $X_i \sim$ Bernoulli(0.5)) as n ranges from 1 to 1000 (left panel). The plot shows $\frac{x_1}{1}, \frac{x_1+x_2}{2}, \ldots, \frac{x_1+\cdots+x_{1000}}{1000}$. The value of $\bar{x}$ begins above $\mu = 0.5$, rapidly approaches μ, then moves away again, then comes back and again moves away, but as $n \to 1000$ it seems to approach μ. This illustrates the fact that the random variable $\hat{\mu}_n$ is not guaranteed to be close to μ, but the probability that a given realization $\bar{x}$ is far away gets small as $n \to \infty$. On the right the results of that same draw are plotted again, along with the results of three more draws. None of these is consistently close to μ, but the law of large numbers says that the probability that any one of these estimates $\bar{x}$ will be more than a given distance ε from $\mu = \frac{1}{2}$ is less than $\frac{\sigma^2}{n\varepsilon^2} = \frac{1}{4n\varepsilon^2}$, which gets small as n gets large.*

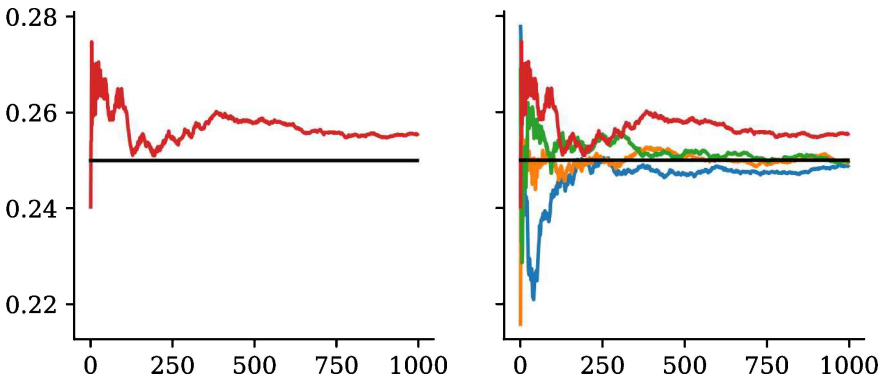


Figure 6.3. *Plot of $\bar{x} = \hat{\mu}_n(\mathbf{x})$ for a sequence of draws $\mathbf{x}$ from* Beta$(10, 30)$ *as n ranges from 1 to 1000 (left side). The plot shows $\frac{x_1}{1}, \frac{x_1+x_2}{2}, \ldots, \frac{x_1+\cdots+x_{1000}}{1000}$. In this case as $n \to 1000$ the value of $\bar{x}$ does not seem to approach μ very rapidly. The experiment is repeated three more times and plotted in the right panel. The value of $\bar{x}$ is closer to μ near $n = 1000$ for the other three experiments (green, blue, and yellow) than it is for the first (red). The law of large numbers says that the probability that any one of these experiments will be more than a given distance ε from μ gets small as n gets large.*

arising from the estimator $\hat{\mu}_n$ for a draw $\mathbf{x} = (x_1, \ldots, x_n)$ from a sample $X_1, \ldots, X_n$ of a random variable $X \sim$ Bernoulli(0.5) as n ranges from 1 to 1000. Figure 6.3 shows the same thing for $X \sim$ Beta$(10, 30)$.

Theorem 6.2.8 (Weak Law of Large Numbers). *Let $X_1, X_2, \ldots$ be a sequence of i.i.d. random variables, each having mean μ and variance σ^2. For each $n \in \mathbb{Z}^+$ let $\hat{\mu}_n$ be the estimator $\hat{\mu}_n = \frac{1}{n}(X_1 + X_2 + \cdots + X_n)$. For all $\varepsilon > 0$, we have that*

$$P\left(|\hat{\mu}_n - \mu| \geq \varepsilon\right) \to 0$$

as $n \to \infty$. More specifically, for every $\varepsilon > 0$

$$P\left(|\hat{\mu}_n - \mu| \geq \varepsilon\right) \leq \frac{\sigma^2}{n\varepsilon^2}. \tag{6.15}$$

Proof. By linearity of expectation (Theorem 5.4.14) we have

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}(X_1 + X_2 + \cdots + X_n)\right] = \frac{1}{n}(\mathbb{E}[X_1] + \mathbb{E}[X_2] + \cdots \mathbb{E}[X_n]) = \mu$$

and by the additivity of variance (5.15) for independent random variables, we have

$$\begin{aligned}\operatorname{Var}(\hat{\mu}_n) &= \operatorname{Var}\left[\frac{1}{n}(X_1 + X_2 + \cdots + X_n)\right] = \frac{1}{n^2}\operatorname{Var}(X_1 + X_2 + \cdots + X_n) \\ &= \frac{1}{n^2}(\operatorname{Var}(X_1) + \operatorname{Var}(X_2) + \cdots + \operatorname{Var}(X_n)) = \frac{\sigma^2}{n}. \end{aligned} \tag{6.16}$$

Using Chebyshev's inequality, we have

$$P\left(\left|\frac{1}{n}(X_1 + X_2 + \cdots + X_n) - \mu\right| \geq \varepsilon\right) \leq \frac{\sigma^2}{n\varepsilon^2},$$

which gives the result in the limit as $n \to \infty$. □

There is also a stronger version of the law of large numbers called the *strong law of large numbers.*

Example 6.2.9. Flipping a fair coin n times corresponds to n i.i.d. random variables X_i, each with expected value $\frac{1}{2}$ and variance $\frac{1}{4}$. The law of large numbers tells us for any $\varepsilon > 0$ that

$$P\left(\left|\hat{\mu}_n - \frac{1}{2}\right| \geq \varepsilon\right) \leq \frac{1}{4n\varepsilon^2}.$$

This goes to 0 as n gets large; that is, the probability that $\hat{\mu}_n$ is any given distance away from $\frac{1}{2}$ gets arbitrarily small. This corresponds to the intuition that the ratio $\hat{\mu}_n$ should approach $\frac{1}{2}$ as n gets large. This is depicted in Figure 6.2. More generally, in a sequence of Bernoulli trials with probability p of each success, we have

$$P\left(|\hat{\mu}_n - p| \geq \varepsilon\right) \leq \frac{p(1-p)}{n\varepsilon^2}.$$

Nota Bene 6.2.10. The law of large numbers does *not* say that the sum $S_n = X_1 + \cdots + X_n$ is likely to approach $n\mu$. As an example of this, consider the situation with a fair coin ($X \sim$ Bernoulli(0.5)). For any $\varepsilon > 0$ the law of large numbers says that

$$P\left(|S_n - n\mu| \geq n\varepsilon\right) = P\left(\left|\frac{1}{n}S_n - \mu\right| \geq \varepsilon\right) \leq \frac{1}{4n\varepsilon^2}. \tag{6.17}$$

But for any constant k and any choice of ε, as $n \to \infty$ we eventually have $n\varepsilon > k$, so (6.17) tells us nothing about $P\left(|S_n - n\mu| \geq k\right)$. In fact, a more careful analysis shows that $|S_n - \frac{n}{2}| \to \infty$ with probability 1.

Nota Bene 6.2.11. The law of large numbers also does not say that $\hat{\mu}_n$ must approach μ, only that it becomes increasingly likely to be near μ as n gets large.

For a fair coin, it says that the probability that $\frac{S_{100}}{100}$ is outside the range $(\frac{1}{4}, \frac{3}{4})$ is no more than $\frac{1}{4 \cdot 100 \cdot (\frac{1}{4})^2} = \frac{1}{25}$. While $\frac{1}{25}$ may seem unlikely (depending

on the circumstances), it is certainly not impossible. Similarly, the probability of being outside the range $(\frac{1}{4}, \frac{3}{4})$ if $n = 1000$ is no more than $\frac{1}{250}$, which is also not impossible.

This can be seen in Figure 6.2, where $\hat{\mu}_n$ is often moving farther away from μ rather than getting closer, but the likelihood that it will be more than a certain amount ε from μ gets small as n gets large.

Example 6.2.12. Let X_i be the outcome of the ith roll of a six-sided die. If the die is fair, then the expectation of each X_i is $\mathbb{E}[X_i] = \frac{7}{2}$ and $\text{Var}(X_i) = \frac{105}{36}$. By the law of large numbers, we should have

$$P\left(\left|\frac{\sum_{i=1}^{n} X_i}{n} - \frac{7}{2}\right| \geq \varepsilon\right) \leq \frac{105}{36n\varepsilon^2}$$

for any n and $\varepsilon > 0$. If the die is fair, then the probability is low that a given draw $\mathbf{x} = (x_1, \ldots, x_n)$ will make $\bar{x} = \hat{\mu}_n(\mathbf{x})$ very far away from $\frac{7}{2}$. Specifically, after 105 rolls of the die, taking $\varepsilon = 1$, the probability that $\bar{x}$ lies outside the interval $[\frac{5}{2}, \frac{9}{2}]$ is bounded above by $\frac{105}{36 \cdot 105} = \frac{1}{36} \approx 0.027$, and after 315 rolls it is bounded above by $\frac{1}{36 \cdot 3} \approx 0.009$.

6.3 The Central Limit Theorem

The [central limit theorem] would have been personified by the Greeks and deified, if they had known of it. It reigns with serenity and in complete self-effacement, amidst the wildest confusion. The huger the mob, and the greater the apparent anarchy, the more perfect is its sway. It is the supreme law of Unreason.
—Sir Francis Galton

As its name suggests, the central limit theorem is central to probability theory. It is also the key behind our ability to draw inferences. We give the proof of the central limit theorem in Section 6.4. In this section we state the theorem and discuss a few of its many applications.

6.3.1 The Central Limit Theorem

The central limit theorem says that for i.i.d. random variables $X_1, X_2, \ldots, X_n$ with finite mean μ and variance σ^2, if n is large, then the sample mean $\hat{\mu}_n$ is approximately distributed as $\mathcal{N}(\mu, \frac{\sigma^2}{n})$. This is both surprising and powerful because it holds regardless of how X_i is distributed.

Theorem 6.3.1 (Central Limit Theorem). *Let $X_1, X_2, \ldots, X_n$ be a sequence of i.i.d. random variables, each having mean μ and variance σ^2. Define random variables $S_n = X_1 + X_2 + \cdots + X_n$ and*

$$Y_n = \frac{S_n - n\mu}{\sqrt{n}\sigma}.$$

The sequence $(F_{Y_n})_{n=1}^{\infty}$ of c.d.f.s converges pointwise to the standard normal c.d.f. as $n \to \infty$. In other words, for each $y \in \mathbb{R}$, we have

$$P(Y_n \le y) \to \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} e^{-x^2/2}\, dx \tag{6.18}$$

as $n \to \infty$.

The fact that the sample is i.i.d. implies that the sample mean $\hat{\mu}_n = \frac{S_n}{n}$ has mean μ and variance $\frac{\sigma^2}{n}$ (see (6.16)). We can reformulate the central limit theorem to say that as n gets large, $\hat{\mu}_n$ approaches a normally distributed random variable with mean μ and variance $\frac{\sigma^2}{n}$.

Corollary 6.3.2. *For sufficiently large values of n the sample mean $\hat{\mu}_n$ is approximately distributed as $\mathcal{N}(\mu, \frac{\sigma^2}{n})$, and the sum S_n is approximately distributed as $\mathcal{N}(n\mu, n\sigma^2)$.*

Proof. For large n we have

$$\begin{aligned} P(\hat{\mu}_n \le z) &= P\left(\frac{\frac{S_n}{n} - \mu}{\frac{\sigma}{\sqrt{n}}} \le \frac{z-\mu}{\frac{\sigma}{\sqrt{n}}} \right) = P\left(Y_n \le \frac{\sqrt{n}}{\sigma}(z-\mu) \right) \\ &\approx \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{\sqrt{n}}{\sigma}(z-\mu)} e^{-x^2/2}\, dx. \end{aligned}$$

Making the substitution $x = \frac{\sqrt{n}}{\sigma}(v-\mu)$ gives

$$P(\hat{\mu}_n \le z) \approx \frac{\sqrt{n}}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{z} e^{-\frac{(v-\mu)^2}{2(\sigma^2/n)}}\, dv,$$

which is the c.d.f. for $\mathcal{N}(\mu, \frac{\sigma^2}{n})$.

A similar argument (see Exercise 6.12) shows that

$$P(S_n \le w) \approx \frac{1}{\sqrt{2\pi n\sigma^2}} \int_{-\infty}^{w} e^{-\frac{(v-n\mu)^2}{2(n\sigma^2)}}\, dv,$$

and hence S_n is approximately distributed as $\mathcal{N}(n\mu, n\sigma^2)$. ∎

A key point of the central limit theorem is that *the random variables $X_1, \ldots, X_n$ need not be normally distributed.* They can be as unnormal as you like—for example, heavily skewed to one side or bimodal—but as long as they have a well-defined (and finite) mean and variance, the average always approaches a normal distribution as $n \to \infty$. Of course, the farther away from normal the original distribution is, the larger n must be before the distribution of $\hat{\mu}_n$ is approximately normal. But for n large enough, $\hat{\mu}_n$ will be close to normal; see Figure 6.4.

The central limit theorem also explains the great prevalence of normal distributions in many applications. Whenever a quantity is a sum of many i.i.d. effects, the central limit theorem (or appropriate generalizations) says that quantity will be approximately normally distributed, regardless of the distribution of the pieces. Thus we see the normal distribution in quantities like exam scores, total annual snowfall, and radio noise, all of which are determined by sums of many independent factors.

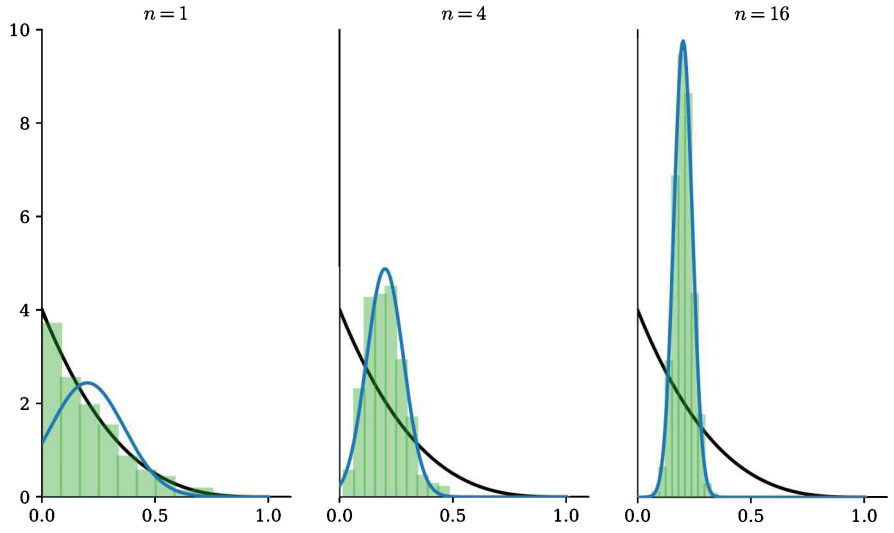


Figure 6.4. *Illustration of the central limit theorem for the distribution* $\text{Beta}(1,4)$ *(p.d.f. in black). We have sampled* $X_1, \ldots, X_n$ *from* $\text{Beta}(1,4)$ *and constructed* $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$*. For each* $n \in \{1, 4, 16\}$*, the corresponding histogram (green) shows the result of drawing from* $\hat{\mu}_n$ *a thousand times, giving an approximation of the distribution of* $\hat{\mu}_n$*. The central limit theorem guarantees that when* n *is large, the distribution of* $\hat{\mu}_n$ *is close to the normal distribution* $\mathcal{N}(\frac{1}{5}, \frac{2}{75n})$ *(p.d.f. in blue).*

Example 6.3.3. As discussed in Example 6.2.9, the law of large numbers guarantees for repeated trials of a fair coin flip, for any $\varepsilon > 0$, we have

$$P\left(\left|\hat{\mu}_n - \frac{1}{2}\right| \geq \varepsilon\right) \leq \frac{1}{4n\varepsilon^2}.$$

In the case that $\varepsilon = \frac{1}{2\sqrt{n}}$, this gives the unhelpful result that

$$P\left(\left|\hat{\mu}_n - \frac{1}{2}\right| \geq \frac{1}{2\sqrt{n}}\right) \leq 1.$$

But the central limit theorem says that $\hat{\mu}_n$ is approximately normally distributed with mean $\frac{1}{2}$ and variance $\frac{1}{4n}$. The 68-95-99.7 rule (see Exercise 5.40) says that the probability that $\hat{\mu}_n$ is farther away than one standard deviation $\varepsilon = \frac{1}{2\sqrt{n}}$ from $\frac{1}{2}$ is approximately

$$P\left(\left|\hat{\mu}_n - \frac{1}{2}\right| \geq \frac{1}{2\sqrt{n}}\right) \approx 1 - 0.68 = 0.32.$$

Similarly, the law of large numbers says that the probability that $\hat{\mu}_n$ is more than two standard deviations ($\varepsilon = \frac{1}{\sqrt{n}}$) away from $\frac{1}{2}$ is bounded by

$$P\left(\left|\hat{\mu}_n - \frac{1}{2}\right| \geq \frac{1}{\sqrt{n}}\right) \leq 0.25.$$

But the central limit theorem, combined with the 68-95-99.7 rule, says that

$$P\left(\left|\hat{\mu}_n - \frac{1}{2}\right| \geq \frac{1}{\sqrt{n}}\right) \approx 1 - 0.95 = 0.05.$$

6.3.2 Approximation of Common Distributions by Normal

Many distributions correspond to sums of i.i.d. random variables. The central limit theorem says that when the number of terms in the sum is large enough, the distribution is close to a normal distribution. This allows us to approximate many distributions with a normal distribution.

Binomial

If $X \sim \text{Binomial}(n, p)$, then X has the same distribution as a sum $\sum_{i=1}^{n} X_i$, where the $X_i \sim \text{Bernoulli}(p)$ are independent. Recall that the mean and variance of the X_i are p and $p(1-p)$, respectively. Therefore, when n is large enough, the central limit theorem says that X is approximately distributed as $\mathcal{N}(np, np(1-p))$, that is,

$$\text{Binomial}(n, p) \approx \mathcal{N}(np, np(1-p)).$$

This means that if $Y \sim \mathcal{N}(np, np(1-p))$, then

$$F_X(x) \approx F_Y(x) = \frac{1}{\sqrt{2\pi np(1-p)}} \int_{-\infty}^{x} e^{-\frac{(t-np)^2}{2np(1-p)}}\, dt. \tag{6.19}$$

See Figure 6.5 for a plot of the binomial p.m.f. and the corresponding normal p.d.f.

Although this is a good approximation for large n, we do still have a problem arising from the fact that X is discrete, while Y is continuous. Thus, $P(X = k)$ is nonzero while $P(Y = k) = 0$. But observe that $P(X = k) = P\left(k - \frac{1}{2} \leq X \leq k + \frac{1}{2}\right)$, and this is well approximated by $P\left(k - \frac{1}{2} \leq Y \leq k + \frac{1}{2}\right)$:

$$\begin{aligned} P(X = k) &= P\left(k - \frac{1}{2} \leq X \leq k + \frac{1}{2}\right) \\ &\approx P\left(k - \frac{1}{2} \leq Y \leq k + \frac{1}{2}\right) = F_Y\left(k + \frac{1}{2}\right) - F_Y\left(k - \frac{1}{2}\right). \end{aligned}$$

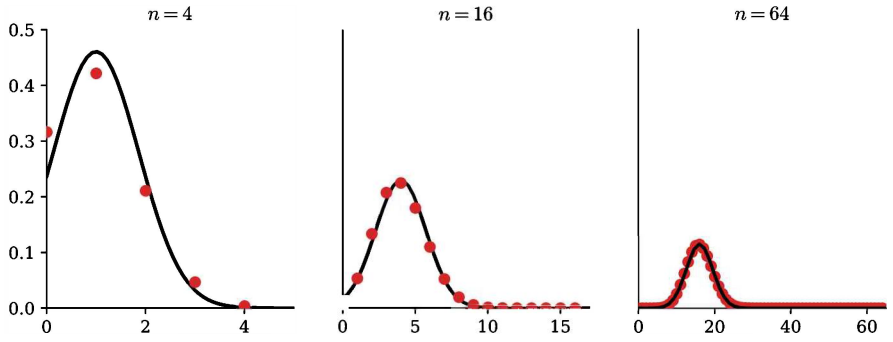


Figure 6.5. *Plots of the p.m.f.s for* Binomial(n,p) *(red) and p.d.f.s for* $\mathcal{N}(np, np(1-p))$ *(black) for* $p = 0.25$ *and various values of* n*. As* n *grows the binomial distribution is increasingly well approximated by the normal distribution.*

This is called the *continuity correction*. It also applies to larger intervals, namely, for any $k, \ell \in \{0, \dots, n\}$ we have

$$\begin{aligned} P(k \le X \le \ell) &= P\left(k - \frac{1}{2} \le X \le \ell + \frac{1}{2}\right) = \sum_{m=k}^{\ell} P\left(m - \frac{1}{2} \le X \le m + \frac{1}{2}\right) \\ &\approx \sum_{m=k}^{\ell} P\left(m - \frac{1}{2} \le Y \le m + \frac{1}{2}\right) = P\left(k - \frac{1}{2} \le Y \le \ell + \frac{1}{2}\right) \\ &= F_Y\left(\ell + \frac{1}{2}\right) - F_Y\left(k - \frac{1}{2}\right). \end{aligned}$$

Example 6.3.4. A really bored student rolls a fair die 900 times and records each outcome. What is the probability that the number 6 appears between 150 and 200 times? The number X of times that 6 appears is binomially distributed with parameters $n = 900$ and $p = \frac{1}{6}$, so this probability could be found as

$$P(150 \le X \le 200) = \sum_{x=150}^{200} g_X(x) = \sum_{x=150}^{200} \binom{900}{x} p^x (1-p)^{900-x}.$$

Alternatively, observe that X is a sum of 900 i.i.d. Bernoulli random variables, with $p = \frac{1}{6}$, so by the central limit theorem this should be closely approximated by the normal distribution $\mathcal{N}(np, np(1-p)) = \mathcal{N}(150, 125)$. Therefore, we have

$$P(150 \le X \le 200) \approx \frac{1}{\sqrt{250\pi}} \int_{149.5}^{200.5} e^{-\frac{(t-150)^2}{250}}\, dt = 0.5178.$$

Compare this to the exact answer of 0.5138677670284817.

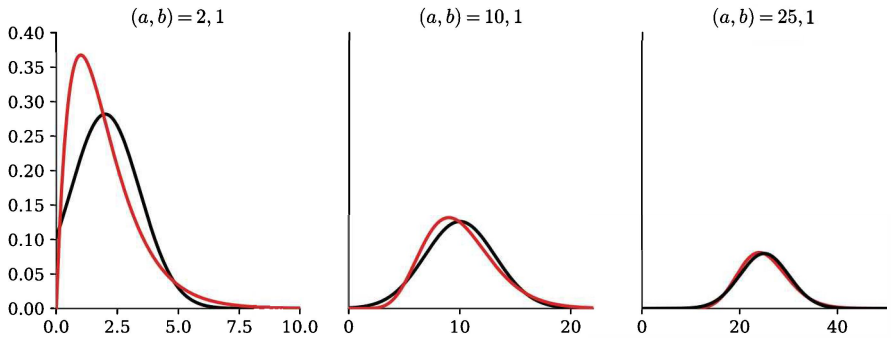


Figure 6.6. *Plot of the p.d.f.s for* $\text{Gamma}(a,b)$ *(red) and* $\mathcal{N}(\frac{a}{b}, \frac{a}{b^2})$ *(black) for various values of* a *(with* b *fixed at* 1*). As* a *grows, the gamma distribution is increasingly well approximated by the normal distribution.*

Gamma

Recall that if X is the waiting time for a events to occur in a homogeneous Poisson process of rate b, then $X \sim \text{Gamma}(a,b)$ (see Section 5.6.2). If we let X_i be the waiting time from event number $i-1$ to event number i, then X has the same distribution as $\sum_{i=1}^{a} X_i$. Moreover, the X_i are independent and each X_i is also the waiting time for a single event with the same rate b to occur, so $X_i \sim \text{Gamma}(1,b)$ with $\mu = \frac{1}{b}$ and $\sigma^2 = \frac{1}{b^2}$. By the central limit theorem, we have

$$\text{Gamma}(a,b) \approx \mathcal{N}(a\mu, a\sigma^2) = \mathcal{N}\left(\frac{a}{b}, \frac{a}{b^2}\right)$$

when a is large. An example of this is shown in Figure 6.6.

Poisson

The sum of two independent Poisson-distributed random variables is again Poisson distributed. Specifically, if $X \sim \text{Poisson}(\lambda_1)$ and $Y \sim \text{Poisson}(\lambda_2)$ are independent, then

$$\begin{aligned} g_{X+Y}(z) = P(X+Y=z) &= P\left(\bigcup_{x+y=z} (\{X=x\} \cap \{Y=y\})\right) \\ &= \sum_{x+y=z} g_X(x) g_Y(y) = \sum_{x=0}^{z} g_X(x) g_Y(z-x) \\ &= \sum_{x=0}^{z} \frac{\lambda_1^x e^{-\lambda_1}}{x!} \frac{\lambda_2^{z-x} e^{-\lambda_2}}{(z-x)!} = e^{-\lambda_1-\lambda_2} \sum_{x=0}^{z} \frac{\lambda_1^x}{x!} \frac{\lambda_2^{z-x}}{(z-x)!} \\ &= e^{-\lambda_1-\lambda_2} \frac{(\lambda_1+\lambda_2)^z}{z!}, \end{aligned}$$

where the last equality follows from the binomial theorem. This implies that $X + Y \sim \text{Poisson}(\lambda_1 + \lambda_2)$. Therefore, if $X \sim \text{Poisson}(\lambda)$, we can write X as $\sum_{i=1}^{n} X_i$ with $X_i \sim \text{Poisson}(\frac{\lambda}{n})$. The central limit theorem gives a normal approximation:

$$\text{Poisson}(\lambda) \approx \mathcal{N}(\lambda, \lambda).$$

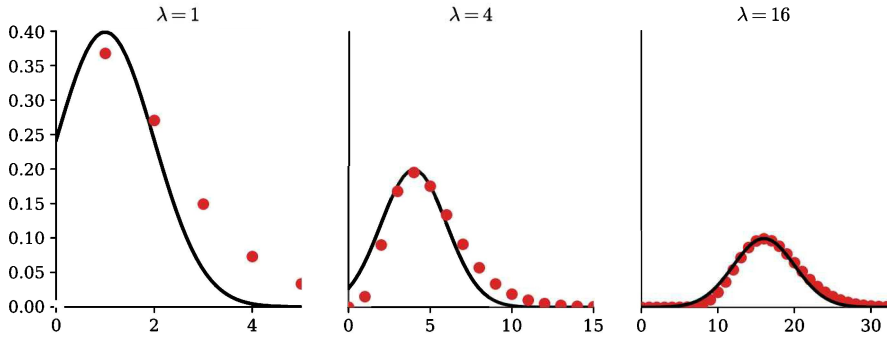


Figure 6.7. *Plot of the p.m.f.s for* Poisson(λ) *(red) and the p.d.f.s for* $\mathcal{N}(\lambda, \lambda)$ *(black) for various values of* λ*. As* λ *grows the Poisson distribution is increasingly well approximated by the normal distribution.*

See Figure 6.7 for a plot of the Poisson p.m.f. and the corresponding normal p.d.f.

As in the case of the binomial, since the Poisson distribution is discrete, the approximation can be improved by using a continuity correction:

$$\begin{aligned} P(X = k) &= P\left(k - \frac{1}{2} \leq X \leq k + \frac{1}{2}\right) \approx P\left(k - \frac{1}{2} \leq Y \leq k + \frac{1}{2}\right) \\ &= F_Y\left(k + \frac{1}{2}\right) - F_Y\left(k - \frac{1}{2}\right) \end{aligned}$$

and

$$P(k \leq X \leq \ell) \approx F_Y\left(\ell + \frac{1}{2}\right) - F_Y\left(k - \frac{1}{2}\right),$$

where $X \sim \text{Poisson}(\lambda)$ and $Y \sim \mathcal{N}(\lambda, \lambda)$.

6.4 *Proof of the Central Limit Theorem

In this section we prove the central limit theorem (Theorem 6.3.1). To complete this proof we first need to develop the idea of *characteristic functions*.

6.4.1 Characteristic Functions

Definition 6.4.1. *Let* $X : \Omega \to \mathbb{R}$ *be a univariate random variable. The function* $\varphi_X(t) = E[e^{itX}]$ *is called the* characteristic function *of the random variable* X.[30]

Proposition 6.4.2. *For any discrete or continuous random variable* $X : \Omega \to \mathbb{R}$*, the characteristic function* $\varphi_X(t)$ *exists for all* $t \in \mathbb{R}$.

Proof. If X is a continuous random variable with p.d.f. $f_X(x)$, then for any $t \in \mathbb{R}$, the value $\varphi_X(t)$ exists if the integral $\int_{\mathbb{R}} e^{itx} f_X(x)\, dx$ is absolutely convergent. This

[30] Those who are familiar with the Fourier transform should recognize that the characteristic function of a continuous random variable X is equal to the Fourier transform of the p.d.f. of X, up to a sign. That is, if $f_X(x)$ is the p.d.f. of X, then $\varphi_X(t) = \int_{-\infty}^{\infty} f_X(x) e^{itx}\, dx = \hat{f}_X(-t)$.

is clear because $|e^{itx}| = 1$ for all x and t, giving

$$\int_{\mathbb{R}} \left|e^{itx} f_X(x)\right| \, dx = \int_{\mathbb{R}} f_X(x) \, dx = 1.$$

The proof in the discrete case is similar. □

Example 6.4.3. Let Z have standard normal distribution with p.d.f.

$$f_Z(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

By completing the square, we have

$$\begin{aligned}
\varphi_Z(t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{x^2}{2}} e^{itx} \, dx = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x^2+2ixt)} \, dx \\
&= \frac{e^{-\frac{t^2}{2}}}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x+it)^2} \, dx \\
&= e^{-\frac{t^2}{2}}.
\end{aligned}$$

The integral $\int_{-\infty}^{\infty} e^{-\frac{1}{2}(x+it)^2} \, dx$ in the penultimate line is equal to $\sqrt{2\pi}$, as can be seen with a substitution, using the results of Exercise 2.5. Thus

$$\varphi_Z(t) = e^{-t^2/2}.$$

Proposition 6.4.4. *For any univariate random variables X and Y, the following properties of characteristic functions hold:*

(i) $\varphi_X(0) = 1$ *and* $|\varphi_X(t)| \leq 1$ *for all* $t \in \mathbb{R}$.

(ii) $\varphi_{-X}(t) = \overline{\varphi_X(t)}$.

(iii) *If X and Y are independent with $Z = X + Y$, then* $\varphi_Z(t) = \varphi_X(t)\varphi_Y(t)$.

(iv) *For any constant $a \in \mathbb{C}$ the function* $\varphi_{aX}(t) = \varphi_X(at)$.

Proof. The proof is Exercise 6.19. □

We need one additional theorem about convergence of characteristic functions. This theorem says that pointwise convergence of characteristic functions implies pointwise convergence of the corresponding distributions. We do not include the proof here because it would take us too far afield, but the interested reader can find it in many standard books on advanced probability, such as [Fel71, Kle08] or [Shi84].

Theorem 6.4.5 (Lévy Continuity [Kle08, Theorem 15.23]). *If $X_1, X_2, \ldots$ is a sequence of univariate random variables, and if there exists a univariate random variable X such that $\varphi_{X_i} \to \varphi_X$ pointwise, as $n \to \infty$, then the c.d.f.s F_{X_i} converge pointwise to F_X as $n \to \infty$.*

6.4.2 Proof of the Central Limit Theorem

We can now use characteristic functions to give a proof of the central limit theorem. First we need two lemmata.

Lemma 6.4.6. *If $(w_1, \ldots, w_n)$ and $(z_1, \ldots, z_n)$ satisfy $|w_i| \leq 1$ and $|z_i| \leq 1$ for all $i \in \{1, \ldots, n\}$, then*

$$\left| \prod_{i=1}^{n} z_i - \prod_{i=1}^{n} w_i \right| \leq \sum_{i=1}^{n} |z_i - w_i|. \tag{6.20}$$

Proof. The proof is by induction. The base case of $n = 1$ is immediate. Assume now that the lemma holds for $n - 1$. A little algebraic manipulation shows that

$$\prod_{i=1}^{n} z_i - \prod_{i=1}^{n} w_i = (z_n - w_n) \left(\prod_{i=1}^{n-1} z_i \right) + w_n \left(\prod_{i=1}^{n-1} z_i - \prod_{i=1}^{n-1} w_i \right).$$

Applying the triangle inequality and the bounds $|w_i| \leq 1$ and $|z_i| \leq 1$ gives

$$\left| \prod_{i=1}^{n} z_i - \prod_{i=1}^{n} w_i \right| \leq |z_n - w_n| + \left| \prod_{i=1}^{n-1} z_i - \prod_{i=1}^{n-1} w_i \right|. \tag{6.21}$$

Applying the induction hypothesis to (6.21) gives (6.20). ∎

Lemma 6.4.7. *Fix $c > 0$ and let $\psi(n, t) = 1 - \frac{ct^2}{n} + h(n, t)$ be a function of $n \in \mathbb{Z}^+$ and $t \in \mathbb{R}$ such that $|h(n, t)| \in o\left(\frac{1}{n}\right)$. We have*

$$\psi(n, t)^n \to e^{-ct^2} \qquad \text{pointwise, as } n \to \infty. \tag{6.22}$$

Proof. Since $|h(n, t)| \in o\left(\frac{1}{n}\right)$ as $n \to \infty$, we have $|h(n, t)| \leq |\frac{ct^2}{n}| \leq 1$ for fixed t and large n. Therefore, $|\psi(n, t)| \leq 1$ for large n. Applying (6.20) with $z_i = \psi(n, t)$ and $w_i = (1 - ct^2/n)$ for every $i \in \{1, \ldots, n\}$ gives

$$|\psi(n, t)^n - (1 - ct^2/n)^n| \leq n|h(n, t)|.$$

Taking the limit as $n \to \infty$ gives $\lim_{n \to \infty} \psi(n, t)^n = e^{-ct^2}$, as required. ∎

We are now ready to prove the central limit theorem (Theorem 6.3.1).

Proof. It suffices to prove the result for $\mu = 0$ and $\sigma = 1$ and $Y_n = S_n/\sqrt{n}$ (see Exercise 6.17). Since the X_i are identically distributed, we have $\varphi_{X_i}(t) = \varphi_{X_j}(t)$ for any $i, j \in \{1, \ldots, n\}$. Denote their common characteristic function by $\varphi_X(t)$.

By Proposition 6.4.4 we have

$$\varphi_{Y_n}(t) = \varphi_{S_n/\sqrt{n}}(t) = \varphi_{\sum \frac{X_i}{\sqrt{n}}}(t) = \left(\varphi_{\frac{X}{\sqrt{n}}}(t)\right)^n = \left(\varphi_X\left(\frac{t}{\sqrt{n}}\right)\right)^n.$$

We rewrite the characteristic function $\varphi_X\left(\frac{t}{\sqrt{n}}\right)$ by expanding the exponential as a Taylor series (see Theorem 10.3.7) and using the fact that $\mu = 0$ and $\sigma = 1$:

$$\begin{aligned}
\varphi_X\left(\frac{t}{\sqrt{n}}\right) &= \int_{-\infty}^{\infty} e^{itx/\sqrt{n}} f_X(x)\,dx \\
&= \int_{-\infty}^{\infty} (1 + \frac{itx}{1!\sqrt{n}} + \frac{(itx)^2}{2!n} + \frac{(itx)^3}{3!n^{3/2}} + \cdots) f_X(x)\,dx \\
&= \int_{-\infty}^{\infty} f_X(x)\,dx + \frac{it}{\sqrt{n}} \int_{-\infty}^{\infty} x f_X(x)\,dx + \frac{(it)^2}{2n} \int_{-\infty}^{\infty} x^2 f_X(x)\,dx \\
&\quad + \frac{(it)^3}{6n^{3/2}} \int_{-\infty}^{\infty} (x^3 + \cdots) f_X(x)\,dx \\
&= 1 + \frac{it}{\sqrt{n}}\mu - \frac{t^2}{2n}\sigma^2 + \frac{(it)^3}{6n^{3/2}} \int_{-\infty}^{\infty} (x^3 + \cdots) f_X(x)\,dx \\
&= 1 - \frac{t^2}{2n} + h(n,t),
\end{aligned}$$

where $h(n,t) \in o\left(\frac{1}{n}\right)$. Lemma 6.4.7 gives

$$\varphi_{Y_n}(t) = \left(\varphi_X\left(\frac{t}{\sqrt{n}}\right)\right)^n \to e^{-\frac{t^2}{2}}.$$

So the characteristic functions $\varphi_{Y_n}(t)$ of Y_n converge to the characteristic function $\varphi_Z(t)$ of the standard normal distribution $Z \sim \mathcal{N}(0,1)$. By Lévy continuity (Theorem 6.4.5), we have $F_{Y_n} \to F_Z$ pointwise, as $n \to \infty$. In particular, (6.18) holds for all points $y \in \mathbb{R}$. □

6.5 Bayesian Statistics

Probabilities do not describe reality—only our information about reality.
—E. T. Jaynes

Maximum likelihood and other methods of estimation discussed earlier in this chapter are all about making a single point estimate for an unknown parameter. In Bayesian statistics, rather than computing a single point estimate, we compute a distribution for the parameter. A key feature of Bayesian statistics is recognizing that if a parameter is unknown, it should be treated as a random variable in its own right and therefore should have a corresponding distribution. New data can be incorporated into the model via the conditional probability of the parameter, given the data. This is done using Bayes' rule, which gives an improved estimate of the distribution of the parameter. The initial distribution for the parameter is called the *prior distribution* and the updated distribution, accounting for the data, is called the *posterior distribution*.

More precisely, let $X_1, \ldots, X_n$ be a sample of random variable X whose distribution $P(x \mid \theta) = P(X = x \mid \Theta = \theta)$ depends on an unknown parameter Θ. Assume Θ is a random variable with an initial (prior) distribution $P(\theta) = P(\Theta = \theta)$. Given a draw $\mathbf{x} = (x_1, \ldots, x_n)$, the conditional probability $P(\theta \mid \mathbf{x}) = P(\Theta = \theta \mid X = \mathbf{x})$ gives an updated (posterior) distribution for Θ that takes into account the information from the data. This conditional probability can be calculated using Bayes' rule. If X and Θ have discrete distributions, then we have

$$P(\theta \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \theta)P(\theta)}{\sum_{\theta'} P(\mathbf{x} \mid \theta')P(\theta')} = \frac{\prod_{i=1}^{n} P(x_i \mid \theta)P(\theta)}{\sum_{\theta'} \prod_{i=1}^{n} P(x_i \mid \theta')P(\theta')}, \tag{6.23}$$

where the sums in the denominators run over all possible values of the parameter θ'.

Example 6.5.1. Suppose there are two coins, one fair and one which comes up heads with probability 0.25. The coins look and feel identical. One of the coins is randomly selected and we are asked to determine whether it is the fair coin or the unfair coin. The natural way to approach this is by flipping the coin repeatedly and recording the results.

The outcome X of flipping the coin has a Bernoulli distribution with unknown parameter Θ:

$$P(x \mid \theta) = P(X = x \mid \Theta = \theta) = g_X(x) = \theta^x (1 - \theta)^{1-x}.$$

Since the coin was chosen randomly, the probability of each of the two possibilities is the same:

$$P(\Theta = 0.25) = P(\Theta = 0.5) = 0.5.$$

This is called the *prior distribution* of Θ. Flipping the coin once does not give enough information to determine the value of Θ, but we can incorporate the result into the model using conditional probability. Assume the outcome of the first flip is T (so $X_1 = 0$). By Bayes' rule, we have

$$\begin{aligned} &P(\Theta = 0.25 \mid X = 0) \\ &\quad = \frac{P(X = 0 \mid \Theta = 0.25)P(\Theta = 0.25)}{P(X = 0 \mid \Theta = 0.25)P(\Theta = 0.25) + P(X = 0 \mid \Theta = 0.5)P(\Theta = 0.5)} \\ &\quad = \frac{0.75 \cdot 0.5}{0.75 \cdot 0.5 + 0.5 \cdot 0.5} = 0.6. \end{aligned}$$

So we have a new (posterior) distribution for Θ:

$$P(\theta) = \begin{cases} 0.6 & \text{if } \theta = 0.25, \\ 0.4 & \text{if } \theta = 0.5. \end{cases}$$

This procedure can now be repeated. The posterior distribution for the last trial becomes the prior for the next trial. Assume that flipping the coin a

second time gives the outcome $X_2 = 0$. Again Bayes' rule gives the conditional probability

$$\begin{aligned} &P(\Theta = 0.25 \mid X = 0) \\ &\quad = \frac{P(X = 0 \mid \Theta = 0.25)P(\Theta = 0.25)}{P(X = 0 \mid \Theta = 0.25)P(\Theta = 0.25) + P(X = 0 \mid \Theta = 0.5)P(\Theta = 0.5)} \\ &\quad = \frac{0.75 \cdot 0.6}{0.75 \cdot 0.6 + 0.5 \cdot 0.4} \approx 0.692, \end{aligned}$$

giving a new (posterior) distribution for Θ:

$$P(\theta) = \begin{cases} 0.692 & \text{if } \theta = 0.25, \\ 0.307 & \text{if } \theta = 0.5. \end{cases}$$

For continuous distributions the corresponding version of Bayes' rule[31] replaces probability with density functions: $P(x \mid \theta)$ is replaced with the p.d.f. $f(x \mid \theta)$ and the probability $P(\mathbf{x} \mid \theta)$ with the likelihood $\prod_{i=1}^{n} f(x_i \mid \theta)$; the probability $P(\theta)$ is also replaced with a p.d.f., which we still denote by $P(\theta)$, and the conditional probability $P(\theta \mid \mathbf{x})$ is replaced with a conditional p.d.f., which we also denote $P(\theta \mid \mathbf{x})$. We have

$$P(\theta \mid \mathbf{x}) = \frac{f(\mathbf{x} \mid \theta)P(\theta)}{\int f(\mathbf{x} \mid \theta')P(\theta')\,d\theta'}, \tag{6.24}$$

where, as in the discrete case, the integral in the denominator runs over all possible values of θ'.

Remark 6.5.2. It is common to use the notation of (6.24), regardless of whether the distributions of X and Θ are discrete or continuous, with the understanding that the obvious substitutions (integrals to sums and p.d.f.s to p.m.f.s) should be made as needed.

Remark 6.5.3. If needed, we can take Θ to be multivariate, that is, a vector of parameters. For example, if the parameter distribution were normal, we would write $\Theta = (\mu, \sigma^2)$ to represent the two parameters and the marginalization in (6.24) would be an integral over the domain of both variables.

Nota Bene 6.5.4. Almost everyone who does Bayesian statistics abuses notation and uses the same symbol for both θ, in the numerator, and θ', in the denominator, of (6.24). This is problematic because θ is a parameter for which we want to find a posterior distribution, but θ' is a dummy variable that runs over all possible values, so the entire denominator is actually a constant, independent of θ. In this book we do not indulge in this notational tradition because we want to be able to look in the mirror without shame.

[31] We do not prove the continuous version of Bayes' rule in this book, but the theorem itself is a straightforward generalization of the discrete case. We give the proof and a complete treatment in Volume 3.

6.5.1 Example: The Bernoulli Distribution

Suppose that we have a coin that may or may not be fair, but unlike Example 6.5.1 the parameter p determining the probability of heads is not limited to just two possibilities—it could be anything in the interval $[0, 1]$. To estimate p we draw the data $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ by flipping the coin n times, where each $x_i \in \{0, 1\}$. We have

$$f(\mathbf{x} \mid p) = \prod_{i=1}^{n} f(x_i \mid p) = p^{n\bar{x}}(1-p)^{n(1-\bar{x})},$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. For reasons explained below, we choose the beta distribution as the prior, with some chosen values for a and b. Thus, we have

$$X \sim \text{Bernoulli}(p) \qquad \text{and} \qquad p \sim \text{Beta}(a, b).$$

The prior density is given by

$$P(p) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} p^{a-1}(1-p)^{b-1},$$

and thus we have

$$\begin{aligned}
P(p \mid \mathbf{x}) &= \frac{\prod_{i=1}^{n} f(x_i \mid p) P(p)}{\int_0^1 \prod_{i=1}^{n} f(x_i \mid p') P(p')\, dp'} \\
&= \frac{p^{n\bar{x}}(1-p)^{n(1-\bar{x})} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} p^{a-1}(1-p)^{b-1}}{\int_0^1 (p')^{n\bar{x}}(1-p')^{n(1-\bar{x})} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} (p')^{a-1}(1-p')^{b-1}\, dp'} \\
&= \frac{p^{a+n\bar{x}-1}(1-p)^{b+n(1-\bar{x})-1}}{\int_0^1 (p')^{a+n\bar{x}-1}(1-p')^{b+n(1-\bar{x})-1}\, dp'} \\
&= \frac{\Gamma(a+b+n)}{\Gamma(a+n\bar{x})\Gamma(b+n(1-\bar{x}))} p^{a+n\bar{x}-1}(1-p)^{b+n(1-\bar{x})-1}.
\end{aligned}$$

The last equality follows because the integral in the denominator is a beta function; see (2.13). Alternatively, note that the numerator $p^{a+n\bar{x}-1}(1-p)^{b+n(1-\bar{x})-1}$ differs from the p.d.f. $f_B(p)$ of $\text{Beta}(a + n\bar{x}, b + n(1-\bar{x}))$ by a constant multiple. Since $P(p \mid \mathbf{x})$ and $f_B(p)$ are both p.d.f.s, they must both integrate to 1, and so the constant multiple must also be 1.

All this shows that $P(p \mid \mathbf{x})$ is distributed as $\text{Beta}(a + n\bar{x}, b + n(1-\bar{x}))$, so the posterior is also a beta distribution, but with parameters a plus the number of successes and b plus the number of failures. This is one reason that a beta distribution is a preferred choice for a prior of the Bernoulli distribution—the corresponding posterior has a very nice form. But, of course, the fact that it is convenient does not necessarily mean it is good reflection of reality.

A better reason to choose one of the beta distributions as a prior is that the family of beta distributions can easily represent a wide range of prior beliefs about

the parameter p of the coin. If we have reason to believe that the coin is fair, then we could choose values of a and b that make $\text{Beta}(a, b)$ have mean near 0.5. Since $\mu = \frac{a}{a+b}$ (see (5.26)), this could be accomplished by choosing $a \approx b$. The more certain we are of the fairness, the smaller we want the variance of the prior to be. The variance $\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}$ of $\text{Beta}(a, b)$ gets smaller as a and b get larger, and it goes to zero as the parameters get large. If we have great initial confidence that the coin is fair, we could reflect that by choosing large values of $a \approx b$ in the prior, and if we have low initial confidence that it is fair, we can reflect that in the prior by taking a and b small.

If we have no prior reason to believe that the coin is any more likely to have one probability than any other, we can reflect that by taking $a = b = 1$, which gives $\text{Beta}(1, 1) = \text{Uniform}([0, 1])$. Even bimodal priors are possible with the beta family when $a, b < 1$; if $a = b$ is less than 1, then the beta distribution approaches infinity at each end; see Figure 5.13. Thus the beta family of distributions is fairly expressive and can match many different prior beliefs about p.

Now suppose that the coin was weighted so that it landed on heads 40% of the time; see Figure 6.8. Over the long run the posterior would look like $\text{Beta}(a + 0.4n, b + 0.6n)$, which is essentially indistinguishable from $\text{Beta}(0.4n, 0.6n)$ when n is large enough. In other words, in the long run, regardless of the value of a and b, the prior becomes less and less relevant and the data dominate the shape of the posterior. When this happens, it is said that the data *swamps* the prior.

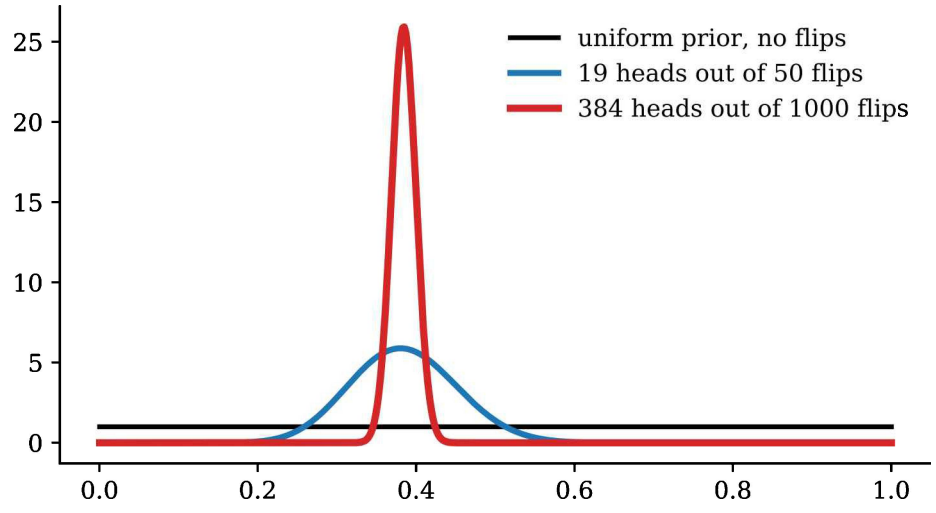


Figure 6.8. *Graph of the p.d.f.s for* $\text{Beta}(1, 1)$, $\text{Beta}(20, 32)$, *and* $\text{Beta}(385, 617)$ *corresponding to draws of the weighted coin for* $n = 0$, $n = 50$, *and* $n = 1000$, *respectively. In the limit as* $n \to \infty$, *the variance shrinks to zero and the probability distribution takes on the shape of an infinitely tall, infinitesimally narrow spike of area* 1. *Assuming that the coin is actually weighted to land on heads* 40% *of the time, the spike should occur at* 0.4. *In this example after* 1000 *flips, we don't have exactly* 400 *heads, of course, but the likelihood of* $p = 0.4$ *in the corresponding posterior distribution is very high.*

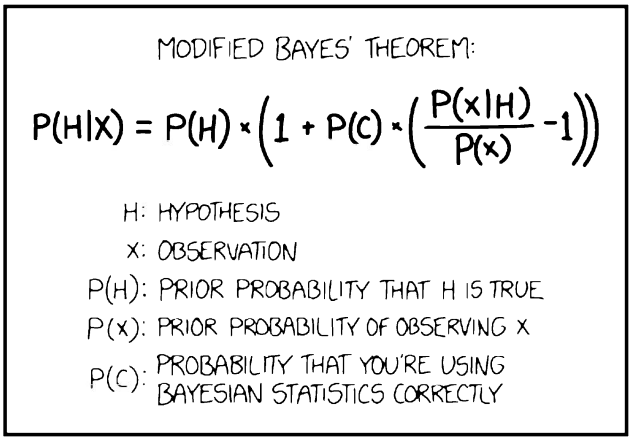


Figure 6.9. *Bayesian statistics can be very powerful when used correctly. Source: XKCD, Randall Munroe.* `http://xkcd.com/2059/`

6.5.2 MAP Estimate

Unlike the other estimation methods discussed earlier in this chapter, the Bayesian approach does not give a single point estimate of the parameter Θ but instead gives a distribution for Θ. If we must choose a single value $\Theta = \theta$, a natural choice would be the mode (the value that maximizes the p.d.f.) of the posterior $P(\theta \mid \mathbf{x})$, assuming this exists and is unique.[32] The mode of the posterior distribution, if it exists and is unique, is called the *maximum a posteriori estimate (MAP).*

Example 6.5.5. In the case of the Bernoulli distribution with Beta(a, b) as prior and Beta$(a + n\bar{x}, b + n(1 - \bar{x}))$ as posterior, the MAP is found by maximizing the p.d.f. $f(p)$ of the posterior. This is done by differentiating $\log(f)$:

$$\begin{aligned}\frac{d}{dp}\log(f(p)) &= \frac{d}{dp}\left((a + n\bar{x} - 1)\log(p) + (b + n(1 - \bar{x}) - 1)\log(1 - p)\right) \\ &= \frac{a + n\bar{x} - 1}{p} - \frac{b + n(1 - \bar{x}) - 1}{1 - p}.\end{aligned}$$

Setting this to zero and solving for p gives the MAP

$$\hat{p}_{MAP} = \frac{a + n\bar{x} - 1}{a + b + n - 2}. \tag{6.25}$$

The MAP depends on the choice of prior, but when the prior is uniform, the MAP agrees with the MLE. In the case of the Bernoulli distribution, as discussed in the previous subsection, the MLE for p is $\hat{p}_{\text{MLE}} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. But if the

[32] Not every distribution has a mode. For example, the distribution Beta$(0.5, 0.5)$ has no mode, because its p.d.f. approaches infinity at either end of the interval $(0, 1)$; see Figure 5.13.

prior is uniform, the MAP is the mode of $\text{Beta}(1 + n\bar{x}, 1 + n(1 - \bar{x}))$, which is also $\bar{x}$, by (6.25). That is to say, starting with the uniform distribution as the prior and taking the most likely value of the parameter p with the posterior distribution gives exactly the MLE for the parameter. The relationship between the MAP and the MLE is a general phenomenon for uniform priors, as the next proposition shows.

Proposition 6.5.6. *For any given p.d.f. $f(\mathbf{x}, \theta)$, if θ is known to lie in the interval $[a, b]$ and the prior distribution $P(\theta)$ is uniform on $[a, b]$, then the mode of the posterior distribution is the MLE for θ:*

$$\hat{\theta}_{\text{MLE}} = \text{mode}(P(\theta \mid \mathbf{x})).$$

Proof. The proof is Exercise 6.26. □

6.5.3 Conjugacy

As described in Section 6.5.1, the Bernoulli distribution with a beta prior results in a posterior distribution that is also beta distributed. But in general there is no reason to expect the prior and posterior distributions to be of the same type. In the special case that they are of the same type, we say that type of distribution is *conjugate* to the likelihood. The beta distribution (or rather the beta family of distributions) is conjugate to the Bernoulli likelihood; see Table 6.1 for a list of common likelihoods and their priors.

Conjugacy is a truly special relationship not enjoyed by most distributions. In general, one has to compute nasty integrals to get the posterior. Since there's no simple functional form for most of these integrals, these are usually computed numerically using quadrature (see Sections 9.6 and 9.7) or Monte Carlo methods (see Section 7.1).

Only in recent years have computers become fast enough and algorithms good enough that it is practical to compute these integrals in general. As a result, Bayesian statistics wasn't taken seriously as a reasonable way to do statistics until nearly the end of the 20th century. Today, however, the tide has changed, and Bayesian statistics has taken a dominant position in statistical estimation theory, particularly for highly complex problems where other methods fail miserably.

Likelihood	Conjugate Prior
Bernoulli	Beta
Binomial	Beta
Negative Binomial	Beta
Poisson	Gamma
Gamma (shape fixed)	Gamma
Normal (variance fixed)	Normal
Normal (mean fixed)	Inverse Gamma
Multivariate Normal (covariance fixed)	Multivariate Normal

Table 6.1. *Table of some common conjugate priors. Despite what this table may seem to imply, most distributions do not have a conjugate prior.*

6.5.4 Example: $\mathbf{Gamma}(\tau, \theta)$ with Fixed Shape τ

Let $X \sim \text{Gamma}(\tau, \theta)$ for some fixed shape $\tau > 0$ and an unknown rate $\theta > 0$. The density function takes the form $\frac{1}{\Gamma(\tau)}(\theta^\tau x^{\tau-1} e^{-\theta x})$ (see (5.24)); thus, the likelihood takes the form

$$f(\mathbf{x} \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta) = \frac{\theta^{n\tau}}{\Gamma(\tau)^n} \prod_{i=1}^{n} x_i^{\tau-1} e^{-\theta x_i} = \frac{1}{\Gamma(\tau)^n} \theta^{n\tau} e^{-\theta n \bar{x}} \prod_{i=1}^{n} x_i^{\tau-1}.$$

Choosing the gamma distribution with some fixed parameters a and b for the prior $P(\theta)$ gives

$$\begin{aligned} P(\theta \mid \mathbf{x}) &= \frac{\frac{1}{\Gamma(\tau)^n} \theta^{n\tau} e^{-\theta n \bar{x}} \prod_{i=1}^{n} x_i^{\tau-1} \cdot \left(\frac{b^a}{\Gamma(a)}\right) \theta^{a-1} e^{-b\theta}}{\int_0^\infty \frac{1}{\Gamma(\tau)^n} (\theta')^{n\tau} e^{-\theta' n \bar{x}} \prod_{i=1}^{n} x_i^{\tau-1} \cdot \left(\frac{b^a}{\Gamma(a)}\right) (\theta')^{a-1} e^{-b\theta;} \, d\theta'} \\ &= Z\, \theta^{n\tau+a-1} e^{-\theta(n\bar{x}+b)}, \end{aligned} \tag{6.26}$$

where Z is independent of θ (the integral in the denominator runs over all values of θ', so the denominator is independent of any particular value of θ). Notice that, as a function of θ, the last line of (6.26) is a constant times the p.d.f. of $\text{Gamma}(a + n\tau, b + n\bar{x})$. Since both (6.26) and the p.d.f. of $\text{Gamma}(a + n\tau, b + n\bar{x})$ are p.d.f.s, they must both integrate to 1, and hence they must be the same function. Therefore, $P(\theta \mid \mathbf{x})$ is distributed as $\text{Gamma}(a+n\tau, b+n\bar{x})$. This shows that gamma is a conjugate prior to the distribution $\text{Gamma}(\tau, \theta)$ with fixed shape $\tau > 0$.

Remark 6.5.7. The expected value of the posterior $\text{Gamma}(a+n\tau, b+n\bar{x})$ is $\frac{a+n\tau}{b+n\bar{x}}$. The MAP for θ is the mode of the posterior, which is $\frac{a+n\tau-1}{b+n\bar{x}}$ whenever $a + n\tau \geq 1$. As in the case of the Bernoulli distribution with a beta prior, as n increases the data swamps the prior and the posterior look increasingly like $\text{Gamma}(n\tau, n\bar{x})$, which has its expected value equal to its mode of $\frac{\tau}{\bar{x}}$. This is the MLE estimate for θ given $\mathbf{x}$ with the original exponential distribution.

Example 6.5.8. Assume that the lifespan of a projector bulb can be modeled as a random variable X with an exponential distribution. Recall that the exponential distribution is a special case of the gamma distribution with shape 1, so $X \sim \text{Gamma}(1, \lambda)$. Suppose that our prior experience consists of observing 5 bulbs having an average lifespan of 4 months. We would like to estimate the distribution of the parameter λ, describing the rate of failure of the bulbs.

Since the observed average rate of failure $\hat{\lambda}$ is $\frac{1}{4}$, it is natural to choose a prior that has $\mathbb{E}[\lambda] = \frac{1}{4}$. Since the gamma distribution is a conjugate prior to the exponential distribution, it is convenient to choose $\text{Gamma}(a, b)$ as the prior for λ, with expected value $\frac{a}{b} = \frac{1}{4}$; for example, we could choose $\text{Gamma}(5, 20)$, corresponding to the 5 observed failures in approximately $5 \times 4 = 20$ bulb-months.

Sampling four more bulbs with a total lifespan of 26.93 bulb-months (an average lifespan of 6.73 months each) gives a posterior distribution of Gamma$(5+4, 20+26.93)$ = Gamma$(9, 46.93)$ for λ. This distribution has mean 0.19 and mode (MAP) 0.17.

Sampling another 16 bulbs with a total lifespan of 112.57 bulb-months (an average lifespan of 7.04 months each) gives a posterior distribution of Gamma$(25, 159.5)$ for λ. This has mean 0.16 and mode 0.15.

These distributions are depicted in Figure 6.10. Notice how incorporating more data gives distributions for λ with smaller and smaller variance, and the mean of each distribution is equal to the average failure rate of all the bulbs tested.

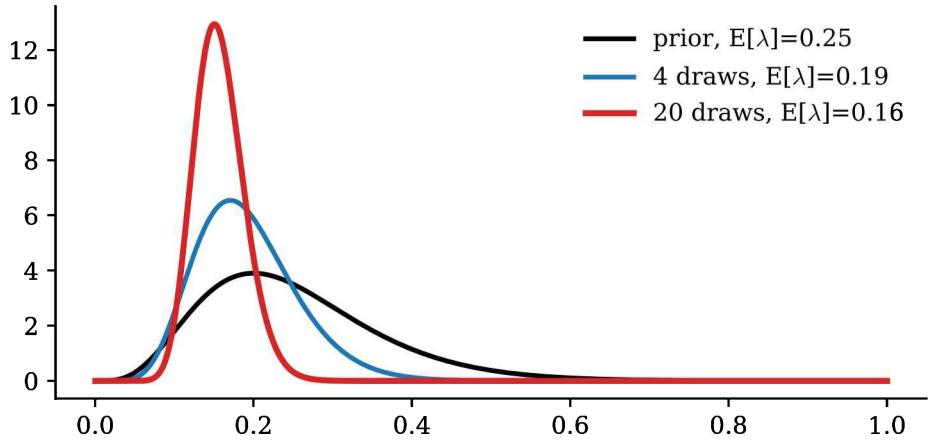


Figure 6.10. *The evolution of the p.d.f. for Example 6.5.8, beginning with a prior (black) of* Gamma$(5, 20)$*, incorporating data for four more bulbs (blue), and then for 16 more bulbs (red). In this case the mean of the final posterior distribution is* 0.16*. Note how the variance shrinks and the height (likelihood) at the MAP increases as more data are incorporated.*

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text.

Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

6.1. Prove that $\hat{s}^2$ is an unbiased estimator of σ^2, as claimed in Proposition 6.1.7.

6.2. A binomial distribution has parameters n and p. Show that for a sample $X_1, \ldots, X_N$, the estimator $\hat{p} = \frac{1}{nN}\sum_{i=1}^{N} X_i$ is an unbiased estimator for p.

6.3. Given a sample $X_1, \ldots, X_n$ of a Poisson distribution with parameter $\lambda > 0$, find the maximum likelihood estimator for λ.

6.4. Given a sample $X_1, \ldots, X_n$ of the exponential distribution Gamma$(1, \lambda)$ with rate $\lambda > 0$, find the maximum likelihood estimator for λ.

6.5. If a small number of draws (say, two or three) from the Bernoulli distribution were taken and they were all equal to 1, what would be the MLE of p? Explain how this might be a weakness of MLE.

6.6.* Let $b \in \mathbb{N}$. Let $X_1, \ldots, X_n$ be a sample without replacement from a uniform distribution on $S = \{1, 2, \ldots, b\}$ (that is, the probability of drawing $x_1 \in S$ is $\frac{1}{b}$; but then, since there is no replacement, the probability of drawing $x_2 \in S$ is $\frac{1}{b-1}$ if $x_2 \neq x_1$ and zero otherwise). Let $M = \max(X_1, \ldots, X_n)$. For $k \in S$ show that if $k < n$, then $P(M \leq k) = 0$, and if $k \geq n$, then $P(M \leq k) = \binom{k}{n}/\binom{b}{n}$. Use this to show $P(M = k) = \binom{k-1}{n-1}/\binom{b}{n}$.

6.7.* If $X_1, \ldots, X_n$ is a sample without replacement from a uniform distribution on $\{1, 2, \ldots, b\}$, and if $M = \max(X_1, \ldots, X_n)$, then show $\mathbb{E}[M] = (b+1)n/(n+1)$. Hint: Consider using (1.22). Use this to show that

$$\hat{b} = \frac{n+1}{n}\max(X_1, \ldots, X_n) - 1$$

is an unbiased estimator for b. This estimator was used by the Allies in World War II to estimate the number of tanks built by the Nazis, based on serial numbers on parts of tanks that were captured or destroyed.

6.8. Write code to sample from the Bernoulli(0.5) distribution 1000 times, simulating a repeated coin flip, and compute $\hat{\mu}_{1000}$. Repeat the experiment 100 times, saving the result each time. For each value of $\varepsilon \in \{0.1, 0.01, 0.001\}$ do the following:

(i) Find the upper bound from the law of large numbers for $n = 1000$.

(ii) Of those 100 trials, calculate the proportion of times that $|\hat{\mu}_{1000} - \mu| \geq \varepsilon$ and compare your result to the bound from the law of large numbers.

6.9. Repeat the previous problem using the distribution Beta$(1, 9)$ instead of Bernoulli(0.5).

6.10. Let $B \sim$ Binomial(n, p) be a binomial random variable for n trials with parameter p. Prove for any $\varepsilon > 0$ that

$$P\left(\left|\frac{B}{n} - p\right| \geq \varepsilon\right) \leq \frac{p(1-p)}{n\varepsilon^2}.$$

6.11. Let $X_1, X_2, \ldots$ be a sequence of mean-zero independent random variables with bounded variances (for all i, $\sigma_i^2 \leq M$ for some $M < \infty$). Prove that for any $\varepsilon > 0$ we have

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i\right| < \varepsilon\right) \to 1$$

as $n \to \infty$.

6.12. Give the details for the second part of Corollary 6.3.2 to show that if $S_n = \sum_{i=1}^{n} X_i$ is a sum of n i.i.d. random variables with mean μ and variance σ^2, then for large values of n, the distribution of S_n is approximately $\mathcal{N}(n\mu, n\sigma^2)$.

6.13. An elevator can transport a maximum of 2000 pounds. The safety plate on the elevator says the maximum occupancy is 10 persons. Experience has shown that passengers on this elevator have a mean weight of 176 pounds and standard deviation 30 pounds. Use this information and the central limit theorem to estimate the probability that a full elevator will not exceed its safe carrying capacity. Hint: Most numerical computation software (like Python or R) has built-in functions or a library with the c.d.f. of the normal distribution.

6.14. A university wants to enroll 5000 new freshmen students each year, and the most they can handle is 5500. Data on past admissions and enrollments show that 80.1% of all students admitted to the university actually decide to enroll (as opposed to going elsewhere or not going to school at all). Assuming that each student's decision to enroll is independent of the others, and each has a probability 0.801 of enrolling (that is, enrollments can be modeled as Bernoulli trials), use the central limit theorem to estimate the probability that the number of students enrolling will exceed 5500, provided the university admits 6242 students (so the expected number of enrollments is $6242 \times 0.801 = 5000$).

6.15. A fair four-sided die is rolled 800 times and each outcome recorded. Using the central limit theorem, approximate the probability that the number 4 appears between 150 and 250 times.

6.16. For each of the two distributions Beta($\frac{1}{2}, \frac{1}{2}$) and Uniform($[0, 1]$), and for each $n \in \{1, 2, 4, 8, 16, 32\}$, do the following:

(i) Find the mean μ and variance σ^2, and plot the p.d.f. of the distribution (the results of this problem do not depend on n, but you will need a separate one of these plots for each n).

Hint: Many computational systems have a built in method or a library for sampling from common distributions like the uniform and beta distributions, as well as prebuilt functions for the c.d.f. and p.d.f. of those distributions.

(ii) Plot (on the same graph as before) the p.d.f. of the normal distribution with mean μ and standard deviation $\sigma/\sqrt{n}$.

(iii) Do the following 1000 times:

(a) Draw $x_1, \ldots, x_n$ from the distribution.

(b) Compute $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

(iv) Plot a normed histogram of the results of the previous item (scaled so that the total area of the histogram is one—like a p.d.f.) on the same graph as your original distribution and the normal.

6.17.* Prove the claim at the beginning of the proof of the central limit theorem: "It suffices to prove the result for $\mu = 0$ and $\sigma = 1$ and $Y_n = S_n/\sqrt{n}$."

6.18.* Compute the characteristic function for the following distributions:

(i) Bernoulli.

(ii) Binomial.

(iii) Poisson.

6.19.* Prove Proposition 6.4.4.

6.20.* For any univariate random variable X with a characteristic function $\varphi_X(t)$ that is differentiable at $t = 0$, show that $\mathbb{E}[X] = -i\varphi'_X(0)$ (assume that derivatives and expectation commute)[33] and, more generally, that $\mathbb{E}[X^k] = (-i)^k \varphi_X^{(k)}(0)$ for any $k \in \mathbb{N}$, provided $\varphi_X^{(j)}(0)$ exists for all $j \leq k$.

6.21.* It can be shown that the characteristic function of an exponentially distributed random variable $X \sim \text{Gamma}(1, \lambda)$ is $\phi_X(t) = \frac{\lambda}{\lambda - it}$. Use this fact to give a closed formula for $\mathbb{E}[X^k]$ for all $k \in \mathbb{N}$.

6.22.* The *Fourier inversion formula* says that the characteristic function uniquely determines the distribution. In particular, if X and Y are random variables with the same characteristic function, then their c.d.f.s are equal. Use this fact to show that if $X_1 \sim \text{Poisson}(\lambda_1)$ and $X_1 \sim \text{Poisson}(\lambda_2)$ are independent, then $Y = X_1 + X_2 \sim \text{Poisson}(\lambda_1 + \lambda_2)$.

6.23.* Use characteristic functions and the Fourier inversion formula to show that if $X_1 \sim \text{Binomial}(n, p)$ and $X_1 \sim \text{Binomial}(m, p)$ are independent, then $Y = X_1 + X_2 \sim \text{Binomial}(n + m, p)$.

6.24. A coin is flipped 8 times with the outcomes H, H, H, T, T, H, H, H. Using a uniform prior for the probability p of heads, what is the posterior probability that $p \leq 0.6$? What is the probability that $p > 0.8$?

6.25. Assume that the lifespan of a projector bulb can be modeled as a random variable X with an exponential distribution of unknown parameter λ. Suppose that your data consists of observing 7 bulbs which lasted 2, 3.3, 4.5, 1.8, 3.1, 2.7, and 2.2 months, respectively.
Using the prior $\text{Gamma}(2, 6)$, find the posterior p.d.f. for λ. What is the posterior probability that $\lambda \leq \frac{1}{4}$ (corresponding to an average lifespan of at least 4 months)? Hint: The syntax of some computational systems uses the *scale* $\frac{1}{b}$ in the gamma distribution instead of the parameter b.

6.26. Prove Proposition 6.5.6, which states that whenever we start with a uniform prior on any interval $[a, b]$, the mode (the value θ which maximizes the p.d.f.) of the posterior distribution is precisely the same as the MLE $\hat{\theta}_{\text{MLE}}$. Hint:

[33] Commuting expectation and derivatives is not always possible, although it works in this case. Differentiation under an integral on a compact region is governed by Leibniz' integral rule (Volume 1, Theorem 8.6.9). Applying this to an unbounded region requires commuting limits with derivatives (see Volume 1, Theorem 6.5.11).

Use Bayes' formula to show that the likelihood $L(\theta)$ only differs from the posterior $f(\theta \mid \mathbf{x})$ by a constant multiple.

6.27. Let $X_1, \ldots, X_n$ be i.i.d. random variables with a Poisson distribution of parameter λ, and let $\mathbf{x} = (x_1, \ldots, x_n)$ be a corresponding draw. Prove that if the prior for λ is Gamma(a, b), then the posterior is Gamma$(a + n\bar{x}, b + n)$. This shows that the gamma distribution is a conjugate prior for the Poisson distribution. What is the MAP in this case? Compare the MAP to the MLE for λ, given $\mathbf{x}$.

6.28. ⚠ Given a draw $\mathbf{x} = (x_1, x_2)$ of a sample X_1, X_2 of a random variable X depending on an unknown parameter θ, there are two different ways to use Bayes' rule to compute the Bayesian posterior $P(\theta \mid \mathbf{x})$. The first way is to do it in a single step as

$$P(\theta \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \theta)P(\theta)}{\int P(\mathbf{x} \mid \theta')P(\theta')\, d\theta'}.$$

This is the method used in Section 6.5.1. The second way is to compute it in two steps by first computing

$$P(\theta \mid x_1) = \frac{P(x_1 \mid \theta)P(\theta)}{\int P(x_1 \mid \theta')P(\theta')\, d\theta'},$$

and then taking $\widetilde{P}(\theta) = P(\theta \mid x_1)$ as a new prior, computing

$$P(\theta \mid x_2) = \frac{P(x_2 \mid \theta)\widetilde{P}(\theta)}{\int P(x_2 \mid \theta')\widetilde{P}(\theta')\, d\theta'}.$$

This two-step method was used in Example 6.5.1. Prove that for any likelihood $P(x \mid \theta)$ and any prior $P(\theta)$, the final posterior distribution $P(\theta \mid \mathbf{x})$ is the same, regardless of which method is used.

6.29. Let $X \sim \mathcal{N}(\mu, \sigma^2)$ for a fixed, known value of σ^2 and an unknown value of μ. Assume that $\mu \sim \mathcal{N}(\nu, \tau^2)$ for some given, prior values of ν and τ^2.

(i) Given a single draw x of X, show that the Bayesian posterior distribution of μ is

$$\mathcal{N}\left(\frac{\tau^2 x + \sigma^2 \nu}{\sigma^2 + \tau^2}, \frac{\sigma^2 \tau^2}{\sigma^2 + \tau^2}\right).$$

(ii) Give a formula for the posterior distribution $P(\mu \mid \mathbf{x})$ of μ for data $\mathbf{x} = (x_1, \ldots, x_n)$ of n draws.

(iii) Show that the MAP converges to the MLE as $n \to \infty$.

(iv) Show that for any n and any ν, the MAP converges to the MLE as $\tau^2 \to \infty$. Note that there is no uniform distribution on $\mathbb{R}$, but as τ^2 gets large the distribution $\mathcal{N}(\nu, \tau^2)$ can be thought of as a good surrogate for a uniform distribution, so the fact that the MAP converges to the MLE can be thought of as an analogue to Proposition 6.5.6 (Exercise 6.26).

Notes

Additional introductory references about the ideas in this chapter include [Kur15, BH15, GS03, Ros07, Ros14, Was04]. Our treatment of the law of large numbers is inspired in part by [GS03]. For more about the fundamental bridge, see [BH15].

7 Random Algorithms

Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin.
—John von Neumann

A random algorithm is one that uses some notion of randomness as part of its logic. In theory, they are algorithms that use random variables; in practice, they are algorithms that use draws of those random variables.

Among the most important random algorithms are *Monte Carlo methods*, which give powerful tools for estimating quantities like the value of an integral or, equivalently, the expected value of a random variable. These are discussed in Section 7.1. In Section 7.2 we discuss methods of sampling both to compute expectations (integrals) and to draw samples from various nonuniform distributions.

In the rest of the chapter we discuss three other types of random algorithms, namely, *hashing*, *simulated annealing*, and *genetic algorithms*. Hashing is a fundamental tool for producing efficient data structures like dictionaries and sets. Simulated annealing and genetic algorithms are important methods for optimization. They are especially useful in situations where the function to optimize is not differentiable or where its derivative is not easily calculated.

7.1 Monte Carlo Methods

Monte Carlo methods[34] form a broad class of techniques that use random sampling to estimate various quantities, including high-dimensional integrals, parameters of distributions, and probabilities of various events. For example, with Monte Carlo sampling, one can estimate the expected value $\mathbb{E}[X]$ of a random variable X by computing its sample mean $\hat{\mu}$ (see (6.1) and (6.3)). In fact, many of the Monte Carlo techniques boil down to computing some kind of expectation, and in most cases the central limit theorem can be used to analyze the convergence properties of these methods.

[34]The name comes from the Monte Carlo Casino in Monaco.

Monte Carlo methods are useful in a wide variety of settings. They are often easy to implement in situations where analytic methods are difficult or even impossible. While they are not usually considered very computationally efficient, they are usually parallelizable, and for many high-dimensional problems they are the only feasible approach. Moreover, they can be applied in almost any situation where the desired answer has a probabilistic interpretation. For example, integration can be interpreted as computing an expected value, and thus Monte Carlo methods can be applied. In fact, the best methods for numerical integration in high dimensions are Monte Carlo methods.

Example 7.1.1. Since the area of a circle of radius r is $A = \pi r^2$, one way to estimate π is to estimate the area of the unit circle. A Monte Carlo approach to this problem is to uniformly sample points in the square $[-1, 1] \times [-1, 1]$ and then count the percentage of points that land within the unit circle. The percentage of points within the circle approximates the percentage of the area occupied by the circle. Multiplying this percentage by 4 (the area of the square $[-1, 1] \times [-1, 1]$) gives an estimate for the area of the circle.

The results of three such experiments, with 500, 2000, and 16,000 points, respectively, are shown in Figure 7.1. The corresponding estimates for π are 3.0880, 3.1980, and 3.1412.

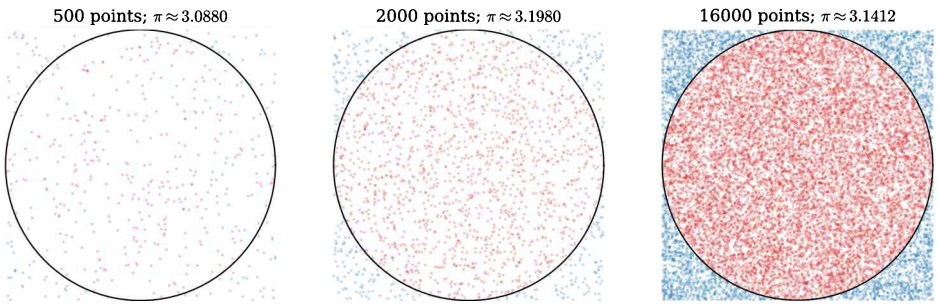


Figure 7.1. *Monte Carlo estimation of the area of the unit circle. Points are chosen uniformly from the square* $[-1, 1] \times [-1, 1]$. *The area of the square times the percentage of points landing in the circle gives an estimate for the area* π *of the circle; see Example 7.1.1.*

7.1.1 Expected Value via Monte Carlo

One of the most basic problems in the class of random algorithms is estimating the expected value $\mathbb{E}[X]$ of a random variable X. To estimate $\mu = \mathbb{E}[X]$, we take a draw $\mathbf{x} = (x_1, \ldots, x_n)$ from a sample $X_1, \ldots, X_n$ of X and compute the value of the sample mean $\hat{\mu}_{\mathbf{x}} = \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$. The law of large numbers guarantees that $\hat{\mu}_{\mathbf{x}} \to \mu$ with probability 1 as $n \to \infty$, and the central limit theorem gives additional probabilistic information on the quality of the estimate as a function of n.

More precisely, the central limit theorem (Theorem 6.3.1) guarantees that for large n the sample mean $\hat{\mu}$ is approximately distributed as $\mathcal{N}(\mu, \frac{\sigma^2}{n})$, so by the 68-95-99.7-rule (see Exercise 5.40)), the probability that $\hat{\mu}$ is within one standard deviation $\frac{\sigma}{\sqrt{n}}$ of μ is approximately 68%, and the probability that it is within two standard deviations is approximately 95%.

The standard deviation $\frac{\sigma}{\sqrt{n}}$ of the sample mean is called the *standard error* (SE) of the mean. Of course, if the purpose of the experiment is to approximate μ, then we probably also don't know σ^2, but we can use the unbiased sample variance

$$\hat{s}_{\mathbf{x}}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

(see (6.5)). Thus, a good estimate for the standard error is

$$SE = \frac{\sigma}{\sqrt{n}} \approx \frac{\hat{s}_{\mathbf{x}}}{\sqrt{n}}.$$

Therefore, the result $\bar{x}$ of a Monte Carlo experiment to estimate $\mu = \mathbb{E}[X]$ with n samples will, with approximately 95% probability, lie within two standard errors (that is, $\frac{2\hat{s}_{\mathbf{x}}}{\sqrt{n}}$) of the true mean μ.

More generally, to compute the expected value of $h(X)$ for some well-behaved function $h : \mathbb{R} \to \mathbb{R}$, observe that $Y = h(X)$ is itself a random variable with expected value $\mathbb{E}[Y] = \mathbb{E}[h(X)]$. Again, we can estimate this value with a Monte Carlo experiment, by drawing $\mathbf{y} = (y_1, \ldots, y_n)$ from a sample $Y_1, \ldots, Y_n$. But each Y_i is $h(X_i)$, so to sample from Y we can apply h to a sample from X. This gives

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{n}\sum_{i=1}^{n} h(x_i) \approx \mathbb{E}[h(X)],$$

and $P(|\bar{y} - \mathbb{E}[h(X)]| \leq \frac{2\hat{s}_{\mathbf{y}}}{\sqrt{n}})$ is approximately 95%, where

$$\hat{s}_{\mathbf{y}}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \frac{1}{n-1}\sum_{i=1}^{n}(h(x_i) - \bar{y})^2.$$

Example 7.1.2. Let $X \sim \text{Uniform}([0,1])$. To estimate the expected value of $Y = X^2$, we sample from X and compute $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} x_i^2$. Drawing $n = 10^6$ times, we found $\bar{y} = 0.333298$ and $\hat{s}_{\mathbf{y}}^2 = 0.088995$, which gives $SE = \frac{\hat{s}_{\mathbf{y}}}{\sqrt{n}} = 0.0003$. We conclude from this that the probability that $|\hat{y} - \mu|$ is no more than 0.0003 is about 68%, the probability that it is less than 0.0006 is about 95%, and the probability that it is less than 0.0009 is about 99.7%.

In many cases the expected value cannot be computed analytically, but in this special case the true answer can be computed as $\mathbb{E}[X^2] = \int_0^1 x^2\,dx = \frac{1}{3}$, and we have $|\bar{y} - \mu| \approx 0.000035$, which is well within one standard error of the mean.

Example 7.1.3. Consider a game where you repeatedly roll three distinct six-sided dice. With each roll of the three dice, if there are no doubles or triples, then you win the total amount shown (the sum of the three dice); otherwise you lose everything you have won so far.

We can use Monte Carlo methods to estimate the expected value with very little work. Let the random variable Y be the value of your stake after 10 rolls. With only a few lines of code, we can draw from $\{1, \ldots, 6\}$ uniformly for each of the three dice and compute the effect of each roll on the total stake. The result y of repeating this 10 times constitutes one experiment. Here is an example:

Roll	Outcome	Winnings	Roll	Outcome	Winnings
1	3 6 3	0	6	5 2 2	0
2	2 5 2	0	7	1 5 3	9
3	2 6 4	12	8	6 5 4	24
4	4 2 5	23	9	6 6 4	0
5	2 6 4	35	10	4 3 3	0

In this example, the final winnings are $y = 0$. Repeating the 10-roll experiment n times gives a draw $y_1, y_2, \ldots, y_n$ from a sample $Y_1, \ldots, Y_n$ of Y. From this we can quickly compute $\bar{y}$ and the standard error SE, increasing the number of samples until we are sufficiently confident in the quality of the estimate $\bar{y} \approx \mathbb{E}[Y]$.

We ran this experiment 10^5 times and found $\bar{y} = 13.1$, with $SE \approx 0.06$. So the probability that the true value of $\mathbb{E}[Y]$ lies in the interval $(12.9, 13.3)$ is greater than 99.7%; see Algorithm 7.1 for details. The analytical solution of this problem shows that $\mathbb{E}[Y] = 13.0882$, which is very close to our Monte Carlo estimate.

7.1.2 Monte Carlo Integration with Uniform Distributions

In calculus, we learn that the average of a function on the interval $[a, b]$ can be computed via the integral

$$\frac{1}{b-a}\int_a^b f(x)\,dx.$$

We can connect integration to Monte Carlo methods because this quantity is equal to $E[Y]$, where $Y = f(X)$ for $X \sim \text{Uniform}([a, b])$. This means that we can estimate the integral as

$$\int_a^b f(x)\,dx = (b-a)E[Y] \approx \frac{b-a}{n}\sum_{i=1}^{n} f(x_i),$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a draw from the sample $X_1, X_2, \ldots, X_n$ of the random variable $X \sim \text{Uniform}([a, b])$.

More generally, consider an integral of the form

$$\int_{[\mathbf{a},\mathbf{b}]} k(\mathbf{x})\,d\mathbf{x},$$

```python
import numpy as np # module for efficient linear algebra

def MC_dice_game(n=10**5):
    # Monte Carlo estimate of expected value of the dice game.

    def trial(n_rolls=10,n_dice=3):
        """ Perform a single experiment of n_rolls dice throws
        and return total_winnings (a draw y of Y)."""

        total_winnings = 0
        rolls = randint(1,7,[n_rolls,n_dice])  # array of rolls
        for roll in rolls:
            if len(np.unique(roll)) == n_dice: # if all distinct
                total_winnings += np.sum(roll)
            else:
                total_winnings = 0
        return total_winnings

    # run n total experiments and compute ybar
    trials = np.array([trial() for x in range(n)]) # n trials
    ybar = trials.mean()                    # Estimate of E[Y]
    svar = np.sum((trials-ybar)**2)/n       # Sample variance
    SE = np.sqrt(svar/n)                    # Standard error
    return(ybar, SE)
```

Algorithm 7.1. *Routine for computing a Monte Carlo estimate $\bar{y}$ of the expected value $\mathbb{E}[Y]$ for the dice game of Example 7.1.3. While the analytic computation of $\mathbb{E}[Y]$ is difficult, writing and running the code for the Monte Carlo estimate is easy.*

over a bounded interval $[\mathbf{a}, \mathbf{b}] = [a_1, b_1] \times \cdots \times [a_d, b_d] \subset \mathbb{R}^d$, for an integrable function $k(\mathbf{x})$ (the symbol $\mathbf{x}$ here is not a draw from a sample but rather just a point in $\mathbb{R}^d$). The p.d.f. of the uniform distribution on $[\mathbf{a}, \mathbf{b}]$ is $f(\mathbf{x}) = \frac{1}{\lambda([\mathbf{a},\mathbf{b}])} \mathbb{1}_{[\mathbf{a},\mathbf{b}]}$, where $\lambda([\mathbf{a}, \mathbf{b}]) = \prod_{j=1}^{d}(b_j - a_j)$, so we have

$$\begin{aligned}\int_{[\mathbf{a},\mathbf{b}]} k(\mathbf{x})\, d\mathbf{x} &= \lambda([\mathbf{a}, \mathbf{b}]) \int_{\mathbb{R}^d} k(\mathbf{x}) f(\mathbf{x})\, d\mathbf{x} \\ &= \lambda([\mathbf{a}, \mathbf{b}])\mathbb{E}[k(X)] \approx \frac{\lambda([\mathbf{a}, \mathbf{b}])}{n} \sum_{i=1}^{n} k(\mathbf{x}_i),\end{aligned}$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is a draw from the uniform distribution on $[\mathbf{a}, \mathbf{b}]$. Geometrically, this is similar to computing Riemann sums, except that Riemann sums correspond (when $d = 1$) to taking one point x_i in each interval $[a + i\Delta, a + (i + 1)\Delta]$ for each $i \in \{0, 1, \ldots, n - 1\}$, with $\Delta = \frac{b-a}{n}$, whereas the Monte Carlo method just takes n points sampled uniformly from the interval $[a, b]$; see Figure 7.2.

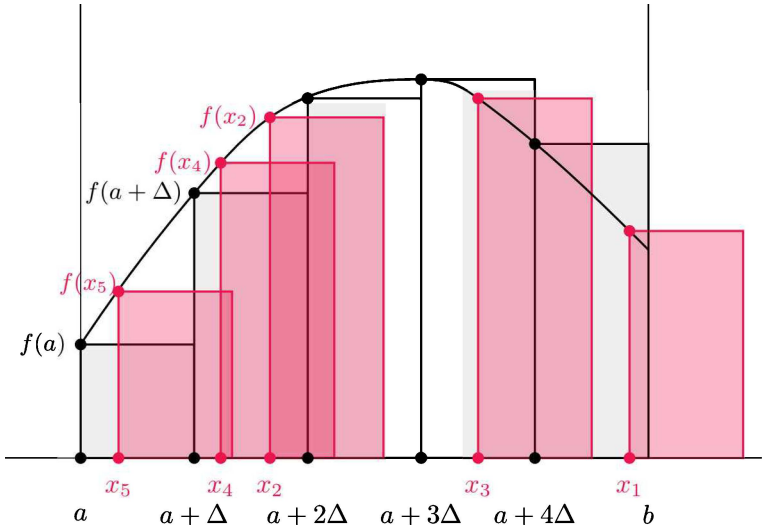


Figure 7.2. *Depiction of left Riemann sums (black/gray) versus Monte Carlo integration (red) to estimate the integral* $\int_a^b f(x)\,dx$. *Riemann sums take one sample (the leftmost point) from each interval* $[a+i\Delta, a+(i+1)\Delta]$, *whereas the Monte Carlo method takes samples drawn uniformly from the interval* $[a,b]$.

The standard error for this Monte Carlo integration is given by (we drop the subscript on $\hat{s}$)

$$SE \approx \lambda([\mathbf{a},\mathbf{b}])\frac{\hat{s}}{\sqrt{n}} = \lambda([\mathbf{a},\mathbf{b}])\sqrt{\frac{\sum_{i=1}^{n}(k(\mathbf{x}_i)-\bar{\mathbf{y}})^2}{n(n-1)}}, \tag{7.1}$$

where $\bar{\mathbf{y}} = \frac{1}{n}\sum_{i=1}^{n} k(\mathbf{x}_i)$ and $\hat{s}$ is the square root of the unbiased sample variance for the draw $k(\mathbf{x}_1),\ldots,k(\mathbf{x}_n)$.

Example 7.1.4. Example 7.1.1 gives one way to estimate the numerical value of π using Monte Carlo methods. Another approach to estimating π is to numerically estimate the integral $\int_0^1 \sqrt{1-x^2}\,dx$, which gives the area of one fourth of the unit circle, and then multiply by 4. This gives

$$\pi = 4\int_0^1 \sqrt{1-x^2}\,dx \approx \frac{4}{n}\sum_{i=1}^{n}\sqrt{1-x_i^2},$$

for a draw $x_1,\ldots,x_n$ from the uniform distribution on $[0,1]$. Drawing 500, 2000, and 16,000 times gave the results 3.12053, 3.13163, and 3.14146, respectively, and with standard errors of approximately 0.04065, 0.02006, and 0.00705, respectively. This is better than the results of Example 7.1.1, although our error of only 0.00013 in the last case of 16,000 draws seems to have been rather lucky, since the standard error is more than 5 times larger than that.

Example 7.1.5. The method used in Example 7.1.1 can also be considered an example of Monte Carlo integration of the indicator function $\mathbb{1}_A$ of the unit circle A on the square $[-1,1]\times[-1,1]$. Thus, we have

$$\pi = \int_{[-1,1]^2} \mathbb{1}_A(\mathbf{x})\,d\mathbf{x} \approx \frac{4}{n}\sum_{i=1}^{n} \mathbb{1}_A(\mathbf{x}_i).$$

7.1.3 Accuracy and High-Dimensional Integration

To improve the accuracy of the estimate $\bar{y}$ by one decimal place, we must shrink the standard error by a factor of 10. But because the denominator of the standard error is $\sqrt{n}$, to shrink it by a factor of 10 requires that n increase by a factor of 100. For one-dimensional integrals, other numerical techniques, like the quadrature methods in Sections 9.6 and 9.7, usually give more accurate results more efficiently—for example, the error in Simpson's rule shrinks like n^{-4} (assuming $f \in C^4([a,b];\mathbb{R})$) rather than $n^{-\frac{1}{2}}$, so to get one more digit of accuracy using Simpson's rule requires only that n grow by a factor of $10^{\frac{1}{4}} \approx 1.8$ instead of 100.

In two dimensions, numerical quadrature still outperforms Monte Carlo methods, but the natural analogue of Simpson's rule using n points only improves like n^{-2} instead of n^{-4}, so to improve by one digit of accuracy requires $\sqrt{10}$ times more points. In d dimensions the number of points needed for quadrature methods to achieve a given level of accuracy grows exponentially (like k^d for some constant k), so these methods rapidly become unusable. But the standard error in Monte Carlo methods is always $\frac{\hat{s}_{\mathbf{x}}}{\sqrt{n}}$, which is independent of d; see (7.1). For this reason Monte Carlo methods are still effective in high dimensions, even when other methods fail completely.

Nota Bene 7.1.6. Beware that when an integral does not converge, a Monte Carlo estimate still usually returns a finite number. You can often identify that the integral diverges by watching how the Monte Carlo estimate changes as the number of points increases.

7.2 Importance, Inversion, and Rejection Sampling

In this section we extend Monte Carlo integration to nonuniform distributions. We also discuss a method of choosing a good distribution for Monte Carlo integration called *importance sampling*. We then discuss two different methods of producing draws from various distributions, called *inversion sampling*, and *rejection sampling*.

7.2.1 Monte Carlo Integration with Nonuniform Distributions

Uniform distributions are not the only distributions that can be used for Monte Carlo integration. We can apply these ideas to any integral of the form

$$\int_{\mathbb{R}^d} k(\mathbf{x}) f_X(\mathbf{x})\,d\mathbf{x}$$

for any distribution X with p.d.f. $f_X(\mathbf{x})$, provided $k(\mathbf{x})$ is sufficiently well behaved, and provided we can sample from X. We have

$$\int_{\mathbb{R}^d} k(\mathbf{x}) f_X(\mathbf{x})\, d\mathbf{x} = \mathbb{E}[k(X)] \approx \frac{1}{n}\sum_{i=1}^{n} k(\mathbf{x}_i),$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is a draw from the distribution X instead of from the uniform distribution. Again, the standard error is (we drop the subscript)

$$SE \approx \frac{\hat{s}}{\sqrt{n}} = \frac{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(k(\mathbf{x}_i) - \bar{\mathbf{y}})^2}}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^{n}(k(\mathbf{x}_i) - \bar{\mathbf{y}})^2}{n(n-1)}},$$

where $\bar{\mathbf{y}} = \frac{1}{n}\sum_{i=1}^{n} k(\mathbf{x}_i)$.

If $f_X(\mathbf{x})$ never vanishes, then for any $h(\mathbf{x})$ we can set $k(\mathbf{x}) = \frac{h(\mathbf{x})}{f_X(\mathbf{x})}$ to calculate

$$\int_{\mathbb{R}^d} h(\mathbf{x})\, d\mathbf{x} = \int_{\mathbb{R}^d} \frac{h(\mathbf{x})}{f_X(\mathbf{x})} f_X(\mathbf{x})\, d\mathbf{x} = \mathbb{E}\left[\frac{h(\mathbf{x})}{f_X(\mathbf{x})}\right] \approx \frac{1}{n}\sum_{i=1}^{n} \frac{h(\mathbf{x}_i)}{f_X(\mathbf{x}_i)}. \tag{7.2}$$

The uniform distribution is only well defined on sets of finite volume, so if the integration runs over an infinite domain, we cannot use the uniform distribution. A natural choice when integrating over $\mathbb{R}^n$ is the normal distribution, in part because many good techniques have been developed for sampling from the standard normal distribution.

Example 7.2.1. Let $X \sim \mathcal{N}(0,1)$ have standard normal distribution. There are many efficient algorithms for sampling from the standard normal, so we can estimate the c.d.f. $F_X(a) = P(X \le a) = \int_{-\infty}^{a} f_X(x)\, dx$ by sampling from the standard normal:

$$\int_{-\infty}^{a} f_X(x)\, dx = \int_{-\infty}^{\infty} \mathbb{1}_{(-\infty,a]}(x) f_X(x)\, dx = \mathbb{E}[\mathbb{1}_{(-\infty,a]}] \approx \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{(-\infty,a]}(x_i).$$

Using this method to calculate $F_X(1)$, and taking 10^3, 10^4, and 10^5 samples, we gained the estimates 0.8350, 0.8451, and 0.8422, respectively, with standard error of 0.012, 0.004, and 0.001, respectively.

7.2.2 Importance Sampling

One of the main reasons to use a nonuniform p.d.f. is to reduce the variance of the sample and hence reduce the size of the standard error of the Monte Carlo estimate. In the rare case that $f_X(\mathbf{x})$ is exactly proportional to $h(\mathbf{x})$, the ratio $\frac{h(\mathbf{x}_i)}{f_X(\mathbf{x}_i)}$ is constant for all $\mathbf{x}_i$, which means that $\hat{s}^2 = 0$ and $SE = 0$. Choosing $f_X(\mathbf{x})$ to be close to $Mh(\mathbf{x})$ for some constant M makes the standard error of the Monte Carlo estimate small. This translates into computational efficiency gains, since fewer samples are required to achieve the desired accuracy.

Integrating by sampling from a distribution whose p.d.f. is nearly proportional to $h(\mathbf{x})$ is called *importance sampling*, because the sampling favors the more important regions that really contribute to the integral over the less important regions.

Example 7.2.2. Let $h(x) = \frac{1}{2}\operatorname{sech}(x^2)$, and consider the integral $\int_{-5}^{5} h(x)\,dx$; see Figure 7.3. Since this function has a maximum at the origin and then drops off rapidly as x moves away from the origin, it makes sense to try sampling from a normal distribution rather than from a uniform distribution. Let $X \sim \mathcal{N}(0,1)$ with p.d.f. $f_X(x)$. We can estimate $\int_{-5}^{5} h(x)\,dx$ as

$$\begin{aligned}\int_{-5}^{5} h(x)\,dx &= \int_{-\infty}^{\infty} \frac{\mathbb{1}_{[-5,5]}(x)h(x)}{f_X(x)} f_X(x)\,dx \\ &= \mathbb{E}\left[\frac{\mathbb{1}_{[-5,5]}(x)h(x)}{f_X(x)}\right] \approx \frac{1}{n}\sum_{i=1}^{n} \frac{\mathbb{1}_{[-5,5]}(x_i)h(x_i)}{f_X(x_i)},\end{aligned}$$

where $x_1, \ldots, x_n$ is a draw from X. When we run this Monte Carlo estimate with $n = 10^6$, we roughly find that $\bar{y} = 1.18361$ and $SE \approx 0.00032$. This is better than the Monte Carlo estimate using the uniform distribution, which gives roughly $\bar{y} = 1.18558$ and $SE = 0.00183$.

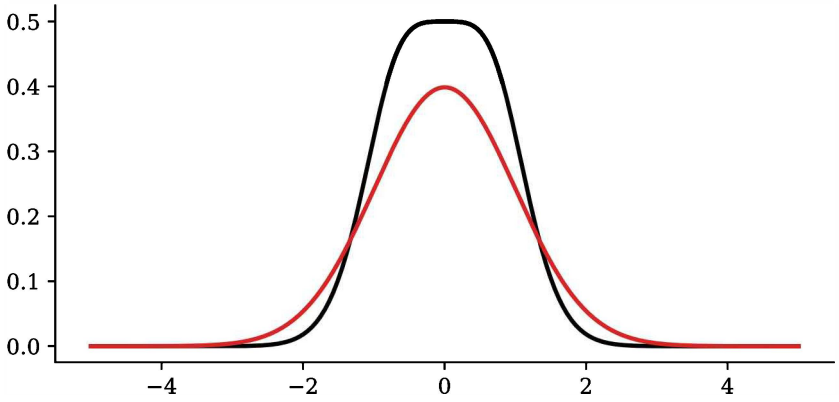


Figure 7.3. *Plot of the function $h(x) = \frac{1}{2}\operatorname{sech}(x^2)$ (black) and the p.d.f. of the standard normal distribution (red). Because the p.d.f. of the standard normal has a shape similar to h, Monte Carlo estimation of the integral $\int_{-5}^{5} h(x)\,dx$ has improved accuracy when importance sampling is used with draws taken from the standard normal distribution instead of from the uniform distribution, as discussed in Example 7.2.2.*

Unfortunately, even if the overall shape of the sampled distribution $f_X(x)$ is similar to that of $h(x)$, if there are places where $f_X(x)$ is much smaller than $h(x)$, then for a draw $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ from those places, the ratio $\frac{h(x_i)}{f_X(x_i)}$ can be very large.

This can cause the unbiased sample variance

$$\hat{s}_{\mathbf{x}}^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{h(x_i)}{f_X(x_i)} - \bar{y}\right)^2$$

to grow uncontrollably, especially if $\frac{h(x_i)}{f_X(x_i)}$ is large on an unbounded region, that is, in the tails of the distributions. Since the goal of importance sampling is to reduce the variance, rather than to let it grow, it is essential to choose a distribution $f_X(x)$ with tails that are large (fat), compared to the tails of the original function $h(x)$.

Unexample 7.2.3. Let $Y \sim \text{Gamma}(8,1)$. The obvious Monte Carlo method for computing probabilities like $P(Y \geq c) = \int_c^\infty f_Y(x)\,dx$ is to use the fundamental bridge (see Remark 6.2.3) and sample from Y to get

$$P(Y \geq c) = \mathbb{E}[\mathbb{1}_{[c,\infty)}] \approx \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{[c,\infty)}(y_i),$$

where y_i is drawn from Y. But this can work only if we have a good way to sample from Y. If not, it seems natural to try to use a normal distribution $X \sim \mathcal{N}(8,8)$, since $\mathcal{N}(8,8)$ is a good approximation to Gamma$(8,1)$; see Section 6.3.2. Trying this gives

$$\int_c^\infty f_Y(t)\,dt = \int_{-\infty}^\infty \mathbb{1}_{[c,\infty)}(t)\frac{f_Y(t)}{f_X(t)}f_X(t)\,dt \approx \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{[c,\infty)}(x_i)\frac{f_Y(x_i)}{f_X(x_i)},$$

where $x_1,\ldots,x_n$ are drawn from $\mathcal{N}(8,8)$. Unfortunately, actually running the computation with $n = 10^4$, 10^5, and 10^6, with $c = 8$ we find completely nonsensical estimates 771.9, 1455.7, and 3226.4, respectively, with standard error 185.5, 206.6, and 810.8, respectively. So the estimates and the standard error are both failing to converge. This is because the tail of $\mathcal{N}(8,8)$ is substantially smaller than that of Gamma$(8,1)$. The ratio of the two is

$$\frac{\sqrt{2\pi}t^7e^{-t}}{e^{-\frac{(t-8)^2}{16}}},$$

which diverges to infinity as $t \to \infty$.

This can be remedied by sampling from a different distribution with a larger tail. For example, the distribution on $[0,\infty)$ with p.d.f. equal to $\frac{1}{(1+t)^2}$ will do, since

$$\lim_{t\to\infty}\frac{t^7e^{-t}}{7!/(1+t)^2} = \lim_{t\to\infty}\frac{t^7(1+t)^2}{7!e^t} = 0.$$

We show how to sample from $\frac{1}{(1+t)^2}$ in Example 7.2.7.

7.2.3 Inversion Sampling

Monte Carlo methods rely on the generation of random (or pseudorandom) samples from various distributions. Most modern computing systems have high-quality methods for generating uniformly distributed and normally distributed pseudorandom numbers, but in many cases (for example, in importance sampling) one needs to sample from other distributions. A key tool for doing this is the following theorem.

Theorem 7.2.4 (Universality of the Uniform).

(i) *Let $F : (a, b) \to (0, 1)$ be bijective and increasing, with inverse F^{-1}. If $U \sim \text{Uniform}((0, 1))$, then $X = F^{-1}(U)$ is a random variable with c.d.f. equal to F (with the obvious extension that $F(x) = 0$ for all $x \leq a$ and $F(x) = 1$ for all $x \geq b$).*

(ii) *If X is a random variable with a continuous c.d.f. F, then $Y = F(X)$ is a random variable with $Y \sim \text{Uniform}((0, 1))$.*

Proof. (i) The function $F^{-1} : (0, 1) \to (a, b)$ exists and is both increasing and bijective because $F : (a, b) \to (0, 1)$ is increasing and bijective. We now show that F^{-1} is continuous by showing that for all $r, s \in (a, b)$ with $r < s$ the set $(F^{-1})^{-1}(r, s) = F((r, s))$ is equal to $(F(r), F(s))$ and hence is open in $(0, 1)$ (see Volume 1, Theorem 5.2.3). To see this, note that F is increasing, so for every $x \in (r, s)$ we have $F(r) < F(x) < F(s)$, and, hence, $F(x) \in (F(r), F(s))$ and $F((r, s)) \subset (F(r), F(s))$. But bijectivity of F implies that for every $y \in (F(r), F(s))$, there exists $z \in (a, b)$ with $y = F(z)$, and the fact that F^{-1} is increasing implies that $z \in (r, s)$; hence, $F((r, s)) = (F(r), F(s))$, which implies that F^{-1} is continuous.

Since F^{-1} is continuous, the function $X = F^{-1}(U)$ is a random variable. Since $P(U \leq u) = u$ for any $u \in (0, 1)$, the c.d.f. of X is

$$P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = \begin{cases} 0 & \text{if } F(x) \leq 0, \\ F(x) & \text{if } F(x) \in (0, 1), \\ 1 & \text{if } F(x) \geq 1. \end{cases}$$

(ii) If F is continuous, then $Y = F(X)$ is also a random variable. Its c.d.f. is

$$P(Y \leq y) = P(F(X) \leq y) = P(X \leq F^{-1}(y)) = F(F^{-1}(y)) = y.$$

Therefore, $Y \sim \text{Uniform}((0, 1))$. □

This theorem is useful for sampling from a given distribution with c.d.f. equal to F, because whenever the inverse F^{-1} is known, we can generate a sample of the original distribution by taking a sample U of the uniform distribution and computing $F^{-1}(U)$. This is called *inversion sampling*.

Remark 7.2.5. Part (i) of the theorem also holds in the case that F maps to $[0, 1]$ instead of to $(0, 1)$. The proof is essentially identical to the one given here for $(0, 1)$.

Example 7.2.6. The distribution Beta$(a, 1)$ has p.d.f. equal to $f(x) = ax^{a-1}$, and thus its c.d.f. is $F(x) = x^a$, which is strictly increasing on the support $[0, 1]$ of the distribution, and hence bijective there. Its inverse $F^{-1}(u) = u^{1/a}$ is also continuous. Therefore, to sample from Beta$(a, 1)$ we may take a sample U from Uniform$([0, 1])$ and compute $U^{1/a}$.

Example 7.2.7. Let D be a distribution with p.d.f. equal to $f(x) = \frac{1}{(1+x)^2}$ defined on $[0, \infty)$. The c.d.f. is $F(x) = \int_0^x \frac{dt}{(1+t)^2} = \frac{x}{1+x}$ and its inverse is $F^{-1}(u) = \frac{u}{1-u}$. Therefore, given a sample $U \sim$ Uniform$([0, 1))$, taking $\frac{U}{1-U}$ gives a sample from D.

7.2.4 Rejection Sampling

Unfortunately it is not always possible to compute a closed-form expression for the inverse of the c.d.f. of a distribution. Hence, inversion sampling is not always feasible. Another approach is *rejection sampling*, which uses the following two main ideas:

(i) To sample from a continuous distribution P, one can sample uniformly from the region in the plane $\mathbb{R}^2$ bounded above by the p.d.f. $f_P(x)$ of P and then project each sample down to the x-axis.

(ii) To sample uniformly from any region C, one can sample uniformly from a larger region containing C and throw away (reject) any samples that do not lie in C.

Idea (i) is illustrated in the left panel of Figure 7.4. The probability that sample $X \sim P$ lies in an interval $[a, b]$ on the x-axis is the area $P(X \in [a, b]) = \int_a^b f_P(x)\,dx$ under the p.d.f. This is the same as the probability that a uniformly chosen point with coordinates (s, t) will lie in the region below the graph of $f_P(x)$ and above the interval $[a, b]$ of the x-axis.

Idea (ii) is exactly the same idea used to approximate π in Example 7.1.1: To sample uniformly from a region C, sample uniformly from a larger region and discard (reject) any sample that does not lie in C. See the right panel of Figure 7.4 for an illustration.

The two ideas are combined in the following way. If we know how to sample from a distribution Q with known p.d.f. $f_Q(x)$ (call this the *proposal distribution*), and we want to sample instead from a distribution P (the *target distribution*) with known p.d.f. $f_P(x)$, we can do this if there exists an M such that $Mf_Q(x) \geq f_P(x)$ for all x. In this case, the region in the plane bounded above by the curve $y = Mf_Q(x)$ contains the region bounded above by the curve $y = f_P(x)$. In the right panel of Figure 7.4, the black curve is the graph of Mf_Q, the blue curve is the graph of f_P. Now draw z from the proposal distribution Q and u from Uniform$(0, Mf_Q(z))$. The point (z, u) corresponds to a uniform draw from the region in the plane below the

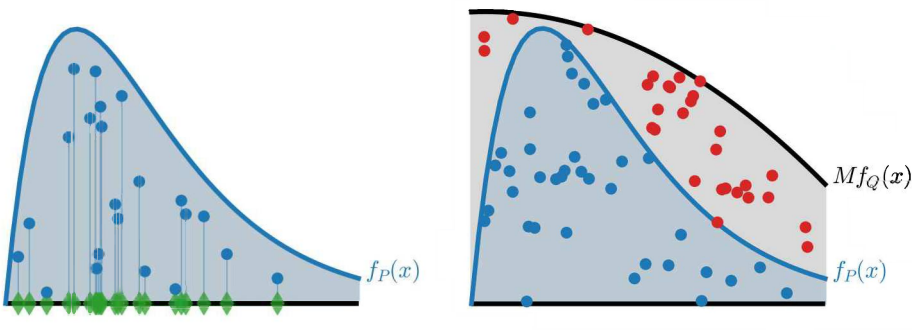


Figure 7.4. *Illustration of the two main ideas behind rejection sampling. The first idea, illustrated in the left panel, is that sampling from a distribution is equivalent to sampling uniformly (blue dots) from the region between the graph of the p.d.f. and the x-axis and then projecting down to the x-axis (green diamonds). The second idea, illustrated in the right panel, is that sampling uniformly from one region (blue) can be accomplished by sampling uniformly from a larger region (gray and blue) and rejecting any samples (red dots) that do not lie inside the smaller region.*

curve $y = Mf_Q(x)$. If $u \leq f_P(z)$, then (z, u) lies inside the region bounded above by $f_P(x)$ and hence the first coordinate z is a draw from X; otherwise reject z and repeat the process.

One minor adjustment is usually made to this process: instead of drawing u from Uniform$(0, Mf_Q(z))$, it is traditional (and sometimes more efficient) to draw $\tilde{u}$ from Uniform$(0, 1)$ and then use the acceptance rule $\tilde{u} \leq \frac{f_P(z)}{Mf_Q(z)}$. Combining all these parts gives the rejection sampling algorithm:

(i) Choose M such that $Mf_Q(x) \geq f_P(x)$ for all x.

(ii) Draw z from Q and $\tilde{u}$ from Uniform$(0, 1)$.

(iii) If $\tilde{u} \leq \frac{f_P(z)}{Mf_Q(z)}$, then accept z as a draw from X; otherwise reject z and go back to (ii).

Remark 7.2.8. A given draw z has a probability $\frac{f_P(z)}{Mf_Q(z)}$ of being accepted, and one can show that the expected number of draws from Q needed to get one acceptable draw from P is proportional to M. Thus, it is generally best to choose a Q for which we can find a small M satisfying $f_P(x) \leq Mf_Q(x)$ for all x, and it is best to take the smallest M that satisfies the condition.

Example 7.2.9. Let P be a *truncated* exponential distribution on $[0, 20]$ with p.d.f. $f_P(x) = \frac{1}{Z}e^{-x}$, where $Z = \int_0^{20} e^{-x}\,dx$. Since $e^{-x} \leq 1$ for all $x \geq 0$, one possible choice of proposal distribution is the uniform distribution Q on $[0, 20]$ with $M = \frac{20}{Z}$, so that $f_P(x) \leq Mf_Q(x) = \frac{1}{Z}$ for all $x \in [0, 20]$. To use the method with this proposal distribution, draw z from Q and $\tilde{u}$

from Uniform($[0,1]$) and reject any z whose corresponding u is greater than $\frac{f_P(z)}{Mf_Q(z)} = e^{-z}$. Implementing this and drawing one million times, we find that roughly 950,000 proposals are rejected and only 50,000 are accepted.

We can improve the efficiency of this rejection sampler by choosing a proposal distribution with a shape that is closer to that of the target. For example, it is easy to check that $e^{-x} \leq (1+x)^{-1}$ for all $x \in [0,\infty)$, and $(1+x)^{-1}$ has a shape much more like that of e^{-x}. Define a new proposal distribution R with

$$f_R(x) = \frac{1}{W}\frac{1}{1+x},$$

where $W = \int_0^{20} \frac{dx}{1+x} = \log(21)$. Setting $M = \frac{W}{Z}$ we have

$$f_P(x) \leq Mf_R(x)$$

for all $x \in [0,20]$. It is easy to sample from R using inversion sampling. We have

$$F_R(x) = \frac{1}{W}\int_0^x \frac{dt}{1+t} = \frac{1}{W}\log(1+x)$$

and

$$F_R^{-1}(v) = e^{Wv} - 1.$$

So the rejection sampling algorithm in this case consists of drawing both u and v from Uniform($[0,1]$), letting $z = e^{Wv} - 1$, and rejecting z if $u > \frac{f_P(z)}{Mf_R(z)} = \frac{1+z}{e^z}$. Implementing this and drawing one million times, we find that roughly $2/3$ of the proposals are rejected and $1/3$ are accepted—a better success rate than with the uniform proposal.

7.3 Hashing

Computers can determine whether two numbers are equal in just one clock cycle. The ability to make such a comparison is built into the hardware. To check whether two strings (or two other, more general objects) are the same, however, is a much more intricate process. For example, to naïvely compare whether "John A. Smith" and "John A. Smyth" are the same, we would successively compare each entry in the first string against its corresponding entry in the second string until either we find a difference or we run out of entries. With this example we could conclude on the 11th iteration that the strings are not the same. Whenever two strings are the same, or differ only in the last character, we must compare every single character to decide whether they are equal. In applications where many strings or lists are likely to be compared, a more efficient way to do this is to use *hashing*, that is, to use a function (called a *hash function*) that assigns an integer to each string. A computer can compare any two hash values in a single clock cycle, so this makes it easy to compare different strings (assuming the hash function is easy to compute).

A related problem is that of searching for a given object in an unordered list (or array of pointers) of length n. This has average temporal complexity $O(n)$, because

the expected number of entries to examine[35] is $\frac{1}{n}\sum_{i=0}^{n-1}(i+1) = \frac{n+1}{2}$. Of course, if the data are sorted and stored in an ordered list or in a BST, then search time drops to $O(\log(n))$. But there are ways to make this process much faster, based on the fact that accessing the ith entry in an array is a very fast, constant-time operation.

Again, the idea is to hash the data, that is, to use a hash function to assign an integer to each object, and then use the integer as the index of the object in an array. Thus, each object x is placed into entry number $h(x)$ of the array, where h is the hash function. This construction is called a *hash table*. In this section we discuss hashing, hash tables, and related algorithms and applications.

7.3.1 Dictionaries and Sets

Hashing is used to implement some of the most useful and important abstract data types in computer science, including *dictionaries* and *sets*. A *dictionary* (also called an *associative array* or a *map*) consists of a collection of objects (often called *values*), each indexed by a unique key, with the following operations:

(i) Search for a given key, and return the corresponding value if the key exists.

(ii) Add a value to the dictionary, with its key.

(iii) Delete a value and its key.

For now we assume that keys are unique, and so, for a given key, adding another value with that same key overwrites the old value.

Dictionaries are useful any time you need to define a rule, or a mapping, from one set (the keys) to another (the values). For example, they can be used for storing computer usernames and passwords, where each username is a key and the user's password is the associated value. They could be used to store prices of entrées in a restaurant menu, where the name of each entrée is a key and the price of that entrée is the associated value. They could be used to store hyphenation rules for English words, where the keys are the English words and the associated values are the hyphenation rules. They are also used in compilers and interpreters for matching variable names (keys) to their values and for countless other applications.

Example 7.3.1. In Python a dictionary can be constructed with a sequence of pairs of the form `key:value`, as follows:

```
menu = { "ham":8, "spam":7, "lobster":42}
```

The dictionary name is `menu`; the keys are `"ham"`, `"spam"`, and `"lobster"`; and the corresponding values are the integers 8, 7, and 42, respectively. To search for the `"spam"` key, use `menu["spam"]`, which returns 7. Searching for `"beans"` raises a `KeyError`, since that key is not in the dictionary.

[35] This is assuming the location of the object is uniformly distributed over all the positions, and assuming the number of entries we must examine is $i+1$, when the target is in position i.

A closely related data structure is that of a *set*, which is essentially the same idea as a mathematical set. It is a collection of elements without any order, and with the following fundamental operations:

(i) Identify whether an object is an element of the set.

(ii) Add a new element to the set.

(iii) Remove an element from the set.

Example 7.3.2. In Python a set can be constructed as a sequence of elements inside of braces, as follows:

```
S = { "x", 42, 3.14 }
```

The set name is `S` and the elements are `"x"`, 42, and 3.14. To identify whether the element 42 is in the set, use the command `42 in S`, which returns `True`, whereas `"spam" in S` returns `False`.

7.3.2 Hash Tables

If the keys have an order on them, then a dictionary or a set can be implemented with a BST. But searching a BST takes $\log(n)$ time if there are n entries in the tree. We can do much better with a *hash table*. A hash table consists of two things:

(i) An integer-valued hash function h whose domain includes all keys.

(ii) An array (or a list) of size at least as big as the largest possible value of the hash function. The value associated with the key k is placed in position $h(k)$ (or a pointer to the value is placed in that location).

Example 7.3.3. The `menu` dictionary of Example 7.3.1 can be implemented as a hash table by first defining a hash function on the collection of possible entrée names. One simple example of a hash function is the function that assigns each letter its position in the alphabet and then adds up the values of all the letters in the word and reduces modulo 48 (the number 48 is an arbitrary choice), thus

$$\begin{aligned} h(\texttt{"abc"}) &= 0 + 1 + 2 = 3, \\ h(\texttt{"ham"}) &= 7 + 0 + 12 = 19, \\ h(\texttt{"lobster"}) &= 11 + 14 + 1 + 18 + 19 + 4 + 17 = 84 \equiv 36 \pmod{48}. \end{aligned}$$

There are problems with this particular choice of hash function (we discuss some of these below). But despite its problems, we can use this hash function to define a hash table for the menu dictionary. To do this, begin by constructing a list `menu` of length 48 with `None` in each entry. For each of the foods, put

the associated price in the position given by the hash:

$$\begin{aligned} h(\texttt{"ham"}) &= 19 \quad \text{so} \quad \texttt{menu}[19] = 8, \\ h(\texttt{"spam"}) &= 45 \quad \text{so} \quad \texttt{menu}[45] = 7, \\ h(\texttt{"lobster"}) &= 36 \quad \text{so} \quad \texttt{menu}[36] = 42. \end{aligned}$$

All remaining entries of `menu` are left at `None`. To find the price of key x, calculate its hash value $h(x)$ and retrieve the value of `menu[h(x)]`.

If the hash function can be computed in constant time for all the possible keys, and if all keys have unique hash values (the hash function is injective), then finding a value in the hash table can also be done in constant time, since accessing a known position in an array is a constant-time operation. Moreover, inserting a new value into the table involves computing the new key's hash, and then putting the object into the table at the hashed position, so this is also a constant-time operation, as is removing a value from the table. No matter how big the table is, if the hash function is injective and can be computed in constant time, then we can perform any of the three hash table operations (search, insert, or remove) in constant time. Unfortunately, however, most hash functions are not injective.

7.3.3 Hash Collisions

If the hash function is not injective, then differing keys could produce the same hash value and thus also be assigned to the same position in the hash table. This is called a *hash collision*.

Unexample 7.3.4. For the simple hash table `menu` of Example 7.3.3, if we try to insert the pair `"beans":5` into the table, we get a hash collision because, $h(\texttt{"beans"}) = 36 = h(\texttt{"lobster"})$. Therefore, this particular implementation of the menu dictionary cannot accommodate entries for both `"lobster"` and `"beans"` simultaneously.

An injective hash function will have no hash collisions, but to use such a function to construct a hash table in the obvious way with no hash collisions, the table must have at least as many entries as the collection (also called *universe*) of all possible keys.

Example 7.3.5. If we know that all keys will consist of three-letter words, we can construct an injective hash function h from the universe of three-letter strings by sending the first letter to its position in the alphabet, the second letter to 26 times its position, and the third to 26^2 times its position, and then

summing these three. This hash function gives

$$h(\texttt{"abc"}) = 0 + 1 \times 26 + 2 \times 26^2 = 1378,$$
$$h(\texttt{"ham"}) = 7 + 0 \times 26 + 12 \times 26^2 = 8119,$$
$$h(\texttt{"egg"}) = 4 + 6 \times 26 + 6 \times 26^2 = 4216.$$

To build a collision-free hash table with this hash function, we must allocate an array with 17,576 entries (as many entries as the number of possible three-letter words) and assign each key k to the hashed position. Such a table is sometimes called a *direct address table*.

Of course this table is guaranteed to be collision free, but it is spatially expensive if the total number of keys to be used is much smaller than the universe of all possible keys (of size 17,576).

Remark 7.3.6. Some people call an injective hash function a *perfect hash function*.

Usually we do not know in advance which keys will be used or even how many keys will be used. Consequently, the universe of possible keys could be very large, and it is not practical to make a hash table that is as large as that universe. For example, the data might consist of strings of arbitrary length. In this case the universe of possible objects to hash is infinite, but we don't want to make an infinite hash table. When the hash table is smaller than the universe of possible keys, an injective hash function is impossible.

The strategy for dealing with this is to go ahead and use a hash function that is not injective, but then expand the algorithm to handle a hash collision. The goal is to choose a hash function that minimizes the probability of a hash collision, and then handle the collisions as efficiently as possible.

Probability of a Hash Collision

When a hash function is not injective, hash collisions can occur. Assuming that the hash values are uniformly distributed among the possible indices in the hash table (we call such a hash function *simply uniform*), the probability of a hash collision can be computed in the same way we computed the probability of a birthday match in the birthday problem (Example 5.1.17). If the number of possible outputs of the hash function (and indices of the hash table) is n and the total number of keys is k, then following the argument of Example 5.1.17 shows that the probability $P(k)$ of a hash collision is

$$P(k) = 1 - \frac{n!}{(n-k)!n^k},$$

provided $0 < k \leq n$. If $k > n$, then $P(k) = 1$, by the pigeonhole principle.

The discussion of the birthday problem shows that if a hash table has $n = 365$ possible entries, then the probability of a collision is 50% or more if $k \geq 23$. So even with a uniform distribution of hash values, collisions are likely with relatively few keys. Figure 7.5 shows the probability of a hash collision for k keys in a table of size 10^5.

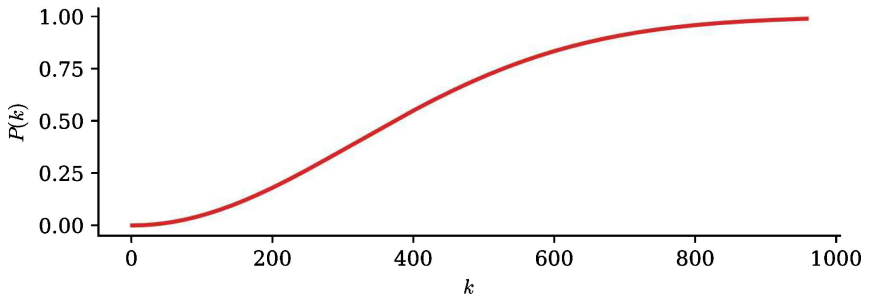


Figure 7.5. *Plot of the probability $P(k)$ of a hash collision for k keys with a simply uniform hash function whose values are uniformly distributed among $n = 10^5$ possible values. Note that the probability is greater than 50% when $k \geq 372$, and it is greater than 99% for $k \geq 1000$.*

With a 32-bit hash function there are over four billion possible hash values ($2^{32} - 1$), yet the probability of a hash collision is more than 50% if the number of keys is at least 77,164 (see Exercise 7.15). Again, relatively speaking, it does not take a very large list of keys to cause a collision. Nevertheless, hashing is usually much more efficient than search trees or any other table lookup method. As a result, hashing is used widely throughout computer science.

Handling Hash Collisions

One common way to handle hash collisions is the method of *chaining*. In this method, each position in the array corresponds to a linked list, and when a new key is hashed to a given position, the key and corresponding value are placed at the head of the corresponding linked list.

Example 7.3.7. For a chained version of the `menu` hash table of Example 7.3.3 and Unexample 7.3.4, if the table is to include both `"beans"` and `"lobster"`, then at position $36 = h(\texttt{"beans"}) = h(\texttt{"lobster"})$, we insert a linked list whose tail contains the data `"lobster":42` and whose head contains the data `"beans":5`. Searching for `"lobster"` now involves first hashing to get position 36 and then searching the linked list to find the key `"lobster"`.

In the worst case, chaining can perform very poorly. If all the keys are hashed to the same value, then every value is contained in one linked list. Both the average and the worst-case complexity of searching a linked list with k keys is $O(k)$. If, however, the hash function is simply uniform, taking n possible values, then the expected number of keys in any of the linked lists is k/n, so the average complexity of searching such a chained hash table is $O(1 + k/n)$. The number k/n is often called the *load factor* of the hash table.

If the table has a fixed size n, then as k grows, we have $O(1 + k/n) = O(k)$; so the average asymptotic complexity of searching the table is the same as searching a linked list, but with a much better leading coefficient (by a factor of $1/n$). It is

common practice to resize the hash table if the load factor grows too large. Often the bound on the load factor is taken to be less than 1. Once the load factor exceeds the bound, a new array is allocated with a larger number n' of positions, a new hash function is chosen which produces n' hash values, and all the entries in the old table are hashed with the new hash function and placed in the new table. The construction of the new hash table has a cost of $O(k)$, but it need only occur when k/n exceeds the desired bound.

Alternatively, if rehashing the entire table to construct a new, larger table is too expensive, one may choose simply to build a second hash table, make all new additions in the new table, and then search both tables for each lookup. This doubles the time it takes to perform each search (at least until ongoing deletions have removed all the elements from the original table) but does not require a complete rehash of the original table.

One other method for handling hash collisions is called *open addressing*. In this method, when a hash collision occurs, the algorithm searches for another open address in the table to put the key into. For example, one could search for the next available slot after the one addressed by the hash (modulo n). A drawback of open addressing is that you can never have $k > n$. As in the case of chained hash tables, if all the keys hash to the same value, then searching an open-address hash table is no better than searching a list.

Example 7.3.8. When using open addressing for the `menu` hash table of Example 7.3.3 and Unexample 7.3.4, we add the key `"beans"` to position 37, since there is a collision with lobster at position $36 = h(\texttt{"beans"}) = h(\texttt{"lobster"})$. Searching for `"beans"` now involves first hashing to get position 36 and then moving consecutively through the array from that point until finding the desired key. Note that `"cheddar"` also hashes to 36, so if we wish to add `"cheddar"` to the menu, it cannot go into position 36 nor into position 37, so we put `"cheddar"` into position 38. The word `"paté"` hashes to 38 (treating `é` the same as `e`), but 38 has already been taken, so we put `"paté"` in position 39.

7.4 *Simulated Annealing

A fundamental problem in mathematics and applications is to find the optimum (minimum or maximum) of a function and the optimizer (the point which yields the optimum). For concreteness, we consider the optimization problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \mathbf{x} \in \Omega \end{aligned} \tag{7.3}$$

for some function f and some set Ω. This is equivalent to maximizing $-f(\mathbf{x})$ subject to $\mathbf{x} \in \Omega$, so we lose nothing by focusing solely on minimization. Part IV of this text discusses many methods for finding local minima when the *objective function* f is differentiable. Chapter 4 discusses some deterministic methods for solving combinatorial optimization problems, where the search space Ω is discrete, where differentiability doesn't even make sense.

In this section and the next we discuss some probabilistic methods for optimizing functions without differentiation. One obvious way to try to find an optimizer on a discrete space is the brute force exhaustive method; that is, try every possible input and see which one gives the best solution. The exhaustive method has the advantage of guaranteeing the correct solution, and it does not use differentiation, so it makes no assumptions about differentiability or smoothness, but it is usually too computationally expensive to be useful.

One way to try to find an optimizer without an assumption of differentiability and without trying every point in Ω is a Monte Carlo method—sample Ω randomly and return the sample that produced the smallest value of the objective function. This is potentially much cheaper than a brute force method, and if the sample is dense enough in Ω, it should give a good approximation to the global optimizer. But random sampling does not take advantage of any information gained by previous samples. For example, it might be better to sample near the current best estimate, rather than choosing each point completely at random.

Simulated annealing and *genetic algorithms* are two methods that use some randomness in their sampling but are sometimes more effective than purely random sampling because they try to leverage information gained by previous samples in choosing new samples. We discuss simulated annealing in this section and genetic algorithms in the next section (Section 7.5).

Neither of these methods assumes differentiability, but they work best if the objective function is approximately continuous, in the sense that small changes in the input do not produce large changes in the output. If the output of the objective function changes wildly with even the smallest change in the inputs, then simulated annealing and genetic algorithms have no significant advantage over completely random sampling. Finally, we note that, like random sampling, simulated annealing and genetic algorithms provide no guarantee of finding the optimal solution, but they often find a solution that is not far from optimal, which, in many cases, is good enough.

7.4.1 Stochastic Hill Sliding

Before discussing simulated annealing and genetic algorithms, we begin with a more naïve sampling method that is sometimes called *stochastic hill climbing* when the objective function is being maximized, but since we are minimizing the objective, we call it *stochastic hill sliding*. This is an iterative method for minimization:

(i) Set $k = 0$. Choose an initial $\mathbf{x}_0 \in \Omega$.

(ii) Draw a candidate $\mathbf{z}_k$ from a neighborhood $N(\mathbf{x}_k)$ of $\mathbf{x}_k$.

(iii) If $f(\mathbf{z}_k) < f(\mathbf{x}_k)$, then set $\mathbf{x}_{k+1} = \mathbf{z}_k$; otherwise set $\mathbf{x}_{k+1} = \mathbf{x}_k$.

(iv) Set $k = k + 1$ and go to (ii) (unless some stopping criterion is met).

The choice of the neighborhood $N(\mathbf{x})$ and the distribution used to draw from that neighborhood have a large effect on the final result of the algorithm. If $N(\mathbf{x})$ is always a small neighborhood of $\mathbf{x}$, the method will tend toward a local minimum. However, if the neighborhood $N(\mathbf{x})$ is large and the sampling method tends to choose points far away from $\mathbf{x}$, then the algorithm is unlikely to be trapped near a

local minimizer, but it may also have to sample many more points to find a value that descends ($f(\mathbf{z}_k) < f(\mathbf{x}_k)$). If every neighborhood is the entire feasible set, and if the samples are drawn from the uniform distribution, then the stochastic hill-sliding algorithm is just random sampling.

Example 7.4.1. Let $f(x) = x^4 - x^2 + \frac{x}{10}$; this is plotted in Figure 7.6. This function has two local minimizers: one near $x = 0.681$ and the other (the global minimizer) near $x = -0.731$. We implemented the stochastic hill-sliding algorithm for minimizing this function, with $x_0 = 0$, and drawing each z_k from $\mathcal{N}(x_k, \frac{1}{4})$. The result was as follows:

$x_0 = 0$	$f(x_0) = 0$	$z_0 = -0.041$	$f(z_0) = -0.006$	accept z_0
$x_1 = z_0$	$f(x_1) = -0.006$	$z_1 = 0.133$	$f(z_1) = -0.004$	reject z_1
$x_2 = x_1$	$f(x_2) = -0.006$	$z_2 = 0.331$	$f(z_2) = -0.0644$	accept z_2
⋮	⋮	⋮	⋮	⋮
$x_8 = 0.6886$	$f(x_8) = -0.180$	$z_8 = 0.395$	$f(z_8) = -0.0922$	reject z_8
⋮	⋮	⋮	⋮	⋮
$x_{100} = 0.680$	$f(x_{100}) = -0.181$			

The algorithm arrived at an answer fairly near the positive local minimizer in 8 steps. But it still had not reached the global minimizer after 100 steps.

We repeated the process with a larger neighborhood, by choosing z_k from $\mathcal{N}(x_k, 1)$, again starting at $x_0 = 0$. In the first move it went to $x_1 = 0.759$ and remained there for 10 steps, after which it moved to -0.828 for 19 steps. Then it moved to -0.706, where it remained until step 100.

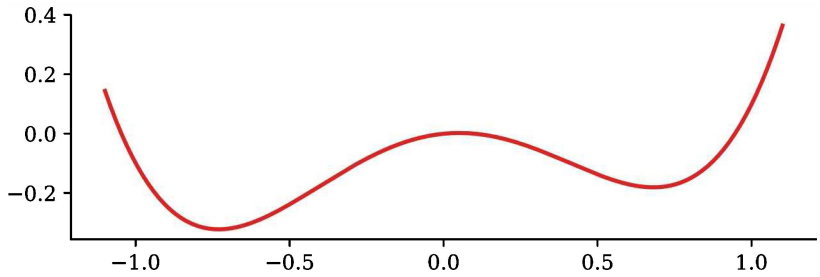
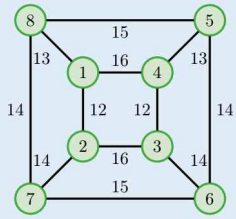
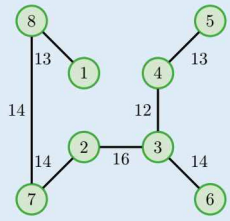


Figure 7.6. *A plot of the function minimized in Examples 7.4.1 and 7.4.4. Starting stochastic hill sliding at $x_0 = 0$ and drawing each new candidate with a low variance ($\sigma^2 = \frac{1}{4}$) in Example 7.4.1 leads us to the positive local minimizer (on the right), but we never approach the global minimizer (on the left). Starting at the same place but drawing with a higher variance ($\sigma^2 = 1$) allowed us to escape from the local minimizer on the right and get near the global minimizer on the left. But, because of the high variance, our experiment never got better than -0.706, while the global minimizer is actually at -0.731. Simulated annealing in Example 7.4.4 did better: starting at $x_0 = 0$ it got in the ballpark of the global minimum ($x_{13} = -0.680$) in 13 steps and was very close ($x_{100} = -0.730$) after 100 steps.*

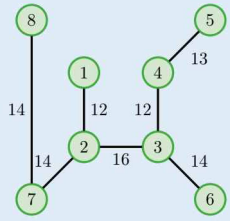
Example 7.4.2. Consider the problem of finding the MST for the graph G in Figure 4.5 and Example 4.3.1, which we include here again for ease of reference:



Of course, Prim's algorithm (see Section 4.3.1) gives a deterministic method for finding an MST, but we can also apply the stochastic hill-sliding algorithm. Let Ω be the set of all spanning trees in G. We let a neighborhood of a given spanning tree T be the set of all spanning trees of G that can be built from T by deleting one edge and inserting one edge. So, for example, if T_0 is the spanning tree



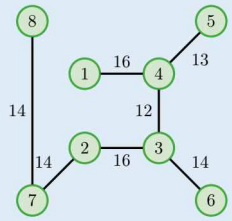
then deleting the edge $(1, 8)$ and adding the edge $(1, 2)$ shows that the resulting spanning tree T_1



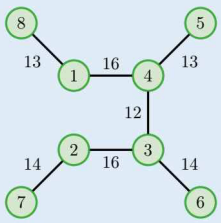
is in the neighborhood of T_0. The objective function is the length of the spanning tree. The hill-sliding algorithm is now straightforward: Given any spanning tree T_k randomly select a tree in the neighborhood of T_k by selecting a node, an edge from that node to remove, and an edge from that node to add. If the result is not a tree, reject it and try again. If the result is a tree, compute its length. If the length is not less than the length of T_k, reject and try again.

Starting with T_0, which has length 96, assume the new tree T_1 (above) is randomly selected as a proposal. Since an edge of length 13 has been replaced by an edge of length 12, it has length 95, and it is accepted as the new spanning

tree. At the next step, suppose the edge (1,2) is chosen to be replaced by the edge (1,4), giving



The result is a spanning tree, but it has length 99, which is longer, so it is rejected. Repeating the process, now suppose edge (7,8) is chosen to be replaced by edge (1,8), giving the spanning tree



which has length 94, so the proposed tree is accepted. The process is repeated until some maximum number of iterations has been reached or until it appears that the proposed solution is no longer improving.

In this example there is no guarantee that every possible spanning tree can be reached by this process of swapping edges. So it is possible that an MST could not be reached if the initial starting tree was poorly (or unluckily) chosen.

7.4.2 Simulated Annealing

Simulated annealing differs from stochastic hill sliding by sometimes allowing $\mathbf{z}_{k+1}$ to be accepted even if $f(\mathbf{z}_{k+1}) > f(\mathbf{x}_k)$. In Example 7.4.1 using larger neighborhoods for stochastic hill sliding allowed the algorithm to probe farther away, but most of those probes were unsuccessful and the algorithm did not converge well. On the other hand, using small neighborhoods meant that the algorithm could not probe far enough away to break away from the local minimizer, but many of the probes did improve the result, so it converged to a good approximation of the local minimizer. The idea with simulated annealing is to begin by sampling from Ω fairly broadly, by allowing more uphill moves, but as time goes on, focus near the current estimate and only accept a proposal if the result is actually an improvement.

Specifically, choose a monotone decreasing sequence $(t_k)_{k\in\mathbb{N}}$ that converges to zero (corresponding to temperature in the annealing process). This is called the

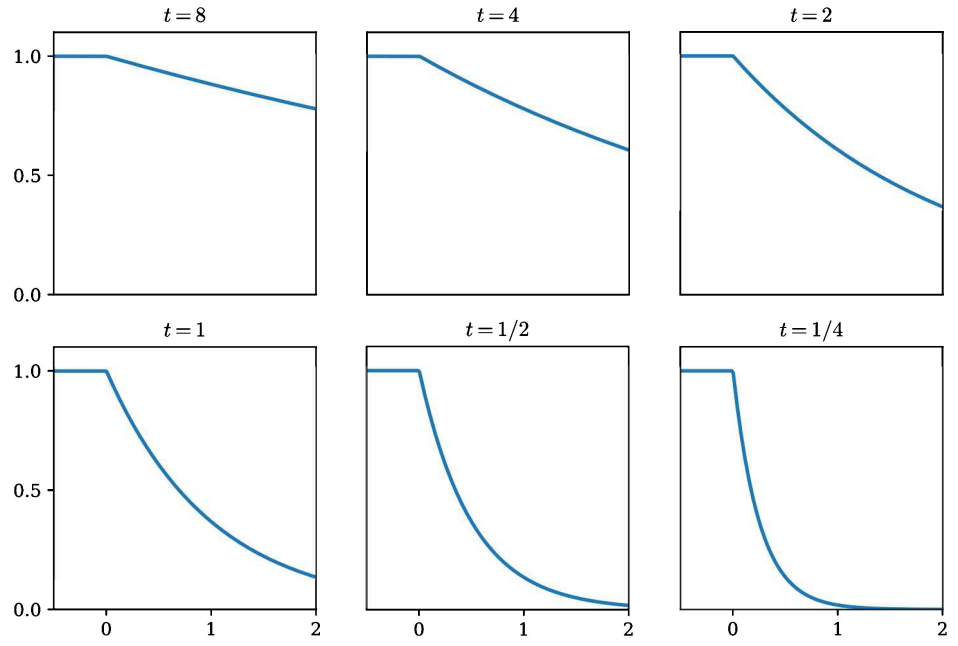


Figure 7.7. *Simulated annealing might accept a $\mathbf{z}_k$ even when $f(\mathbf{z}_k) > f(\mathbf{x}_k)$. This figure shows a plot of the acceptance probabilities (7.4), where the x-axis in the plot corresponds to the difference $f(\mathbf{z}_k) - f(\mathbf{x}_k)$. As t_k gets smaller, the values of $\mathbf{z}_k$ with $f(\mathbf{z}_k) > f(\mathbf{x}_k)$ are less and less likely to be chosen.*

cooling schedule or *annealing schedule*. For each $k \in \mathbb{N}$ let

$$p_k(f(\mathbf{x}_k), f(\mathbf{z}_k)) = \min\left\{1, \exp\left(-\frac{f(\mathbf{z}_k) - f(\mathbf{x}_k)}{t_k}\right)\right\}. \tag{7.4}$$

Now replace step (iii), above, with the following rule:

(iii) With probability $p_k(f(\mathbf{x}_k), f(\mathbf{z}_k))$, set $\mathbf{x}_{k+1} = \mathbf{z}_k$. Otherwise, set $\mathbf{x}_{k+1} = \mathbf{x}_k$.

Note that $p_k = 1$ whenever $f(\mathbf{z}_k) < f(\mathbf{x}_k)$, and thus $\mathbf{z}_k$ is always accepted in that case. When $f(\mathbf{z}_k) > f(\mathbf{x}_k)$, there is still some probability that $\mathbf{z}_k$ will be accepted, but that probability goes down as $f(\mathbf{z}_k) - f(\mathbf{x}_k)$ increases and as t_k decreases; see Figure 7.7 for a plot of some of these probabilities.

As the temperature t_k becomes very small, the sequence $(\mathbf{x}_k)_{k=1}^{\infty}$ will usually converge to a local minimizer. But this is not guaranteed to be a global minimizer, and it might not even be the best value seen by the algorithm. Thus, it is usually a good idea to remember the point with the best value seen so far. As the temperature decreases, it sometimes also makes sense to occasionally restart the algorithm with the best value seen so far.

Remark 7.4.3. One easy way to implement the rule of accepting $\mathbf{z}_k$ with probability $p_k(f(\mathbf{x}_k), f(\mathbf{z}_k))$ is the following: draw u from the uniform distribution on $[0, 1]$ and then accept $\mathbf{z}_k$ if $u < p_k(f(\mathbf{x}_k), f(\mathbf{z}_k))$.

Example 7.4.4. Let $f(x) = x^4 - x^2 + \frac{x}{10}$, as in Example 7.4.1, and plotted in Figure 7.6. We implemented the simulated annealing algorithm for minimizing this function, with $t_k = 10/(k+1)^{2.5}$, starting at $x_0 = 0$, and drawing each z_k from $\mathcal{N}(x_k, 0.5)$. The result was as follows:

$x_0 = 0.000$	$f(x_0) = -0.003$	$z_0 = 0.122$	$f(z_0) = -0.003$	$p = 1.00$	accept
$x_1 = 0.122$	$f(x_1) = -0.121$	$z_1 = 0.460$	$f(z_1) = -0.121$	$p = 1.00$	accept
$x_2 = 0.460$	$f(x_2) = -0.108$	$z_2 = 0.430$	$f(z_2) = -0.108$	$p = 0.98$	accept
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_{13} = -0.680$	$f(x_{13}) = -0.317$	$z_{13} = -0.535$	$f(z_{13}) = -0.258$	$p = 0.03$	reject
$x_{14} = -0.680$	$f(x_{14}) = -0.320$	$z_{14} = -0.757$	$f(z_{14}) = -0.320$	$p = 1.00$	accept
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_{100} = -0.730$	$f(x_{100}) = -0.322$				

The algorithm was in the ballpark of the global minimizer by step 13, and by step 100 it was close to the global minimizer of -0.731.

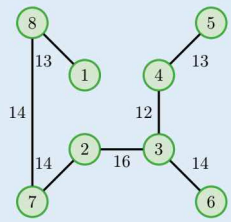
The results of simulated annealing depend a lot on the choice of the cooling schedule $(t_k)_{k\in\mathbb{N}}$. A slower cooling schedule, such as $t_k = 10/(k+1)$, will leave the algorithm bouncing around quite a bit, so it behaves more like a random sampler. A fast cooling schedule, such as $t_k = 10/(k+1)^4$, will tend to converge rapidly to the local minimizer at 0.681 and never see the global minimizer.

Remark 7.4.5. The sequence p_k need not be defined by (7.4)—other choices of p_k can be useful in some settings. The key requirement is that the probability of accepting $\mathbf{z}_k$ should decrease both as k increases and as $f(\mathbf{z}_k) - f(\mathbf{x}_k)$ increases.

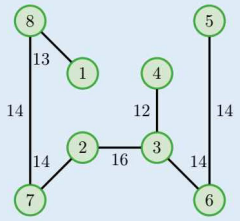
Example 7.4.6. Simulated annealing can also be used on the problem of finding the MST for the graph G of Example 7.4.2. As in that example, let Ω be the set of all spanning trees in G; let the neighborhood of a given spanning tree T be the set of all spanning trees of G that can be built from T by deleting one edge and inserting one edge; and let the objective function be the length $\ell(T)$ of the spanning tree T.

As in the hill-sliding algorithm, given any spanning tree T_k, the algorithm randomly selects a new spanning tree z_k in the neighborhood of T_k, computes its length $\ell(z_k)$, and accepts z_k as the next tree T_{k+1} with probability $p_k(z_k, T_k)$. Otherwise it sets $T_{k+1} = T_k$.

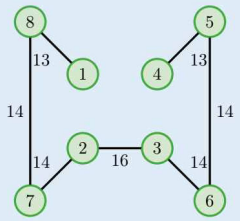
As an explicit example, assume the cooling schedule is $t_k = \frac{1}{k+1}$ and start again with T_0,



which has length $\ell = 96$. Assume the new tree z_0



is randomly selected as a proposal. Since an edge of length 13 has been replaced by an edge of length 14, it has length 97, so $p_0(z_0, T_0) = \exp(-(97 - 96)) = e^{-1} \approx 0.368$. Thus the algorithm has a 37% chance of accepting z_0 as the next spanning tree, despite the fact that its length is greater than T_0. Assuming that z_0 is accepted, we have $T_1 = z_0$. Assume that the spanning tree z_1



is proposed for the next step. The length of z_1 is 98, but it could still be accepted with probability $p_1(z_1, T_1) = \exp(-2(98 - 97)) = e^{-2} \approx 0.135$. This process is repeated until some maximum number of iterations has been reached or until it appears that the algorithm has converged.

As in the case of stochastic hill sliding, there is no guarantee that every possible spanning tree can be reached by this process.

Remark 7.4.7. If it is known that the objective has only one local minimum, which is also the global minimum, then stochastic hill sliding is probably a more efficient search method than simulated annealing. But if there are likely to be many local minima that are not global minima, then stochastic hill sliding can easily get stuck near a local minimum, while simulated annealing has a better chance of avoiding that trap.

7.5 *Genetic Algorithms

Genetic algorithms are the third best way to do almost anything.
—John Denker

Genetic algorithms provide another stochastic method for optimization on some discrete domains. Rather than choosing sample points from a neighborhood of

one previous sample point, they use an idea inspired by genetics to construct new samples from a collection (the current *population*) of several previous points. They are applied to domains where every point can be described as a finite sequence of symbols, analogous to DNA in genetics.

At each iteration (*generation*) some members of the population are paired as parents to produce children that should have characteristics of each of the parents. These children are added to the population, and some less fit individuals are selected out of the population.

There are three main ingredients in a genetic algorithm:

(i) Crossover: children that are a mixture of their parents.

(ii) Mutation: random variation in the genetic information.

(iii) Selection: elimination of inferior species.

Genetic algorithms that incorporate all three of these tend to be more successful than those that omit one or two of them.

7.5.1 Crossover

Given two points (parents) from the current population, crossover produces two new points (children) that are a mixture of their parents. If each parent is represented by a string of length n, a random point k is chosen and two children are created by taking the first k terms of one parent and the remaining $n - k$ terms of the other.

Example 7.5.1. If two parents are described as the 8-bit sequences

$$01101110 \quad \text{and} \quad 10111101$$

and $k = 4$, then the two children resulting from crossover are a mix of the parents, taking the first four digits from one parent and swapping them with the last four from the other parent.

$$01101101 \quad \text{and} \quad 10111110.$$

It is not essential that two parents produce exactly two children—they may produce any number through mixing their sequences in some prescribed manner.

7.5.2 Mutation

With *mutation*, terms in the sequence can occasionally be changed at random to allow for greater diversity in the population and thus allow the algorithm to sample the domain more broadly.

Example 7.5.2. Assume the domain consists of strings of five binary digits and the current population is

$$01010, \quad 11101 \quad 11111, \quad 10010.$$

If the second bit of the third child is chosen to be mutated, then the new generation would be

$$01010, \quad 11101 \quad 10111, \quad 10010.$$

7.5.3 Selection

To prevent the population from growing too large, one could simply replace all parents by their children after each generation, but it is usually helpful to remove less fit individuals from the population. This is called *selection*.

With each generation, each individual in the population is evaluated in terms of the objective function and those that have a better objective function value are considered more *fit*. A successful genetic algorithm normally includes some sort of selection method to remove less fit candidates from the gene pool and to increase the likelihood that a fit candidate will mate with another fit candidate.

Example 7.5.3. Suppose that the domain of an optimization problem is $\Omega = \{0, 1, 2, \ldots, 31\}$ and we want to maximize the function $f(x) = (x - 15)^2$.

Representing each point as a sequence of five binary numbers provides a binary encoding.[a]

Suppose that the initial population is $13, 27, 30, 17$, encoded as

$$01101, \quad 11011 \quad 11110, \quad 10001.$$

The fitness of these individuals is given by the objective function f:

$$(-2)^2 = 4, \quad 12^2 = 144, \quad 15^2 = 225, \quad 2^2 = 4.$$

Suppose we choose to pair the most fit individual 11110 with the second most fit 11011 and also with 10001, and we randomly choose crossover points for each pair. Suppose the first crossover point is 3 and the second is 2. This generates the following children:

$$11111, \quad 11010 \quad 11001, \quad 10010.$$

Mutation now randomly chooses a few bits to change. Suppose the second bit of the first child is chosen and the first bit of the fourth child is chosen, to give

$$10111, \quad 11010 \quad 11001, \quad 00010.$$

So, starting with $13, 27, 30, 17$, we applied selection, then crossover and finally mutation to get the children $23, 26, 25, 2$. To avoid doubling the population at each step we can either replace all the parents by their children (remembering also that the point 11110 is the most fit individual seen so far), or we can select the four most fit individuals to keep, discarding the others.

[a]Both this problem and the binary encoding are ill suited for a genetic algorithm. Nevertheless, this simple example gives a good illustration of some of the main ideas of genetic algorithms.

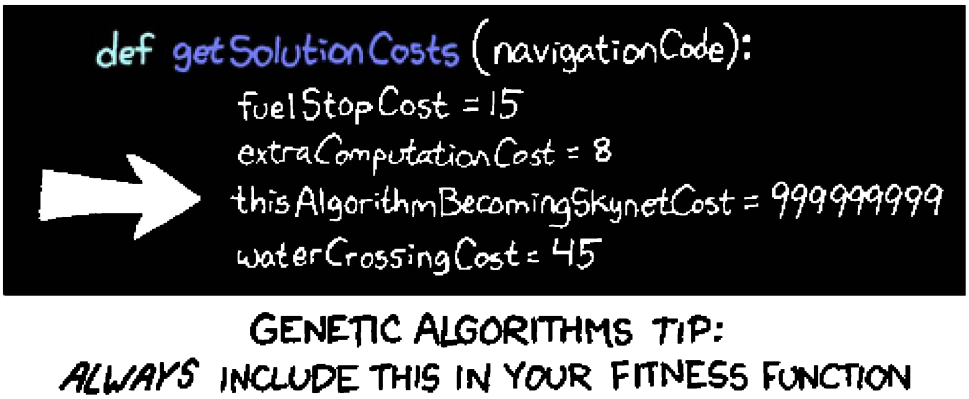


Figure 7.8. *In a genetic algorithm individuals are selected based on their fitness. Source: XKCD, Randall Munroe.* `http://xkcd.com/534`

7.5.4 Encoding

Typically the domain of an optimization problem is not given in the form of a set of sequences. So we need some sort of *genetic encoding* that represents each possible optimal point as a sequence and each sequence as a legitimate point in the domain Ω.

For example, if the domain has (or can be endowed with) a natural binary tree structure, then a suitable encoding might be binary, where each binary address represents a terminal branch of a tree. Or if the domain corresponds to corners of a hypercube, then a possible encoding might represent each corner as a binary address.

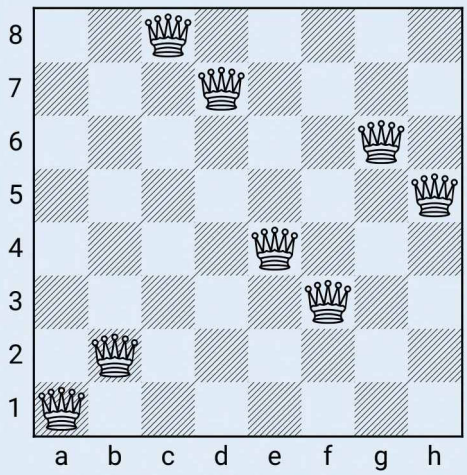
It is also important that the encoding reflect meaningful aspects of the problem. The types of variation in the encoding that occur with each new generation should also correspond to children that have some characteristics in common with the parents. In other words, genetic changes seen in just one or two generations should not produce large changes in the objective function.

It is not essential that every point in the domain be represented by the encoding, provided the points that are left out are guaranteed not to be optimal.

Example 7.5.4. Suppose you are asked to put all the integers from 1 up to n into two sets, call them A and B, in such a way that 10 times the sum of the numbers in A is as close as possible to the product of the numbers in B. To encode the sets as sequences for use with a genetic algorithm, we can write each choice of sets as a binary sequence of length n, where the kth term of the sequence is 1 if the number k is included in A and 0 otherwise. Thus $[1,1,1,0,1,0,0,1]$ corresponds to the two sets $A = \{1,2,3,5,8\}$ and $B = \{4,6,7\}$.

Example 7.5.5. The *eight queens problem* is to place eight queens on a standard chessboard in such a way that no queen threatens another. To try to solve this using a genetic algorithm, we must choose an objective function and find an encoding of all the possible configurations. A natural choice of objective function is the number of pairs of queens that threaten each other.

One way to encode the configurations as sequences is to first restrict the domain by assuming that no two queens will be placed in the same column (otherwise they would be threatening each other) and then observe that every column must contain exactly one queen (because there are eight of them placed in eight columns). Now encode a configuration by listing the position of the queen in each column. So, for example, the configuration



would be encoded as $[1,2,8,7,4,3,6,5]$. To compute the objective function f we must identify threatening pairs. In a configuration coded as a list L, queens in a pair (corresponding to two positions i and j in L) threaten each other if they lie in the same row ($L[i] = L[j]$) or if they are on the same diagonal

$(L[i] - L[j] = \pm(i-j))$. So

$$f(L) = \sum_{i=1}^{7} \sum_{j=i+1}^{8} D_{ij}(L),$$

where

$$D_{ij}(L) = \begin{cases} 1 & \text{if } L[i] = L[j], \\ 1 & \text{if } L[i] - L[j] = \pm(i-j), \\ 0 & \text{otherwise.} \end{cases}$$

If the domain consists of integers or floating-point numbers and the objective function depends on the input values, then the usual binary (or decimal) representation of these numbers is usually a poor choice of encoding, and a genetic algorithm is unlikely to successfully combine two good points to produce another good point. This is because changing even a single bit of the representation can result in a huge change in the value of the represented number and hence of the objective function.

Unexample 7.5.6. As in Example 7.5.3, suppose that the domain is

$$\Omega = \{0, 1, 2, \ldots, 31\}$$

and we want to maximize the function $f(x) = (x - 15)^2$. Representing each point as a sequence of five binary numbers provides an encoding, but this encoding, and indeed the entire problem, is poorly suited to genetic algorithms, because the types of changes that occur are not likely to produce offspring that are at all like the parents.

For example, suppose that we have an initial population of two individuals

$$00000, \quad 11110.$$

The objective function at these points takes the values $(-15)^2$ and 15^2, so both are very fit. To create the next generation, choose a crossover point—say 2—for the pairing, and combine the two parents at that point. This gives a new population

$$00110, \quad 11000$$

with objective function values $(-9)^2$ and 9^2. Neither of the offspring is anywhere near as fit as the parents. The problem is that crossover of the binary representations has no real meaning in terms of the objective function. In a situation like this, a genetic algorithm is usually a poor choice, and simulated annealing (and possibly even a Monte Carlo method) would probably perform much better.

7.5.5 Adjusting Crossover and Mutation

For some encodings the crossover and mutation operations described in Sections 7.5.1 and 7.5.2 do not result in valid children. In these situations the operations must be adjusted.

Example 7.5.7. In the traveling salesman problem (see Section 4.5.2), if the n cities to be visited are labeled 1 through n, then we can encode a tour simply as a permutation of these n integers. Thus $[3, 1, 2]$ corresponds to first visiting city 3, then city 1, and finally city 2. But using this encoding, both crossover and mutation fail to produce valid tours, if they are done as described in Sections 7.5.1 and 7.5.2.

Instead, we define a mutation by taking a random pair of indices (i, j) with $i \neq j$ and swap the positions of the corresponding cities. This produces a legitimate tour. For example, if the original tour is $[5, 4, 3, 2, 1, 0]$ and the indices chosen at random are 0 and 3, then the new tour is $[2, 4, 3, 5, 1, 0]$.

To perform a crossover between tours T and T', choose a random value k, as in the usual crossover, and start the child with the first k terms of T. The remaining cities are then put into the child tour in the same order that they appear in T'. For example, if $T = [5, 4, 3, 2, 1, 0]$ and $T' = [1, 2, 5, 3, 0, 4]$, with $k = 3$, then the child is $[5, 4, 3, 1, 2, 0]$. Both the modified mutation and the modified crossover operation clearly produce legitimate tours if they begin with legitimate tours.

7.5.6 Additional Considerations

Stopping Criteria

Stopping criteria for genetic algorithms depend on the problem. In some cases we may know that the objective is nonnegative (as in the eight queens problem), and so when the algorithm gets to an input that yields 0, it must be a global minimizer. But in many cases we have no way to identify that we have actually found the optimizer. In these cases the algorithm must continue generating populations for some fixed number of generations, always remembering what the best candidate is so far. Alternatively, the algorithm could run until a feasible point is found that meets some standard of quality—for example, the objective function falls below some predetermined threshold.

Other Methods

The quote at the beginning of this section, that "genetic algorithms are the third best way to do almost anything," reflects the fact that genetic algorithms can be used to produce a passable solution for many problems, but for a given problem, there are often other (usually more specialized) methods that work better.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

7.1. Let $X = Z^2$, where $Z \sim \mathcal{N}(0,1)$ is standard normal. The random variable X has a *chi-squared* distribution with 1 degree of freedom. For each of the following problems calculate the answer in the following two ways:

(A) Using Monte Carlo methods, sampling from a standard normal distribution, and taking the number of samples equal to 10^k for each value of $k = 2, 4, 6$.

(B) Using the appropriate built-in functions from your preferred computational tool.

Compare the results of the various computations.

(i) Plot the p.d.f. of the random variable X (experiment to find a good number of bins for your histogram).

(ii) Compute the c.d.f. $F_X(x)$ for $x \in \{0.5, 1.0, 1.5\}$.

(iii) Compute the expected value $\mathbb{E}[X]$.

(iv) Compute the variance $\mathrm{Var}(X)$.

7.2. Write code to approximate π using the Monte Carlo methods in the reading, by sampling pairs from the uniform distribution on $[-1,1]\times[-1,1]$ and counting the proportion that lie inside the circle $x^2 + y^2 \leq 1$. Sample 10^k times for $k \in \{2, 4, 6\}$ and compare the results to the true value of π. Calculate the (approximate) standard error for each of these estimates.

7.3. There are at least two different Monte Carlo methods you could use to approximate the area under the curve $y = e^{\cos(x^2)}$ for $x \in [0, 2]$.

(i) Estimate the integral

$$\int_0^2 h(x)\,dx = 2\int_0^2 h(x) f_{\mathrm{Uniform}([0,2])}(x)\,dx = 2\mathbb{E}[h \circ X]$$

by approximating $\mathbb{E}[h \circ X]$, with sampling from $\mathrm{Uniform}([0,2])$ at least 10^5 times. What is the (approximate value of the) standard error?

(ii) Estimate the area under the curve by taking a 2d-sample

$$Z \sim \text{Uniform}([0,2]) \times \text{Uniform}([0,b])$$

for some b with $b \geq \max_{\{x\in[0,2]\}} e^{\cos(x^2)}$ and then count (and scale appropriately) the samples that lie under the curve. This corresponds to estimating a multiple of $\mathbb{E}[k \circ Z]$, where $k = \mathbb{1}_A$ and $A = \{(x,y) \mid 0 \leq y \leq e^{\cos(x^2)}\}$. Sample at least 5×10^4 times. What is the (approximate value of the) standard error?

Compare the results of the two methods.

7.4. Use Monte Carlo methods to estimate the probability that a sample from Beta$(2,5)$ will be less than a sample from Beta$(20,55)$. How many samples are required to give 95% confidence that your answer is correct to three decimal places?

7.5. Consider a game where you roll a fair four-sided die with sides labeled 1, 2, 3, and 4. You win \$1 if the die shows 1 or 2, you win \$2 if the die shows 3, and you lose \$1 if the die shows 4. Use Monte Carlo simulation methods to estimate the probability that your winnings will be negative after 10 rolls of the die. Justify your choice of the number of samples to use.

7.6. Sampling from the standard normal distribution 10^5 times, calculate a Monte Carlo estimate of the value of $\int_3^\infty e^{-\frac{x^2}{2}}\,dx$, and estimate the standard error. Repeat the problem, sampling instead from $\mathcal{N}(3,1)$, as in (7.2).

7.7. Use importance sampling, drawing from Beta(a,b) for various values of a and b, to estimate the integral $\int_0^{2\pi} \frac{dx}{x^3+x+1}$. Find values of a and b and sample size n that will give a standard error less than 10^{-3}.
Hint: The domain of Beta(a,b) is $[0,1]$, but the integral is to be evaluated over $[0,2\pi]$, so you'll need to do a change of variables to be able to calculate this with samples from Beta(a,b).

7.8. The c.d.f. of the exponential distribution Gamma$(1,\lambda)$ is $F(x) = 1 - e^{-\lambda x}$.

(i) Show that the inverse is $F^{-1}(y) = -\frac{\log(1-y)}{\lambda}$.

(ii) Prove that if $Y \sim \text{Uniform}(0,1)$, then $1 - Y \sim \text{Uniform}(0,1)$.

(iii) Thus, a draw from the exponential distribution can be constructed by drawing u from Uniform$(0,1)$ and computing $-\frac{\log(u)}{\lambda}$. Write code to implement this, and use your code to draw 10^5 times from Gamma$(1,2)$. Plot a normed histogram of your results, and, on the same graph, plot the p.d.f. of the exponential distribution for comparison.

7.9. The *logistic distribution* has p.d.f. $f(x) = \frac{e^{-x}}{(1+e^{-x})^2}$ and c.d.f. $F(x) = \frac{1}{1+e^{-x}}$.

(i) Compute the inverse of F.

(ii) Code up the inversion sampling algorithm to sample from the logistic distribution, and use your code to draw from the logistic distribution 10^5 times. Plot a normed histogram of your results, and on the same graph plot the p.d.f. of the logistic distribution for comparison.

(iii) Using your sample, estimate the mean and variance of the distribution.

7.10. Code up a method for estimating the volume of the unit ball in d-dimensional space using rejection sampling. Compute estimates for these values for $d \in \{1, \ldots, 10\}$ using enough samples to get a standard error less than 10^{-2}.

7.11. Let P be a distribution on $[0, \infty)$ with p.d.f. equal to $f_P(x) = \frac{1}{Z} e^{-x^2 - x^3}$ for some constant $Z > 0$ (the constant is $Z = \int_0^\infty e^{-x^2-x^3}\, dx$). Use rejection sampling with proposal distribution $Q \sim \text{Gamma}(1, 1)$ to sample from this distribution as follows:

(i) Find the smallest m for which $e^{-x^2-x^3} \leq m e^{-x}$.

(ii) Find the smallest M (expressed in terms of m and Z) for which $f_P(x) \leq M f_Q(x)$ for all $x \in [0, \infty)$. Show that the quantity $\frac{f_P(x)}{M f_Q(x)}$ can be computed without knowing Z.

(iii) Code up a rejection sampler that draws z from Gamma$(1, 1)$ (using either the sampler you wrote for Exercise 7.8 or another sampler), draws u from Uniform$([0, 1])$, and rejects any z such that the corresponding u is greater than $\frac{f_P(z)}{M f_Q(z)}$.

(iv) Use your sampling method to draw 10^5 times from P, and plot a normed histogram of the results, along with a plot of the p.d.f. $f_P(x)$ (this last plot will require you to approximate Z).

7.12. Insert the sequence of keys F O R G I V E, in that order, into an initially empty hash table using chaining with linked lists. Assume the table consists of $n = 4$ linked lists, and use the hash function that maps the jth letter in the alphabet to $3j \pmod n$. Start counting at 0, so, for example, `hash(A) = 3 * 0 % 4 = 0` and `hash(B) = 3 * 1 % 4 = 3`. Give the contents of the table at each step.

7.13. Let S be the set consisting of all 10 digits and all upper- and lowercase letters. Let U be the set of all (ordered) strings consisting of exactly four elements from S. Construct an injective hash function $h : U \to \mathbb{N}$ with the smallest possible value of $\max_U h(u)$. Modify your hash function to give a simply uniform map h' to the set $\{0, \ldots, 30\}$. In other words, if U is a probability space with $P(u) = P(u')$ for all $u, u' \in U$, then the random variable $h' : U \to \{0, \ldots, 30\}$ should have a uniform distribution.

7.14. Hashing is important for computer security. Rather than store passwords in a file, many systems only store hashed passwords. To check if a user has entered a correct password, the system hashes the entered password and compares it to the hashed password in the file. This makes checking the password easy, but it can also make the password file more secure.

(i) Assuming the hash function is easy to evaluate but hard to invert, if you had access to the entire file of hashed passwords, how could you try to find a password to break into the system?

(ii) Assuming the number of possible hash values is much larger than the universe of possible passwords, that the hash function can be evaluated in constant time, and that the hash function is injective, what is the temporal complexity of your password search, as a function of the size K of the universe of all possible passwords?

(iii) Calculate the complexity of the search if passwords consist of only lowercase letters and are only six characters long. Compare this to the complexity if passwords are eight characters long and may include both upper- and lowercase letters, as well as digits (but no special characters).

7.15. Prove that the probability of a hash collision with a simply uniform 32-bit hash function is at least 50% if the number of keys is at least 77,164. Hint: Naïve application of the obvious formula will probably not work (why not?).

7.16. Let S be a finite set (the universe of possible keys for a hash table) of size K, and assume that all keys in S are equally likely. Thus any hash function $h : S \to \{0, \dots, n-1\}$ is a random variable with a discrete distribution

$$g_h(x) = P(h(s) = x) = \frac{1}{K}|\{s \in S \mid h(s) = x\}|.$$

(i) Prove that if two keys are chosen at random (uniformly in S), the probability $P_h(2)$ of a hash collision is

$$P_h(2) = 1 - \sum_{x_1=0}^{n-1} \sum_{x_2 \neq x_1} g_h(x_1) g_h(x_2) = \sum_{x=0}^{n-1} g_h(x)^2.$$

(ii)* Prove that $P_h(2)$ is minimized when h is simply uniform (that is, $g_h(x) = \frac{1}{n}$ for all x). Hint: You can either use Lagrange multipliers to account for the constraint $\sum_{x=0}^{n-1} g_h(x) = 1$ or use the constraint to solve for $g_h(n-1)$ in terms of each of the other values of $g_h(x)$ and then find where the gradient of $P_h(2)$ (as a function of the values of $g_h(x)$ for $x \in \{0, \dots, n-2\}$) vanishes.

7.17. Implement the stochastic hill-sliding algorithm on $\Omega = \mathbb{R}$, where at stage k the proposal z_k is drawn from $\mathcal{N}(x_k, \sigma^2)$. Your code should accept an initial guess x_0, a callable function f, the variance σ^2, and an integer n. At each stage it should draw z_k from $\mathcal{N}(x_k, \sigma^2)$ and accept it only if $f(z_k) < f(x_k)$. It should terminate after n iterations and return its best estimate x_n of the minimizer. For each choice of $\sigma^2 \in \{2^{-3}, 2^{-2}, 2^{-1}, 2^0, 2^1\}$, apply your code to the function f in Example 7.4.1, starting at $x_0 = 0$, and running for $n = 100$ steps. Plot the value of $f(x_k)$ as a function of k. Compare the results. Explain.

7.18. Implement simulated annealing on $\Omega = \mathbb{R}$, where at stage k the proposal z_k is drawn from $\mathcal{N}(x_k, \sigma^2)$. Your code should accept an initial guess x_0, a callable function f, the variance σ^2, an integer n, and a monotone decreasing cooling schedule $(t_k)_{k \in \mathbb{N}}$ defined as a function of k. At each stage it should draw z_k from $\mathcal{N}(x_k, \sigma^2)$ and accept z_k with probability $p_k(z_k, x_k)$. It should terminate after n iterations and return its best estimate of the minimizer (not necessarily the final point x_n). For each of the cooling schedules $t_k = 10/(k+1)$, $t_k = 10/(k+1)^2$, $t_k = 10/(k+1)^3$, and $t_k = 2^{-(k+1)}$, apply your code to the function f in Example 7.4.1, with $\sigma^2 = \frac{1}{4}$, starting at $x_0 = 0$, and running for $n = 100$ steps. Plot the value of $f(x_k)$ as a function of k.

Compare the results for the different cooling schedules to each other and to the results of the previous problem. Explain.

7.19. Adapt your code from the previous problem to work in $\mathbb{R}^2$ and use it to find the global minimizer of the function

$$f(x,y) = \exp(\sin(50x)) + \sin(60e^y) + \sin(70\sin x) + \sin(\sin(80y)) - \sin(10(x+y)) + \frac{1}{4}(x^2+y^2).$$

Hint: The current best-known estimate of the minimum value is -3.3068686.

7.20. Implement the stochastic hill-sliding algorithm for solving the TSP (see Section 4.5.2) in the plane. Assuming that the distance between any two cities is the Euclidean distance

$$d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}.$$

Your code should accept an integer n, representing the maximum number of iterations, and a list $L = [(x_0,y_0),\ldots,(x_n,y_n)]$ of ordered pairs, representing Cartesian coordinates of the city locations in the plane, listed in the current order that the cities will be visited, assuming the traveler's home is located at the point (x_0,y_0). Given one route (list of cities), construct a new proposed route by choosing a pair of indices (i,j) at random and swapping the position of the two cities at those indices. Your code should return a list corresponding to the route that minimizes the total distance traveled. Hint: Remember that the traveler must also return home at the end of the trip.
Use your code to find the best route you can for a list of 15 pairs of random integers drawn uniformly from $\{-20,-19,\ldots,19,20\}$. Let the code run until it stops improving. Plot the cities, the initial route, and the final route.

7.21. Modify your code in the previous problem to implement the simulated annealing algorithm for solving the TSP (see Section 4.5.2) in the plane. (In addition to the inputs from the previous problem, your code should also accept a cooling schedule $(t_k)_{k\in\mathbb{N}}$, represented as a function of k.)
Apply your code to the same list used in the previous problem. Plot the cities, the initial route, and the final route. Compare the results and the number of steps it takes to converge for a variety of cooling schedules.

7.22. Consider a genetic algorithm on binary sequences of length 4 that allows only crossover, but never mutation or selection (so the population grows at every generation). Give a population of two individuals from which it would not be possible for the algorithm to generate all possible length-4 binary sequences; that is, no matter how long the algorithm runs and no matter which choices of k are chosen at each step, no amount of crossover (as described in Section 7.5.1) could ever generate all of the 16 possible sequences. Are there two individuals from whom all the sequences could be generated? Prove or disprove.

7.23. Describe the primary benefits and disadvantages of having a very large population or a very small population in each generation. What if there is a very large population, but only the fittest individuals in that population are allowed to cross?

7.24. Implement the genetic algorithm described in Example 7.5.3 to maximize the function $f(x) = (x - 15)^2$ on the domain $\Omega = \{0, 1, 2, \ldots, 31\}$. Show the population and the value of the objective f on the most fit individual after each generation.

7.25. Implement the genetic algorithm described in Example 7.5.5, starting with a population of four configurations. At each generation cross the two most fit individuals and also the most fit individual with one (randomly selected) of the two remaining individuals, resulting in four new children for a total of eight individuals. Then perform random mutation on one of the eight. Finally, evaluate all eight for fitness and keep only the four best to start the next generation. Use your code to solve the eight queens problem. Show the results of each generation and the value of the objective function for the most fit individual in each population.

7.26. Implement a genetic algorithm for solving the TSP in the plane using the encoding, mutation, and crossover operations described in Example 7.5.7. Apply your code to the same list of cities used in Exercise 7.21 and compare the results, as well as the speed of convergence.

Notes

Sections 7.1 and 7.2 are inspired in part by [BH15] and [Was04]. For more on importance sampling, see [Was04, Section 25.3]. Exercise 7.4 and Exercise 7.5 are inspired by [Kur15]. We are grateful to Chris Grant for showing us a slick way to compute the exact solution of the problem in Example 7.1.3 using a nice symmetry argument.

A more complete version of John Denker's quote, as cited in [RN03], is "Neural networks are the second best way to do almost anything, and genetic algorithms are the third."

Part II

Approximation

8 Harmonic Analysis

If you want to find the secrets of the universe, think in terms of energy, frequency, and vibration.
—Nikola Tesla

The field of *harmonic analysis* is concerned with both the representation and the approximation of functions, including signals[36] and images, as linear combinations of basic waves. In this chapter we consider two branches of harmonic analysis corresponding to two different classes of "basic waves." The first comes from the class of trigonometric functions (sines and cosines) and is called *Fourier analysis*. The second deals with self-similar waves and is called *wavelet analysis*.

Fourier analysis shows how to represent, or closely approximate, most well-behaved functions as sums of trigonometric functions with varying frequencies and amplitudes. This allows us to decompose a signal into different frequencies and analyze the contribution of each frequency toward a given signal. The net result is a powerful set of tools that can be used in myriad applications.

For example, wireless communication relies on the ability to isolate and extract the part of a transmitted signal corresponding to a specific frequency or narrow band of frequencies. The electromagnetic spectrum includes signals of all frequencies from the various wireless devices that transmit and receive information, both across the world and throughout space and time, as well as interference from the sun, microwave ovens, and other sources of radiation. The *superposition principle* states that when waves of differing frequencies travel through the same medium at the same time, the net displacement of the medium at any point in space or time is the sum of the individual wave displacements. Or, said more simply, the resulting wave is just a sum of the individual frequencies. This is what allows a cell phone to pick up a broad range of signals from countless sources of electromagnetic energy and filter out all the frequencies except those specific to the cell tower. Fourier analysis is the mathematical tool that allows the design and optimization of the algorithms and circuits used to filter out all the undesired frequencies.

[36] In many quantitative disciplines a *signal* is a function that contains information about the behavior or attributes of some phenomenon. In this text, when we say *signal* we mean a function that depends on time.

More generally, the field of signal processing deals with taking readings that come from various electronic devices or sensors (known as *the signal*), and separating out and analyzing the desired part of that signal from the undesired part (called *noise*). For example, when producing music in a recording studio, one usually wants to filter out and discard background noises such as the hum from air conditioning and ventilation systems. Before the age of digital electronics, both audio recordings and long-distance phone calls contained a lot of static noise, or interference, that often made it difficult to hear what someone was saying. Today, signal processing algorithms are able to remove the interference and provide crisp, clear communications that are mostly free of distortions or noise. It should be of no surprise that much of the theory of signal processing came out of the telephone industry, most notably Bell Labs.

Applications of Fourier analysis go well beyond the physics of light and sound. For example, stock market analysts may want to separate out intraday trading effects, which have a 1-day frequency, from day-of-week effects, which have a 7-day frequency, and seasonal effects, which have an annual frequency. This may help them understand whether a particular decline in market activity is signaling a change in investor interest or whether it's just lunchtime.

Regardless of the application, a good rule of thumb is that oscillatory signals are likely to be amenable to Fourier analysis. In other words, greater understanding will likely come from decomposing the signal into trigonometric functions of varying frequency and amplitude.

We begin this chapter by showing how to represent or approximate a function as a linear combination of trigonometric functions of varying frequencies. These sums are called *Fourier series*. There are two conventions for representing Fourier series. The first represents functions as sums of exponential functions with imaginary exponents. The second convention represents functions as sums of sines and cosines. Because of Euler's identity (8.1), these two conventions are equivalent, but, as we show in Section 8.5, the exponential form is better for computation. With either convention, the process of mapping a signal into a Fourier series corresponds to an orthogonal projection into a certain function space, where the Fourier series is the image expressed in an orthonormal basis with respect to a certain inner product.

A second important topic in this chapter is the idea of approximating a function by sampling[37] at equally spaced points in time, that is, going from a continuous-time signal to an equally spaced discrete-time signal. In Section 8.5, we introduce the *discrete Fourier transform* (DFT), which computes a Fourier series of a given discrete-time signal. Because there are infinitely many possible Fourier series that could produce the same discrete-time signal, we give the "best" Fourier series, in a sense to be described in Section 8.7.

Transforming a discrete-time signal into a Fourier series with the DFT provides a powerful way to analyze, filter, and compress the signal. The naïve way to compute

[37]To *sample* means to observe or measure. In Chapter 7, we used the term sampling to mean observing realizations (or draws) from a random process like flips of a coin or rolls of a die. In this chapter, there is no randomness or variation in the observations, and thus repeated experiments yield identical observations—it's the same signal each time—but we still use the term *sample* to indicate we are observing or measuring something.

the DFT on a sample of n data points takes $O(n^2)$ time. However, there is a very fast algorithm for computing the DFT called the *fast Fourier transform (FFT)*, which takes only $O(n \log n)$ time. For large n, this makes a substantial difference in computing times. This is especially the case for data-rich fields like medical imaging, where the FFT has profoundly sped up computations (and, consequently, aided in the saving of countless lives). Indeed, the FFT is considered to be one of the most important algorithms of all time and is ubiquitous in signal and image processing.

A third major topic considered in Fourier analysis is an important theorem called the *periodic sampling theorem*, which guarantees that, for periodic functions, the original signal can be perfectly reconstructed from a finite number of samples, provided that the signal is sampled often enough and is *band limited.* More precisely, the theorem states that perfect reconstruction of a band-limited signal is guaranteed whenever the sampling frequency is more than double the highest frequency occurring in the signal.

In spite of the great usefulness of Fourier analysis, it does have weaknesses because trigonometric functions do not gracefully approximate discontinuous signals. As we show in several examples in this chapter, in the presence of a discontinuity the coefficients of the Fourier series do not converge to zero very quickly, resulting in poor approximations. Hence, for signals with many discontinuities and especially for images of objects with sharp edges, Fourier analysis is probably not the best way to proceed.

As an alternative, we consider another class of basic waves that are useful in signal and image processing, called *wavelets*. Wavelet analysis uses rescaling and translation of the basic wavelets to build an orthonormal basis that forms a function space capable of representing or approximating most well-behaved functions. Wavelet bases can usually be chosen so that the wavelet representation is *sparse*, meaning that it has only a small number of nonzero entries. One of the big advantages of wavelets is that they can be made to gracefully and efficiently approximate functions with discontinuities; see Sections 8.8 and 8.10 for details. Hence, they are great for analyzing signals with many discontinuities and images of objects with sharp edges.

There are many similarities between wavelet analysis and Fourier analysis. Just as the trigonometric functions in Fourier analysis, wavelets form an orthonormal basis, so that an inner product can be used to identify the coefficients for each basis function. In fact, in Sections 8.9 and 8.11, we discuss the *discrete wavelet transform (DWT)*, which is an algorithm used to represent a discrete signal in a wavelet basis, just as the DFT represents a discrete signal in the Fourier basis. Finally, just as the DFT can be made fast with the FFT, the DWT can be made fast with the *fast wavelet transform (FWT)*, which requires only $O(n)$ time.

But there are some key differences between wavelet analysis and Fourier analysis. In addition to being able to gracefully deal with discontinuities, wavelet basis functions are well suited to representing local behavior. Their ability to easily express both local properties and discontinuities make wavelets very useful for analyzing nonperiodic, piecewise-continuous functions, such as those observed in images, videos, and other digital media. By contrast, Fourier basis functions are periodic on $\mathbb{R}$ (and therefore not of compact support), and thus they are better suited to representing global behavior.

8.1 Complex Numbers

In this section[38] we briefly review some fundamental properties of the complex numbers, which are essential in Fourier analysis. The main results that we need are Euler's identity (8.1), the "inverse" of Euler's identity (8.3), a relation on sums of roots of unity (Proposition 8.1.6), and a relation for powers of roots of unity (Proposition 8.1.7). Students who are already familiar with complex numbers, including these four results, may skip to the next section.

8.1.1 Basics of Complex Numbers

Definition 8.1.1. *Let i be a formal symbol (representing a square root of -1). Let $\mathbb{C}$ denote the set*

$$\mathbb{C} = \{a + bi \mid a, b \in \mathbb{R}\}.$$

Elements of $\mathbb{C}$ are called complex numbers. *We define addition of complex numbers by*

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

and we define multiplication of complex numbers by

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i.$$

Example 8.1.2. Complex numbers of the form $a+0i$ are usually written just as a, and those of the form $0 + bi$ are usually written just as bi.

We verify that i has the expected property:

$$i^2 = (0 + 1i)^2 = (0 + 1i)(0 + 1i) = (0 - 1) + (0 + 0)i = -1.$$

Definition 8.1.3. *For any $z = a + bi \in \mathbb{C}$ we define the* complex conjugate *of z to be $\bar{z} = a - bi$, and we define the* modulus *(sometimes also called the* norm*) of z to be $|z| = \sqrt{z\bar{z}} = \sqrt{a^2 + b^2} \in \mathbb{R}$. We also define the real part $\Re(z) = a$, and the imaginary part $\Im(z) = b$.*

Proposition 8.1.4. *Addition and multiplication of complex numbers satisfy the following properties. For any $z, w, v \in \mathbb{C}$ we have*

(i) *associativity of addition:* $(v + w) + z = v + (w + z)$;

(ii) *commutativity of addition:* $z + w = w + z$;

(iii) *associativity of multiplication:* $(vw)z = v(wz)$;

(iv) *commutativity of multiplication:* $zw = wz$;

(v) *distributivity:* $v(w + z) = vw + vz$;

(vi) *additive identity:* $0 + z = z = z + 0$;

[38] This section has also been published as an appendix in Volume 1.

(vii) *multiplicative identity:* $1 \cdot z = z = z \cdot 1$;

(viii) *additive inverses: if* $z = a + bi$, *then* $-z = -a - bi$ *satisfies* $z + (-z) = 0$;

(ix) *multiplicative inverses: if* $z = a + bi \neq 0$, *then* $|z|^{-2} \in \mathbb{R}$ *and so*

$$z^{-1} = \bar{z}|z|^{-2} = \frac{a - bi}{a^2 + b^2}.$$

Proof. All of the properties are straightforward algebraic manipulations. We give one example and leave the rest to the reader.

For (ix) first note that since $z \neq 0$ we have $|z|^2 = a^2 + b^2 \neq 0$, so its multiplicative inverse $(a^2 + b^2)^{-1}$ is also in $\mathbb{R}$. We have

$$z\left(\bar{z}|z|^{-2}\right) = z\bar{z}(z\bar{z})^{-1} = 1,$$

so $\left(\bar{z}|z|^{-2}\right)$ is the multiplicative inverse to z. □

8.1.2 Euler's Formula and Graphical Representation

Euler's Formula

For any $z \in \mathbb{C}$ we define the exponential e^z using the Taylor series

$$e^z = \sum_{k=0}^{\infty} \frac{z^k}{k!}.$$

One of the most important identities for complex numbers is *Euler's formula* (see Proposition 11.2.12 in Volume 1):

$$e^{it} = \cos(t) + i\sin(t), \quad t \in \mathbb{R}. \tag{8.1}$$

As a consequence of Euler's formula, we have *De Moivre's formula*:

$$(\cos(t) + i\sin(t))^n = (e^{it})^n = e^{int} = \cos(nt) + i\sin(nt), \quad n \in \mathbb{N}, t \in \mathbb{R}. \tag{8.2}$$

Inverting Euler's Formula

By taking the real and imaginary parts of e^{it} we can invert Euler's identity and write the sine and cosine formulas in terms of exponentials. Thus we have

$$\cos(t) = \Re(e^{it}) = \frac{e^{it} + e^{-it}}{2} \qquad \text{and} \qquad \sin(t) = \Im(e^{it}) = \frac{e^{it} - e^{-it}}{2i}. \tag{8.3}$$

Trig Identities

Using (8.1), we can derive some key identities from trigonometry. Expanding $e^{i(x \pm y)} = e^{ix}e^{\pm iy}$ into real and imaginary parts gives

$$\begin{aligned}
&\cos(x \pm y) + i\sin(x \pm y) \\
&\quad = (\cos(x) + i\sin(x))\,(\cos(y) \pm i\sin(y)) \\
&\quad = \cos(x)\cos(y) \pm \sin(x)\sin(y) + i\left[\sin(x)\cos(y) \pm \cos(x)\sin(y)\right].
\end{aligned}$$

Thus taking the real and imaginary parts gives the following identities:

$$\cos(x \pm y) = \cos(x)\cos(y) \mp \sin(x)\sin(y), \tag{8.4a}$$
$$\sin(x \pm y) = \sin(x)\cos(y) \pm \cos(x)\sin(y). \tag{8.4b}$$

From here it's not hard to get the following three identities:

$$2\cos(x)\cos(y) = \cos(x+y) + \cos(x-y), \tag{8.5a}$$
$$2\sin(x)\cos(y) = \sin(x+y) + \sin(x-y), \tag{8.5b}$$
$$2\sin(x)\sin(y) = \cos(x-y) - \cos(x+y). \tag{8.5c}$$

Graphical Representation

The complex numbers have a very useful graphical representation as points in the plane, where we associate the complex number $z = a+bi$ with the point $(a,b) \in \mathbb{R}^2$; see Figure 8.1. In this representation real numbers lie along the x-axis and imaginary numbers lie along the y-axis. The modulus $|z|$ of z is the distance from the origin to z in the plane and the complex conjugate $\bar{z}$ is the image of z under a reflection through the x-axis.

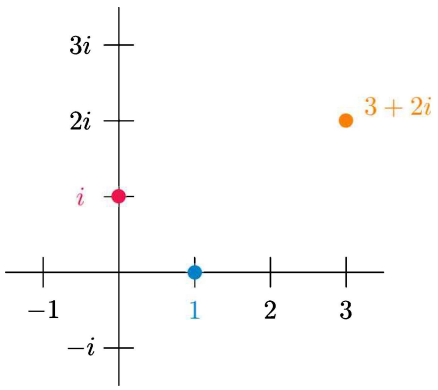


Figure 8.1. *A complex number $x+iy$ with $x,y \in \mathbb{R}$ is usually represented graphically in the plane as the point (x,y). This figure shows the graphical representation of the complex numbers i (red), 1 (blue), and $3+2i$ (orange).*

Addition of complex numbers is just the same as vector addition in the plane, so geometrically, the complex number $z+w$ is the point in the plane corresponding to the far corner of the parallelogram whose other corners are 0, z, and w; see Figure 8.2.

Using (8.1), we can represent any point in the plane in polar form as $z = r(\cos(\theta) + i\sin(\theta))$ for some $\theta \in [0, 2\pi)$ and some $r \in \mathbb{R}$ with $r \geq 0$. Combining this with Euler's formula means that we can write every complex number in the form $z = re^{i\theta}$. In this form we have

$$|z| = |re^{i\theta}| = |r(\cos(\theta) + i\sin(\theta))| = r$$

and

$$\bar{z} = r(\cos(\theta) - i\sin(\theta)) = re^{-i\theta}.$$

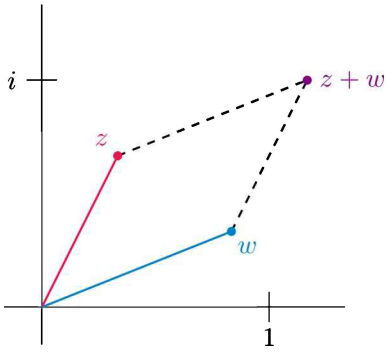


Figure 8.2. *Graphical representation of complex addition. Thinking of complex numbers $z = a + bi$ (red) and $w = c + di$ (blue) as the points in the plane (a, b) and (c, d), respectively, their sum $z + w = (a + c) + (b + d)i$ (purple) corresponds to the usual vector sum $(a, b) + (c, d) = (a + c, b + d)$ in the plane.*

We define the *sign* of $z = re^{i\theta} \in \mathbb{C}$ to be

$$\operatorname{sign}(z) = \begin{cases} e^{i\theta} = \frac{z}{|z|} & \text{if } z \neq 0, \\ 1 & \text{if } z = 0. \end{cases} \tag{8.6}$$

The polar form gives a geometric interpretation for multiplication of complex numbers. If $z = re^{it}$ and $w = \rho e^{is}$, then

$$wz = r\rho e^{i(t+s)} = |z||w|(\cos(t + s) + i \sin(t + s)).$$

This shows that multiplication of two complex numbers in polar form multiplies the moduli and adds the angles; see Figure 8.3.

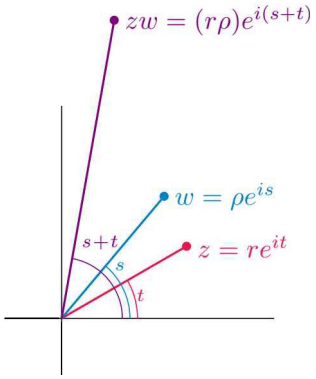


Figure 8.3. *Complex multiplication adds the polar angles $(s + t)$ and multiplies the moduli $(r\rho)$.*

Similarly, $z^{-1} = \bar{z}|z|^{-2} = re^{-it}r^{-2} = r^{-1}e^{-it}$, so the multiplicative inverse changes the sign of the angle $(t \mapsto -t)$ and inverts the modulus $(r \mapsto r^{-1})$. But the complex conjugate leaves the modulus unchanged and changes the sign of the angle; see Figure 8.4.

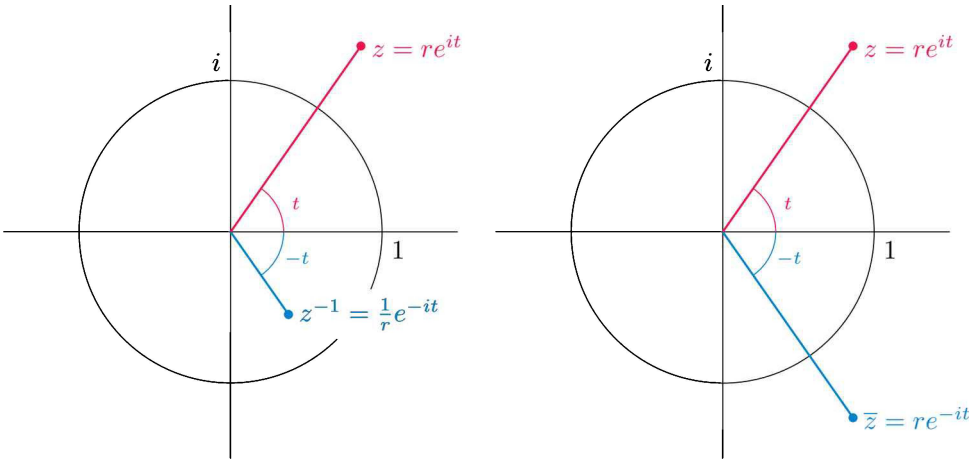


Figure 8.4. *Graphical representation of multiplicative inverse* (a)*, and complex conjugate* (b)*. The multiplicative inverse of a complex number changes the sign of the polar angle and inverts the modulus. The complex conjugate also changes the sign of the polar angle but leaves the modulus unchanged.*

8.1.3 Roots of Unity

Definition 8.1.5. *For $n \in \mathbb{Z}^+$ an* nth root of unity *is any solution to the equation $z^n = 1$ in $\mathbb{C}$. The complex number $\omega_n = e^{2\pi i/n}$ is called the* primitive nth root of unity.

By the fundamental theorem of algebra there are exactly n of these nth roots of unity in $\mathbb{C}$. Euler's formula tells us that

$$\omega_n = \cos(2\pi/n) + i\sin(2\pi/n)$$

is the point on the unit circle in the complex plane corresponding to an angle of $2\pi/n$ radians, and

$$\omega_n^k = e^{2\pi ik/n} = \cos(2\pi k/n) + i\sin(2\pi k/n).$$

Thus we have

$$\omega_n^n = e^{2\pi i} = 1,$$

so ω_n^k is a root of unity for every $k \in \mathbb{Z}$; see Figure 8.5.

If $k' \equiv k \pmod{n}$, then $k' = k + mn$ for some $m \in \mathbb{Z}$, and thus

$$\omega_n^{k'} = \omega_n^{(k+mn)} = \omega_n^k(\omega_n^n)^m = \omega_n^k.$$

The nth roots of unity are uniformly distributed around the unit circle, so their average is 0. The next proposition makes that precise.

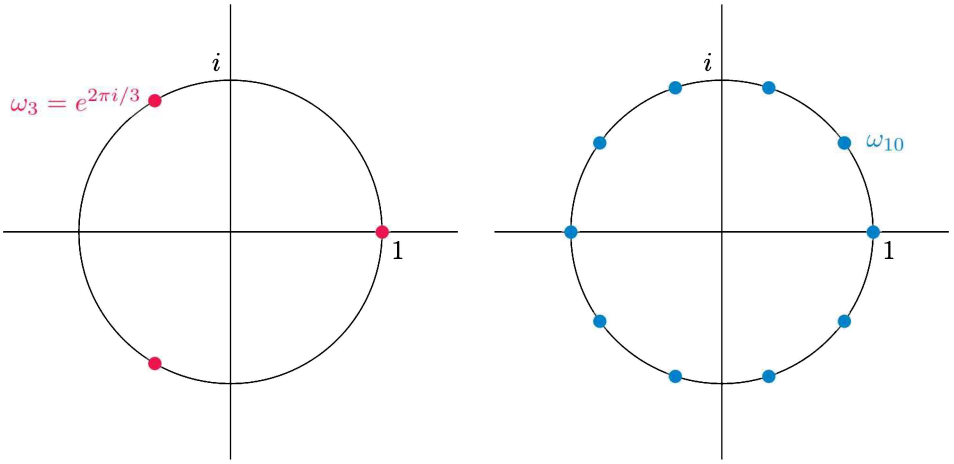


Figure 8.5. *Plots of all the third (on the left) and tenth (on the right) roots of unity. The roots are uniformly distributed around the unit circle, so their sum is* 0.

Proposition 8.1.6. *For any $n \in \mathbb{Z}^+$ and any $k \in \mathbb{Z}$ we have*

$$\frac{1}{n}\sum_{\ell=0}^{n-1} \omega_n^{k\ell} = \frac{1}{n}\sum_{\ell=0}^{n-1} e^{2\pi i k\ell/n} = \begin{cases} 0 & \text{if } k \not\equiv 0 \pmod{n}, \\ 1 & \text{if } k \equiv 0 \pmod{n}. \end{cases} \tag{8.7}$$

Proof. The sum $\sum_{\ell=0}^{n-1}(\omega_n^k)^\ell$ is a geometric series, so if $k \not\equiv 0 \pmod{n}$ we have

$$\sum_{\ell=0}^{n-1} \omega_n^{k\ell} = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = 0.$$

But if $k \equiv 0 \pmod{n}$, then

$$\frac{1}{n}\sum_{\ell=0}^{n-1} \omega_n^{k\ell} = \frac{1}{n}\sum_{\ell=0}^{n-1} 1 = 1. \quad \square$$

We conclude this section with a simple observation that turns out to be very powerful. The proof is immediate.

Proposition 8.1.7. *For any divisor d of n and any $k \in \mathbb{Z}$, we have*

$$\omega_n^{kd} = \omega_{n/d}^k. \tag{8.8}$$

The relation (8.8) is key in the derivation of the FFT; see Section 8.5.4.

8.2 Fourier Series

The idea of Fourier series is to use an orthonormal basis of trigonometric functions to efficiently represent or approximate a broad class of well-behaved functions (to be made precise in Section 8.4). By writing functions as a linear combination of these

basis functions, we can process and transform a signal in many useful ways. As discussed above, there are many situations where one may want to isolate certain frequencies in a signal or, alternatively, filter out unwanted frequencies that add noise or interference to the signal. But the implications of Fourier analysis are much farther reaching, and deep insights can often come from analyzing the different frequencies that contribute to a signal. For example, if a machine is rattling at a certain frequency, one may be able to identify the specific cause of the problem inside the machine, based in part on the frequency of the rattle.

In this section we develop Fourier series on the interval $[0, T]$. As mentioned in the introduction to this chapter, there are two different, but equivalent, ways of expressing Fourier series. The first and more modern version is expressed in terms of complex-exponential functions of the form $e^{\pm i\omega kt}$ with $k \in \mathbb{Z}$, whereas the second, more classical, version is expressed in terms of the trigonometric functions $\sin(\omega kt)$ and $\cos(\omega kt)$ with $k \in \mathbb{N}$. Throughout this section, we assume that

$$\omega = \frac{2\pi}{T}$$

for both versions.

8.2.1 Complex-Exponential Fourier Series

Recall that $\mathbb{F}$ refers to the field of either real or complex numbers. Throughout this chapter let $L^2([0,T];\mathbb{F})$ denote the vector space of $\mathbb{F}$-valued square-integrable functions; that is, we assume every $f \in L^2([0,T];\mathbb{F})$ satisfies

$$\|f\|^2 = \int_0^T |f(t)|^2 \, dt < \infty. \tag{8.9}$$

And we assume this space is endowed with the inner product

$$\langle f, g \rangle = \frac{1}{T} \int_0^T \overline{f(t)} g(t) \, dt. \tag{8.10}$$

We show that $E = \{e^{i\omega kt}\}_{k\in\mathbb{Z}}$ is an orthonormal set in the space $L^2([0,T];\mathbb{C})$. The Fourier series of a function $f \in L^2([0,T];\mathbb{C})$ is just the orthogonal projection of f onto the subspace $X = \operatorname{span}(E)$.

Theorem 8.2.1. *The set $E = \{e^{i\omega kt}\}_{k\in\mathbb{Z}}$, with $\omega = 2\pi/T$, is orthonormal in $L^2([0,T];\mathbb{C})$.*

Proof. A straightforward integration (see Exercise 8.2) shows that

$$\left\langle e^{i\omega kt}, e^{i\omega \ell t} \right\rangle = \frac{1}{T} \int_0^T e^{i\omega(\ell - k)t} \, dt = \delta_{k\ell} = \begin{cases} 1 \text{ if } k = \ell, \\ 0 \text{ if } k \neq \ell, \end{cases}$$

where $\delta_{k\ell}$ is the *Kronecker delta.* □

Corollary 8.2.2. *If $f \in L^2([0,T];\mathbb{F})$ satisfies*

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega k t} \tag{8.11}$$

for some coefficients $(c_k)_{k\in\mathbb{Z}}$ in $\mathbb{C}$, then for each $k \in \mathbb{Z}$ we have

$$c_k = \langle e^{i\omega k t}, f\rangle = \frac{1}{T}\int_0^T f(t)e^{-i\omega k t}dt. \tag{8.12}$$

This corollary inspires the following definition of the Fourier series of a given function to be its projection onto the span of the basis vectors.

Definition 8.2.3. *Given $f \in L^2([0,T];\mathbb{F})$ and $k \in \mathbb{Z}$ we define the* kth Fourier coefficient of f *to be c_k, as given in (8.12). We define the* nth truncated Fourier series *(also called the nth partial sum of the Fourier series) to be*

$$S_n[f](t) = \sum_{k=-n}^{n} c_k e^{i\omega k t}, \tag{8.13}$$

where each c_k is the kth Fourier coefficient of f. Taking $n \to \infty$ gives the complete Fourier series

$$S[f](t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega k t}. \tag{8.14}$$

Remark 8.2.4. The definition of the Fourier coefficients and the Fourier series makes sense even if f cannot be written as a series of the form (8.11). As indicated above, the truncated and complete Fourier series are simply orthogonal projections of f onto the subspaces $X_n = \operatorname{span}(E_n)$ and $X = \operatorname{span}(E)$, respectively, where $E_n = \{e^{i\omega k t}\}_{k=-n}^{n} \subset E$. In other words,

$$\operatorname{proj}_{X_n} f = S_n[f] \quad \text{and} \quad \operatorname{proj}_X f = S[f].$$

Remark 8.2.5. Since the complex exponentials $e^{\pm i\omega k t}$ are periodic on $[0,T]$, so are the truncated and complete Fourier series (8.13) and (8.14), respectively, even if the original function is not.

Example 8.2.6. Consider the function $f(t) = t$ defined on $[0, 2\pi]$. To find the Fourier series we must find its Fourier coefficients. For $k \neq 0$, we have

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} e^{-ikt}t\,dt = \frac{1}{2\pi}\left[\frac{te^{-ikt}}{-ik}\Big|_0^{2\pi} + \int_0^{2\pi}\frac{e^{-ikt}}{-ik}\,dt\right] = \frac{i}{k},$$

and for $k = 0$ one can check that $c_0 = \pi$. Thus we have

$$S[f](t) = \sum_{k=-\infty}^{\infty} c_k e^{ikt} = \pi + i\sum_{k=1}^{\infty}\frac{e^{ikt} - e^{-ikt}}{k}. \tag{8.15}$$

Using (8.3), this can be rewritten as

$$S[f](t) = \pi - 2\sum_{k=1}^{\infty} \frac{\sin kt}{k}. \tag{8.16}$$

Figure 8.6 shows the wave and some of the partial sums of its Fourier series.

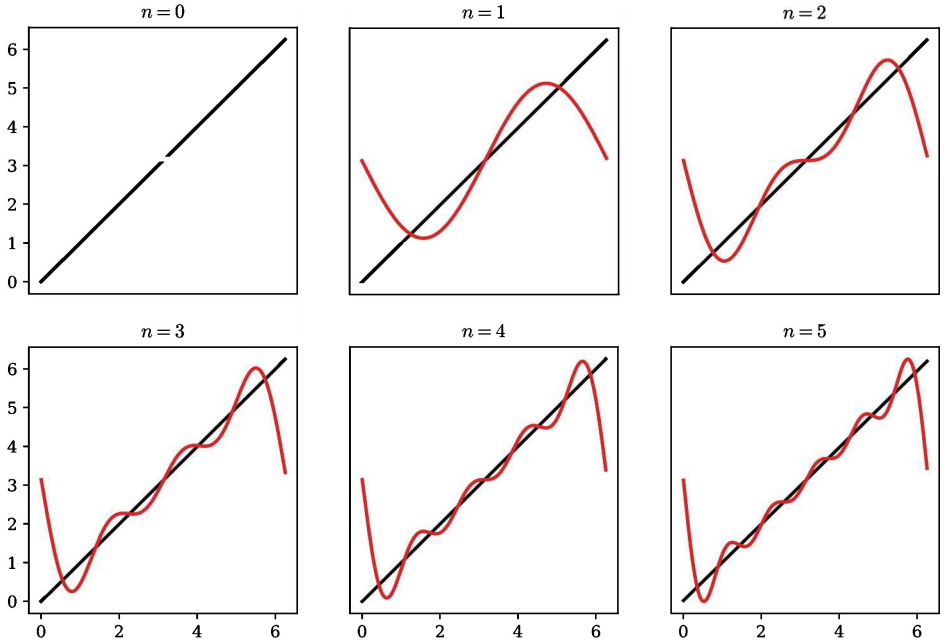


Figure 8.6. *Graphs (in red) of the nth partial sums of the Fourier series for the function $f(t) = t$ (black) on the interval $[0, 2\pi]$, as in Example 8.2.6.*

Remark 8.2.7. Example 8.2.6 shows that the Fourier series (8.15) isn't identical to the underlying function f. In particular, we have $S[f](0) = S[f](2\pi) = \pi$, but $f(0) = 0$ and $f(2\pi) = 2\pi$. However, as we show in Theorem 8.2.16, the two functions match everywhere else on the interval; that is,

$$S[f](t) = \begin{cases} t & \text{if } t \in (0, 2\pi), \\ \pi & \text{if } t = 0 \text{ or } t = 2\pi. \end{cases} \tag{8.17}$$

Remark 8.2.8. Recall that applying any projection twice is the same as applying it once. Since $S[f]$ is the projection of f onto X, as described in Remark 8.2.4, the Fourier series of f is $S[f]$ and the Fourier series of $S[f]$ is also $S[f]$.

In the case of the series (8.15), it is straightforward to verify this property if (8.17) holds, because then f and $S[f]$ differ only at the endpoints. Changing the values of the function at the endpoints does not change the values of the integrals in Example 8.2.6, and thus the coefficients c_k remain the same.

Remark 8.2.9. Since the function $S[f]$ in (8.15) is periodic, it can be extended to the entire real line; the resulting function is a sawtooth wave.

Example 8.2.10. Let $f : [0, 2\pi] \to \mathbb{R}$ satisfy

$$f(t) = \begin{cases} 1 & \text{if } t \in (0, \pi), \\ 0 & \text{if } t \in (\pi, 2\pi), \\ \frac{1}{2} & \text{if } t \in \{0, \pi, 2\pi\}. \end{cases} \tag{8.18}$$

Recall that $e^{-i\pi k} = (-1)^k$ for $k \in \mathbb{Z}$. For $k \neq 0$ we have

$$c_k = \frac{1}{2\pi} \int_0^{2\pi} f(t) e^{-ikt}\, dt = \frac{1}{2\pi} \int_0^{\pi} e^{-ikt}\, dt = \frac{e^{-ikt}}{-2\pi ik}\bigg|_0^{\pi} = \begin{cases} \frac{1}{\pi ik} & \text{if } k \text{ is odd}, \\ 0 & \text{if } k \text{ is even}. \end{cases}$$

For $k = 0$ one can show that $c_0 = \frac{1}{2}$. Thus we have

$$S[f](t) = \frac{1}{2} + \frac{1}{\pi i} \left(\sum_{k=-1}^{-\infty} \frac{e^{i(2k+1)t}}{2k+1} + \sum_{k=1}^{\infty} \frac{e^{i(2k-1)t}}{2k-1} \right). \tag{8.19}$$

Using (8.3) and Exercise 8.1 shows that (8.19) is the same as

$$\frac{1}{2} + \frac{2}{\pi} \sum_{k=1}^{\infty} \frac{\sin((2k-1)t)}{2k-1}. \tag{8.20}$$

Remark 8.2.11. Since the function $S[f]$ in (8.19) is periodic on the interval $[0, 2\pi]$, it can be extended to the entire real line, and the resulting function is a square wave.

In both Examples 8.2.6 and 8.2.10, the complex Fourier series simplifies to a real-valued function. This is not coincidence, and, in fact, it happens whenever the original function is real valued.

Proposition 8.2.12. *If $f \in L^2([0, T]; \mathbb{R})$, then $\overline{c_k} = c_{-k}$ for each $k \in \mathbb{Z}$.*

Proof. Since $\overline{f(t)} = f(t)$ and $\overline{e^{-i\omega kt}} = e^{i\omega kt}$, we have

$$\overline{c_k} = \overline{\frac{1}{T} \int_0^T f(t) e^{-i\omega kt} dt} = \frac{1}{T} \int_0^T f(t) e^{i\omega kt} dt = c_{-k}. \qquad \square$$

Remark 8.2.13. When computing the Fourier coefficients $\{c_k\}_{k \in \mathbb{Z}}$ of a real-valued function, we need only compute the coefficients c_k for nonnegative values $k \in \mathbb{N}$, because $c_{-k} = \overline{c_k}$.

As mentioned above, not every function is equal to its Fourier series. One useful sufficient condition for equality is given in Theorem 8.2.16, below. To state the theorem we first need one definition.

Definition 8.2.14. *A function $f : [a,b] \to \mathbb{F}$ is* piecewise continuous *if it is continuous on $[a,b]$ except at finitely many points and the limit at each point of discontinuity exists and is finite. We say that $f : [a,b] \to \mathbb{F}$ is* piecewise Lipschitz *if it is piecewise continuous on $[a,b]$ and there is a constant L such that on every interval I of continuity, we have*

$$|f(x) - f(y)| \leq L|x - y| \tag{8.21}$$

for all $x, y \in I$.

Example 8.2.15. If $f : [a,b] \to \mathbb{R}$ is continuously differentiable, meaning that f' exists and continuous on $[a,b]$ and both $f'(a)$ and $f'(b)$ are well defined in the sense of left and right limits, respectively, then by the mean value theorem (8.21) holds where

$$L = \sup_{x \in [a,b]} |f'(x)|.$$

Theorem 8.2.16. *For any function $f : [0,T] \to \mathbb{F}$, denote $f(t^+) = \lim_{\tau \to t^+} f(\tau)$ and $f(t^-) = \lim_{\tau \to t^-} f(\tau)$. If f is piecewise Lipschitz, then*

$$\lim_{n \to \infty} S_n[f](t) = \begin{cases} \frac{1}{2}(f(t^+) + f(t^-)) & \text{if } t \in (0,T), \\ \frac{1}{2}(f(T) + f(0)) & \text{if } t \in \{0,T\}. \end{cases} \tag{8.22}$$

In particular, if f is continuously differentiable on $(0,T)$ with finite right and left derivatives at $t = 0$ and $t = T$, respectively, with $f(0) = f(T)$, then for each $t \in [0,T]$ we have $S_n[f](t) \to f(t)$ as $n \to \infty$.

Proof. The proof is given in Section 8.4. □

8.2.2 The Theory of Music

A pure note is a sinusoid playing at a frequency in the audible range. For example, the A note just above middle C can be represented by the signal $\sin(2\pi 440t)$ (with t measured in seconds), which means that the signal cycles 440 times per second (440 Hz). This note is often called A4 because it's in the fourth octave of a standard 88-key piano.

Increasing the frequency increases the pitch of the note. The A note that's one octave higher is A5, corresponding to the signal $\sin(2\pi 880t)$ with twice the frequency (880 Hz). The signal $\sin(2\pi 660t)$ with 3/2 the frequency of A4 corresponds to E4, which is the E above A4.

Increasing the amplitude (the modulus of the coefficient in front) of the signal increases the volume of the note we hear without changing the pitch. Thus the signal $2\sin(2\pi 440t)$ is an A4 that sounds louder than the signal $\sin(2\pi 440t)$, which is the same A4, but softer.

When we hear a sound wave of the form

$$f(t) = 4\sin(2\pi 440t) + 6\sin(2\pi 660t) + 2\sin(2\pi 880t), \tag{8.23}$$

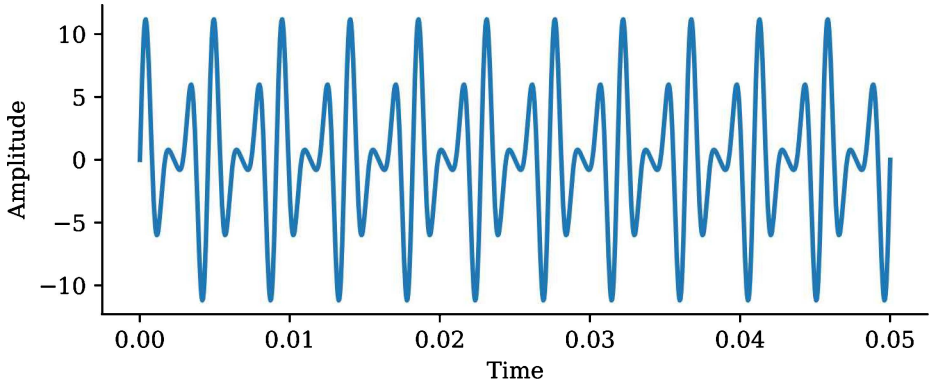


Figure 8.7. *A plot of the signal in* (8.23)*, where time is measured in seconds and* $T = 0.05$*. This signal is the sum of three separate pure sinusoidal tones, one of frequency* 440 *Hz, one of frequency* 660 *Hz, and one of frequency* 880 *Hz. A trained musician can hear and identify the separate tones in the signal, which essentially corresponds to doing a Fourier decomposition.*

we hear (or at least an experienced musician hears) a three-note chord, consisting of A4, E4, and A5, with the E4 sounding louder and the A5 sounding softer than the A4. For a plot of this signal, see Figure 8.7.

Most humans cannot hear frequencies outside the range of 20 to 20,000 Hz. So if we decompose an audio signal into its Fourier series and discard any frequencies outside the audible range, the simplified signal will sound identical to the original, but it will require less data to store the signal, and less energy to reproduce it.

Once we have the Fourier decomposition of the signal, we can adjust the amplitudes of the various frequencies. This is what a sound system's *equalizer* does. Some misguided teenagers like to increase the bass (low frequencies) on their sound systems and decrease the treble (higher frequencies).

In some settings a signal may have unwanted noise. For example, the electrical power source in a typical American home is an alternating current with a frequency of 60 Hz, and this sometimes introduces an undesirable hum into the audio system. This can be filtered out of a signal by setting the Fourier coefficient corresponding to 60 Hz (and the others near 60 Hz) to zero and leaving all the other coefficients unchanged.

Vista 8.2.17. The ideas of Fourier series can be extended by taking a limit as the interval $[0, T]$ is expanded to the entire real line $\mathbb{R}$. This extension is called the *Fourier transform*. Fourier transforms are an essential tool in many areas of applied mathematics and are discussed in Volume 4.

In Fourier series the collection of coefficients $\{c_k\}_{k\in\mathbb{Z}}$ can be thought of as a function $\hat{f} : \mathbb{Z} \to \mathbb{C}$, where $\hat{f}(k) = c_k$. In the Fourier transform, the index $k \in \mathbb{Z}$ is replaced with a continuous variable $\xi \in \mathbb{R}$, and the new function takes the form $\hat{f} : \mathbb{R} \to \mathbb{C}$, given by $\hat{f}(\xi) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\xi t}\, dt$.

8.2.3 *Gibbs Phenomenon

A careful look at Figure 8.6 shows that the distance from the graph of f down to the bottom of the lowest valley does not shrink as the number of terms in the approximation increases. Although the valleys and peaks get narrower as the number of terms increases, the valleys and peaks nearest the endpoints don't actually get closer to the graph of f. This is called the *Gibbs phenomenon*.

Extending a function from the interval $[0, T]$ to the whole line by making it T-periodic causes a discontinuity if $f(0) \neq f(T)$. Near such a discontinuity, any finite Fourier series will overshoot the graph of the function being approximated; that is, it will have a neighborhood where the series is a fixed distance $\alpha > 0$ away from the desired function. Adding more terms in the series will not shrink α, but it will shrink the size of the bad neighborhood. For more on this phenomenon, see Exercises 8.21–22.

8.3 *Trigonometric Fourier Series

In this section we develop the theory of trigonometric Fourier series, using a basis of trigonometric functions of the form $\cos(\omega kt)$ and $\sin(\omega kt)$ instead of complex-exponential functions. It should not be surprising that the complex-exponential version is equivalent to the trigonometric version. Indeed Euler's identity (8.1) allows us to express any of the basis functions in $E = \{e^{i\omega kt}\}_{k\in\mathbb{Z}}$ in terms of trigonometric functions of the form $\cos(\omega kt)$ and $\sin(\omega kt)$; see, for example, (8.16) and (8.20). Moreover, Euler's identity can be inverted to express sine and cosine in terms of complex exponentials (see (8.3)). The details of the equivalence are given in this section.

Throughout this section, we assume that functions are contained in the space $L^2([0, T]; \mathbb{R})$ of real-valued functions satisfying

$$\int_0^T |f(t)|^2 \, dt < \infty.$$

But for convenience we use the weighted inner product

$$\langle f, g \rangle = \frac{2}{T} \int_0^T f(t) g(t) \, dt, \tag{8.24}$$

which has a different scaling coefficient than the inner product used in the complex-exponential Fourier series (8.10).

8.3.1 Formulation

Theorem 8.3.1. *The set*

$$\left\{ \frac{1}{\sqrt{2}}, \cos(\omega t), \sin(\omega t), \ldots, \cos(\omega kt), \sin(\omega kt), \ldots \right\} \tag{8.25}$$

is orthonormal in $L^2([0, T]; \mathbb{R})$. Here the set (8.25) includes only positive integer values of k.

Proof. It is straightforward to check that the first vector has unit length, and the inner product of the first vector with any of the others is zero. The other inner products all follow from the trigonometric identities (8.5), which give the following integrals for $k, \ell \in \mathbb{Z}^+$:

$$\begin{aligned}
\frac{2}{T}\int_0^T \sin(\omega k t)\cos(\omega \ell t)\,dt &= \frac{1}{T}\int_0^T \left[\sin(\omega(k+\ell)t) + \sin(\omega(k-\ell)t)\right] dt = 0,\\
\frac{2}{T}\int_0^T \cos(\omega k t)\cos(\omega \ell t)dt &= \frac{1}{T}\int_0^T \left[\cos(\omega(k+\ell)t) + \cos(\omega(k-\ell)t)\right] dt = \delta_{k\ell},\\
\frac{2}{T}\int_0^T \sin(\omega k t)\sin(\omega \ell t)dt &= \frac{1}{T}\int_0^T \left[\cos(\omega(k-\ell)t) - \cos(\omega(k+\ell)t)\right] dt = \delta_{k\ell}. \qquad \square
\end{aligned}$$

Corollary 8.3.2. *If f can be expressed in the form*

$$f(t) = \frac{a_0}{\sqrt{2}} + \sum_{k=1}^{\infty} \left(a_k \cos(\omega k t) + b_k \sin(\omega k t)\right), \tag{8.26}$$

then the coefficients satisfy the following relations:

$$a_k = \langle \cos(\omega k t), f\rangle = \frac{2}{T}\int_0^T f(t)\cos(\omega k t)\,dt, \tag{8.27a}$$

$$b_k = \langle \sin(\omega k t), f\rangle = \frac{2}{T}\int_0^T f(t)\sin(\omega k t)\,dt, \tag{8.27b}$$

for $k \in \mathbb{Z}^+$, and

$$a_0 = \left\langle \frac{1}{\sqrt{2}}, f \right\rangle = \frac{\sqrt{2}}{T}\int_0^T f(t)\,dt. \tag{8.28}$$

As with the complex-exponential case, Corollary 8.3.2 inspires the following definition of the Fourier series of a given function f as the projection of f onto the span of the basis vectors.

Definition 8.3.3. *Given $f \in L^2([0,T];\mathbb{R})$, we define the nth* truncated Fourier series *(also called the nth partial sum of the Fourier series) to be*

$$S_n[f](t) = \frac{a_0}{\sqrt{2}} + \sum_{k=1}^{n} \left(a_k \cos(\omega k t) + b_k \sin(\omega k t)\right), \tag{8.29}$$

where each a_k and b_k satisfy (8.27) *and* (8.28)*. Taking the limit as $n \to \infty$ gives the* complete Fourier series

$$S[f](t) = \frac{a_0}{\sqrt{2}} + \sum_{k=1}^{\infty} \left(a_k \cos(\omega k t) + b_k \sin(\omega k t)\right). \tag{8.30}$$

Remark 8.3.4. As with the complex-exponential case, the truncated and complete Fourier series defined here are also the orthogonal projections of f onto the corresponding truncated and complete subspaces of trigonometric functions.

Remark 8.3.5. Note that the constant term in (8.30) satisfies

$$\frac{a_0}{\sqrt{2}} = \frac{1}{T}\int_0^T f(t)\,dt,$$

which is the average value of f on the interval $[0, T]$. Some authors write $\langle f \rangle$ for the average of f and thus express (8.30) as

$$S[f](t) = \langle f \rangle + \sum_{k=1}^{\infty} \left(a_k \cos(\omega k t) + b_k \sin(\omega k t)\right). \tag{8.31}$$

This notation is useful because adding or subtracting a constant to the function only changes its average, not the terms which oscillate at a certain frequency.

Example 8.3.6. Following Example 8.2.6, consider the function $f(t) = t$ defined on $[0, 2\pi]$. We integrate to find the Fourier coefficients. It is straightforward to check that $\langle f \rangle = \pi$. For $k \in \mathbb{Z}^+$, we can show

$$a_k = \frac{1}{\pi}\int_0^{2\pi} t\cos(kt)\,dt = 0, \qquad b_k = \frac{1}{\pi}\int_0^{2\pi} t\sin(kt)\,dt = -\frac{2}{k}.$$

Thus

$$S[f](t) = \pi - 2\sum_{k=1}^{\infty} \frac{\sin kt}{k},$$

which agrees with (8.16).

Example 8.3.7. Following Example 8.2.10, let $f : [0, 2\pi] \to \mathbb{R}$ satisfy (8.18). We integrate to find the Fourier coefficients. It is straightforward to check that that $\langle f \rangle = \frac{1}{2}$. For $k \in \mathbb{Z}^+$, we have

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)\,dt = \frac{1}{\pi}\int_0^{\pi} \cos(kt)\,dt = 0,$$
$$b_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(kt)\,dt = \frac{1}{\pi}\int_0^{\pi} \sin(kt)\,dt = \begin{cases} \frac{2}{\pi k} & \text{if } k \text{ is odd,} \\ 0 & \text{if } k \text{ is even.} \end{cases}$$

Thus

$$S[f](t) = \frac{1}{2} + \frac{2}{\pi}\sum_{k \text{ odd}} \frac{\sin(kt)}{k} = \frac{1}{2} + \frac{2}{\pi}\sum_{k=1}^{\infty} \frac{\sin\left((2k-1)t\right)}{2k-1},$$

which agrees with (8.20).

8.3.2 Equivalency

We have developed two conventions for the Fourier series. We now show that the conventions are equivalent by using Euler's identity (8.1), which states that

$$e^{\pm i\omega kt} = \cos(\omega kt) \pm i\sin(\omega kt) \tag{8.32}$$

for all $k \in \mathbb{N}$, $t \in \mathbb{R}$. This means that any complex-exponential Fourier series can be written in terms of $\sin(\omega kt)$ and $\cos(\omega kt)$. Conversely, by inverting Euler's identity (see (8.3)), we can also write a trigonometric Fourier series in terms of complex exponentials via

$$\cos(\omega kt) = \frac{e^{i\omega kt} + e^{-i\omega kt}}{2} \qquad \text{and} \qquad \sin(\omega kt) = \frac{e^{i\omega kt} - e^{-i\omega kt}}{2i}. \tag{8.33}$$

Theorem 8.3.8. *For $f \in L^2([0,T];\mathbb{F})$ the two conventions for Fourier series are equivalent. In particular, we have*

$$\sum_{k=-\infty}^{\infty} c_k e^{i\omega kt} = \langle f \rangle + \sum_{k=1}^{\infty} a_k \cos(\omega kt) + b_k \sin(\omega kt).$$

Proof. Note that

$$\begin{aligned}
\sum_{k=-\infty}^{\infty} c_k e^{i\omega kt} &= c_0 + \sum_{k=1}^{\infty} c_k e^{i\omega kt} + \sum_{k=1}^{\infty} c_{-k} e^{-i\omega kt} \\
&= c_0 + \sum_{k=1}^{\infty} c_k(\cos(\omega kt) + i\sin(\omega kt)) + c_{-k}(\cos(\omega kt) - i\sin(\omega kt)) \\
&= c_0 + \sum_{k=1}^{\infty} (c_k + c_{-k})\cos(\omega kt) + i(c_k - c_{-k})\sin(\omega kt).
\end{aligned}$$

Thus it suffices to show that

$$c_0 = \langle f \rangle, \tag{8.34a}$$
$$c_k + c_{-k} = a_k, \tag{8.34b}$$
$$i(c_k - c_{-k}) = b_k. \tag{8.34c}$$

The verification of (8.34a) is left to the reader. To prove (8.34b), we have

$$\begin{aligned}
c_k + c_{-k} &= \frac{1}{T}\int_0^T f(t)(e^{-i\omega kt} + e^{i\omega kt})\,dt = \frac{2}{T}\int_0^T f(t)\frac{e^{i\omega kt} + e^{-i\omega kt}}{2}\,dt \\
&= \frac{2}{T}\int_0^T f(t)\cos(\omega kt)\,dt = a_k.
\end{aligned}$$

To prove (8.34c), we have

$$\begin{aligned}
i(c_k - c_{-k}) &= \frac{i}{T}\int_0^T f(t)(e^{-i\omega kt} - e^{i\omega kt})\,dt = \frac{2}{T}\int_0^T f(t)\frac{e^{i\omega kt} - e^{-i\omega kt}}{2i}\,dt \\
&= \frac{2}{T}\int_0^T f(t)\sin(\omega kt)\,dt = b_k. \qquad \square
\end{aligned}$$

8.4 Convergence of Fourier Series

In this section we prove Theorem 8.2.16, which gives conditions for the convergence of Fourier series. To do so, we first prove the Riemann–Lebesgue theorem, which states that for a sufficiently well-behaved function f, the Fourier coefficients converge to zero as $k \to \pm\infty$. We also develop the theory of Dirichlet kernels, which provides a useful representation of Fourier series. Combining the Riemann–Lebesgue theorem with the theory of Dirichlet kernels gives a nice proof of Theorem 8.2.16.

Throughout this section we assume $T = 2\pi$ and thus $\omega = 1$, noting that this causes no loss of generality because we can always rescale the domain.

8.4.1 The Riemann–Lebesgue Theorem

There are several versions of the Riemann–Lebesgue theorem, but they all essentially say that the Fourier coefficients of a given well-behaved function f converge to zero in the high-frequency limit. We provide two such versions (Lemma 8.4.1 and Theorem 8.4.2).

Lemma 8.4.1 (Riemann–Lebesgue). *If $f \in L^2([0, 2\pi]; \mathbb{F})$, then the coefficients c_k of the complex-exponential Fourier series of f converge to zero as $k \to \pm\infty$. Moreover the coefficients a_k and b_k of the trigonometric Fourier series of f converge to zero as $k \to \infty$.*

Proof. By Theorem 8.2.1, the set $E = \{e^{ikt}\}_{k \in \mathbb{Z}}$ is orthonormal in $L^2([0, 2\pi]; \mathbb{F})$. Hence, by the Pythagorean theorem (Volume 1, Theorem 3.2.9), the coefficients satisfy

$$\sum_{k=-n}^{n} |c_k|^2 = \|S_n[f]\|^2 \leq \|S_n[f]\|^2 + \|f - S_n[f]\|^2 = \|f\|^2 < \infty,$$

where the norm $\|\cdot\|$ is given in (8.9). Since this holds for all $n \in \mathbb{N}$, we have

$$\sum_{k=-\infty}^{\infty} |c_k|^2 \leq \|f\|^2 < \infty,$$

which implies $|c_k|^2 \to 0$ as $k \to \pm\infty$. Moreover, $\max\{|a_k|, |b_k|\} \leq |c_k| \to 0$ as $k \to \infty$, which gives the desired result. □

Theorem 8.4.2 (Riemann–Lebesgue). *If $f \in L^2([a, b]; \mathbb{F})$ and $\phi \in \mathbb{R}$, then*

$$\lim_{k \to \infty} \int_a^b f(t) e^{i(kt + \phi)} \, dt = 0. \tag{8.35}$$

Proof. First assume $b - a \leq \pi$; otherwise split the interval $[a, b]$ into subintervals each of length at most π and proceed with each integral separately. Then apply a change of variables corresponding to a translation of $[a, b]$ by some integer multiple of π so that the resulting interval is a subset of $[0, 2\pi]$; note that the magnitude of the integral does not change when doing so. Finally, extend f to the interval $[0, 2\pi]$

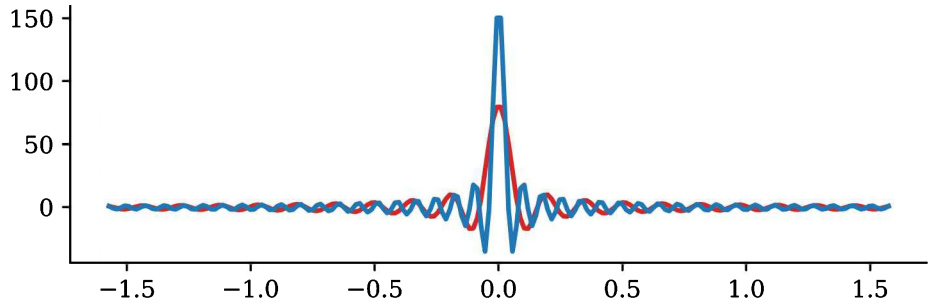


Figure 8.8. *A plot of the Dirichlet kernel D_n for $n = 40$ (red) and $n = 80$ (blue).*

by setting it to zero outside of the interval $[a, b]$. All together, this shows we can assume, without loss of generality, that $f \in L^2([0, 2\pi]; \mathbb{F})$.

By the lemma, the coefficients satisfy $\lim_{k\to\pm\infty} |c_k| = 0$. Thus as $k \to \infty$, we have

$$\left| \int_0^{2\pi} f(t) e^{i(kt+\phi)}\, dt \right| = |e^{i\phi}| \left| \int_0^{2\pi} f(t) e^{ikt}\, dt \right| = 2\pi |c_{-k}| \to 0. \quad \square$$

8.4.2 The Dirichlet Kernel

We define the Dirichlet kernel and prove several lemmata needed to prove the convergence of Fourier series.

Definition 8.4.3. *For $n \in \mathbb{N}$, the Dirichlet kernel D_n is the function*

$$D_n(t) = \sum_{k=-n}^{n} e^{ikt}. \tag{8.36}$$

Lemma 8.4.4. *For $n \in \mathbb{N}$ and $t \in \mathbb{R}$, the Dirichlet kernel takes the form*

$$D_n(t) = 1 + 2 \sum_{k=1}^{n} \cos(kt). \tag{8.37}$$

It follows that D_n is real valued.

Proof. This follows by pairing up the positive and negative exponentials in (8.36) and using (8.3). $\square$

Lemma 8.4.5. *Let $f \in L^2([0, 2\pi]; \mathbb{F})$. The truncated Fourier series of f can be rewritten as*

$$S_n[f](t) = \frac{1}{2\pi} \int_0^{2\pi} f(s) D_n(t - s)\, ds. \tag{8.38}$$

Proof. We have

$$\begin{aligned}\frac{1}{2\pi}\int_0^{2\pi} f(s)D_n(t-s)\,ds &= \frac{1}{2\pi}\int_0^{2\pi} f(s)\left(\sum_{k=-n}^{n} e^{ik(t-s)}\right)ds \\ &= \sum_{k=-n}^{n}\left(\frac{1}{2\pi}\int_0^{2\pi} f(s)e^{-iks}\,ds\right)e^{ikt} \\ &= \sum_{k=-n}^{n} c_k e^{ikt} = S_n[f](t). \quad \square\end{aligned}$$

Lemma 8.4.6. *For $n \in \mathbb{N}$, the Dirichlet kernel D_n satisfies the following:*

(i) *D_n has area 2π on the interval $[0, 2\pi]$; that is,*

$$\int_0^{2\pi} D_n(t)\,dt = 2\pi.$$

(ii) *D_n is 2π-periodic; that is, $D_n(t) = D_n(t + 2\pi)$ for all $t \in \mathbb{R}$.*

(iii) *D_n is even; that is, $D_n(-t) = D_n(t)$ for all $t \in \mathbb{R}$.*

Proof. These all follow directly from Lemma 8.4.4. $\square$

Remark 8.4.7. Combining the results in Lemma 8.4.6 gives

$$2\int_0^{\pi} D_n(t)\,dt = 2\pi.$$

Theorem 8.4.8. *Let $f \in L^2([0, 2\pi]; \mathbb{F})$. The truncated Fourier series of f can be rewritten as*

$$S_n[f](t) = \frac{1}{2\pi}\int_0^{\pi} (f(t+s) + f(t-s))D_n(s)\,ds, \tag{8.39}$$

where we extend f on both sides of the interval so that (8.39) is well defined; in particular, we assume $f(t) = f(t - 2\pi)$ for all $t > 2\pi$ and $f(t) = f(t + 2\pi)$ for all $t < 0$.

Proof. Adding the identities in Exercise 8.19 shows that

$$S_n[f](t) = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left(\frac{f(t+s) + f(t-s)}{2}\right)D_n(s)\,ds.$$

Since the integrand is even, as a function of s, we can halve the interval and double its value giving (8.39). $\square$

We conclude the discussion of the Dirichlet kernel with a useful identity that we use in the proof of Theorem 8.2.16 below.

Lemma 8.4.9. *When $t = 2\pi k$ for some $k \in \mathbb{Z}$ we have $D_n(t) = 2n+1$. Otherwise, the Dirichlet kernel can also be expressed as*

$$D_n(t) = \frac{e^{(2n+1)it/2} - e^{-(2n+1)it/2}}{2i\sin(t/2)} = \frac{\sin\left((2n+1)t/2\right)}{\sin(t/2)}. \tag{8.40}$$

Proof. The proof is Exercise 8.20. □

Remark 8.4.10. We can use (8.40) to plot the Dirichlet kernel D_n for varying n. In Figure 8.8, we see what it looks like for $n = 40$ and $n = 80$. For increasing values of n, we can see that $D_n(t)$ takes the value of $2n+1$ at $t = 0$ and gets closer to zero for nonzero values of t.

8.4.3 Proof of Theorem 8.2.16

Let $t \in [0, 2\pi]$ be fixed. By Remark 8.4.7, the right and left limits of t satisfy

$$\pi f(t^+) = f(t^+)\int_0^\pi D_n(s)\,ds \quad \text{and} \quad \pi f(t^-) = f(t^-)\int_0^\pi D_n(s)\,ds,$$

with the edge cases $f(0^-)$ and $f(2\pi^+)$ defined as $f(2\pi)$ and $f(0)$, respectively. By Theorem 8.4.8, we have

$$\begin{aligned} &2\pi\left(S_n[f](t) - \frac{f(t^+) + f(t^-)}{2}\right) \\ &\quad = \int_0^\pi (f(t+s) + f(t-s))D_n(s)\,ds - \int_0^\pi (f(t^+) + f(t^-))D_n(s)\,ds \\ &\quad = \int_0^\pi (f(t+s) - f(t^+))D_n(s)\,ds + \int_0^\pi (f(t-s) - f(t^-))D_n(s)\,ds. \end{aligned}$$

Hence, it suffices to show that for each $\varepsilon > 0$, there exists N such that

$$\left|\int_0^\pi (f(t \pm s) - f(t^\pm))D_n(s)\,ds\right| < \varepsilon \tag{8.41}$$

whenever $n > N$.

Let L be the Lipschitz constant for f. Given $\varepsilon > 0$, choose $\delta < \min\left(\frac{\varepsilon}{\pi L}, \pi\right)$ so that f is continuous on $[t-\delta, t)$ and $(t, t+\delta]$, respectively. Hence

$$|f(t \pm s) - f(t^\pm)| \leq Ls \tag{8.42}$$

whenever $0 < s \leq \delta$. Since the sine function is concave on $s \in [0, \pi]$, we have

$$\frac{|s|}{\pi} < \left|\sin\left(\frac{s}{2}\right)\right| \tag{8.43}$$

for all $s \in [-\pi, \pi]$. Hence, for $0 < s \leq \delta$, (8.40) gives

$$|D_n(s)| = \left|\frac{\sin\left((2n+1)s/2\right)}{\sin(s/2)}\right| \leq \frac{\pi}{2s}.$$

Thus combining (8.42) and (8.43) yields

$$\left| \int_0^\delta (f(t \pm s) - f(t^\pm)) D_n(s)\, ds \right| \leq \int_0^\delta Ls \frac{\pi}{2s}\, ds \leq \frac{\pi L \delta}{2} < \frac{\varepsilon}{2}.$$

Also, (8.40) and Theorem 8.4.2 imply that for sufficiently large n we have

$$\left| \int_\delta^\pi (f(t \pm s) - f(t^\pm)) D_n(s)\, ds \right| = \left| \int_\delta^\pi g_\pm(s)(e^{(n+\frac{1}{2})it} - e^{-(n+\frac{1}{2})it})\, ds \right| < \frac{\varepsilon}{2},$$

where

$$g_\pm(s) = \frac{f(t \pm s) - f(t^\pm)}{2i \sin(s/2)} \qquad \text{for } s \in [\delta, \pi],$$

which is piecewise Lipschitz on $[\delta, \pi]$ and therefore in L^2. Combining the two bounds implies (8.41), which concludes the proof.

8.5 The Discrete Fourier Transform

Sampling a signal corresponds to observing a continuous-time function at (discrete) equally spaced points in time. In nature, most signals are in continuous time. In technological applications, most signals are in discrete time. Therefore the interplay between nature and technology requires going back and forth between continuous-time and discrete-time signals.

Given a discrete-time signal, we can compute its *discrete Fourier transform (DFT)*, which gives the Fourier series of the discrete-time signal, or rather the Fourier series of a continuous-time function that, when sampled, gives the same discrete-time signal. Of course there are infinitely many continuous-time functions that yield the same discrete-time sample, so we provide the one that is most efficient, in a sense to be described later. In short, the DFT provides a way to transform a discrete-time signal into a Fourier series, thus providing a powerful way to analyze, filter, and compress the sample.

In addition to discussing the DFT, we also describe a fast way to compute it. Given a signal of n data points, the naïve DFT algorithm takes $O(n^2)$ time. However, the DFT can be computed very quickly using an algorithm called the *fast Fourier transform* (FFT), which requires only $O(n \log n)$ time. This difference is substantial when n is large.

8.5.1 Sampled Functions and the Discrete Inner Product

Definition 8.5.1. *Let $f \in L^2([0,T];\mathbb{F})$ and $n \in \mathbb{Z}^+$. A* sample *of f is an n-tuple*

$$\mathbf{f} = (f_0, f_1, \ldots, f_{n-1}) = (f(t_0), \ldots, f(t_{n-1})) \in \mathbb{F}^n, \tag{8.44}$$

where the points $0 = t_0 < \cdots < t_n = T$ are equally spaced on $[0,T]$ and each t_k satisfies $t_k = k\Delta_n$ for $\Delta_n = T/n$. The map $\Phi_n : L^2([0,T];\mathbb{F}) \to \mathbb{F}^n$ that takes f to $(f(t_0), \ldots, f(t_{n-1}))$ is called the sampling operator *with sample time Δ_n.*

Remark 8.5.2. It is straightforward to check that Φ_n is a linear transformation.

Definition 8.5.3. *The* discrete inner product *on the space* $\mathbb{F}^n$ *is given by*

$$\langle \mathbf{f}, \mathbf{g} \rangle_n = \frac{1}{n} \sum_{k=0}^{n-1} \overline{f_k} g_k, \tag{8.45}$$

which is $\frac{1}{n}$ *times the standard inner product on* $\mathbb{F}^n$*; that is,* $\langle \mathbf{f}, \mathbf{g} \rangle_n = \frac{1}{n} \overline{\mathbf{f}}^{\mathsf{T}} \mathbf{g} = \frac{1}{n} \langle \mathbf{f}, \mathbf{g} \rangle$.

Remark 8.5.4. One nice property of the discrete inner product (8.45) is that it approximates the continuous inner product

$$\langle f, g \rangle = \frac{1}{T} \int_0^T \overline{f(t)} g(t)\, dt$$

if n is large and the functions f and g are sufficiently well behaved. Specifically, we have

$$\frac{1}{T} \int_0^T \overline{f(t)} g(t)\, dt \approx \frac{1}{T} \sum_{\ell=0}^{n-1} \overline{f(t_\ell)} g(t_\ell) \frac{T}{n} = \frac{1}{n} \sum_{\ell=0}^{n-1} \overline{f(t_\ell)} g(t_\ell) = \langle \Phi_n(f), \Phi_n(g) \rangle_n .$$

8.5.2 The Discrete Fourier Transform

From Theorem 8.2.1 we know that the set $\{e^{i\omega k t}\}_{k=0}^{n-1}$ is orthogonal in $L^2([0,T];\mathbb{F})$. Correspondingly, the samples of the same basis functions $\{\Phi_n(e^{i\omega k t})\}_{k=0}^{n-1}$ form an orthonormal basis in $\mathbb{F}^n$ with the discrete inner product (8.45).

Proposition 8.5.5. *Let* $n \in \mathbb{Z}^+$ *be fixed. For each* $k \in \mathbb{Z}_n$ *(see Definition 1.9.7), let*

$$\Phi_n(e^{i\omega k t}) = \mathbf{w}_n^{(k)} = (\omega_n^0, \omega_n^k, \ldots, \omega_n^{(n-1)k}), \tag{8.46}$$

where $\omega_n = \exp(2\pi i/n)$ *is a primitive* n*th root of unity (see Section 8.1.3 for details on the roots of unity). The set* $\{\mathbf{w}_n^{(k)}\}_{k=0}^{n-1}$ *is orthonormal in* $\mathbb{C}^n$ *with the discrete inner product* (8.45) *and thus forms a basis for* $\mathbb{C}^n$*. We call the set* $\{\mathbf{w}_n^{(k)}\}_{k=0}^{n-1}$ *the* discrete Fourier basis *on* $\mathbb{C}^n$.

Proof. It suffices to show that $\{\mathbf{w}_n^{(k)}\}_{k=0}^{n-1}$ is orthonormal. For any $j, k \in \mathbb{Z}_n$ we have

$$\left\langle \mathbf{w}_n^{(k)}, \mathbf{w}_n^{(j)} \right\rangle_n = \frac{1}{n} \sum_{\ell=0}^{n-1} \omega_n^{-k\ell} \omega_n^{j\ell} = \frac{1}{n} \sum_{\ell=0}^{n-1} \omega_n^{(j-k)\ell} = \delta_{jk} = \begin{cases} 0 & \text{if } j \not\equiv k \pmod{n}, \\ 1 & \text{if } j \equiv k \pmod{n}. \end{cases}$$

The last equality follows from Proposition 8.1.6. ∎

Many good properties of Fourier series arise from the fact that they express functions in terms of an orthonormal basis. In the discrete setting, we get these same benefits since the discrete Fourier basis $\{\mathbf{w}_n^{(k)}\}_{k=0}^{n-1}$ is orthonormal.

Definition 8.5.6. *Let* $\mathbf{f} = (f_0, \dots, f_{n-1}) \in \mathbb{F}^n$. *The* discrete Fourier transform (DFT) *of* $\mathbf{f}$ *is the vector* $\widehat{\mathbf{f}} \in \mathbb{C}^n$ *of the coefficients of* $\mathbf{f}$ *expressed in terms of the discrete Fourier basis. In other words,* $\widehat{\mathbf{f}} = (\hat{f}_0, \dots, \hat{f}_{n-1})$ *satisfies*

$$\mathbf{f} = (f_0, \dots, f_{n-1}) = \sum_{k=0}^{n-1} \hat{f}_k \mathbf{w}_n^{(k)}.$$

The components can be found via the relation

$$\hat{f}_k = \left\langle \mathbf{w}_n^{(k)}, \mathbf{f} \right\rangle_n = \frac{1}{n} \sum_{\ell=0}^{n-1} \omega_n^{-k\ell} f_\ell. \tag{8.47}$$

Remark 8.5.7. Although the DFT is defined in terms of a vector $\mathbf{f} \in \mathbb{C}^n$, we often talk about the DFT of a function f, by which we mean the DFT of the sampled function $\mathbf{f} = \Phi_n(f)$.

The DFT $\mathbf{f} \mapsto \widehat{\mathbf{f}}$, given by (8.47), is a linear transformation from $\mathbb{C}^n$ to $\mathbb{C}^n$. Because it is linear, we can express it in matrix notation. From here on out, we write $\mathbf{f}$ and $\widehat{\mathbf{f}}$ as vectors. The matrix W_n representing the DFT, with $\widehat{\mathbf{f}} = W_n \mathbf{f}$, is given by

$$W_n = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)^2} \end{bmatrix}. \tag{8.48}$$

For an implementation of the DFT, see Algorithm 8.1.

```
import numpy as np  # module for efficient linear algebra
def DFT(f):
    """ Compute the discrete Fourier Transform of
    the 1D array f."""

    n = len(f)
    m = np.arange(n).reshape(n,1)
    W = np.exp((-2j * np.pi/n) * m @ m.T)
    return W @ f / n
```

Algorithm 8.1. *The DFT algorithm, which amounts to computing* $\widehat{\mathbf{f}} = W_n \mathbf{f}$, *where the matrix* W_n, *is given in* (8.48). *Note that* `@` *is the NumPy syntax for matrix multiplication and* `-2j` *denotes the complex number* $-2i$. *To compute the DFT of the vector in Example 8.5.8 call* `DFT(np.array([1,6,2,4]))`.

Example 8.5.8. If $\mathbf{f} = (1, 6, 2, 4)$, then $\omega_4 = e^{2\pi i/4} = i$ and

$$W_4 = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & \omega_4^{-2} & \omega_4^{-3} \\ 1 & \omega_4^{-2} & \omega_4^{-4} & \omega_4^{-6} \\ 1 & \omega_4^{-3} & \omega_4^{-6} & \omega_4^{-9} \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}.$$

Hence, $\widehat{\mathbf{f}} = W_4\mathbf{f} = \frac{1}{4}(13, -1 - 2i, -7, -1 + 2i)$.

Proposition 8.5.9. *The matrix W_n is symmetric, that is, $W_n^{\mathsf{T}} = W_n$.*

Proof. The (j, k) entry of the matrix W_n (indexed from 0 to $n - 1$) is given by $\frac{1}{n}\omega_n^{-jk}$. This is the same as the (k, j) entry. Thus W_n is symmetric. □

Theorem 8.5.10. *The matrix $\sqrt{n}W_n$ is orthonormal, that is, $nW_n^{\mathsf{H}}W_n = I$, where W_n^{H} is the Hermitian (conjugate transpose) of W_n.*

Proof. Noting that $\overline{\omega_n^{-k\ell}} = \omega_n^{k\ell}$, we compute

$$(nW_n^{\mathsf{H}}W_n)_{km} = \frac{1}{n}\sum_{\ell=0}^{n-1}\omega_n^{k\ell}\omega_n^{-\ell m} = \frac{1}{n}\sum_{\ell=0}^{n-1}\omega_n^{(k-m)\ell} = \delta_{km} = \begin{cases} 0 & \text{if } k \not\equiv m \pmod{n}, \\ 1 & \text{if } k \equiv m \pmod{n}, \end{cases}$$

where the penultimate equality follows from (8.7). □

Example 8.5.11. If $\mathbf{f} = (1, 1, \ldots, 1)$, then the kth component of the DFT is given by

$$\widehat{f}_k = \frac{1}{n}\sum_{\ell=0}^{n-1}\omega_n^{-k\ell} = \delta_{1k} = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{if } k \in \{1, 2, \ldots, n-1\}. \end{cases}$$

In other words, $\widehat{\mathbf{f}} = \mathbf{e}_0$, which is the first standard basis element for $\mathbb{F}^n$, counting from zero. This makes sense, since the signal is constant and the first basis vector corresponds to the average.

Example 8.5.12. Let $\mathbf{f} = (1, -1, 1, -1, \ldots, -1) \in \mathbb{F}^n$, where $n = 2m$, that is, n is even. Note that $f_\ell = (-1)^\ell$ and $-1 = e^{i\pi} = \omega_n^m$. Thus the kth component of the DFT is given by

$$\widehat{f}_k = \frac{1}{n}\sum_{\ell=0}^{n-1}(-1)^\ell\omega_n^{-k\ell} = \delta_{km} = \frac{1}{n}\sum_{\ell=0}^{n-1}\omega_n^{(m-k)\ell} = \begin{cases} 1 & \text{if } k = m, \\ 0 & \text{if } k \neq m. \end{cases}$$

In other words, $\widehat{\mathbf{f}} = \mathbf{e}_m$, the mth standard basis element.

Example 8.5.13. Let $\mathbf{f} = \mathbf{w}_n^{(m)} = (\omega_n^0, \omega_n^m, \omega_n^{2m}, \dots, \omega_n^{(n-1)m})$ for some fixed $m \in \mathbb{Z}$. The kth component of the DFT is

$$\widehat{f}_k = \frac{1}{n}\sum_{\ell=0}^{n-1} \omega_n^{m\ell}\omega_n^{-k\ell} = \frac{1}{n}\sum_{\ell=0}^{n-1} \omega_n^{(m-k)\ell} = \delta_{km} = \begin{cases} 1 & \text{if } k = m, \\ 0 & \text{if } k \neq m. \end{cases}$$

In other words, $\widehat{\mathbf{f}} = \mathbf{e}_m$. In hindsight, this should be obvious, since $\mathbf{w}_n^{(m)}$ is orthogonal to the other basis vectors.

Example 8.5.14. Exercise 8.9 shows how to write a signal as a sum (8.94) of linear oscillators. By sampling a real-valued signal on $[0, T]$ at n equally spaced points and taking its DFT, we can estimate the contribution of each frequency to the signal by plotting the amplitude $2|\widehat{f}_k|$ as a function of the frequency k/T for $0 \leq k \leq \frac{n}{2}$ (assume n is even).

In Figure 8.9, we sample the audio signal (8.23) at $n = 1024$ points. We then compute the DFT of the sample and then plot the amplitude $2|\widehat{f}_k|$ as a function of the frequency k/T (where $T = 0.05$ and so each point in the figure increments by 20 Hz). Note that the peaks are roughly at the frequencies 440 Hz, 660 Hz, and 880 Hz, with amplitudes of approximately 4, 6, and 2, respectively. In other words, we can use the DFT to decompose a signal into a sum of linear oscillators of varying frequencies.

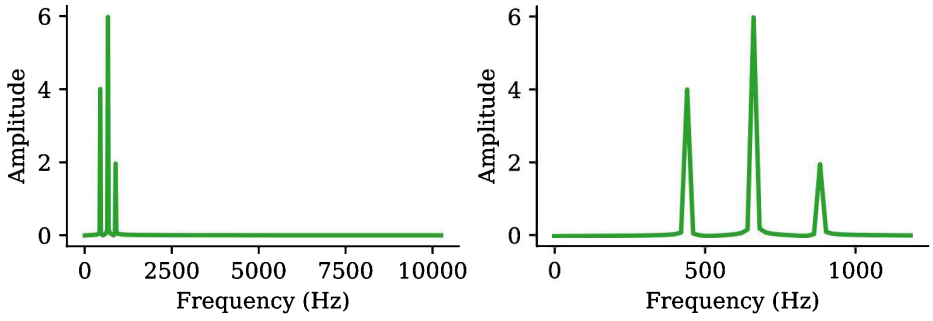


Figure 8.9. *A plot of the Fourier amplitudes of a sample of $n = 1024$ points from the audio signal in (8.23) (left) as described in Example 8.5.14 and a zoomed-in version of the same (right). The amplitudes roughly match those of the signal with approximate frequencies of 440 Hz, 660 Hz, and 880 Hz.*

8.5.3 The Inverse DFT

An immediate corollary of Theorem 8.5.10 is that the DFT matrix W_n has an easily computed inverse, which allows us to transform the Fourier coefficients $\widehat{\mathbf{f}}$ back to the original sample $\mathbf{f}$ by matrix-vector multiplication. This operation is called the *inverse discrete Fourier transform* (IDFT).

Corollary 8.5.15. *The IDFT is given by* $\mathbf{f} = W_n^{-1}\widehat{\mathbf{f}}$, *where*

$$W_n^{-1} = nW_n^{\mathsf{H}} = n\overline{W}_n = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix}. \tag{8.49}$$

Proof. Since $nW_n^{\mathsf{H}}W_n = I$ and W_n is square, we have $W_n^{-1} = nW_n^{\mathsf{H}} = n\overline{W}_n$. The rest of the proof follows from the observation that $\overline{\omega_n^{-k\ell}} = \omega_n^{k\ell}$. ∎

Nota Bene 8.5.16. There are several different conventions for the DFT and IDFT. These correspond to various choices of $\kappa \in \{1, \sqrt{n}, n\}$ and $\sigma \in \{-1, +1\}$ in the sum

$$\widehat{f}_k = \frac{1}{\kappa}\sum_{\ell=0}^{n-1} f(t_\ell)\omega_n^{\sigma k\ell}.$$

In this text we have used the convention of $\kappa = n$ and $\sigma = -1$, and our IDFT corresponds to $\kappa = 1$ and $\sigma = +1$. The reader should be aware of other conventions, especially when using software libraries.

8.5.4 Fast Fourier Transform

Both the DFT and the IDFT can be implemented by matrix-vector multiplication via (8.48) and (8.49), respectively, both of which have a temporal complexity of $O(n^2)$. In this section we show how both of these transforms can be sped up to $O(n \log n)$ by using the *fast Fourier transform* (FFT). This drastic and remarkable improvement allows for very fast real-time computation of the DFT in many settings, including radar, image processing, audio filtering, magnetic resonance imaging (MRI), and many other applications.

Lemma 8.5.17. *Let* $n = 2^m$ *for some* $m \in \mathbb{Z}^+$. *If* $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{F}^n$, *then the DFT* $\widehat{\mathbf{f}} = (\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{n-1}) \in \mathbb{F}^n$ *satisfies*

$$\widehat{f}_k = \frac{1}{2}\left(\frac{1}{n/2}\sum_{j=0}^{n/2-1} f_{2j}\omega_{n/2}^{-jk} + \omega_n^{-k}\frac{1}{n/2}\sum_{j=0}^{n/2-1} f_{2j+1}\omega_{n/2}^{-jk}\right). \tag{8.50}$$

Proof. Separate (8.47) into the even and odd powers of ω_n to get

$$\begin{aligned}\widehat{f}_k = \frac{1}{n}\sum_{\ell=0}^{n-1} f_\ell \omega_n^{-k\ell} &= \frac{1}{n}\sum_{j=0}^{n/2-1} f_{2j}\omega_n^{-k(2j)} + \frac{1}{n}\sum_{j=0}^{n/2-1} f_{2j+1}\omega_n^{-k(2j+1)} \\ &= \frac{1}{2}\left(\frac{1}{n/2}\sum_{j=0}^{n/2-1} f_{2j}\omega_{n/2}^{-jk} + \omega_n^{-k}\frac{1}{n/2}\sum_{j=0}^{n/2-1} f_{2j+1}\omega_{n/2}^{-jk}\right).\end{aligned}$$

The last step uses (8.8) to get $\omega_n^{-2jk} = \omega_{n/2}^{-jk}$. ∎

Let $\mathbf{f}_\mathrm{e}$ denote the vector of length $n/2$ consisting of the even-indexed entries of $\mathbf{f}$, and let $\widehat{\mathbf{f}}_\mathrm{e}$ denote its DFT. Similarly, let $\mathbf{f}_\mathrm{o}$ denote the vector of odd-indexed entries of $\mathbf{f}$ and $\widehat{\mathbf{f}}_\mathrm{o}$ its DFT. If $k < n/2$, then the first sum in (8.50) is exactly the index-k part of $\widehat{\mathbf{f}}_\mathrm{e}$, and the second sum is exactly the index-k part of $\widehat{\mathbf{f}}_\mathrm{o}$. If $k \geq n/2$, then $\omega_{n/2}^k = \omega_{n/2}^{k-n/2}$, and so each sum corresponds to the entry with index $k - n/2$ from $\widehat{\mathbf{f}}_\mathrm{e}$ or $\widehat{\mathbf{f}}_\mathrm{o}$, respectively. Thus we can construct the full DFT of $\mathbf{f}$ recursively, by computing $\widehat{\mathbf{f}}_\mathrm{e}$ and $\widehat{\mathbf{f}}_\mathrm{o}$ rescaling the odd part by ω_n^{-k} (often called the *twiddle factor* in this setting), and summing them.

Theorem 8.5.18 (Fast Fourier Transform). *Let $n = 2^m$ for some $m \in \mathbb{Z}^+$. By applying* (8.50) *recursively, we can compute the DFT in $O(n \log n)$ time.*

Proof. The lemma converts the DFT of a vector of length $n = 2^m$ into the sum of two DFTs of length $n/2$. The second term must be multiplied by a root of unity and then added to the first. The multiplication and sum have temporal complexity $O(1)$ for a single coefficient $\widehat{f}_k$, and since we are computing them for n coefficients, they contribute $O(n)$ to the temporal complexity. If $T(n)$ is the time it takes to compute all n coefficients of the DFT of a vector of length $n = 2^m$, then we have

$$T(n) \leq 2T(n/2) + cn$$

for some constant $c > 0$. By the master theorem (Theorem 1.10.2), it follows that $T(n) \in O(n \log n)$. This recursive approach to computing the DFT is called the *fast Fourier transform.* ∎

Remark 8.5.19. When n is not an exact power of 2, we can pad the signal with extra zeros at the end until its length is a power of 2, and then perform the FFT. The maximum number of zeros required is $n-1$ (when $n = 2^m+1$), and the padded FFT will be run at worst on a sample of length $2n$. Thus the temporal complexity will be at worst $O(2n \log(2n)) = O(n \log n)$.

Remark 8.5.20. Since the IDFT is constructed in a manner almost identical to the DFT, an argument similar to the one above shows that the IDFT also has a fast divide-and-conquer implementation of temporal complexity $O(n \log n)$.

```python
def FFT(f):
    """Perform the FFT algorithm on the numpy array `f`
    """

    n = len(f) # assumed to be a power of 2

    if n <= 4:  # this cutoff to be optimized, also a power of 2
        return DFT(f)
    else:
        f_even = FFT(f[::2]) # FFT of even indexed entries of f
        f_odd = FFT(f[1::2]) # FFT of odd indexed entries of f
        w = np.exp((-2j * np.pi/n) * np.arange(n))
        first_sum = f_even + w[:n//2] * f_odd
        second_sum = f_even + w[n//2:] * f_odd
        return 0.5 * np.concatenate([first_sum, second_sum])
```

Algorithm 8.2. *The FFT algorithm, which computes the DFT recursively using* (8.50)*. At the end of the recursion, for the vectors of length less than some cutoff (here we use* 4*, but that choice should be optimized), the algorithm uses the* `DFT` *code from Algorithm 8.1. As observed after the proof of Lemma 8.5.17, the vectors* $\widehat{\mathbf{f}}_e$ *and* $\widehat{\mathbf{f}}_o$ *are used once for the terms of index* $k < n/2$ *and again for the terms of index* $k \geq n/2$*. Theorem 8.5.18 shows that this algorithm has temporal complexity* $O(n\log(n))$*, which is a big speedup over the complexity* $O(n^2)$ *of the naïve DFT using matrix multiplication.*

Remark 8.5.21. There are very efficient implementations of the FFT that have relatively small leading coefficients on the $n \log n$. Algorithm 8.2 is just an illustration of how to implement the FFT, but it is not especially efficient.

8.5.5 A Foray into Filtering with Fourier

We conclude this section by presenting two simple methods of filtering noise from a signal. Consider the audio signal given in (8.23). If some Gaussian noise is added to the signal (for example, if it is transmitted along a noisy channel), then we get a new signal, as in the top panel of Figure 8.10. Moreover, plotting the amplitudes of the various frequencies shows nonzero coefficients distributed jaggedly, but somewhat uniformly, across all frequencies; see the bottom panel of Figure 8.10. This is because Gaussian noise is uniform across frequencies. We call this *white noise* because in the visual spectrum the color white is a uniform combination of all frequencies.

There are two easy ways to filter the noisy signal in Figure 8.10. In both cases we carry out the filtering by sampling, using DFT, applying a function to the transformed values, and then using the IDFT to get back a filtered signal.

One way to filter the noisy signal is to use the fact that the original signal consisted only of frequencies between 400 and 900 Hz, so we remove all frequencies less than 400 Hz and greater than 900 Hz. This is called a *band-pass filter*, and it is

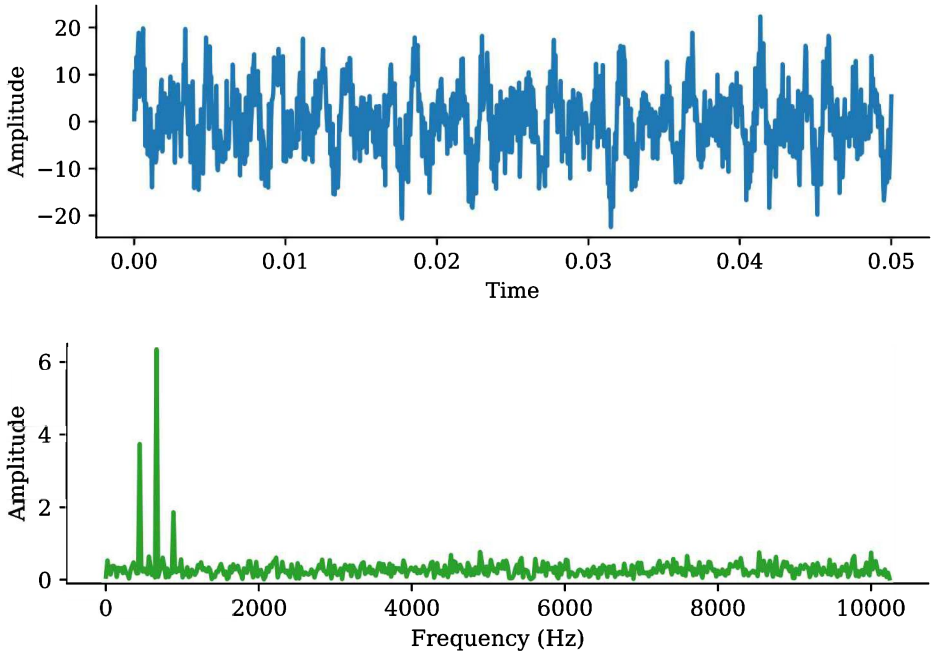


Figure 8.10. *A plot of* (8.23) *with added Gaussian noise. The top panel shows the noisy signal as a function of time (in seconds), and the bottom panel shows the net amplitudes as a function of frequency (the result of applying the DFT).*

accomplished by applying the DFT, setting all the resulting coefficients outside the range of 400 and 900 Hz to zero, and then using the IDFT to construct the filtered signal.

Another way to filter the noisy signal is to remove any frequency that has small amplitude, with the expectation that the noise should have relatively small amplitude relative to the original signal. This is accomplished by applying the DFT, setting all the resulting coefficients with amplitude less than some value to zero (in Figure 8.11 we used $\frac{1}{2}$ as the cutoff), and then applying the IDFT to construct the filtered signal. Both filtered signals are shown in Figure 8.11.

8.6 Convolution

In most modern applications, signals and images are typically represented in discrete time and space as large arrays of numbers indicating an intensity or magnitude at that specific point in space or time. Mathematically we can consider these arrays as vectors in a high-dimensional vector space. This allows for methods from linear algebra and multivariable calculus to be used to analyze and process the data. However, the dimensions used are often too high to be immediately useful.

For example, when dealing with images, slight translations, rotations, or changes in resolution, scale, or color will have no significant perceptual differences to a human, but the vector representations can be wildly different from each other and make it difficult or even impossible to make sense out of the data. One useful

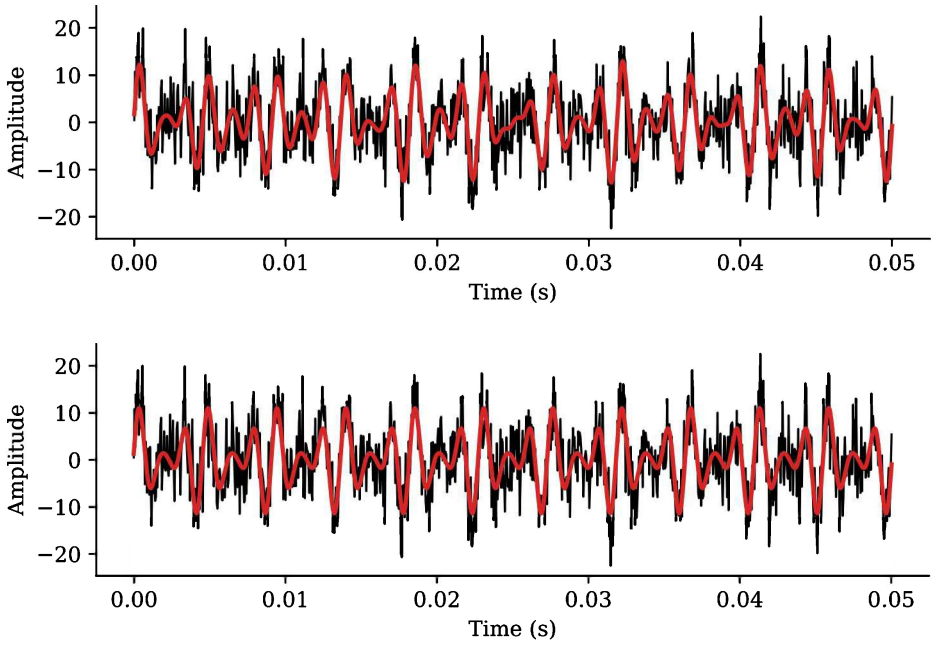


Figure 8.11. *The two filtering methods discussed in Section 8.5.5. The plots show the filtered signal in black and the original, noisy signal in red. The method used for the upper panel is a band-pass filter that removes all frequencies outside of a* 400 *to* 900 *Hz band. The method used for the lower panel is a filter that removes all frequencies with small amplitude (below* $\frac{1}{2}$*).*

approach is to reduce the variation through filtering so that perceptually similar signals and images have similar vector representations.

For example, Section 8.5.5 showed that a low-pass filter can remove noise from a signal. In this section we generalize the notion of a filter using a mathematical construct called *convolution*, which, when applied judiciously, can often help simplify or allow better analysis of the resulting vector. While this doesn't reduce the dimensionality of the vector representation, it can reduce the variability caused by noise as well as provide better ways of understanding how to match signals and images that have undergone translations, rotations, or changes in resolution, scale, or color. Convolutions can average data locally, in a way that reduces variability while preserving perceptual similarity. They can also be used to identify whether a certain feature in the data is present. Moreover, multiple convolutions can be composed in succession to extract features that help identify patterns in data.

8.6.1 Circular Convolution

Throughout this subsection, we assume that signals are periodic with period T and thus correspondingly their samples are periodic with period n. Hence, given a vector $\mathbf{f} = (f_0, f_1, f_2, \ldots, f_{n-1}) \in \mathbb{F}^n$ we assume that $f_{k+n} = f_k$ for each $k \in \mathbb{Z}$. We call such a vector a *periodic vector*.

Definition 8.6.1. *Given periodic vectors* $\mathbf{f}$ *and* $\mathbf{g}$ *in* $\mathbb{F}^n$*, we define their* circular convolution *as the periodic vector* $\mathbf{f} * \mathbf{g} \in \mathbb{F}^n$ *whose* k*th component is given by*

$$(\mathbf{f} * \mathbf{g})_k = \sum_{j=0}^{n-1} f_{k-j} g_j. \tag{8.51}$$

Example 8.6.2. If $\mathbf{f} = (1,3,2,0)$ and $\mathbf{g} = (1,1,0,1)$ are periodic vectors, then

$$\begin{aligned}(\mathbf{f} * \mathbf{g})_0 &= 1 \cdot 1 + 0 \cdot 1 + 2 \cdot 0 + 3 \cdot 1 = 4,\\ (\mathbf{f} * \mathbf{g})_1 &= 3 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 2 \cdot 1 = 6,\\ (\mathbf{f} * \mathbf{g})_2 &= 2 \cdot 1 + 3 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 5,\\ (\mathbf{f} * \mathbf{g})_3 &= 0 \cdot 1 + 2 \cdot 1 + 3 \cdot 0 + 1 \cdot 1 = 3,\end{aligned}$$

and so $\mathbf{f} * \mathbf{g} = (4,6,5,3)$.

Remark 8.6.3. Given periodic vectors $\mathbf{f}, \mathbf{g} \in \mathbb{F}^n$, the circular convolution $\mathbf{f} * \mathbf{g}$ is compatible with the requirement of periodic vectors that $f_{k+n} = f_k$ for each $k \in \mathbb{Z}$. Indeed, for any k we have

$$(\mathbf{f} * \mathbf{g})_{n+k} = \sum_{j=0}^{n-1} f_{n+k-j} g_j = \sum_{j=0}^{n-1} f_{k-j} g_j = (\mathbf{f} * \mathbf{g})_k.$$

The next result lists some basic properties of the convolution operator.

Theorem 8.6.4. *Let* $\mathbf{f}, \mathbf{g}, \mathbf{h} \in \mathbb{F}^n$ *be periodic. For any* $a, b \in \mathbb{F}$*, we have the following properties:*

(i) $\mathbf{f} * \mathbf{g} = \mathbf{g} * \mathbf{f}$.

(ii) $\mathbf{f} * (\mathbf{g} * \mathbf{h}) = (\mathbf{f} * \mathbf{g}) * \mathbf{h}$.

(iii) $(a\mathbf{f} + b\mathbf{g}) * \mathbf{h} = a(\mathbf{f} * \mathbf{h}) + b(\mathbf{g} * \mathbf{h})$.

Proof. First observe that for any periodic vector $\mathbf{f} \in \mathbb{F}^n$, the sum over n consecutive terms is the same no matter where the sum begins:

$$\sum_{i=k}^{n-1+k} f_i = \sum_{i=0}^{n-1} f_i \qquad \forall k \in \mathbb{Z}.$$

To see this, reindex to put all indices in the range $\{0, 1, 2, \ldots, n-1\}$. For example, when $0 \le k \le n-1$ we have

$$\sum_{i=k}^{n-1+k} f_i = \sum_{i=k}^{n-1} f_i + \sum_{i=n}^{n-1+k} f_i = \sum_{i=k}^{n-1} f_i + \sum_{j=0}^{k-1} f_j = \sum_{i=0}^{n-1} f_i.$$

Now we prove each of the parts of the theorem.

(i) Using $m = k - j$ to reindex the sum defining $(\mathbf{f} * \mathbf{g})_k$ (and reversing the order of summation) gives

$$\begin{aligned}(\mathbf{f} * \mathbf{g})_k &= \sum_{j=0}^{n-1} f_{k-j} g_j = \sum_{k-n+1}^{m=k} f_m g_{k-m} \\ &= \sum_{m=0}^{n-1} g_{k-m} f_m = (\mathbf{g} * \mathbf{f})_k.\end{aligned}$$

(ii) Using $m = j - i$ to reindex the sum defining $(\mathbf{g} * \mathbf{h})_j$ gives

$$\begin{aligned}(\mathbf{f} * (\mathbf{g} * \mathbf{h}))_k &= \sum_{j=0}^{n-1} f_{k-j} \left(\sum_{i=0}^{n-1} g_{j-i} h_i \right) \\ &= \sum_{i=0}^{n-1} \sum_{m=-i}^{n-1-i} f_{k-m-i} g_m h_i \\ &= \sum_{i=0}^{n-1} \left(\sum_{m=0}^{n-1} f_{k-i-m} g_m \right) h_i \\ &= ((\mathbf{f} * \mathbf{g}) * \mathbf{h})_k .\end{aligned}$$

(iii) Expanding gives

$$\begin{aligned}((a\mathbf{f} + b\mathbf{g}) * \mathbf{h})_k &= \sum_{j=0}^{n-1} (a f_{k-j} + b g_{k-j}) h_j \\ &= a \sum_{j=0}^{n-1} f_{k-j} h_j + b \sum_{j=0}^{n-1} g_{k-j} h_j \\ &= a(\mathbf{f} * \mathbf{h})_k + b(\mathbf{g} * \mathbf{h})_k. \quad \square\end{aligned}$$

Remark 8.6.5. Properties (i) and (iii) in Theorem 8.6.4 can be combined to show that the convolution is also linear in the second argument. Thus convolution is a *bilinear* operation.

Example 8.6.6. Let $\mathbf{e}_i$ denote the periodic vector whose ith entry is one, and all other entries zero; so the jth component of $\mathbf{e}_i$ is δ_{ij}. Thus

$$(\mathbf{f} * \mathbf{e}_i)_k = \sum_{j=0}^{n-1} f_{k-j} \delta_{ij} = f_{k-i}.$$

In other words, convolving $\mathbf{f}$ with $\mathbf{e}_i$ shifts $\mathbf{f}$ to the right by i entries.

Example 8.6.7. Convolving an input signal $\mathbf{f}$ with a sum $\mathbf{g} = \mathbf{e}_1 - 0.5\mathbf{e}_{20}$ gives

$$\mathbf{f} * \mathbf{g} = \mathbf{f} * \mathbf{e}_1 - 0.5\mathbf{f} * \mathbf{e}_{20}.$$

The term $\mathbf{f} * \mathbf{e}_1$ gives a copy of $\mathbf{f}$ shifted one unit to the right. And the term $-0.5\mathbf{f} * \mathbf{e}_{20}$ gives a copy of $\mathbf{f}$ shifted 20 units to the right, inverted, and scaled to half amplitude. Thus this convolution reproduces $\mathbf{f}$ along with an (inverted) echo of $\mathbf{f}$ of half amplitude. See Figure 8.12.

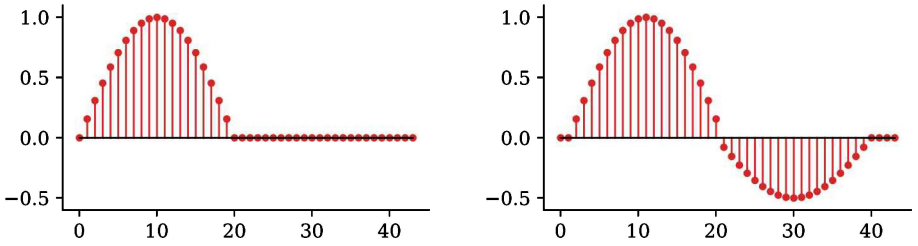


Figure 8.12. *Convolution of an input* $\mathbf{f}$*, consisting of a single hump (left), with a filter* $\mathbf{g} = \mathbf{e}_1 - 0.5\mathbf{e}_{20}$*, as described in Example 8.6.7. The convolution* $\mathbf{f} * \mathbf{g}$ *(left) consists of a copy of* $\mathbf{f}$ *shifted one unit to the right (due to* $\mathbf{e}_1$*) and one more copy of* f *shifted 20 units to the right (due to* $\mathbf{e}_{20}$*), inverted, and scaled to half amplitude.*

8.6.2 The Finite Convolution Theorem

In this subsection we prove the finite convolution theorem, which gives a fast method for computing the convolution using the FFT.

Definition 8.6.8. *For* $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{F}^n$ *and* $\mathbf{g} = (g_0, g_1, \ldots, g_{n-1}) \in \mathbb{F}^n$*, the* Hadamard product *is the componentwise product*

$$\mathbf{f} \odot \mathbf{g} = (f_0 g_0, f_1 g_1, \ldots, f_{n-1} g_{n-1}).$$

Example 8.6.9. If $\mathbf{f} = (1, 3, 2, 0) \in \mathbb{F}^4$ and $\mathbf{g} = (1, 2, 0, 1) \in \mathbb{F}^4$, then

$$\mathbf{f} \odot \mathbf{g} = (1, 6, 0, 0).$$

Theorem 8.6.10 (Finite Convolution Theorem). *If* $\mathbf{f}, \mathbf{g} \in F^n$ *are periodic, then the DFT satisfies the identity*

$$\widehat{(\mathbf{f} * \mathbf{g})} = n\widehat{\mathbf{f}} \odot \widehat{\mathbf{g}}. \tag{8.52}$$

Proof. Assume $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{F}^n$ and $\mathbf{g} = (g_0, g_1, \ldots, g_{n-1}) \in \mathbb{F}^n$. Writing out the kth component of $n\widehat{(\mathbf{f} * \mathbf{g})}$ gives

$$\begin{aligned} n\widehat{(\mathbf{f} * \mathbf{g})}_k &= \sum_{j=0}^{n-1} \omega_n^{-jk} (\mathbf{f} * \mathbf{g})_j = \sum_{j=0}^{n-1} \omega_n^{-jk} \left(\sum_{i=0}^{n-1} f_{j-i} g_i \right) \\ &= \sum_{i=0}^{n-1} \sum_{m=-i}^{n-i-1} \omega_n^{-(m+i)k} f_m g_i = \left(\sum_{m=-i}^{n-i-1} \omega_n^{-mk} f_m \right) \left(\sum_{i=0}^{n-1} \omega_n^{-ik} g_i \right) \\ &= \left(\sum_{m=0}^{n-1} \omega_n^{-mk} f_m \right) \left(\sum_{i=0}^{n-1} \omega_n^{-ik} g_i \right) = (n\widehat{\mathbf{f}})_k (n\widehat{\mathbf{g}})_k. \end{aligned}$$

Hence 8.52 follows. □

Example 8.6.11. Let $\widehat{\mathbf{f}} = (\widehat{f}_0, \ldots, \widehat{f}_k, \ldots, \widehat{f}_{n-1}) \in \mathbb{F}^n$ be the DFT of the vector $\mathbf{f} \in \mathbb{F}^n$. Recall from Example 8.5.13 that the DFT of the kth (periodic) Fourier basis vector $\mathbf{w}_n^{(k)}$ is $\mathbf{e}_k$. Denoting the IDFT by W_n^{-1} and applying Theorem 8.6.10 to the expression $\frac{1}{n}\mathbf{f} * \mathbf{w}_n^{(k)}$ gives

$$\begin{aligned} \frac{1}{n}\mathbf{f} * \mathbf{w}_n^{(k)} &= \frac{1}{n} W_n^{-1}(\widehat{(\mathbf{f} * \mathbf{w}_n^{(k)})}) = W_n^{-1}(\widehat{\mathbf{f}} \odot \widehat{\mathbf{w}_n^{(k)}}) \\ &= W_n^{-1}(\widehat{\mathbf{f}} \odot \mathbf{e}_k) = W_n^{-1}(0, \ldots, 0, \widehat{f}_k, 0 \ldots, 0). \end{aligned}$$

This strips out the component of the kth frequency of $\mathbf{f}$ and discards all the other frequencies. We can use this fact, along with linearity of convolution, to create a filter that removes, attenuates, or amplifies any combination of frequencies in the original signal $\mathbf{f}$. For example, if $\mathbf{g} = \sum_{k \geq m} \mathbf{w}_n^{(k)}$, then convolving with $\mathbf{g}$ keeps all the frequencies greater than or equal to m but removes all the frequencies less than m from $\mathbf{f}$.

8.6.3 Fast Convolution

Taking the IDFT of both sides of (8.52) gives

$$\mathbf{f} * \mathbf{g} = n W_n^{-1} \left(\widehat{\mathbf{f}} \odot \widehat{\mathbf{g}} \right).$$

This gives a fast way to compute a convolution; that is, take the FFT of $\mathbf{f}$ and $\mathbf{g}$, multiply the results componentwise, and then take the inverse FFT. Since each application of the FFT is $O(n \log n)$, and componentwise (Hadamard) multiplication is $O(n)$, the entire calculation is $O(n \log n)$. This is a considerable speedup for convolutions, since the naïve definition (8.51) is $O(n^2)$.

Fast convolution has many applications, including in signal processing, as mentioned above. More examples are given in the computer labs for this volume.

8.6.4 *Linear Convolution

Although circular convolutions are most useful when dealing with Fourier analysis, other variations of convolution are important in many settings. An especially important variant is the *linear convolution*, which we denote by $*'$. Linear convolution maps two vectors $\mathbf{f}, \mathbf{g} \in \mathbb{F}^n$ into a new vector $\mathbf{f} *' \mathbf{g} \in \mathbb{F}^{2n-1}$, given by the rule

$$\begin{aligned}
(\mathbf{f} *' \mathbf{g})_0 &= f_0 g_0, \\
(\mathbf{f} *' \mathbf{g})_1 &= f_1 g_0 + f_0 g_1, \\
&\vdots \\
(\mathbf{f} *' \mathbf{g})_{n-1} &= f_{n-1} g_0 + f_{n-2} g_1 + \cdots + f_0 g_{n-1}, \\
(\mathbf{f} *' \mathbf{g})_n &= f_{n-1} g_1 + \cdots + f_1 g_{n-1}, \\
(\mathbf{f} *' \mathbf{g})_{n+1} &= f_{n-1}\ g_2 + \cdots + f_2 g_{n-1}, \\
&\vdots \\
(\mathbf{f} *' \mathbf{g})_{2n-2} &= f_{n-1} g_{n-1}, \\
(\mathbf{f} *' \mathbf{g})_k &= 0 \qquad \forall k \notin \{0, \ldots, 2n-2\}.
\end{aligned}$$

To compute a linear convolution within the framework of periodic vectors and circular convolution, we can pad the periodic vectors with n zeros before and after the original vectors. That is, set

$$\begin{aligned}
\mathbf{f}' &= (0, 0, \ldots, 0, f_0, \ldots, f_{n-1}, 0, 0, \ldots, 0) \in \mathbb{F}^{3n}, \\
\mathbf{g}' &= (0, 0, \ldots, 0, g_0, \ldots, g_{n-1}, 0, 0, \ldots, 0) \in \mathbb{F}^{3n},
\end{aligned}$$

which gives $(\mathbf{f}' * \mathbf{g}')_k = \sum_{j=0}^{3n-1} f'_{k-j} g'_j$, and then take $(\mathbf{f} *' \mathbf{g})_k = (\mathbf{f}' * \mathbf{g}')_{n+k}$ for $k \in \{0, \ldots, 2n-2\}$.

Similarly, to compute a circular convolution within the framework of the linear convolution, pad the vectors with one more copy, that is, set

$$\begin{aligned}
\mathbf{f}'' &= (f_0, \ldots, f_{n-1}, f_0, \ldots, f_{n-1}) \in \mathbb{F}^{2n}, \\
\mathbf{g}'' &= (g_0, \ldots, g_{n-1}, g_0, \ldots, g_{n-1}) \in \mathbb{F}^{2n},
\end{aligned}$$

which gives $(\mathbf{f}'' *' \mathbf{g}'')_k = \sum_{j=0}^{2n-1} f''_{k-j} g''_j$, and then take $(\mathbf{f} * \mathbf{g})_k = (\mathbf{f}'' *' \mathbf{g}'')_{n+k}$ for $k \in \{0, \ldots, n-1\}$.

8.7 Periodic Sampling Theorem

In this section we address the question of how to reproduce a periodic function from a sample. The FFT and convolution are very powerful tools for understanding and modifying sampled functions. But once all the computations and adjustments have been made to the sample, we need to convert it back to a function, and, as it turns out, there is not a unique way to do this. So the question becomes, What is the best way to reconstruct the signal?

8.7.1 Band-Limited Functions

In most real-world applications, the high-frequency part of a signal is considered noise. For audible signals, crackling and static are high-frequency noises, and it is often considered a good idea to filter all parts of a signal that lie outside of a certain frequency band. This is one example of the common situation where we want to consider only functions that have a Fourier series involving a limited range of frequencies. Such functions are called *band-limited* functions.

Definition 8.7.1. *A function $f \in L^2([0,T];\mathbb{F})$ with Fourier series*

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega k t}$$

is called band limited *if there exists a nonnegative integer ν such that $c_k = 0$ whenever $|k| > \nu$. In this case, the smallest such ν is called the* Nyquist frequency *of f. The* Nyquist rate *of f is twice the Nyquist frequency.*

Example 8.7.2. Consider the function

$$f(t) = 2\sin(6\pi t) + 3\sin(10\pi t + 2) + 5\sin(14\pi t + 4).$$

Using the formulas (8.33) to write the trigonometric functions as exponentials, we have

$$\begin{aligned} f(t) &= \frac{2}{2i}\left(e^{2\pi i 3t} - e^{-2\pi i 3t}\right) + \frac{3}{2i}\left(e^{2\pi i 5t+2i} - e^{-2\pi i 5t-2i}\right) \\ &\quad + \frac{5}{2i}\left(e^{2\pi i 7t+4i} - e^{-2\pi i 7t-4i}\right) \\ &= -\frac{5e^{-4i}}{2i}e^{-2\pi i 7t} - \frac{3e^{-2i}}{2i}e^{-2\pi i 5t} - \frac{1}{i}e^{-2\pi i 3t} + \frac{1}{i}e^{2\pi i 3t} \\ &\quad + \frac{3e^{2i}}{2i}e^{2\pi i 5t} + \frac{5e^{4i}}{2i}e^{2\pi i 7t}. \end{aligned}$$

This exponential Fourier series has $c_{\pm 7} \neq 0$ but all coefficients c_k vanish for $|k| > 7$; thus the Nyquist frequency of f is $\nu = 7$, and the Nyquist rate is $2\nu = 14$.

Application 8.7.3. As mentioned in Section 8.2.2, most humans cannot hear frequencies above 20 kHz. Hence, the difference between an audio signal $f(t)$ and the perturbed signal $f(t) + \sin(\omega \ell t)$, when $\ell > 20{,}000$, is not audible to humans (although your dog might not like it!). Therefore, for audio signal processing, it is common to work with functions having Nyquist frequency of slightly over 20 kHz to cover people with exceptional hearing. This gives a corresponding Nyquist rate of slightly over 40 kHz, and, indeed, the standard sample rate used in audio recordings is $44.1k$ kHz.

8.7.2 Aliasing

Assume that a signal f has Fourier series $S[f](t) = \sum_{-\infty}^{\infty} c_k e^{i\omega kt}$. As before, we sample the interval $[0, T]$ at n equally spaced points $0 = t_0 < t_1 < \cdots < t_{n-1} < T$ with $t_\ell = \frac{\ell T}{n}$. These samples cannot determine the function f uniquely, because the functions $e^{i\omega t}$ and $e^{i\omega(n+1)t}$ both take on the same values at each t_ℓ, that is,

$$e^{i\omega t}\Big|_{t=t_\ell} = e^{2\pi i\ell/n} = e^{2\pi i(\ell+\ell/n)} = e^{2\pi i(n+1)t_\ell} = e^{i\omega(n+1)t}\Big|_{t=t_\ell}. \tag{8.53}$$

The good news is that if the sample comes from a band-limited periodic function, then Theorem 8.7.7 (given below) guarantees the function is uniquely determined by the samples, if there are sufficiently many samples. Moreover, given a sample rate, we can identify a band for which all functions limited to that band can be perfectly and uniquely reconstructed from the samples.

Example 8.7.4. Consider the function $f(t) = c_{-1}e^{-it} + c_0 + c_1 e^{it}$ on $[0, 2\pi]$. Sampling only once on the interval $[0, 2\pi]$, at the point 0, will only determine the value of $f(0) = c_{-1} + c_0 + c_1$. Sampling twice, at 0 and π, will also determine $f(\pi) = -c_{-1} + c_0 - c_1$. With these two values, we can deduce c_0 but not c_{-1} and c_1. Only after sampling three times, at $0, \frac{2\pi}{3}, \frac{4\pi}{3}$, can we deduce the values of all three coefficients. This shouldn't be a surprise since three sample points gives a linear system with three equations and three unknowns.

Remark 8.7.5. Generalizing the previous example, if we consider the family of all functions of the form

$$f(t) = \sum_{k=-n}^{n} c_k e^{i\omega kt}$$

on the domain $[0, T]$, then to uniquely determine the $2n+1$ coefficients $c_{-n}, \ldots, c_n$ requires $2n+1$ samples in order to have at least as many equations as unknowns. This means we need to sample at a rate higher than the Nyquist rate of $2n$ in order to recover the signal. Of course, even if we sample at this higher rate, it is conceivable that there might be linear dependencies among the equations (the matrix representation might not have full rank), but Theorem 8.7.7, below, shows that this is not the case: the Nyquist bound is exactly the right bound, and any sampling rate higher than the Nyquist rate will allow for perfect and unique reconstruction.

Definition 8.7.6. *When the number of samples is insufficient to uniquely identify a band-limited T-periodic function f with Nyquist frequency ν, any other function with the same sample values is called an* alias *of f if its Nyquist frequency is no greater than ν.*

Simple examples of aliasing are given in Figures 8.13 and 8.14. You can see two-dimensional examples of aliasing when watching videos of rotating objects. When the frame rate of the video is very close to an integer multiple of the rotation rate of the object, the object appears to stop rotating. This is the explanation of

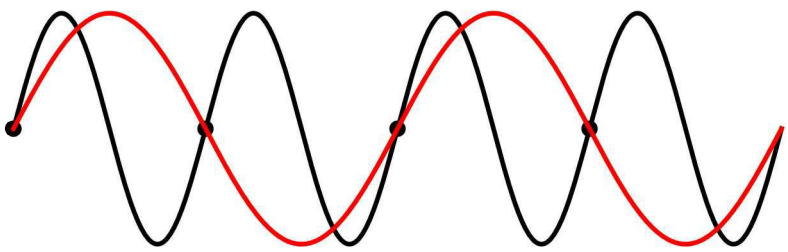


Figure 8.13. *A simple example of aliasing. When the black curve* $y(t) = \sin(4t)$ *is sampled only four times, each sample also lies on the red curve* $y(t) = \sin(2t)$*. The red curve is an alias for the black.*

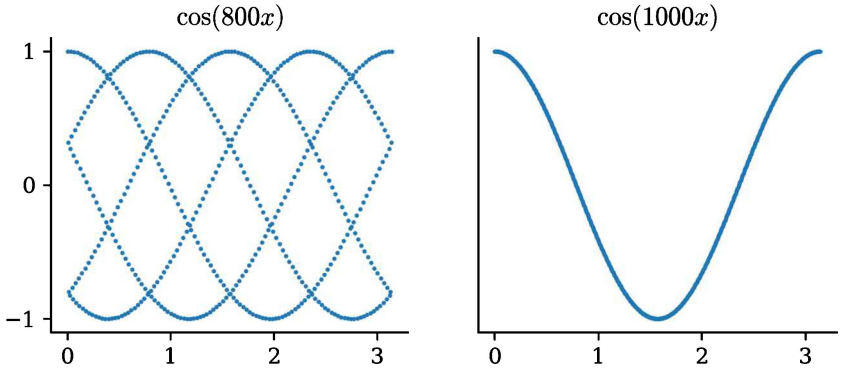


Figure 8.14. *Two examples of aliasing. In the left panel is a plot of* $\cos(800t)$*, and in the right panel a plot of* $\cos(1000t)$*, but neither image is an accurate representation of the functions. Since the number of dots drawn in these images is much smaller than the number of times each curve oscillates, the plots appear to be of an entirely different shape than the true curves. Indeed, we can't hope to plot these curves at this resolution, since the thickness of a line is greater than the wavelength of the oscillations.*

the popular internet videos of flying helicopters whose rotors appear not to spin. Similarly, if the frame rate is only slightly faster or slower than a multiple of the rotation rate of the object, the object appears to rotate very slowly.

8.7.3 Periodic Sampling Theorem

The periodic sampling theorem guarantees that whenever the number of samples of a band-limited periodic function exceeds the Nyquist rate, the function is uniquely determined by the samples.

Theorem 8.7.7 (Periodic Sampling Theorem). *Assume that a band-limited function* $f : [0, T] \to \mathbb{F}$ *has the form* $f(t) = \sum_{k=-\nu}^{\nu} c_k e^{i\omega k t}$*. If* f *is sampled at equally spaced points* $0 = t_0 < t_1 < \cdots < t_{n-1} < T$*, where each* $t_\ell = \frac{\ell T}{n}$*, and* n *exceeds the Nyquist rate of* 2ν*, then* f *is uniquely determined by its sampled values*

$\Phi_n(f) = \mathbf{f} = (f(t_0), f(t_1), \dots, f(t_{n-1}))$. *In particular, the coefficients* c_k *are given by the DFT of the vector of samples, with a shift, that is,*

$$c_k = \begin{cases} \widehat{f}_k & \text{if } k \in [0, \nu], \\ \widehat{f}_{k+n} & \text{if } k \in [-\nu, 0), \end{cases}$$

where $(\widehat{f}_0, \widehat{f}_1, \dots, \widehat{f}_{n-1}) = \widehat{\mathbf{f}} = W_n \mathbf{f}$ *is the DFT of* $\mathbf{f}$.

Proof. The discrete inner product $\langle \Phi_n(f), \Phi_n(g) \rangle_n$ is uniquely determined by the values of f and g at the sample points $t_0, \dots, t_{n-1}$. Moreover, by Proposition 8.5.5, the set $\{\Phi_n(e^{i\omega kt})\}_{k=0}^{n-1}$ is orthonormal with respect to $\langle \cdot, \cdot \rangle_n$. For any $k \in \mathbb{Z}$ and any $\ell \in \{0, \dots, n-1\}$ the sample value of the function $e^{i\omega kt}$ at t_ℓ is equal to the value of the function $e^{i\omega(k-n)t}$ at t_ℓ. Therefore, the set

$$\{\Phi_n(e^{i\omega kt})\}_{k=-\nu}^{n-\nu-1}$$

is also orthonormal with respect to $\langle \cdot, \cdot \rangle_n$. Since $n > 2\nu$, we have $\nu \leq n - \nu - 1$, and therefore, for each integer $k \in [-\nu, \nu]$ the coefficient

$$c_k = \left\langle \Phi_n(f), \Phi_n(e^{i\omega kt}) \right\rangle_n = \begin{cases} \widehat{f}_k & \text{if } k \in [0, \nu), \\ \widehat{f}_{k+n} & \text{if } k \in [-\nu, 0) \end{cases}$$

is uniquely determined by the values of f at the sample points $t_0, \dots, t_{n-1}$. ∎

Example 8.7.8. The function of Example 8.7.2 has Nyquist frequency 7, so the function is uniquely determined (as a periodic band-limited function with $\nu = 7$) by 15 or more samples. In Figure 8.15 we show what happens when we sample 2, 5, 9, 11, 14, and 15 times. In each case the red curve is the periodic function with the smallest Nyquist frequency that passes through all the sample points. The signal is not correctly reconstructed until the number of samples exceeds 14.

Vista 8.7.9. There is also a sampling theorem for nonperiodic functions, called the *Shannon sampling theorem*. Like the periodic sampling theorem, it guarantees that if a function is band limited, then sampling above the Nyquist rate will allow the function to be completely reconstructed. More precisely, it guarantees that $f(t)$ is completely determined by its values at the points $t_k = \frac{k\pi}{n}$ for $k \in \mathbb{Z}$, that $f(t)$ can be written as

$$f(t) = \sum_{k=-\infty}^{\infty} f(t_k) \frac{\sin(nt - k\pi)}{nt - k\pi}, \tag{8.54}$$

and that this series converges uniformly. A full treatment of the Shannon sampling theorem requires the Fourier transform (see Vista 8.2.17), rather than just Fourier series. This is discussed in more depth in Volume 4.

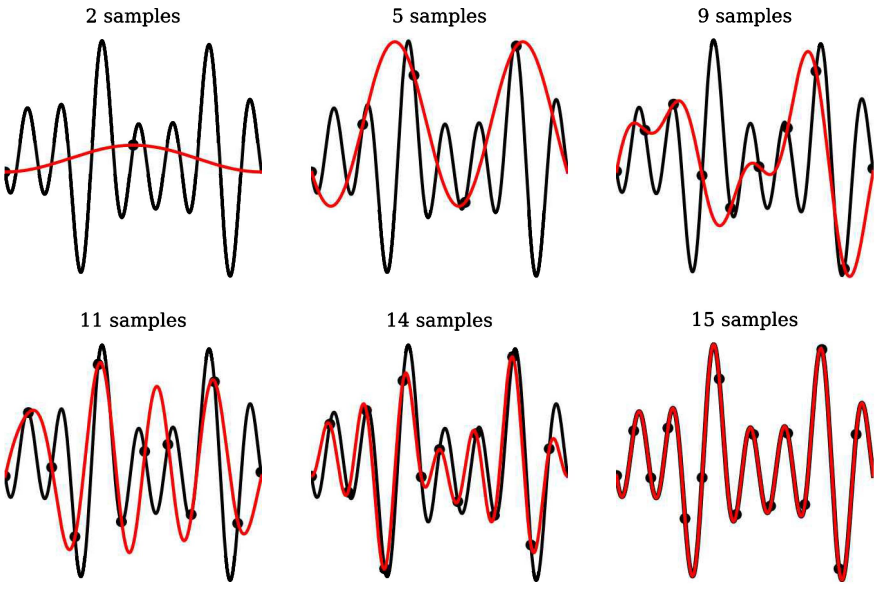


Figure 8.15. *Attempts to reconstruct the f in Example 8.7.8 with* 2, 5, 9, 11, 14, *and* 15 *equally spaced samples. The function f (black) has Nyquist frequency* 7*. In each case the red curve is the periodic function with the smallest Nyquist frequency that passes through all the sample points. As described in the periodic sampling theorem (Theorem 8.7.7), the function is uniquely determined (as a band-limited periodic function with Nyquist frequency* 7*) once the number of samples is strictly greater than the Nyquist rate of* 14*.*

8.7.4 Antialiasing

We have seen that when a band-limited, T-periodic function

$$f(t) = \sum_{k=-\nu}^{\nu} c_k e^{i\omega k t}$$

is sampled at a rate $n \leq 2\nu$, aliasing occurs when we try to reconstruct the function, and the alias may be very different from the original function. One way to improve the situation is to first set all the Fourier coefficients outside the interval $[-\mu, \mu]$ to zero, where $\mu = \lfloor \frac{n-1}{2} \rfloor$; that is, replace f by the function

$$\widetilde{f}(t) = \sum_{k=-\mu}^{\mu} c_k e^{i\omega k t}.$$

Sampling $\widetilde{f}$ at least n times will allow perfect reconstruction of $\widetilde{f}$ from the sample. This is preferable to sampling from f and reconstructing an alias for f because $\widetilde{f}$ is the best approximation to f (measured in the L^2-norm), as the next proposition shows. That means $\widetilde{f}$ is always at least as close to f as the corresponding alias would be.

Proposition 8.7.10. *Define* $\operatorname{span}(E_\nu)$ *to be the space of all band-limited Fourier series on* $[0,T]$ *with Nyquist frequency at most* ν. *If* $f \in L^2([0,T];\mathbb{F})$ *satisfies* $f = S[f]$, *then the truncated Fourier series* $\widetilde{f} = S_\nu[f]$ *satisfies*

$$\|f - \widetilde{f}\| \leq \|f - g\|$$

for all $g \in \operatorname{span}(E_\nu)$.

Proof. The proof is Exercise 8.37. □

The process of replacing f with $\widetilde{f}$ is an example of *low-pass filtering*, so named because it keeps (passes) low frequencies and discards (filters) high frequencies. If it is done before sampling, to reduce aliasing effects, it is called *antialiasing*. With audio signals, antialiasing is often done with an analog electronic circuit, and with video or images, it is often done with a lens or optical filter that slightly blurs the images.

8.8 Haar Wavelets

Fourier analysis deals with the representation and approximation of functions as the superposition (linear combination) of trigonometric functions. In this section and throughout the remainder of this chapter, we consider superpositions of a different class of functions called *wavelets*, which come from sums of two self-similar functions, called the *father* and *mother* wavelets, that are rescaled and translated numerous times to form a representation or approximation of a function. In this section and the next, we focus on a simple class of wavelets called *Haar wavelets*. We consider more general wavelets in Section 8.10.

One of the key features of Fourier analysis is the fact that the Fourier basis functions $\{e^{i\omega kt}\}_{k\in\mathbb{Z}}$ form an orthonormal basis. This allows us to use the inner product to peel off the Fourier coefficients and represent a function as a linear combination of the basis functions. This concept also extends to the discrete case, where a signal can be written as a linear combination of the discrete Fourier basis vectors $\{\mathbf{w}_n^{(k)}\}_{k=0}^{n-1}$, and the coefficients are likewise computed with an inner product; see (8.47). Wavelets work similarly, expressing or approximating a signal or sample as a linear combination of wavelet basis functions, and the coefficients can be determined by computing an inner product.

While Fourier series nicely represent and approximate smooth periodic functions, they are not ideal for functions with discontinuities. Indeed, as discussed in Section 8.2.3, the Fourier series suffers from some undesirable error, called the Gibbs phenomenon, when applied to discontinuous functions. This makes the Fourier approximation less than ideal for discontinuous functions. By contrast, wavelets can gracefully and efficiently represent discontinuities in a signal.

Another advantage of wavelets is that the basis functions can represent local behavior with only a few terms. Contrast this with Fourier series, where all the basis functions are periodic and hence do not naturally represent local behavior. One approach for dealing with this is to divide the domain into small subintervals and use Fourier analysis on the restriction of the function to each subinterval, but then it is difficult to capture behavior that occurs at large time scales. Wavelets provide a different way of handling these problems.

In summary, wavelets are sets of orthogonal functions specially designed for non-periodic, piecewise-continuous functions. In many situations involving piecewise-continuous functions, such as those observed in images, videos, and other digital media, a wavelet basis can be chosen so that the wavelet transform is *sparse*, meaning that it has only a small number of nonzero entries in its matrix representation, thus providing a very efficient representation.

8.8.1 The Haar Father and Sons

We begin with a function called the *Haar scaling function* (or *Haar father* function) and its scaled translates, which we call its *sons*.

Definition 8.8.1. *The left-continuous*[39] *map* $\varphi : \mathbb{R} \to \{0, 1\}$ *given by*

$$\varphi(t) = \begin{cases} 1 & \text{if } 0 \leq t < 1, \\ 0 & \text{otherwise} \end{cases} \tag{8.55}$$

is called the Haar (father) scaling function. *Its graph is given in Figure 8.16. The Haar* sons *are the following scaled and translated versions of the father function:*

$$\varphi_{j,k}(t) = \varphi(2^j t - k) = \begin{cases} 1 & \text{if } \frac{k}{2^j} \leq t < \frac{k+1}{2^j}, \\ 0 & \text{otherwise,} \end{cases}$$

where $j \in \mathbb{N}$ *and* $k \in \mathbb{Z}$.

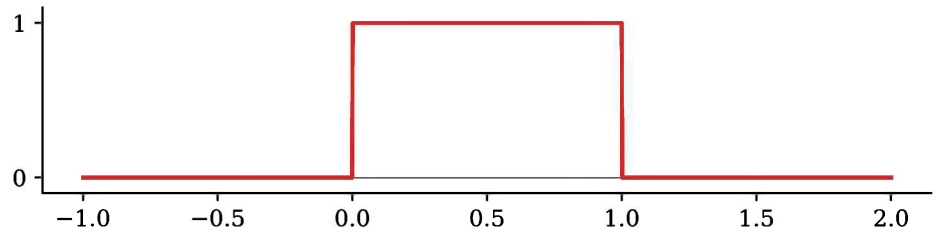


Figure 8.16. *Plot of the Haar father function* φ *given in Definition 8.8.1.*

The Haar father can be thought of as the most basic step function. Any left-continuous step function f with compact support that only jumps at integers can be written as linear combinations of integer translates of the Haar father. More precisely, such an f can be written as

$$f(t) = \sum_{k=-\infty}^{\infty} f(k)\varphi(t - k).$$

Note that since f has compact support, all but a finite number of the sample values $f(k)$ are zero. Hence the sum is well defined.

[39] Recall that a function f is *left continuous* if $\lim_{t \to a^+} f(t) = f(a)$ for every $a \in \mathbb{R}$.

Example 8.8.2. Step functions that change only at integer values can always be written as linear combinations of translates of the Haar father function φ. Consider the function

$$f(t) = 2\varphi(t) + 0.8\varphi(t-1) + 3.1\varphi(t-2) - 2\varphi(t-3),$$

which is illustrated in Figure 8.17.

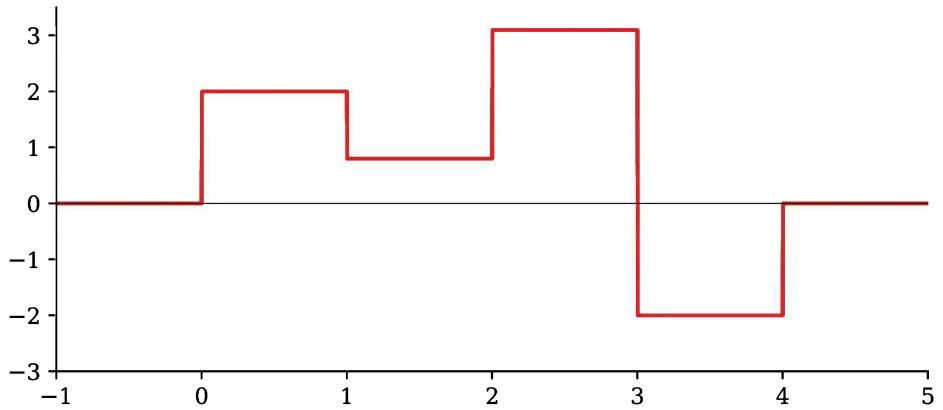


Figure 8.17. *The function f in Example 8.8.2.*

More generally, we can approximate any function $f : \mathbb{R} \to \mathbb{R}$ by choosing $j \in \mathbb{N}$ and sampling f at the points $t_k = \frac{k}{2^j}$ for $k \in \mathbb{Z}$. This gives an approximation $T_j[f](t)$ by left-continuous step functions that have their discontinuities at the values $\frac{k}{2^j}$, by using the sons $\varphi_{j,k}$:

$$T_j[f](t) = \sum_{k=-\infty}^{\infty} f\left(\frac{k}{2^j}\right)\varphi_{j,k}(t) = \sum_{k=-\infty}^{\infty} f\left(\frac{k}{2^j}\right)\varphi(2^j t - k). \tag{8.56}$$

Again since f has compact support, all but a finite number of the sample values $f(\frac{k}{2^j})$ are zero and the sum is well defined. Moreover, as shown at the end of this section, if f is left continuous and has compact support, then the approximation (8.56) converges pointwise to f as $j \to \infty$.

Example 8.8.3. Consider the function

$$f(t) = 100t^2(1-t)\,|\sin(10t/3)|. \tag{8.57}$$

We can approximate f by computing $T_j[f]$ for various values of $j \in \mathbb{N}$; see Figure 8.18.

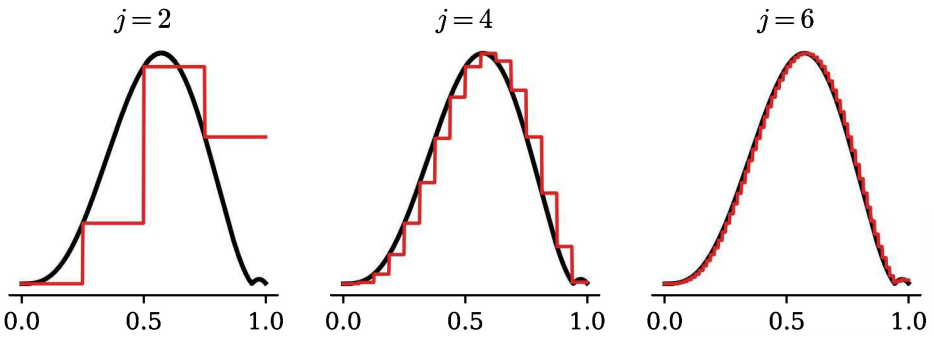


Figure 8.18. *Sampling a function f (black) at points of the form $\frac{k}{2^j}$ for $k \in \mathbb{Z}$ gives an approximation (red) by Haar sons, as $f(t) \approx T_j[f](t)$ as given in* (8.56).

8.8.2 Vector Space Structure

The Haar sons span a vector space that can be used to approximate functions. Moreover, the Haar sons define an orthogonal basis. This is a very natural basis to use for sampled signals, especially those which are discontinuous and nonperiodic.

Throughout the remainder of this chapter, we assume that we are working in the vector space $L^2_c(\mathbb{R};\mathbb{R})$ of square-integrable functions with compact support and with the usual inner product

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(t)g(t)\,dt. \tag{8.58}$$

Definition 8.8.4. *For each $j \in \mathbb{N}$, let V_j denote the span of the set $\{\varphi_{j,k}\}_{k\in\mathbb{Z}}$.*

Every function in V_j is a left-continuous step function with compact support and has discontinuities only on the grid points $\left\{\frac{k}{2^j}\right\}_{k\in\mathbb{Z}}$. Moreover, it is immediate that

$$\varphi(t) = \varphi(2t) + \varphi(2t-1), \tag{8.59}$$

which generalizes to the *scaling relation*

$$\varphi_{j,k}(t) = \varphi_{j+1,2k}(t) + \varphi_{j+1,2k+1}(t), \tag{8.60}$$

which holds for all $j \in \mathbb{N}$ and $k \in \mathbb{Z}$. This shows that V_j is a subspace of V_{j+1}, for each $j \in \mathbb{N}$. Thus we have an increasing chain of vector space inclusions

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_{j-1} \subset V_j \subset \cdots.$$

Proposition 8.8.5. *The set of functions $\{2^{j/2}\varphi_{j,k}\}_{k\in\mathbb{Z}} \subset V_j$ is orthonormal.*

Proof. The proof is Exercise 8.39. □

Assume $f \in L^2_c(\mathbb{R};\mathbb{R})$. We can project f orthogonally onto V_j by computing

$$\operatorname{proj}_{V_j} f(t) = 2^{j/2} \sum_{k=-\infty}^{\infty} \left\langle f, 2^{j/2}\varphi_{j,k}(t)\right\rangle \varphi_{j,k}(t) = 2^j \sum_{k=-\infty}^{\infty} \varphi_{j,k}(t) \int_{\frac{k}{2^j}}^{\frac{k+1}{2^j}} f(t)\,dt.$$

The sum is finite because the support of f is bounded and thus covered by a finite number of intervals of the form $\left[\frac{k}{2^j}, \frac{k+1}{2^j}\right)$. If f is sufficiently well behaved, then as j gets large the integral is well approximated by the area of the rectangle of height $f\left(\frac{k}{2^j}\right)$ and width 2^{-j}. This gives

$$\int_{\frac{k}{2^j}}^{\frac{k+1}{2^j}} f(t)\,dt \approx 2^{-j} f\left(\frac{k}{2^j}\right),$$

which implies that $\operatorname{proj}_{V_j} f(t) \approx T_j[f](t) \in V_j$. In other words, the orthogonal projection of f onto V_j is approximately $T_j[f]$, which also lies in V_j.

8.8.3 Haar Mother and Daughters

Recall that V_j is a subset of V_{j+1}. Here we introduce the Haar *mother* wavelet, and her *daughter* wavelets, which provide an orthonormal basis for the orthogonal complement $V_j^\perp$ in V_{j+1}. This allows us to express elements of V_{j+1} as linear combinations of Haar sons in V_j and Haar daughters in $V_j^\perp$.

Definition 8.8.6. *The* Haar mother wavelet ψ *is the function*

$$\psi(t) = \varphi(2t) - \varphi(2t-1) = \begin{cases} 1 & \text{if } 0 \le t < \frac{1}{2}, \\ -1 & \text{if } \frac{1}{2} \le t < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{8.61}$$

The graph of ψ is given in Figure 8.19. The Haar daughter *wavelets $\psi_{j,k}$ are scaled and translated versions of the mother:*

$$\psi_{j,k}(t) = \psi(2^j t - k) = \varphi_{j+1,2k}(t) - \varphi_{j+1,2k+1}(t) \tag{8.62}$$

for all $j \in \mathbb{N}$ and all $k \in \mathbb{Z}$. Denote the span of $\{\psi_{j,k}\}_{k\in\mathbb{Z}}$ by W_j.

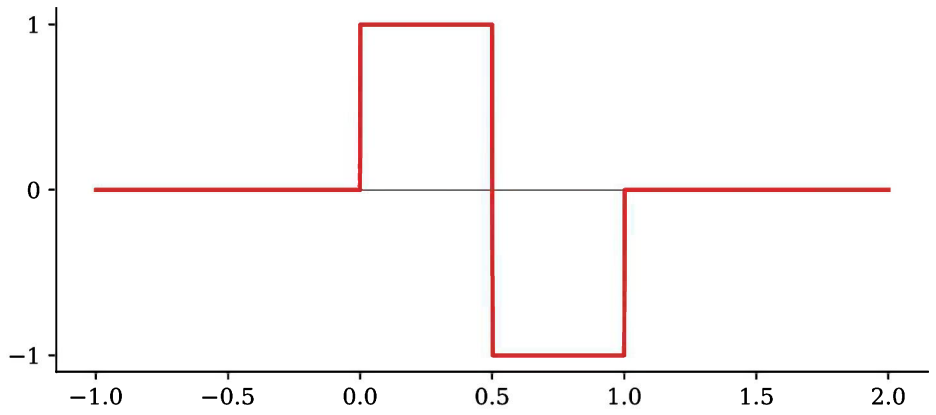


Figure 8.19. *Plot of the Haar mother function ψ given in Definition 8.8.6.*

Proposition 8.8.7. *The following hold for all $j \in \mathbb{N}$ and $k, \ell \in \mathbb{Z}$:*

(i) $\int_{-\infty}^{\infty} \psi_{j,k}(t)\, dt = 0$.

(ii) $\psi_{j,k} \in V_{j+1}$ *and hence* $W_j \subset V_{j+1}$.

(iii) $\langle \psi_{j,k}, \varphi_{j,\ell} \rangle = 0$.

(iv) $\|\psi_{j,k}\|_2^2 = 2^{-j}$.

(v) *The set* $\{\psi_{j,k}\}_{j \in \mathbb{N}, k \in \mathbb{Z}}$ *is orthogonal.*

Proof. The proofs of (i)–(iv) are Exercise 8.41. For (v) observe that if $j < j'$, then by (ii) we have $\psi_{j,k} \in V_{j'}$, but by (iii) $\langle \psi_{j',k'}, g \rangle = 0$ for all $g \in V_{j'}$, so $\langle \psi_{j,k}, \psi_{j',k'} \rangle = 0$. Finally, when $k \neq k'$ the functions $\psi_{j,k}$ and $\psi_{j,k'}$ have disjoint support, and so $\langle \psi_{j,k}, \psi_{j,k'} \rangle = 0$. ∎

8.8.4 Daughters and Sons Are Complements

As shown in Proposition 8.8.7(iii), for a given $j \in \mathbb{N}$ each daughter $\psi_{j,k}$ is orthogonal to each $\varphi_{j,\ell}$ for all $k, \ell \in \mathbb{Z}$; thus each $\psi_{j,k} \in V_j{}^{\perp}$. This implies that $W_j \subset V_j^{\perp}$, where $V_j^{\perp}$ is the orthogonal complement of V_j in V_{j+1}. We now show that $W_j = V_j^{\perp}$.

Theorem 8.8.8. *The space W_j is the orthogonal complement of V_j in V_{j+1}; that is, $V_j \oplus W_j = V_{j+1}$ and $W_j = V_j^{\perp}$. We denote this by*

$$V_{j+1} = V_j \oplus_{\perp} W_j.$$

Proof. Since $W_j \perp V_j$ and $W_j \subset V_{j+1}$, it suffices to prove that $V_j^{\perp} \subset W_j$. Assume that

$$g = \sum_{k=-\infty}^{\infty} a_k \varphi_{j+1,k} \in V_{j+1}$$

with $g \in V_j^{\perp}$; therefore, $g \perp \varphi_{j,m}$ for each $m \in \mathbb{Z}$. Using the scaling relation (8.60) and the orthogonality of sons (Proposition 8.8.5), we have

$$0 = \langle \varphi_{j,m}, g \rangle = \langle \varphi_{j+1,2m}, g \rangle + \langle \varphi_{j+1,2m+1}, g \rangle = \frac{a_{2m} + a_{2m+1}}{2^{j+1}}.$$

Thus $a_{2m+1} = -a_{2m}$ for each $m \in \mathbb{Z}$. It follows that

$$\begin{aligned} g &= \sum_{k=-\infty}^{\infty} a_k \varphi_{j+1,k} = \sum_{m=-\infty}^{\infty} \left(a_{2m} \varphi_{j+1,2m} + a_{2m+1} \varphi_{j+1,2m+1} \right) \\ &= \sum_{m=-\infty}^{\infty} a_{2m} \left(\varphi_{j+1,2m} - \varphi_{j+1,2m+1} \right) = \sum_{m=-\infty}^{\infty} a_{2m} \psi_{j,m} \in W_j. \end{aligned}$$ ∎

The previous theorem allows us to write each V_j as an orthogonal direct sum of Haar daughters.

Corollary 8.8.9. *For any $i \in \mathbb{N}$ with $i \leq j \in \mathbb{Z}^+$ we have*

$$V_j = V_i \oplus_\perp W_i \oplus_\perp W_{i+1} \oplus_\perp \cdots \oplus_\perp W_{j-2} \oplus_\perp W_{j-1}.$$

Proof. For $j \in \mathbb{Z}^+$ we have

$$\begin{aligned} V_j &= V_{j-1} \oplus_\perp W_{j-1} \\ &= V_{j-2} \oplus_\perp W_{j-2} \oplus_\perp W_{j-1} \\ &\vdots \\ &= V_0 \oplus_\perp W_0 \oplus_\perp \cdots \oplus_\perp W_{j-2} \oplus_\perp W_{j-1}. \quad \square \end{aligned}$$

The corollary shows that any function $f \in V_j$ can be written as $f = \tilde{v}_0 + w_0 + w_1 + \cdots + w_{j-1}$, with $\tilde{v}_0 \in V_0$ and $w_i \in W_i$ for each $i \in \{0, 1, \ldots, j-1\}$. Moreover, for any $i \in \mathbb{N}$ with $i < j$, we have

$$f = \tilde{v}_i + \tilde{w}_i, \tag{8.63}$$

where

$$\begin{aligned} \tilde{v}_i &= \tilde{v}_0 + w_0 + \cdots + w_{i-1} \in V_i, \\ \tilde{w}_i &= w_i + \cdots + w_{j-1} \in V_i^\perp. \end{aligned}$$

This allows us to isolate the parts of a signal that change more slowly, namely $\tilde{v}_i \in V_i$, from those parts that change more rapidly, namely $\tilde{w}_i \in V_i^\perp$. For a fixed, relatively small value of $i < j$, we call $\tilde{v}_i$ the *wavelet approximation* of f and call $\tilde{w}_i$ the *detail* of f. By removing the detail from a signal and keeping the approximation, we can filter out unwanted noise and compress the signal.

Example 8.8.10. Consider the function f from Example 8.8.3. We take as the original the sample of f at equally spaced points of distance 2^{-8} apart. The approximation and detail of the sample are given in Figure 8.20 for $j = 6$. The method for computing this decomposition is given in Section 8.9.

Nota Bene 8.8.11. If f is sampled at the points of the form $\frac{k}{2^j}$ to construct $T_j[f] = \sum_k f(\frac{k}{2^j})\varphi_{j,k}$ and $T_j[f]$ is decomposed as $\tilde{v}_i + \tilde{w}_i$, for some $i < j$, then the approximation $\tilde{v}_i$ is not the same as the sampled approximation $T_i[f] = \sum_k f(\frac{k}{2^i})\varphi_{i,k}$.

8.8.5 *Uniform Approximation by Haar Sons

Any continuous function with compact support can be approximated uniformly to arbitrary precision using Haar sons.

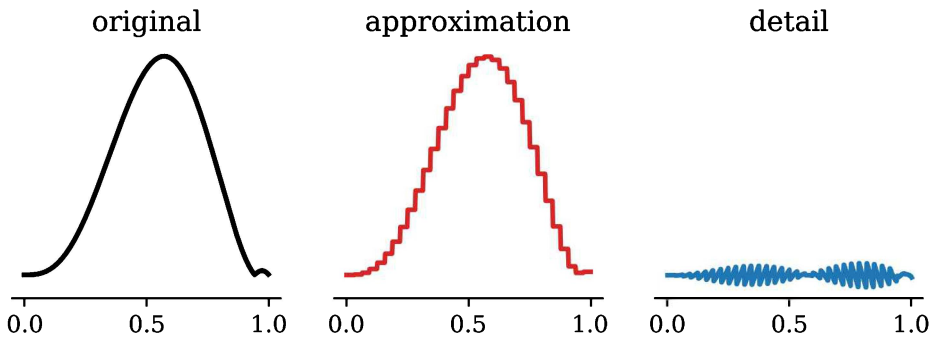


Figure 8.20. *A plot (left panel, black) of $T_8[f] \in V_8$, where f is the function from Example 8.8.3. This is constructed by sampling f at the points of the form $\frac{k}{2^8}$. As described in Corollary 8.8.9, the function $T_8[f]$ can be decomposed into the sum of its Haar wavelet approximation $\tilde{v}_i \in V_i$ and its detail $\tilde{w}_i \in V_i^\perp$. The Haar wavelet approximation $\tilde{v}_6 \in V_6$ of $T_8[f]$ is plotted in the center panel (red), and the detail $\tilde{g}_6 \in V_6^\perp$ is plotted in the right panel (blue).*

Theorem 8.8.12. *Let $f : \mathbb{R} \to \mathbb{R}$ be a uniformly continuous function with compact support. Given $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that $T_j[f] \in V_j$ satisfies*

$$\|f - T_j[f]\|_{L^\infty} = \sup_{t \in \mathbb{R}} |f(t) - T_j[f](t)| < \varepsilon \tag{8.64}$$

whenever $j \geq N$.

Proof. Assume that $\varepsilon > 0$ is given. Since f is uniformly continuous, there exists $\delta > 0$ such that

$$|f(s) - f(t)| < \varepsilon$$

whenever $|s - t| < \delta$. Choose N so that $2^{-N} < \delta$. Then for $j \geq N$ and $t \in \mathbb{R}$, choose $k \in \mathbb{Z}$ so that

$$\frac{k}{2^j} \leq t < \frac{k+1}{2^j}.$$

This gives

$$|f(t) - f(k2^{-j})| < \varepsilon,$$

and thus

$$T_j[f] = \sum_{k=-\infty}^{\infty} f\left(\frac{k}{2^j}\right) \varphi_{j,k} \tag{8.65}$$

satisfies (8.64). Since f has compact support, all but a finite number of the terms of (8.65) are zero, and thus $T_j[f]$ is an element of V_j. □

A similar argument shows that $T_j[f]$ converges pointwise for left-continuous functions with compact support.

Theorem 8.8.13. *If a function f is left continuous with compact support, then the Haar son approximation $T_j[f]$ of f converges pointwise; that is, for every $t \in \mathbb{R}$ and for every $\varepsilon > 0$ there is an integer $N > 0$ such that $|f(t) - T_j[f](t)| < \varepsilon$ whenever $j \geq N$.*

The point is that $T_j[f](t)$ converges to $f(t)$ as $j \to \infty$, and if j is sufficiently large, then $T_j[f]$ is a good approximation of f. In practice, when dealing with slowly varying functions, j does not have to be very large to produce good results. This makes the Haar son approximation a good one for many digital signals and images.

8.9 Discrete Haar Wavelet Transform

As described in the previous section, we can approximate $f \in L_c^2(\mathbb{R};\mathbb{R})$ by projecting it onto V_j and writing it in terms of the Haar sons; that is, we can approximate f by sampling at each $t_k = \frac{k}{2^j}$ to get

$$\operatorname{proj}_{V_j} f \approx T_j[f] = \sum_{k\in\mathbb{Z}} f\left(\frac{k}{2^j}\right)\varphi_{j,k} \in V_j. \tag{8.66}$$

Moreover, Corollary 8.8.9 guarantees that any element of V_j can be decomposed uniquely in terms of the daughters $\psi_{i,k}$, which is generally more useful (see, for example, Applications 8.9.1 and 8.9.8) but which takes a little more work to compute.

Transforming from sons to daughters is called the *wavelet transform*. In this section we consider methods for writing elements of V_j in terms of the Haar wavelet (daughter) basis, and we describe an efficient method for doing this called the *fast wavelet transform*.

Application 8.9.1. The wavelet transform is useful for filtering certain types of noise. When j is small, the approximations in V_j tend not to see high-frequency noise, which tends to show up in the detail (in $V_j^\perp$). Therefore, discarding the detail and keeping only the approximation in V_j is one way to remove the noise. This method is especially effective at removing salt-and-pepper noise (sparsely occurring black and white pixels) from an image.

8.9.1 Sampled Functions and the Discrete Inner Product

While the space V_j is infinite dimensional, we restrict ourselves to functions f that are of compact support, that is, $f \in L_c^2(\mathbb{R};\mathbb{R})$. Thus when we project these functions onto V_j, all but a finite number of the terms in the sum (8.66) are nonzero, and thus the projection can be expressed as a linear combination of basis functions. In the case of Haar wavelets, one typically assumes for simplicity that the function f is supported on the interval $[0,1)$. If the support is some other interval, say, $[a,b)$, then we can translate and rescale as needed to reformulate the problem to having support on $[0,1)$.

The space $V_j \cap L_c^2([0,1);\mathbb{R})$ is spanned by the scaling functions (wavelet sons) $\varphi_{j,k}$ for $k \in \{0,\dots,n-1\}$, so this space is $n = 2^j$ dimensional. Moreover, on the

interval $[0,1)$ the nonzero wavelets in each W_i $(i < j)$ are $\psi_{i,0}, \ldots, \psi_{i,2^i-1}$. Thus the wavelet basis for $V_j \cap L^2_c([0,1);\mathbb{R})$ consists of the functions

$$\{\varphi_{0,0}, \psi_{0,0}, \psi_{1,0}, \psi_{1,1}, \psi_{2,0}, \psi_{2,1}, \psi_{2,2}, \psi_{2,3}, \psi_{3,0}, \ldots, \psi_{j-1,2^{j-1}-1}\}.$$

Note that this basis also has $n = 2^j$ elements because

$$n = 2^j = 1 + \sum_{i=0}^{j-1} 2^i = \dim(V_0) + \sum_{i=0}^{j-1} \dim(W_i).$$

As with the DFT, we consider samples of functions, rather than the functions themselves. For $\Delta_n = 1/n = 2^{-j}$, let $0 = t_0 < \cdots < t_n = 1$ be given by $t_k = k\Delta_n$ for each $k \in \{0, \ldots, n\}$. Given $f \in L^2_c([0,1);\mathbb{R})$, let

$$\Phi_n(f) = \mathbf{f} = (f_0, f_1, \ldots, f_{n-1}) = (f(t_0), \ldots, f(t_{n-1})) \in \mathbb{R}^n. \tag{8.67}$$

The sampling function $\Phi_n : V_j \cap L^2_c([0,1);\mathbb{R}) \to \mathbb{R}^n$ maps each $\varphi_{j,k}$ to the standard basis vector $\mathbf{e}_k \in \mathbb{R}^n$. This is clearly surjective, so Φ_n is an isomorphism of vector spaces.

Since the projected functions are in V_j and supported on $[0,1)$, they are constant on each interval of the form $[t_k, t_{k+1})$. Hence, the inner product (8.58) reduces to the discrete inner product $\langle \cdot, \cdot \rangle_n$:

$$\langle f, g \rangle = \int_0^1 f(t)g(t)\,dt = \frac{1}{n}\sum_{k=0}^{n-1} f(t_k)g(t_k) = \langle \Phi_n(f), \Phi_n(g) \rangle_n .$$

This shows that Φ_n is an orthonormal isomorphism (see Volume 1, Sections 3.2.2 and 3.3.2), so from now on we may work entirely in the inner product space $\mathbb{R}^n$ with the discrete inner product $\langle \cdot, \cdot \rangle_n$. As shorthand we write $\boldsymbol{\varphi}_{i,k} = \Phi_n(\varphi_{i,k})$ for each $i \in \{0, \ldots, j\}$ and each $k \in \{0, \ldots, 2^i - 1\}$. Similarly, we write $\boldsymbol{\psi}_{i,k} = \Phi_n(\psi_{i,k})$ for each $i \in \{0, \ldots, j-1\}$ and each $k \in \{0, \ldots, 2^i - 1\}$.

Example 8.9.2. Taking $j = 2$ (so that $n = 2^2 = 4$), we have

$$\boldsymbol{\varphi}_{2,0} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\varphi}_{2,1} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\varphi}_{2,2} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \text{ and } \boldsymbol{\varphi}_{2,3} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Sampling the functions in the basis $\{\varphi_{0,0}, \psi_{0,0}, \psi_{1,0}, \psi_{1,1}\}$ gives

$$\boldsymbol{\varphi}_{0,0} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\psi}_{0,0} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \quad \boldsymbol{\psi}_{1,0} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}, \text{ and } \boldsymbol{\psi}_{1,1} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}.$$

8.9.2 Wavelet Decomposition

Converting from sons to daughters in $V_j \cap L_c^2([0,1);\mathbb{R})$ corresponds to changing basis from the standard basis in $\mathbb{R}^n$ to the basis $\{\varphi_{0,0}, \psi_{0,0}, \psi_{0,1}, \dots, \psi_{j-1,2^{j-1}-1}\}$.

When $j = 2$, Example 8.9.2 gives the explicit form for the wavelet basis, and expressing any $\mathbf{f}$ in terms of that basis amounts to solving the system

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{8.68}$$

Let H_4 be the matrix in (8.68). Since the columns of H_4 are orthogonal (by Proposition 8.8.7), the inverse of H_4 is the transpose of H_4, with the rows rescaled appropriately.

$$H_4^{-1} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 \end{bmatrix}.$$

This is the matrix representation of the wavelet transform for any $\mathbf{f} \in \mathbb{R}^4$.

Example 8.9.3. Consider the function $f \in V_2$ given by

$$f(t) = 2\varphi_{2,0}(t) + 0.8\varphi_{2,1}(t) + 3.1\varphi_{2,2}(t) - 2\varphi_{2,3}(t).$$

Note that $\Phi_4(f) = \mathbf{f} = (2, 0.8, 3.1, -2)$. The wavelet transform is given by $H_4^{-1}\mathbf{f} = (0.975, 0.425, 0.6, 2.55)$. Thus we have

$$f(t) = 0.975\varphi_{0,0}(t) + 0.425\psi_{0,0}(t) + 0.6\psi_{1,0}(t) + 2.55\psi_{1,1}(t).$$

More generally, for arbitrary $j \in \mathbb{N}$, computing the wavelet transform of f corresponds to solving the system

$$\mathbf{f} = H_n \mathbf{b},$$

where $\mathbf{b} = (b_0, \dots, b_{n-1})$ and the columns of H_n are the discrete wavelet basis vectors. As in the case of $j = 2$, the matrix H_n^{-1} is always a rescaled transpose of H_n.

Since solving this system requires only that we multiply by H_n^{-1}, computing the wavelet transform this way has temporal complexity in $O(n^2)$. This is similar to computing the DFT by matrix multiplication. But, just as in the case of the DFT, there is a way to compute the wavelet transform much more efficiently. Surprisingly, this can be done in $O(n)$ time, as we show in the next subsection.

8.9.3 The Fast Wavelet Transform

To compute the wavelet transform more efficiently, we use the following lemma to write each basis vector $\varphi_{j,k}$ of V_j in terms of elements of V_{j-1} and W_{j-1}.

Lemma 8.9.4. *For $j \in \mathbb{Z}^+$ and $k \in \mathbb{Z}$, we have*

$$\varphi_{j,2k} = \frac{\varphi_{j-1,k} + \psi_{j-1,k}}{2} \quad \textit{and} \quad \varphi_{j,2k+1} = \frac{\varphi_{j-1,k} - \psi_{j-1,k}}{2}.$$

Proof. For $j = 1$ and $k = 0$ the claim of the lemma reduces to

$$\varphi(2t) = \frac{\varphi(t) + \psi(t)}{2} \qquad \text{and} \qquad \varphi(2t-1) = \frac{\varphi(t) - \psi(t)}{2}, \tag{8.69}$$

which follows directly from the definitions. For general j and k, simply take dilates and translates of (8.69). □

Theorem 8.9.5 (Haar Decomposition Theorem). *Any function*

$$f = \sum_{k=-\infty}^{\infty} a_{j,k}\varphi_{j,k} \in V_j \qquad (j \in \mathbb{Z}^+)$$

can be decomposed as

$$f = \tilde{v}_{j-1} + w_{j-1},$$

where

$$\tilde{v}_{j-1} = \sum_{k=-\infty}^{\infty} a_{j-1,k}\varphi_{j-1,k} \quad \textit{and} \quad w_{j-1} = \sum_{k=-\infty}^{\infty} b_{j-1,k}\psi_{j-1,k}$$

with

$$a_{j-1,k} = \frac{a_{j,2k} + a_{j,2k+1}}{2} \quad \textit{and} \quad b_{j-1,k} = \frac{a_{j,2k} - a_{j,2k+1}}{2}. \tag{8.70}$$

Proof. The proof is just a computation, using the previous lemma:

$$\begin{aligned}
f &= \sum_{k=-\infty}^{\infty} a_{j,k}\varphi_{j,k} \\
&= \sum_{k=-\infty}^{\infty} a_{j,2k}\varphi_{j,2k} + \sum_{k=-\infty}^{\infty} a_{j,2k+1}\varphi_{j,2k+1} \\
&= \frac{1}{2}\sum_{k=-\infty}^{\infty} a_{j,2k}\left(\varphi_{j-1,k} + \psi_{j-1,k}\right) + \frac{1}{2}\sum_{k=-\infty}^{\infty} a_{j,2k+1}\left(\varphi_{j-1,k} - \psi_{j-1,k}\right) \\
&= \frac{1}{2}\sum_{k=-\infty}^{\infty} \left(a_{j,2k} + a_{j,2k+1}\right)\varphi_{j-1,k} + \frac{1}{2}\sum_{k=-\infty}^{\infty} \left(a_{j,2k} - a_{j,2k+1}\right)\psi_{j-1,k} \\
&= \sum_{k=-\infty}^{\infty} a_{j-1,k}\varphi_{j-1,k} + \sum_{k=-\infty}^{\infty} b_{j-1,k}\psi_{j-1,k} \\
&= \tilde{v}_{j-1} + w_{j-1}. \quad \square
\end{aligned}$$

Corollary 8.9.6 (Fast Wavelet Transform). *Fix $i < j$, and let $\mathbf{f} \in \mathbb{R}^n$ with $n = 2^j$. Let*

$$\mathbf{f} = \sum_{k=0}^{2^i-1} a_{i,k}\boldsymbol{\varphi}_{i,k} + \sum_{m=i}^{j-1}\sum_{k=0}^{2^m-1} b_{m,k}\boldsymbol{\psi}_{m,k}$$

be the wavelet decomposition of $\mathbf{f}$ to level i, and define $\mathbf{a}_m = (a_{m,0}, \dots, a_{m,2^m-1})$ and $\mathbf{b}_m = (b_{m,0}, \dots, b_{m,2^m-1})$. The wavelet decomposition, consisting of $\mathbf{a}_i$ and $\mathbf{b}_i, \mathbf{b}_{i+1}, \dots, \mathbf{b}_{j-1}$, can be computed using the recursive algorithm below, called the fast wavelet transform (FWT). *For an implementation of this algorithm, see Algorithm 8.3.*

(i) *Initialize by setting $m = j$ and setting $\mathbf{a}_j = \mathbf{f}$.*

(ii) *While $m > i$ repeat the following steps:*

  (a) *Compute $\mathbf{a}_{m-1}$ and $\mathbf{b}_{m-1}$ via (8.70). To vectorize, let $\mathbf{a}_m^{\text{even}}$ and $\mathbf{a}_m^{\text{odd}}$ be, respectively, the even-indexed and odd-indexed parts of $\mathbf{a}_m$. Thus $\mathbf{a}_{m-1} = \frac{1}{2}(\mathbf{a}_m^{\text{even}} + \mathbf{a}_m^{\text{odd}})$ and $\mathbf{b}_{m-1} = \frac{1}{2}(\mathbf{a}_m^{\text{even}} - \mathbf{a}_m^{\text{odd}})$.*

  (b) *Decrement m: $m \leftarrow m - 1$.*

(iii) *Return $\mathbf{a}_i$ and $\mathbf{b}_i, \dots, \mathbf{b}_{j-1}$.*

In particular, the full wavelet decomposition $(\mathbf{a}_0, \mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_{j-1})$ is computed by this algorithm with $i = 0$. The temporal complexity of the algorithm is $\sim 4n$ FLOPs.

Proof. The fact that the algorithm computes the required coefficients is an immediate result of the Haar decomposition theorem (Theorem 8.9.5). The mth step requires 2^{m-1} additions, 2^{m-1} subtractions, and 2^m multiplications by $\frac{1}{2}$ for a total of 2^{m+1} FLOPs per iteration. Summing over all values of m gives $\sum_{m=1}^{j} 2^{m+1} = 4(2^j - 1) \sim 4n$ FLOPs. □

```python
import numpy as np
def FWT(a,j=0):
    """ Haar FWT of `a` down to level j
    """

    # Assume len(a) is an integer power of 2
    m = int(np.log2(len(a)))

    L = []  # List of the partial trasforms
    while m > j:
        L.append(0.5 * (a[::2] - a[1::2]))
        a = 0.5 * (a[::2] + a[1::2])
        m -= 1
    return a,L[::-1]
```

Algorithm 8.3. *An implementation of the FWT algorithm for the Haar wavelet. It computes the wavelet transform of a vector* `a` *of length 2^j down to level i, where $0 \le i < j$.*

The theorem shows that the temporal complexity of the FWT is $O(n)$, where n is the number of samples. Contrast this with the complexity $O(n\log(n))$ of the FFT. The reason that the FWT is faster than the FFT is that the wavelet transform is sparse; that is, many of the entries in the matrix H_n are zero because the wavelets are supported on very small intervals. This can be seen by looking closely at H_n. While the first two columns of H_n have no zeros, the next two columns have only half their entries that are nonzero, the next four have only a quarter that are nonzero, and so forth.

Example 8.9.7. Consider the function f from Example 8.9.3. Instead of computing the wavelet transform by matrix multiplication, we can use the FWT. Setting $\mathbf{a}_2 = \mathbf{f}$ and using (8.70) we have

$$\mathbf{a}_2 = \begin{bmatrix} 2 \\ 0.8 \\ 3.1 \\ -2 \end{bmatrix}, \quad \mathbf{a}_1 = \begin{bmatrix} 1.4 \\ 0.55 \end{bmatrix}, \quad \mathbf{a}_0 = \begin{bmatrix} 0.975 \end{bmatrix}$$

and

$$\mathbf{b}_1 = \begin{bmatrix} 0.6 \\ 2.55 \end{bmatrix}, \quad \mathbf{b}_0 = \begin{bmatrix} 0.425 \end{bmatrix}, \text{ giving } \begin{bmatrix} \mathbf{a}_0 \\ \mathbf{b}_0 \\ \mathbf{b}_1 \end{bmatrix} = \begin{bmatrix} 0.975 \\ 0.425 \\ 0.6 \\ 2.55 \end{bmatrix}$$

as the wavelet transform. Thus

$$\begin{aligned} f(t) &= a_{0,0}\varphi_{0,0}(t) + b_{0,0}\psi_{0,0}(t) + b_{1,0}\psi_{1,0}(t) + b_{1,1}\psi_{1,1}(t) \\ &= 0.975\varphi_{0,0}(t) + 0.425\psi_{0,0}(t) + 0.6\psi_{1,0}(t) + 2.55\psi_{1,1}(t), \end{aligned}$$

which agrees with the result of Example 8.9.3.

Application 8.9.8. The wavelet transform is often used to compress signals or images. In many settings the most important information in the signal is carried in the approximation $\tilde{v}_j \in V_j$, while the information carried in the detail $\tilde{w}_j \in V_j^\perp$ is less important. Keeping the approximation and discarding the detail results in significant compression with minimal information loss. This is the basis of compression used by the JPEG 2000 image compression standard (although these do not use Haar wavelets). This is treated in more depth in the computer labs associated with this volume.

8.10 *General Wavelets

Haar wavelets are reasonably well suited to slowly varying piecewise-continuous functions such as those encountered with many digital signals and images. There are many other examples of wavelets that can also be used, and some tend to have

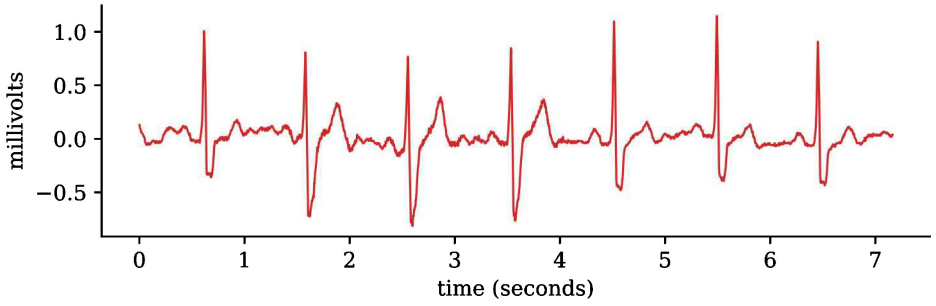


Figure 8.21. *An ECG signal, which is sampled at* 500 *Hz, measuring the electrical activity of the heart (in millivolts) using electrodes placed over the skin.*

more success with specific types of signals. For example, Figure 8.21 shows a typical electrocardiogram (ECG) signal from a beating heart. Because this signal has sharp peaks that occur with each beat and intermittent periods of slow variation between beats, the Haar wavelet is usually less favored than other types of wavelets. But the *Daubechies* wavelet, which is defined in the next section, tends to do fairly well with ECG signals.

In this section, we show how to generalize wavelets to a much more general setting, where many different scaling functions (father wavelets) can be considered, each with a corresponding mother wavelet that couples with the father. These are combined to produce their own wavelet decomposition.

8.10.1 Scaling Function

To generalize the wavelet constructions of the previous sections, we identify the key requirements of a general father function, also known as a scaling function.

Definition 8.10.1. *A function* $\varphi \in L^2_c(\mathbb{R};\mathbb{R})$ *is said to be a* scaling *function (or* father *function) if it satisfies the following:*

(i) *It has positive mass, that is,* $\int_{-\infty}^{\infty} \varphi(x)\,dx > 0$.

(ii) *The set* $\{\varphi(x-k)\}_{k\in\mathbb{Z}}$ *of translated father functions is orthonormal with respect to the usual* L^2 *inner product.*

(iii) *It can be written as a linear combination of half-scaled translates, that is,*

$$\varphi(x) = \sum_{\ell\in\mathbb{Z}} h_\ell \varphi(2x-\ell), \tag{8.71}$$

where all but a finite number of the coefficients $h_\ell \in \mathbb{R}$ *are zero.*

Example 8.10.2. In the case of the Haar scaling function, the scaling relation (8.59) shows that (8.71) holds with $h_0 = h_1 = 1$ and $h_i = 0$ for all $i \notin \{0,1\}$.

Definition 8.10.3. *Given $\varphi \in L^2_c(\mathbb{R};\mathbb{R})$ satisfying Definition 8.10.1, the* sons *are the functions*

$$\varphi_{j,k}(x) = \varphi(2^j x - k), \tag{8.72}$$

where $j \in \mathbb{N}$ and $k \in \mathbb{Z}$.

Proposition 8.10.4 (Scaling Relation). *For each $j \in \mathbb{N}$ and $k \in \mathbb{Z}$ we have*

$$\varphi_{j,k} = \sum_{\ell \in \mathbb{Z}} h_{\ell - 2k} \varphi_{j+1,\ell}. \tag{8.73}$$

Proof. Reindexing the sum via $m = 2k + \ell$, gives

$$\begin{aligned}\varphi_{j,k}(x) = \varphi(2^j x - k) &= \sum_{\ell \in \mathbb{Z}} h_\ell \varphi(2^{j+1}x - 2k - \ell) \\ &= \sum_{m \in \mathbb{Z}} h_{m-2k} \varphi(2^{j+1}x - m) = \sum_{m \in \mathbb{Z}} h_{m-2k} \varphi_{j+1,m}(x). \quad \square\end{aligned}$$

Proposition 8.10.5. *For any $j \in \mathbb{N}$, the set $\{2^{j/2}\varphi_{j,k}\}_{k\in\mathbb{Z}}$ is orthonormal.*

Proof. Using the u-substitution $u = 2^j x - l$, we integrate to get

$$\langle \varphi_{j,k}, \varphi_{j,\ell} \rangle = \int_{-\infty}^{\infty} \varphi(2^j x - k)\varphi(2^j x - \ell)\, dx = 2^{-j}\delta_{k,\ell}. \quad \square$$

Proposition 8.10.6. *For $j \in \mathbb{Z}^+$ and $k, m \in \mathbb{Z}$, we have*

$$\langle \varphi_{j,k}, \varphi_{j-1,m} \rangle = 2^{-j} h_{k-2m}. \tag{8.74}$$

Proof. This is Exercise 8.48. $\square$

Remark 8.10.7. Let $V_j = \operatorname{span}(\{\varphi_{j,k}\}_{k\in\mathbb{Z}})$. As with Haar sons, we have the chain of inclusions

$$V_0 \subset V_1 \subset V_2 \subset \cdots.$$

Remark 8.10.8. Definition 8.10.1 puts many constraints on the function φ and the possible values of the scaling coefficients h_k.

Proposition 8.10.9. *The coefficients h_k satisfy the following properties:*

(i) *For $\ell, m \in \mathbb{Z}$ we have*

$$\sum_{k \in \mathbb{Z}} h_{k-2\ell} h_{k-2m} = 2\delta_{\ell,m}. \tag{8.75}$$

(ii)
$$\sum_{k\in\mathbb{Z}} h_k = 2. \tag{8.76}$$

(iii)
$$\sum_{k\in\mathbb{Z}} h_{2k} = 1 \quad \textit{and} \quad \sum_{k\in\mathbb{Z}} h_{2k+1} = 1. \tag{8.77}$$

Proof.

(i) We have
$$\sum_{k\in\mathbb{Z}} h_{k-2\ell} h_{k-2m} = 2^j \sum_{k\in\mathbb{Z}} h_{k-2\ell} \left\langle \varphi_{j,k}, \varphi_{j-1,m} \right\rangle = 2^j \left\langle \varphi_{j-1,\ell}, \varphi_{j-1,m} \right\rangle = 2\delta_{\ell,m}.$$

(ii) Integrating (8.71) with the substitution $y = 2x - k$ gives
$$2\int_{-\infty}^{\infty} \varphi(x)\,dx = 2\sum_{k=-\infty}^{\infty} h_k \int_{-\infty}^{\infty} \varphi(2x-k)\,dx = \left(\sum_{k=-\infty}^{\infty} h_k\right) \int_{-\infty}^{\infty} \varphi(y)\,dy.$$
Dividing both sides by $\int_{-\infty}^{\infty}\varphi(x)\,dx$, which is nonzero, gives (8.76).

(iii) To prove (8.77), set $\ell = 0$ and sum (8.75) over $m \in \mathbb{Z}$ to get
$$\sum_{m\in\mathbb{Z}} \sum_{k\in\mathbb{Z}} h_{k-2m} h_k = 2.$$
Breaking the inner sum into even ($k = 2j$) and odd ($k = 2j+1$) values of k and changing the order of summation gives
$$\begin{aligned} 2 &= \sum_{m\in\mathbb{Z}} \left(\sum_{j\in\mathbb{Z}} h_{2j-2m} h_{2j} + \sum_{j\in\mathbb{Z}} h_{2j+1-2m} h_{2j+1} \right) \\ &= \sum_{j\in\mathbb{Z}} \left(\sum_{m\in\mathbb{Z}} h_{2j-2m} \right) h_{2j} + \sum_{j\in\mathbb{Z}} \left(\sum_{m\in\mathbb{Z}} h_{2j+1-2m} \right) h_{2j+1}. \end{aligned}$$
Substituting $m \to j - m$ gives
$$2 = \left(\sum_{m\in\mathbb{Z}} h_{2m}\right)\left(\sum_{j\in\mathbb{Z}} h_{2j}\right) + \left(\sum_{m\in\mathbb{Z}} h_{2m+1}\right)\left(\sum_{j\in\mathbb{Z}} h_{2j+1}\right).$$
Setting $r = \sum_m h_{2m}$ and $s = \sum_m h_{2m+1}$ we have $r^2 + s^2 = 2$. Moreover, applying (8.76) gives $r + s = 2$. Solving the system yields $r = s = 1$, which gives the result. $\square$

8.10.2 Wavelets: Mother and Daughters

Definition 8.10.10. *Given a father function φ satisfying Definition 8.10.1, the corresponding* mother wavelet *is the function*

$$\psi(x) = \sum_{\ell \in \mathbb{Z}} (-1)^{\ell} h_{1-\ell} \varphi(2x - \ell), \tag{8.78}$$

where the scaling coefficients h_k are those given in (8.71)*. For each $j \in \mathbb{N}$ and $k \in \mathbb{Z}$ define the* daughter wavelet *$\psi_{j,k}$ to be*

$$\psi_{j,k}(x) = \psi(2^j x - k). \tag{8.79}$$

Moreover, let $W_j = \operatorname{span}(\{\psi_{j,k}\}_{k \in \mathbb{Z}})$ be the span of the daughters.

Proposition 8.10.11. *Given $j \in \mathbb{N}$ and $k \in \mathbb{Z}$, we have $W_j \subset V_{j+1}$ with*

$$\psi_{j,k} = \sum_{\ell \in \mathbb{Z}} (-1)^{\ell} h_{1-\ell+2k} \varphi_{j+1,\ell}. \tag{8.80}$$

Proof. Let $\ell \to \ell - 2k$. Thus

$$\begin{aligned} \psi_{j,k}(x) = \psi(2^j x - k) &= \sum_{\ell \in \mathbb{Z}} (-1)^{\ell} h_{1-\ell} \varphi(2^{j+1} x - 2k - \ell) \\ &= \sum_{\ell \in \mathbb{Z}} (-1)^{\ell} h_{1-\ell+2k} \varphi_{j+1,\ell}(x). \quad \square \end{aligned}$$

Proposition 8.10.12. *Given a scaling function φ satisfying Definition 8.10.1, the corresponding son and daughter wavelets $\varphi_{j,k}$ and $\psi_{j,k}$ (see* (8.72) *and* (8.79)*, respectively) satisfy the following properties for each $j \in \mathbb{N}$:*

(i) $\int_{-\infty}^{\infty} \psi_{j,k}(x)\, dx = 0$ *for each $k \in \mathbb{Z}$.*

(ii) $\langle \psi_{j,k}, \varphi_{j,m} \rangle = 0$ *for all $k, m \in \mathbb{Z}$.*

(iii) *The set $\{2^{j/2} \psi_{j,k}\}_{k \in \mathbb{Z}}$ is orthonormal.*

Proof.

(i) Integrating (8.78) gives

$$\int_{-\infty}^{\infty} \psi(x)\, dx = \sum_{\ell \in \mathbb{Z}} (-1)^{\ell} h_{1-\ell} \int_{-\infty}^{\infty} \varphi(2x - \ell)\, dx = \frac{1}{2} \sum_{\ell \in \mathbb{Z}} (-1)^{\ell} h_{1-\ell} \int_{-\infty}^{\infty} \varphi(y)\, dy.$$

The result follows from Exercise 8.49.

(ii) We have

$$\begin{aligned}\langle \varphi_{j,m}, \psi_{j,k} \rangle &= \left\langle \sum_{i\in\mathbb{Z}} h_{i-2m}\varphi_{j+1,i}, \sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{1-\ell+2k}\varphi_{j+1,\ell} \right\rangle \\ &= \sum_{\ell\in\mathbb{Z}}\sum_{i\in\mathbb{Z}} (-1)^\ell h_{i-2m}h_{1-\ell+2k} \langle \varphi_{j+1,i}, \varphi_{j+1,\ell}\rangle \\ &= 2^{-j-1}\sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{\ell-2m}h_{1-\ell+2k} \\ &= 2^{-j-1}(-1)^{k+m}\sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{k-m+\ell}h_{k-m-\ell+1}.\end{aligned}$$

Note that the negative indices cancel the positive. Specifically,

$$\begin{aligned}\sum_{\ell=-\infty}^{0} (-1)^\ell h_{k-m+\ell}h_{k-m-\ell+1} &= \sum_{\ell=0}^{\infty} (-1)^\ell h_{k-m-\ell}h_{k-m+\ell+1} \\ &= -\sum_{\ell=1}^{\infty} (-1)^\ell h_{k-m-\ell+1}h_{k-m+\ell}.\end{aligned}$$

Thus

$$\sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{k-m+\ell}h_{k-m-\ell+1} = 0.$$

(iii) Reindexing and using (8.75), we have

$$\begin{aligned}2^j \langle \psi_{j,k}, \psi_{j,m} \rangle &= 2^j \left\langle \sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{1-\ell+2k}\varphi_{j+1,\ell}, \sum_{i\in\mathbb{Z}} (-1)^i h_{1-i+2m}\varphi_{j+1,i} \right\rangle \\ &= 2^j \sum_{\ell\in\mathbb{Z}}\sum_{i\in\mathbb{Z}} (-1)^{\ell+i} h_{1-\ell+2k}h_{1-i+2m} \langle \varphi_{j+1,\ell}, \varphi_{j+1,i}\rangle \\ &= \frac{1}{2}\sum_{\ell\in\mathbb{Z}} h_{1-\ell+2k}h_{1-\ell+2m} \\ &= \frac{1}{2}\sum_{\ell\in\mathbb{Z}} h_{\ell+2k}h_{\ell+2m} = \delta_{k,m}. \qquad \square\end{aligned}$$

8.10.3 Wavelet Decomposition

In this section we generalize Theorem 8.8.8. This makes the wavelet transform possible, where an element of V_j can be expressed as a linear combination of wavelet daughters. We describe the general wavelet transform in the next section.

Theorem 8.10.13. *For $j \in \mathbb{N}$, we have $V_{j+1} = V_j \oplus_\perp W_j$.*

Proof. We know that $W_j \perp V_j$, $V_j \subset V_{j+1}$, and $W_j \subset V_{j+1}$. Thus it suffices to show that $V_{j+1} \subset V_j \oplus W_j$. We do this by showing for $j, k \in \mathbb{N}$ that there are sets

$\{a_{j,\ell}\}_{\ell\in\mathbb{Z}}$ and $\{b_{j,\ell}\}_{\ell\in\mathbb{Z}}$ (with all but a finite number being zero) such that

$$\varphi_{j+1,k} = \sum_{\ell\in\mathbb{Z}} a_{j,\ell}\varphi_{j,\ell} + \sum_{\ell\in\mathbb{Z}} b_{j,\ell}\psi_{j,\ell}. \tag{8.81}$$

Assuming the hypothesis, Exercise 8.50 gives

$$a_{j,\ell} = \frac{1}{2}h_{k-2\ell} \quad \text{and} \quad b_{j,\ell} = \frac{1}{2}(-1)^k h_{1-k+2\ell}. \tag{8.82}$$

Thus

$$\begin{aligned} 2\varphi_{j+1,k} &= \sum_{\ell\in\mathbb{Z}} h_{k-2\ell}\varphi_{j,\ell} + \sum_{\ell\in\mathbb{Z}} (-1)^k h_{1-k+2\ell}\psi_{j,\ell} \\ &= \sum_{\ell\in\mathbb{Z}}\sum_{m\in\mathbb{Z}} \left(h_{k-2\ell}h_{m-2\ell} + (-1)^{k+m}h_{1-k+2\ell}h_{1-m+2\ell}\right)\varphi_{j+1,m}. \end{aligned}$$

Taking the inner product with $\varphi_{j+1,k+i}$ gives

$$2\delta_{0,i} = \frac{1}{2}\sum_{\ell\in\mathbb{Z}} \left(h_{k-2\ell}h_{k-2\ell+i} + (-1)^i h_{1-k+2\ell}h_{1-k+2\ell-i}\right). \tag{8.83}$$

Thus to prove the theorem, it suffices to show that (8.83) holds. We do this by proving the case for even ($i = 2s$) and odd ($i = 2s+1$) values of i, respectively.

For the even case, mapping $\ell \to k - \ell$ in the first sum of the right-hand side of (8.83) gives

$$\begin{aligned} &\sum_{\ell\in\mathbb{Z}} (h_{k-2\ell}h_{k-2\ell+2s} + h_{1+2\ell-k}h_{1+2\ell-k-2s}) \\ &\qquad = \sum_{\ell\in\mathbb{Z}} (h_{2\ell-k}h_{2\ell-k+2s} + h_{1+2\ell-k}h_{1+2\ell-k-2s}) \\ &\qquad = \sum_{j\in\mathbb{Z}} h_{j-k}h_{j-k+2s} = \sum_{j\in\mathbb{Z}} h_j h_{j+2s} = 2\delta_{s,0} = 2\delta_{i,0}. \end{aligned}$$

For the odd case, mapping $\ell \to s + k - \ell$ in the second sum of the right-hand side of (8.83) gives

$$\sum_{\ell\in\mathbb{Z}} (-1)^i h_{1+2\ell-k}h_{1+2\ell-k-i} = -\sum_{\ell\in\mathbb{Z}} h_{1+2\ell-k}h_{2\ell-k-2s} = -\sum_{\ell\in\mathbb{Z}} h_{k-2\ell+i}h_{k-2\ell},$$

which cancels with the first term in (8.83). □

Since $V_{j+1} = V_j \oplus_\perp W_j$, we can write any $f \in V_{j+1}$ uniquely as $f = \tilde{v}_j + w_j$ with $\tilde{v}_j \in V_j$ and $w_j \in W_j$. Since $\{\varphi_{j,\ell}\}_{\ell\in\mathbb{Z}}$ is an orthogonal basis for V_j and $\{\psi_{j,\ell}\}_{\ell\in\mathbb{Z}}$ is an orthogonal basis for W_j, we can write $f \in V_{j+1}$ uniquely as a linear combination of elements in these two bases. To find the appropriate linear combination, we generalize (8.82) with the following corollary.

Corollary 8.10.14. *For each $j \in \mathbb{N}$ and $k \in \mathbb{Z}$ we have*

$$\varphi_{j+1,k} = \frac{1}{2}\left(\sum_{\ell\in\mathbb{Z}} h_{k-2\ell}\varphi_{j,\ell} + (-1)^k \sum_{\ell\in\mathbb{Z}} h_{1-k+2\ell}\psi_{j,\ell}\right). \tag{8.84}$$

Example 8.10.15. Haar wavelets satisfy $h_0 = h_1 = 1$ and $h_i = 0$ for all $i \neq 0, 1$. Thus (8.84) gives

$$\varphi_{j+1,2k} = \frac{1}{2}\sum_{\ell\in\mathbb{Z}} h_{2k-2\ell}\varphi_{j,\ell} + h_{1-2k-2\ell}\psi_{j,\ell} = \frac{1}{2}\left(h_0\varphi_{j,k} + h_1\psi_{j,k}\right),$$

$$\varphi_{j+1,2k+1} = \frac{1}{2}\sum_{\ell\in\mathbb{Z}} h_{2k+1-2\ell}\varphi_{j,\ell} - h_{1-(2k+1)-2\ell}\psi_{j,\ell} = \frac{1}{2}\left(h_1\varphi_{j,k} - h_0\psi_{j,k}\right),$$

which are exactly the relations of Lemma 8.9.4.

8.11 *General Fast Wavelet Transform and Examples

In this section we show how to extend the FWT from the Haar case in Section 8.9 to general discrete wavelets, thus providing a general computational framework for wavelet decomposition. As an example, we discuss the famous Daubechies wavelets, which are used widely in applications. We conclude by showing the Daubechies wavelet decomposition of an ECG.

8.11.1 Sampling for General Wavelets

The FWT gives a change of basis to express a function $f \in V_j$ in terms of the daughter wavelets. In particular, since

$$V_j = V_i \oplus_\perp W_i \oplus_\perp W_{i+1} \oplus_\perp \cdots \oplus_\perp W_{j-2} \oplus_\perp W_{j-1}, \tag{8.85}$$

we can use the DWT to decompose f as $f = \tilde{v}_i + \tilde{w}_i$, where $\tilde{v}_i \in V_i$ is the approximation and $\tilde{w}_i \in V_i^\perp = W_i \oplus_\perp W_{i+1} \oplus_\perp \cdots \oplus_\perp W_{j-2} \oplus_\perp W_{j-1}$ is the detail (see, for example, (8.63)). But before we can do this, we must have the function f (or a sample of f) expressed in terms of the basis $\{\varphi_{j,k}\}_{k\in\mathbb{Z}}$ of V_j. In the case of Haar wavelets, this was trivial since we can simply sample the function directly (see (8.66)), but in the general case it's not immediately clear how to take a compactly supported function $f \in L^2_c(\mathbb{R};\mathbb{R})$ and compute the projection $\text{proj}_{V_j} f$ itself or a good approximation of the projection.

Recall that for any function $f \in L^2_c(\mathbb{R};\mathbb{R})$, the nearest element of V_j to f is the orthogonal projection onto V_j given by

$$\text{proj}_{V_j} f = 2^{j/2}\sum_{k=-\infty}^{\infty} \left\langle 2^{j/2}\varphi_{j,k}, f\right\rangle \varphi_{j,k} = 2^j \sum_{k=-\infty}^{\infty} \left(\int_{-\infty}^{\infty} f(x)\varphi_{j,k}(x)\,dx\right)\varphi_{j,k}.$$

Since the support of φ is bounded, the support of $\varphi_{j,k}$ can be made arbitrarily small by choosing j large enough. If f is continuous or otherwise well behaved,

then on a sufficiently small interval, it is almost constant, and the integral above is approximately $2^{-j} f(2^{-j}k) \int_{-\infty}^{\infty} \varphi(x)\,dx$, as the following theorem shows.

Theorem 8.11.1. *If $f \in L^2(\mathbb{R};\mathbb{R})$ is continuous and φ is compactly supported, then for any $\varepsilon > 0$ there exists $j > 0$ such that the coefficients $a_{j,k}$ in the expansion (8.87) of the projection $\operatorname{proj}_{V_j} f$ satisfy*

$$\left|a_{j,k} - \alpha f(2^{-j}k)\right| < \varepsilon,$$

where $\alpha = \int_{-\infty}^{\infty} \varphi(x)\,dx$.

Proof. Assume the support of φ lies in the compact interval $[-L, L]$. Since f is continuous, it is uniformly continuous on $[-L, L]$, so for every $\varepsilon > 0$ there exists a $\delta > 0$ such that $|f(x) - f(y)| < \frac{\varepsilon}{\alpha}$ whenever $|x - y| < \delta$. Choose $j > 0$ such that $2^{-j}L < \delta$. Hence for any $k \in \mathbb{Z}$ the support of $\varphi_{j,k}$ lies in $[2^{-j}(k-L), 2^{-j}(k+L)]$. This implies that

$$\begin{aligned} 2^{-j} a_{j,k} = \int_{-\infty}^{\infty} f(x)\varphi_{j,k}(x)\,dx &= \int_{2^{-j}(k-L)}^{2^{-j}(k+L)} f(x)\varphi(2^j x - k)\,dx \\ &= 2^{-j} \int_{-L}^{L} f(2^{-j}(t+k))\varphi(t)\,dt \\ &= 2^{-j} f(x_0) \int_{-\infty}^{\infty} \varphi(t)\,dt = 2^{-j} f(x_0)\alpha \end{aligned}$$

for some $x_0 = 2^{-j}(k + t_0) \in [2^{-j}(k-L), 2^{-j}(k+L)]$. Since

$$|2^{-j}(k + t_0) - 2^{-j}k| = 2^{-j}|t_0| < 2^{-j}L < \delta,$$

it follows that

$$|f(2^{-j}(k + t_0)) - f(2^{-j}k)| < \frac{\varepsilon}{\alpha}.$$

Thus

$$|a_{j,k} - \alpha f(2^{-j}k)| = |\alpha f(x_0) - \alpha f(2^{-j}k)| = \alpha |f(x_0) - f(2^{-j}k)| < \varepsilon. \quad \square$$

Remark 8.11.2. Since $a_{j,k} \approx \alpha f(2^{-j}k)$, when j is sufficiently large the projection $\operatorname{proj}_{V_j} f$ is well approximated by the sampled sum

$$\operatorname{proj}_{V_j} f = \sum_{k\in\mathbb{Z}} a_{j,k}\varphi_{j,k} \approx \alpha \sum_{k\in\mathbb{Z}} f(2^{-j}k)\varphi_{j,k}. \tag{8.86}$$

Now that we have the approximate the projection, we can consider the DWT.

8.11.2 The FWT for General Wavelets

Fix a value of $j \in \mathbb{N}$ and let $n = 2^j$. Given any function $f \in V_j$ with compact support, assume $f \in \operatorname{span}(\{\varphi_{j,k}\}_{k=0}^{nM-1})$ for some M so that we can write

$$f = \sum_{k=0}^{nM-1} a_{j,k}\varphi_{j,k}. \tag{8.87}$$

As in the case of Haar wavelets, the relations (8.84) can be applied iteratively to write f in terms of the orthogonal wavelet basis

$$\varphi_{0,0}, \dots, \varphi_{0,M-1}, \psi_{0,0}, \dots, \psi_{0,M-1}\psi_{1,0}, \dots, \psi_{1,2M-1}, \dots, \psi_{j-1,2^{j-1}M-1}.$$

Theorem 8.10.13 guarantees that f can be written in terms of the bases for V_{j-1} and W_{j-1} as

$$f = \sum_{k\in\mathbb{Z}} a_{j-1,k}\varphi_{j-1,k} + \sum_{k\in\mathbb{Z}} b_{j-1,k}\psi_{j-1,k}. \tag{8.88}$$

A calculation similar to the proof of Theorem 8.9.5 gives the following theorem.

Theorem 8.11.3. *For $j \in \mathbb{Z}^+$, the coefficients $a_{j-1,k}$ and $b_{j-1,k}$ in (8.88) satisfy*

$$a_{j-1,k} = \frac{1}{2}\sum_{\ell\in\mathbb{Z}} h_{\ell-2k}a_{j,\ell} \quad \textit{and} \quad b_{j-1,k} = \frac{1}{2}\sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{1-\ell+2k}a_{j,\ell}. \tag{8.89}$$

Proof. Taking the inner product of f with the basis function $\varphi_{j-1,k}$ is

$$\begin{aligned} 2^{1-j}a_{j-1,k} = \langle \varphi_{j-1,k}, f\rangle &= \left\langle \sum_{\ell\in\mathbb{Z}} h_{\ell-2k}\varphi_{j,\ell}, \sum_{m\in\mathbb{Z}} a_{j,m}\varphi_{j,m} \right\rangle \\ &= \sum_{\ell\in\mathbb{Z}}\sum_{m\in\mathbb{Z}} h_{\ell-2k}a_{j,m}\langle \varphi_{j,\ell}, \varphi_{j,m}\rangle = 2^{-j}\sum_{\ell\in\mathbb{Z}} h_{\ell-2k}a_{j,\ell}. \end{aligned}$$

Similarly, the inner product of f with the basis function $\psi_{j-1,k}$ gives

$$\begin{aligned} 2^{1-j}b_{j-1,k} = \langle \psi_{j-1,k}, f\rangle &= \left\langle \sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{1-\ell+2k}\varphi_{j,\ell}, \sum_{m\in\mathbb{Z}} a_{j,m}\varphi_{j,m} \right\rangle \\ &= \sum_{\ell\in\mathbb{Z}}\sum_{m\in\mathbb{Z}} (-1)^\ell h_{1-\ell+2k}a_{j,m}\langle \varphi_{j,\ell}, \varphi_{j,m}\rangle = 2^{-j}\sum_{\ell\in\mathbb{Z}} (-1)^\ell h_{1-\ell+2k}a_{j,\ell}. \end{aligned}$$

□

Theorem 8.11.3 shows that the same idea used for the Haar case gives an FWT for general wavelets constructed from a more general scaling function φ. The algorithm is outlined in Algorithm 8.4.

Remark 8.11.4. If there are K nonzero coefficients h_k in the scaling relation (8.71), then the ith step of the FWT iteration (Algorithm 8.4) requires $2^{i-1}K$ multiplications and $2^{i-1}(K-1)$ additions (or subtractions). This adds up to a total of $2^{i-1}(2K-1)$ FLOPs per iteration. Computing the full wavelet transform by this method requires $\sum_{i=1}^{j} 2^{i-1}(2K-1) = (2^j-1)(2K-1) \sim 2^{j+1}K = 2Kn$ FLOPs.

8.11.3 The Daubechies Wavelet

In the 1980s Ingrid Daubechies developed a new class of wavelets, where both the father function φ and the mother function ψ are continuous. The canonical example from this class is known as the *Daubechies db2* scaling function and wavelet. These

(i) Initialize by setting $i = j$.

(ii) For each $k \in \{0, 1, 2, \ldots, 2^{i-1}M - 1\}$, compute (8.89).

(iii) Decrement i: $\quad i \leftarrow i - 1$.

(iv) If $i > 0$, then repeat from step (ii).

Algorithm 8.4. *Outline of the FWT algorithm for general wavelets. Given f as in (8.87), determined by coefficients $\mathbf{a}_j = (a_{j,0}, \ldots, a_{j,nM-1})$, this algorithm computes the coefficients $\mathbf{a}_{j-1}$ and $\mathbf{b}_{j-1}$ of the decomposition (8.88) and then continues to compute $\mathbf{a}_i$ and $\mathbf{b}_i$ iteratively for all $i \in \{j-1, \ldots, 0\}$.*

functions are continuous but cannot be written down in terms of elementary functions; see Figure 8.22. One advantage of the continuous wavelets is that they tend to approximate continuous functions more efficiently than discontinuous wavelets do. After describing the Daubechies *db*2 wavelets, we apply them to the ECG image in Figure 8.21.

The Daubechies *db*2 scaling function $\varphi : \mathbb{R} \to \mathbb{R}$ is characterized by the scaling rule (8.71), satisfying

$$\varphi(x) = h_0\varphi(2x) + h_1\varphi(2x-1) + h_2\varphi(2x-2) + h_3\varphi(2x-3), \tag{8.90}$$

with scaling coefficients

$$h_0 = \frac{1+\sqrt{3}}{4}, \quad h_1 = \frac{3+\sqrt{3}}{4}, \quad h_2 = \frac{3-\sqrt{3}}{4}, \quad \text{and} \quad h_3 = \frac{1-\sqrt{3}}{4}, \tag{8.91}$$

and the corresponding mother wavelet $\psi : \mathbb{R} \to \mathbb{R}$, which is similarly characterized

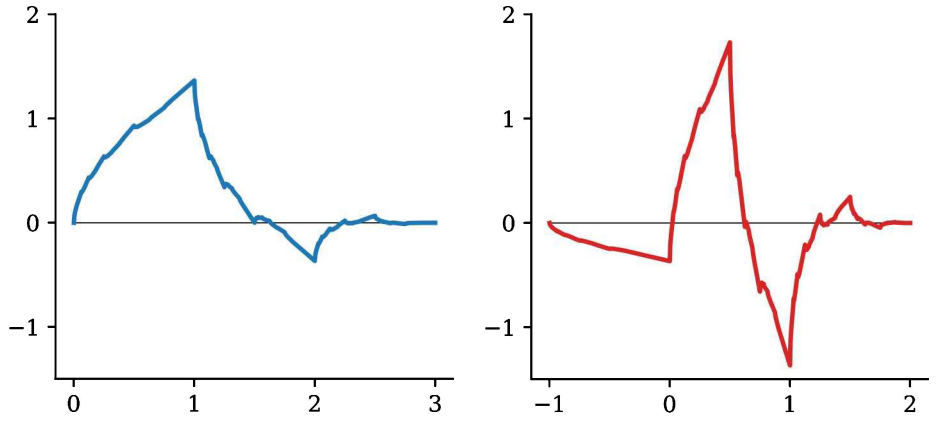


Figure 8.22. *The Daubechies db2 scaling function φ (left), and the mother wavelet ψ (right).*

by the rule (8.78), satisfying

$$\psi(x) = -h_0\varphi(2x-1) + h_1\varphi(2x) - h_2\varphi(2x+1) + h_3\varphi(2x+2).$$

These coefficients are derived from the fact that the Daubechies $db2$ wavelet has a first moment of zero, that is,

$$\int_{-\infty}^{\infty} x\psi(x)\,dx = 0. \tag{8.92}$$

The only choice of four coefficients h_0, h_1, h_2, and h_3 that satisfy the scaling identities in Proposition 8.10.9, as well as (8.92), are those given by (8.91); see Exercise 8.56 for details.

To construct φ and ψ, first define iterative sequences of functions $(\varphi_n)_{n\in\mathbb{N}}$ and $(\psi_n)_{n\in\mathbb{N}}$ by

$$\varphi_{n+1}(x) = h_0\varphi_n(2x) + h_1\varphi_n(2x-1) + h_2\varphi_n(2x-2) + h_3\varphi_n(2x-3)$$

and

$$\psi_{n+1}(x) = -h_0\varphi_n(2x-1) + h_1\varphi_n(2x) - h_2\varphi_n(2x+1) + h_3\varphi_n(2x+2)$$

with initial functions $\varphi_0 = \psi_0 = \mathbb{1}_{(0,1]}$. It can be shown that both φ_n and ψ_n converge uniformly to their limiting functions φ and ψ.

As an alternative construction, we can determine the values of φ and ψ pointwise on each $\frac{k}{2^j}$, where for each fixed $j \in \mathbb{N}$, the numerators k vary between 0 and $3\cdot 2^j$. As j gets larger, the distance between adjacent points grows smaller, and in the limit as $j \to \infty$, we arrive at the values of the functions φ and ψ on a dense set.

To do this start with the initial values

$$\varphi(1) = \frac{1+\sqrt{3}}{2} \qquad \text{and} \qquad \varphi(2) = \frac{1-\sqrt{3}}{2},$$

and assume that $\varphi(x) = 0$ for all $x \notin (0,3)$. Now use (8.90) to find $\varphi(\frac{1}{2})$, $\varphi(\frac{3}{2})$, and $\varphi(\frac{5}{2})$. Then, using the values at $\varphi(\frac{1}{2})$, $\varphi(1)$, $\varphi(\frac{3}{2})$, $\varphi(2)$, and $\varphi(\frac{5}{2})$, we can use (8.90) to compute the values $\varphi(\frac{1}{4})$, $\varphi(\frac{3}{4})$, $\varphi(\frac{5}{4})$, $\varphi(\frac{7}{4})$, $\varphi(\frac{9}{4})$, and $\varphi(\frac{11}{4})$. In other words, by having all of the nonzero values of $\varphi(\frac{k}{2^j})$, for fixed $j \in \mathbb{N}$, we can find all the values of $\varphi(\frac{j}{2^{j+1}})$. Then we can increment j and repeat until we have a dense representation of φ on the interval $(0,3)$. To find the mother wavelet, we follow a similar procedure, but on (8.78) instead. Plots of the Daubechies scaling function and mother wavelets are given in Figure 8.22; see Algorithm 8.5 for the code that generated the figures.

8.11.4 Convolutional Form of FWT

Further inspection of (8.89) shows that the sums can be written as convolutions followed by what is called *downsampling*. We conclude this section by demonstrating how to compute a single iteration of the DWT algorithm using this approach. We apply this technique on the ECG signal in Figure 8.21.

```python
import numpy as np

def daubechies(j):
    """ Produce a sample of the Daubechies d2 scaling function
    and mother wavelet on the interval [0,3].
    """
    signs = np.array([1,-1,1,-1])
    h = (np.array([1,3,3,1])+np.sqrt(3)*np.array([1,1,-1,-1]))/4
    phi = np.zeros(3*2**j+1);
    psi = np.zeros(3*2**j+1);

    idx0 = (2**j)*np.array([1,2]);
    phi[idx0] = 2*np.array([h[0], h[3]]);
    psi[idx0] = 2*np.array([h[3], -h[0]]);

    for k in range(j):
        idx1 = 2**(j-k-1)*np.arange(1,3*2**(k+1),2)
        for l in idx1:
            z = 2*l - (2**j)*np.array([0,1,2,3])
            z = np.array([a*int(a in idx0) for a in z])
            phi[l] = np.dot(h,phi[z]);
            psi[l] = np.dot(h[::-1]*signs,phi[z]);
        idx0 = idx1
    return phi, psi
```

Algorithm 8.5. *The Daubechies algorithm for generating values of the db2 scaling function φ and corresponding mother wavelet ψ, sampled at points of the form $\frac{k}{2^j}$ for $k \in \{0, 1, 2, \ldots, 3 \cdot 2^j\}$.*

Writing the convolution operator as

$$(\mathbf{x} * \mathbf{y})_\ell = \sum_{k \in \mathbb{Z}} x_k y_{\ell - k}$$

simplifies the expressions in (8.89). Let $\mathbf{a}_j = (a_{j,k})_{k \in \mathbb{Z}}$ and $\mathbf{b}_j = (b_{j,k})_{k \in \mathbb{Z}}$ denote the sequence of coefficients, all but a finite number of which are zero. Reindexing $k \to 2\ell - k$ gives the approximation coefficients as

$$a_{j-1,\ell} = \frac{1}{2} \sum_{k \in \mathbb{Z}} h_{k-2\ell} a_{j,k} = \frac{1}{2} \sum_{k \in \mathbb{Z}} h_{-k} a_{j,2\ell-k} = (L * \mathbf{a}_j)_{2\ell},$$

where

$$L = \frac{1}{2}(h_3, h_2, h_1, h_0).$$

The vector L is often called a *low-pass filter*.

Similarly, reindexing $k \to 2\ell - k$ gives the detail coefficients as

$$b_{j-1,\ell} = \frac{1}{2} \sum_{k \in \mathbb{Z}} (-1)^k h_{1-k+2\ell} a_{j,k} = \frac{1}{2} \sum_{k \in \mathbb{Z}} (-1)^k h_{1+k} a_{j,2\ell-k} = (H * \mathbf{a}_j)_{2\ell},$$

where

$$H = \frac{1}{2}\big[-h_0, h_1, -h_2, h_3\big].$$

The vector H is often called a *high-pass filter*.

Both expressions include only the even values of the convolution. We can skip the odd-numbered elements via the *downsampling* operator

$$D\mathbf{x} = (\ldots, x_{-2}, x_0, x_2, \ldots).$$

In other words, we have

$$\mathbf{a}_{j-1} = D(L * \mathbf{a}_j) \qquad \text{and} \qquad \mathbf{b}_{j-1} = D(H * \mathbf{a}_j). \tag{8.93}$$

Remark 8.11.5. In actual code it's customary to implement edge conditions at the beginning and the end of the signal to give the convolution operator extra padding. This is done by adding $K-1$ values at the beginning and the end of the sample, where K is the length of the filters L and H ($K = 4$ in the case of the Daubechies *db*2 wavelet). The leading and trailing padding most used is just the $K-1$ elements of the signal, respectively, in reverse order. So if the actual signal is $a_0, a_1, \ldots, a_n$ and $K = 4$, then we feed the following as the signal into the convolution formulas (8.93):

$$a_3, a_2, a_1, a_0, a_1, a_2, \ldots, a_n, a_{n-1}, a_{n-2}, a_{n-3}.$$

Example 8.11.6. Applying Algorithm 8.6 to the ECG data in Figure 8.21 gives the decompositions shown in Figure 8.23. With each iteration of the DWT, the number of points halves the previous input and therefore so does the number of coefficients required to store the signal. Applying the DWT three times yields a visually similar ECG signal compared to the original, despite being $\frac{1}{8} = 2^{-3}$ the number of coefficients.

```python
import numpy as np

def dwt(X,H,L):
    """ The DWT algorithm using convolutions and downsampling.
    """
    K = len(H)
    l = np.flip(X[0:K-1],0) # left padding
    r = np.flip(X[-(K-1)::],0) # right padding
    X = np.concatenate((l,X,r),axis=0)
    cD = np.convolve(X, H, 'valid')[1::2] # detail
    cA = np.convolve(X, L, 'valid')[1::2] # approximation
    return cA, cD
```

Algorithm 8.6. *Algorithm for a single iteration of the DWT via convolution and downsampling. Left and right padding are attached to the signal as described in Remark 8.11.5.*

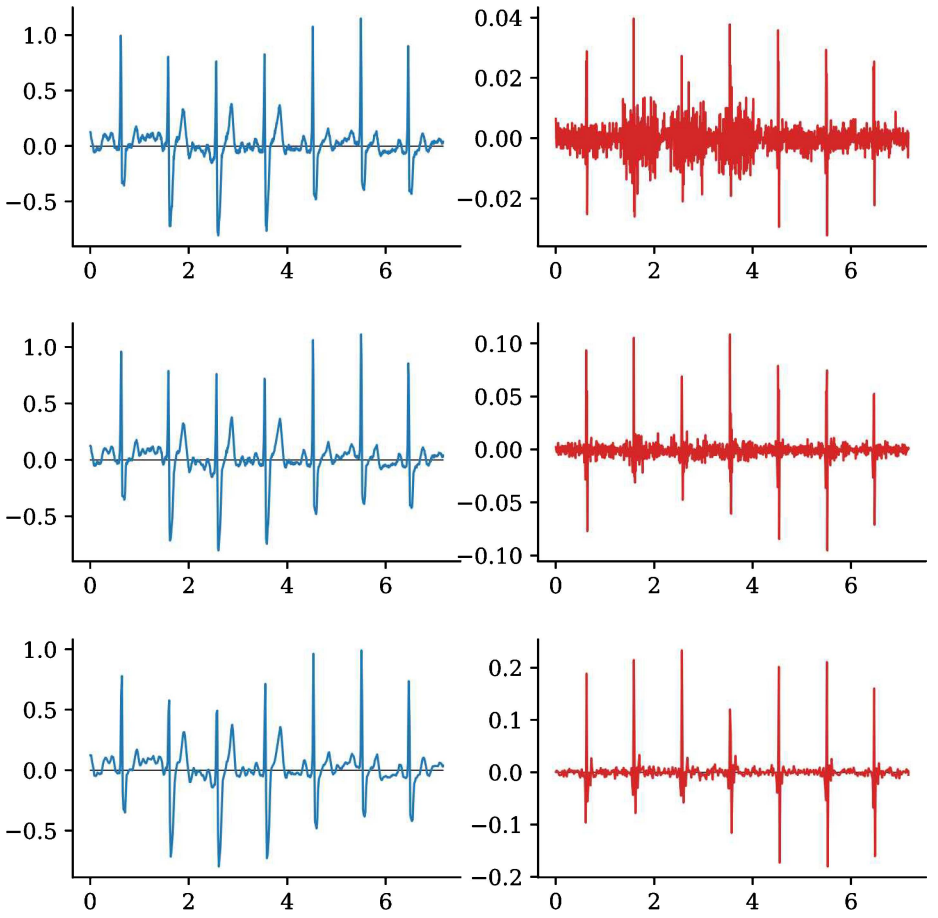


Figure 8.23. *The top pair represents the first db2 wavelet decomposition, taking* 3580 *points from the ECG signal in Figure 8.21 and splitting it into the approximation (left) and the detail (right), each consisting of* 1791 *points. The middle pair is the second db2 wavelet decomposition, resulting in* 897 *points for both the approximation (left) and the detail (right). Finally the bottom pair is the third db2 wavelet decomposition, resulting in* 450 *points for both the approximation (left) and the detail (right). All three approximations are strikingly similar to the original.*

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them,

and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

8.1. Show that

$$\frac{e^{i(b-a)}}{b-a} + \frac{e^{i(a-b)}}{a-b} = \frac{2i\sin(a-b)}{a-b}.$$

8.2. Show for all $k \in \mathbb{Z}$, with $k \neq 0$, that

$$\int_0^{2\pi} e^{ikt}\, dt = 0.$$

8.3. For any $z, w \in \mathbb{C}$ prove that

(i) $\overline{zw} = \bar{z}\bar{w}$,

(ii) $\overline{z \pm w} = \bar{z} \pm \bar{w}$,

(iii) $\overline{\left(\frac{z}{w}\right)} = \frac{\bar{z}}{\bar{w}}$.

8.4. If $|z| = 1$, and $a, b \in \mathbb{C}$, prove that

$$\left|\frac{az+b}{\bar{b}z+\bar{a}}\right| = 1.$$

Hint: Remember that $|w|^2 = w\bar{w}$, and $\overline{rw+s} = \bar{r}\bar{w} + \bar{s}$.

8.5. Express each of the following complex numbers in polar form (i.e., $z = re^{i\theta}$):

(i) $2 + 2\sqrt{3}i$.

(ii) $-2 + 2i$.

8.6. Find all the complex numbers ζ satisfying the relation $\zeta^2 + \zeta + 1 = 0$ as follows:

(i) First determine how many solutions exist.

(ii) Show that any such ζ must satisfy $\zeta^3 = 1$ and $\zeta \neq 1$.

(iii) Use the previous step to find the polar form of all the solutions.

(iv) Now solve the problem using the rectangular form by writing $\zeta = x + iy$ and computing $(x + iy)^2 + (x + iy) + 1$. Setting both the real and imaginary parts to zero gives two equations in two unknowns, whose solutions give the required values of x and y.

(v) Show that the answers you got in polar form agree with the answers you got in rectangular form.

8.7. Simplify the following into the form $a + ib$, where $a, b \in \mathbb{R}$. Is the solution unique? Justify your answer.

(i) $\sqrt{i}$.

(ii) $\sqrt{1+i}$.

(iii) $\sqrt{\sqrt{-i}}$.

8.8. Let $f \in L^2([0,T];\mathbb{F})$. Prove the "almost converse" of Proposition 8.2.12: If $\overline{c_k} = c_{-k}$ for all $k \in \mathbb{Z}$, then $S[f]$ is real valued on $[0,T]$.

8.9. From Proposition 8.2.12, we know that the Fourier coefficients of the function $f \in L([0,T];\mathbb{R})$ satisfy $c_{-k} = \overline{c_k}$. Thus the Fourier series satisfies

$$\begin{aligned} S[f](t) = \sum_{k\in\mathbb{Z}} c_k e^{i\omega k t} &= c_0 + \sum_{k\in\mathbb{Z}^+} \overline{c_k} e^{-i\omega k t} + c_k e^{i\omega k t} \\ &= c_0 + \sum_{k\in\mathbb{Z}^+} (c_k + \overline{c_k})\cos(\omega k t) + i(c_k - \overline{c_k})\sin(\omega k t). \end{aligned}$$

Show that for each $k \in \mathbb{Z}^+$ the following holds:

$$(c_k + \overline{c_k})\cos(\omega k t) + i(c_k - \overline{c_k})\sin(\omega k t) = 2|c_k|\cos(\omega k t + \phi_k),$$

where the real and imaginary parts of c_k satisfy $\Re(c_k) = |c_k|\cos(\phi_k)$ and $\Im(c_k) = |c_k|\sin(\phi_k)$, respectively. In other words, we can decompose the Fourier series $S[f]$ into a sum

$$S[f](t) = c_0 + \sum_{k=1}^{\infty} 2|c_k|\cos(\omega k t + \phi_k) \tag{8.94}$$

of linear oscillators, each having frequency k/T, amplitude $2|c_k|$, and phase angle ϕ_k.

8.10. Find the complex-exponential Fourier series of the function $f(t) = \sin(5\omega t)$ on the interval $[0,T]$.

8.11. Find the complex-exponential Fourier series of the function

$$f(t) = \begin{cases} 0 & \text{if } t \in \{0, 2\pi\}, \\ \pi - t & \text{if } 0 < t < 2\pi \end{cases} \tag{8.95}$$

on the interval $[0, 2\pi]$. By Theorem 8.2.16 the Fourier series converges pointwise to f on $[0,2\pi]$. Hint: We already computed the Fourier series of the sawtooth function in Example 8.2.6.

8.12. Find the complex-exponential Fourier series of the function

$$f(x) = \begin{cases} 1 & \text{if } 0 < x < \pi, \\ 0 & \text{if } x \in \{0, \pi, 2\pi\}, \\ -1 & \text{if } \pi < x < 2\pi \end{cases} \tag{8.96}$$

on the interval $[0, 2\pi]$. Hint: We already computed the Fourier series of the square wave function in Example 8.2.10.

8.13. Show that if $\overline{c_k} = c_{-k}$, then the following hold:

(i) $\overline{a_k} = a_k$ and $\overline{b_k} = b_k$.

(ii) $a_k^2 + b_k^2 = 4c_k c_{-k}$.

(iii) $\sqrt{a_k^2 + b_k^2} = 2|c_k|$.

8.14. Find the trigonometric Fourier series of the function f in Exercise 8.12.

8.15. Use the results from Exercise 8.14 to prove that the following equalities hold:

(i)
$$\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{2k-1} = \frac{\pi}{4}.$$

(ii)
$$\sum_{k=1}^{\infty} \frac{1}{(2k-1)^2} = \frac{\pi^2}{8}.$$

Hint: Show that $\|f\|^2 = 2$.

8.16. Many textbooks develop Fourier series on the domain $[-\pi, \pi]$ instead of $[0, T]$. In this case, the trigonometric Fourier series $S[f]$ of $f : [-\pi, \pi] \to \mathbb{R}$ is still of the form (8.31), but the Fourier coefficients are given by

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos(kt)\, dt, \tag{8.97}$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin(kt)\, dt. \tag{8.98}$$

We call this approach the *centered* trigonometric Fourier series. Now consider the function

$$f(t) = \begin{cases} 1 & \text{if } t \in (-\frac{\pi}{2}, \frac{\pi}{2}), \\ 0 & \text{if } t \in [-\pi, -\frac{\pi}{2}) \cup (\frac{\pi}{2}, \pi], \\ \frac{1}{2} & \text{if } t = \pm\frac{\pi}{2} \end{cases}$$

defined on the interval $[-\pi, \pi]$. Compute the centered trigonometric Fourier series using (8.97) and (8.98).

8.17. Consider the function $f(t) = |t|$ on the interval $[-\pi, \pi]$. Write f as a centered trigonometric Fourier series (see Exercise 8.16). Hint: Write

$$|t| = \begin{cases} t & \text{if } t \geq 0, \\ -t & \text{if } t < 0, \end{cases}$$

and then integrate by parts.

8.18. Prove that the Dirichlet kernel D_n satisfies the following for each $t \in \mathbb{R}$:

(i) $D_n(t) = D_n(2\pi - t)$.

(ii) $D_n(\pi + t) = D_n(\pi - t)$.

8.19. For $f \in L([0, 2\pi]; \mathbb{R})$, extend f to all of $\mathbb{R}$ by letting $f(t) = f(t - 2\pi)$ for all $t > 2\pi$ and $f(t) = f(t + 2\pi)$ for all $t < 0$. Prove the following identities:

(i)

$$\int_0^{2\pi} f(s)D_n(t-s)\,ds = \int_0^{2\pi} f(t-s)D_n(s)\,ds = \int_{-\pi}^{\pi} f(t-s)D_n(s)\,ds.$$

(ii)

$$\int_0^{2\pi} f(s)D_n(t-s)\,ds = \int_0^{2\pi} f(t+s)D_n(s)\,ds = \int_{-\pi}^{\pi} f(t+s)D_n(s)\,ds.$$

Hint: Use integration and change of variable rules with Lemma 8.4.6.

8.20. Prove Lemma 8.4.9. Hint: Use the geometric series formula for (8.36).

8.21. Consider the function

$$f(t) = \begin{cases} t & \text{if } t \in (0, 2\pi), \\ \pi & \text{if } t = 0 \text{ or } t = 2\pi. \end{cases}$$

Let $g_n(t) = f(t) - S_n[f](t)$ be the approximation error between the function f and the nth partial sum of its Fourier series (see Example 8.2.6).

(i) Plot $g_n(x)$ for different values of n to demonstrate the *Gibbs phenomenon*.

(ii) Show that

$$g_n'(t) = \frac{\sin\left((2n+1)t/2\right)}{\sin(t/2)}.$$

Hint: Use Lemmata 8.4.4 and 8.4.9.

(iii) Show that $t_n = \frac{2\pi}{2n+1}$ is the first critical point of g_n to the right of zero. Hence the L^∞-norm (supremum) of the approximation error is at least $g_n(t_n)$.

8.22. Continuing from the previous problem, complete the following steps:

(i) Using the fundamental theorem of calculus, show that

$$g_n(t_n) = \int_0^{t_n} \frac{\sin\left((2n+1)t/2\right)}{\sin(t/2)}\,dt - \pi.$$

Hint: Recall that the fundamental theorem of calculus will apply only to functions that are continuous on the interval $[0, t_n]$, so you will need to replace the integrand with a function that is continuous on that interval.

(ii) Prove that

$$\lim_{n\to\infty} g_n(t_n) = 2\int_0^{\pi} \frac{\sin(t)}{t}\,dt - \pi.$$

(iii) Evaluate the integral numerically to show that

$$2\int_0^{\pi} \frac{\sin(t)}{t}\,dt - \pi \approx 0.562.$$

This shows that the nth truncated Fourier series $S_n[f](t)$ does not converge uniformly to $f(t)$, despite the fact that it converges pointwise. Hint: You can compute the integral with a Riemann sum.

8.23. Compute the following:

(i) Find the DFT of $\mathbf{f} = (1, 1, 0, 0)$.

(ii) Find the inverse DFT of $\widehat{\mathbf{f}} = (1, 0, 1, 0)$.

8.24. (i) If $p(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ is an arbitrary polynomial, what is the value of $p(\omega_n^\ell)$ for arbitrary $\ell \in \mathbb{Z}$? Explain carefully how to use the matrix representation of the DFT to compute this.

(ii) A fairly fast way to evaluate a general polynomial at a single point is *Horner's method*:

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-2} + xa_{n-1}) \dots)),$$

which takes $O(n)$ time. But to evaluate p at n distinct points using Horner's method takes $O(n^2)$ time. Explain carefully how to compute the evaluation of p at the n distinct points $\omega_n^0, \dots, \omega_n^{n-1}$ in $O(n \log(n))$ time using the FFT.

8.25. Prove that applying the DFT twice to $\mathbf{f} = (f_0, f_1, \dots, f_{n-2}, f_{n-1})$ gives $W_n^2 \mathbf{f} = \frac{1}{n}(f_0, f_{n-1}, f_{n-2}, \dots, f_2, f_1)$.

8.26. A matrix $A \in M_n(\mathbb{F})$ is *circulant* if it is of the form

$$A = \begin{bmatrix} a_0 & a_{n-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{bmatrix}.$$

(i) Show that the circulant matrix can be diagonalized by the DFT. That is, show that if $U_n = \sqrt{n} W_n$ is the orthonormal matrix of Theorem 8.5.10, then $U_n^{\mathsf{H}} \operatorname{diag}(c_0, \dots, c_{n-1}) U_n = A$ for some $(c_0, \dots, c_{n-1})$.

(ii) Find the eigenvalues of A.

8.27. The DFT approximates a function $f(t)$ using the orthonormal set $\{e^{i\omega k t}\}_{k=0}^{N-1}$, but if $f(t) = e^{i\omega \lambda t}$ with $\lambda \in \mathbb{R} \setminus \mathbb{Z}$ corresponds to a frequency that is not in that set, we see nonzero effects in all the DFT frequencies k/N. This is called *leakage* from the actual frequency λ/N to the other frequencies k/N.

(i) Prove that the modulus $|\hat{f}(k)|$ of the kth discrete Fourier coefficient $\hat{f}(k)$ of f is given by

$$|\hat{f}(k)| = \left| \frac{\sin \pi(\lambda - k)}{N \sin \frac{\pi(\lambda - k)}{N}} \right|.$$

This formula gives the so-called *leakage amplitude* for the DFT of a noncharacteristic sinusoidal frequency.

(ii) Plot $|\hat{f}(k)|$ as a function of k for several values of λ, taking $N = 100$.

8.28. Let $\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{k}$ be the 4-periodic vectors

$$\begin{aligned}\mathbf{f} &= (1, 2, 3, 4),\\ \mathbf{g} &= (0, 0, 1, 0),\\ \mathbf{h} &= (1, i, -1, -i),\\ \mathbf{k} &= (1, -1, 1, -1).\end{aligned}$$

Compute $\mathbf{f} * \mathbf{g}$, $\mathbf{g} * \mathbf{h}$, $\mathbf{g} * \mathbf{g}$, $\mathbf{h} * \mathbf{k}$, and $\mathbf{h} * \mathbf{h}$.

8.29. Using the same notation as the previous problem,

(i) compute the DFT of each of the vectors $\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{k}$;

(ii) find the products $W_4\mathbf{f} \odot W_4\mathbf{g}$, $W_4\mathbf{g} \odot W_4\mathbf{h}$, $W_4\mathbf{g} \odot W_4\mathbf{g}$, $W_4\mathbf{h} \odot W_4\mathbf{k}$, and $W_4\mathbf{h} \odot W_4\mathbf{h}$;

(iii) verify that $4W_4^{-1}$ applied to each of these products agrees with the corresponding convolution that you computed in the previous problem.

8.30. The naïve algorithm for multiplying two polynomials

$$f = \sum_{k=0}^{n} a_k x^k \qquad \text{and} \qquad g = \sum_{k=0}^{n} b_k x^k$$

of degree $n \in \mathbb{Z}^+$ is to compute every term of

$$fg = \sum_{\ell=0}^{2n} c_\ell x^\ell$$

as

$$c_\ell = \sum_{j=0}^{\ell} a_j b_{\ell - j}.$$

This naïve computation has temporal complexity $O(n^2)$. Explain how to use the DFT to multiply two degree-n polynomials in $O(n \log n)$ time. Hint: Use Exercise 8.24 and the fact that a polynomial of degree $2n$ is uniquely determined by its values at $2n + 1$ distinct points.

8.31. Let A be the circulant matrix

$$A = \begin{bmatrix} a_0 & a_{n-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{bmatrix}$$

and let $\mathbf{g} = (g_0, g_1, \ldots, g_{n-2}, g_{n-1})$. Show that $A\mathbf{g}$ is precisely $\mathbf{a} * \mathbf{g}$, where $\mathbf{a}$ is the periodic vector $\mathbf{a} = (a_0, a_1, \ldots, a_{n-2}, a_{n-1})$.

8.32. Let $f = x$ and $g = \sin(x)$. Using your preferred computational tools, sample both of these functions 1000 times on the interval $[0, 2\pi]$ to get vectors $\mathbf{f}$ and $\mathbf{g}$. Treating $\mathbf{f}$ and $\mathbf{g}$ as periodic vectors, compute the (circular) convolution $\mathbf{f} * \mathbf{g}$ and the Hadamard product $\mathbf{f} \odot \mathbf{g}$. Plot the points (x_k, y_k) where $x_k =$

$2\pi k/1000$ and y_k is the kth coordinate of each of the vectors $\mathbf{f}$, $\mathbf{g}$, $\mathbf{f} * \mathbf{g}$, and $\mathbf{f} \odot \mathbf{g}$. Plot them all separately and then plot them together on the same graph.

8.33. Given $n > 2\nu$ samples $(t_0, f(t_0)), \ldots, (t_{n-1}, f(t_{n-1}))$, implement the periodic sampling theorem algorithm to compute the exponential Fourier series of a function $f : [0, T] \to \mathbb{R}$ with Nyquist frequency ν. Your code should accept integers ν and n, a floating point number T, and a function f and return an array of coefficients $[c_{-\nu}, \ldots, c_0, \ldots, c_\nu]$. Also, implement a method that returns $g(t) = \sum_{k=-\nu}^{\nu} c_k e^{i\omega k t}$ as a callable function.

8.34. Find the period T and the Nyquist frequency of the function

$$f(x) = 1 - 3\sin(12\pi x + 7) + 5\sin(2\pi x - 1) + 5\sin(4\pi x - 3).$$

For each value of $n \in \{3, 7, 11, 13\}$ use your code from the previous problem to sample f at the n points $t_0, \ldots, t_{n-1}$ with $t_\ell = \frac{\ell T}{n}$, and find the unique function $g_n(x)$ with Nyquist frequency less than $n/2$ passing through those n points. Plot f, g_n, and the n sample points on the same graph.

8.35. For any $\nu \in \mathbb{N}$ prove that the set V of all functions on $[0, T]$ with Nyquist frequency no greater than ν is a vector space. Let f have Nyquist frequency ν. Assume f is sampled at $n = 2\nu$ uniformly spaced points on $[0, T)$ as in Definition 8.5.1. Prove that the subset of V consisting of functions that vanish at all the sampled points is a subspace of V of dimension at least 1; therefore, there are an infinite number of functions agreeing with f at all the samples. This shows the conclusion of the periodic sampling theorem fails for $n = 2\nu$.

8.36. Let V be the set of functions on $[0, T]$ with Nyquist frequency ν or less. Fix $n > 2\nu + 1$, and let $\mathbf{f} = \Phi_n(f) = (f_0, \ldots, f_{n-1})$ be the corresponding sampled vector. Prove that there exists at least one vector $\mathbf{a} \in \mathbb{C}^n$ such that for every $f \in V$ we have $\mathbf{a}^{\mathsf{T}}\mathbf{f} = 0$. In particular, this shows that for most choices $\mathbf{y} = (y_0, \ldots, y_{n-1})$ (for all but those satisfying the linear relation $\mathbf{a}^{\mathsf{T}}\mathbf{y} = 0$) there is no function $f \in V$ satisfying $f(t_i) = y_i$ for all $i \in \{0, \ldots, n-1\}$.

8.37. Prove Proposition 8.7.10. Hint: The Fourier basis is orthonormal.

8.38. Prove the Haar scaling relation (8.60).

8.39. Prove Proposition 8.8.5.

8.40. Prove the equality $\psi(2^j x - k) = \varphi_{j+1,2k}(x) - \varphi_{j+1,2k+1}(x)$ from (8.62) holds.

8.41. Prove items (i)–(iv) of Proposition 8.8.7.

8.42. Complete the following:

(i) Code up a method that takes a function $f : [0, 1) \to \mathbb{R}$ and an integer $n > 0$, and returns an array consisting of the values of the function f at the points $k/2^n$ for $k \in \{0, \ldots, 2^n - 1\}$.

(ii) Write a method that accepts an array $\mathbf{a} = [a_0, a_1, \ldots, a_{2^n-1}]$ and a value $x \in [0, 1)$ and returns $\sum_{k=0}^{2^n-1} a_k \varphi_{n,k}(x)$.

(iii) For the function

$$f(t) = \frac{\sin(2\pi t - 5)}{\sqrt{|t - \pi/20|}} \tag{8.99}$$

and for each of the values $n \in \{1, 2, 3, \ldots, 10\}$, use your code to plot f and plot $f_n(x) = \sum_{k=-\infty}^{\infty} f(k/2^n)\varphi_{n,k}(x)$. Hint: It may look like the sum $\sum_{k=0}^{2^n-1} a_k \varphi_{n,k}(x)$ has a lot of terms, but for any x all but one of the terms are zero.

8.43. Consider the function $f : [0, 1) \to \mathbb{R}$ given by

$$f(x) = \begin{cases} -2, & x \in [0, 0.25), \\ 4, & x \in [0.25, 0.5), \\ 2, & x \in [0.50, 0.75), \\ -3, & x \in [0.75, 1.0]. \end{cases}$$

(i) Find the wavelet transform of f by matrix multiplication.

(ii) Find the wavelet transform of f by using the FWT.

(iii) Express f in terms of its component parts V_0, W_0, and W_1.

8.44. Write out the matrix representation of the wavelet transform T_8^{-1} on the subspace of V_3 consisting of functions that have support contained in $[0, 1)$. Consider the function $f : [0, 1) \to \mathbb{R}$ given by

$$f(x) = \begin{cases} 2, & x \in [0, 0.125), \\ 4, & x \in [0.125, 0.25), \\ 3, & x \in [0.25, 0.5), \\ 1, & x \in [0.50, 0.75), \\ -3, & x \in [0.75, 1.0). \end{cases}$$

(i) Find the wavelet transform of f by matrix multiplication.

(ii) Find the wavelet transform of f by the FWT.

(iii) Express f in terms of its component parts W_2, W_1, W_0, and V_0.

8.45. Complete the following:

(i) Code up a function that takes as input the wavelet transform (given by the array $[a_{00}, b_{00}, b_{10}, b_{11}, \ldots]$) and an integer j and returns the approximation $T_j[f] \in V_j$ and detail $g_j \in V_j^{\perp}$ as callable functions.

(ii) Apply your code to the function (8.57) and reproduce the images in Figure 8.20.

8.46. Complete the following:

(i) Use your code from Exercise 8.42 and Exercise 8.45 to construct a method that takes a function f defined on $[0, 1)$ a positive integer ℓ and a positive integer $j < \ell$, samples f to get an approximation f_ℓ, and returns the two functions $T_j[f] \in V_j$ and $g_j \in V_j^{\perp}$ such that $f_\ell = T_j[f] + g_j$.

(ii) Apply your methods to the function (8.99) for $\ell = 10$. For each value of $j \in \{0, 1, \ldots, \ell - 1\}$ plot each of the functions $T_j[f]$ and g_j.

(iii) Compare the plots of each $T_j[f]$ to the corresponding plots in Exercise 8.42. In both cases they are functions in V_j, but they are not identical. Explain the difference. Explain why it occurs.

8.47. Complete the following:

(i) Adapt your code from the previous problem to sample functions on any compact interval $[a, b]$, construct the Haar wavelet transform of the sampled function on that interval, and return the two functions $T_j[f] \in V_j$ and $g_j \in V_j^{\perp}$ such that $f_\ell = T_j[f] + g_j$.

(ii) Apply your method to the function (8.99) on the interval $[-1, 1]$ for $\ell = 10$. For each value of $j \in \{0, 1, \ldots, \ell - 1\}$ plot each of the functions $T_j[f]$ and g_j.

8.48. Prove Proposition 8.10.6.

8.49. Prove that $\sum_{\ell \in \mathbb{Z}} (-1)^\ell h_{1-\ell} = 0$.

8.50. By taking the appropriate inner products, show that the two equations in (8.82) hold.

8.51. Consider the class of scaling functions (8.71) with exactly four coefficients h_0, h_1, h_2, and h_3 satisfying

$$\begin{aligned} h_0 &= \frac{1 + \sqrt{2}\cos\theta}{2}, & h_1 &= \frac{1 + \sqrt{2}\sin\theta}{2}, \\ h_2 &= \frac{1 - \sqrt{2}\cos\theta}{2}, & h_3 &= \frac{1 - \sqrt{2}\sin\theta}{2}. \end{aligned} \tag{8.100}$$

Prove that the three identities in Proposition 8.10.9 hold, specifically:

(i) Following (8.75), show that $h_0^2 + h_1^2 + h_2^2 + h_3^2 = 2$ and $h_0 h_2 + h_1 h_3 = 0$.

(ii) Following (8.76), show that $h_0 + h_1 + h_2 + h_3 = 2$.

(iii) Following (8.77), show that $h_0 + h_2 = 1$ and $h_1 + h_3 = 1$.

8.52. Do the following steps to prove that any scaling function with exactly four coefficients h_0, h_1, h_2, and h_3 can be written as (8.100) for some choice of θ. Assume that the scaling function coefficients satisfy (i)–(iii) from Exercise 8.51.

(i) Show that $(h_0 + h_2)^2 + (h_1 + h_3)^2 = 2$.

(ii) Show that $h_0 + h_2 = h_1 + h_3 = 1$. Hint: Let $h_0 + h_2 = 1 + \varepsilon$ and $h_1 + h_3 = 1 - \varepsilon$ and use the previous identities to show that $\varepsilon = 0$.

(iii) By writing

$$h_0 = \frac{1}{2} + s, \quad h_1 = \frac{1}{2} + t, \quad h_2 = \frac{1}{2} - s, \quad h_3 = \frac{1}{2} - t,$$

show that $s^2 + t^2 = 1/2$.

(iv) Complete the proof by showing that (8.100) holds.

8.53. Show that the coefficients of the Daubechies *db*2 scaling function, given in (8.91), satisfy (8.100). What is the value of θ that makes them equal?

8.54. Show that the first moment of the mother wavelet ψ satisfies

$$\int_{-\infty}^{\infty} x\psi(x)\,dx = \frac{1}{4}\left(\int_{-\infty}^{\infty} \varphi(x)\,dx\right) \sum_{k \in \mathbb{Z}} (-1)^k k h_k. \tag{8.101}$$

8.55. For the Daubechies *db*2 scaling function, prove that $\int_{-\infty}^{\infty} \varphi(x)\,dx = 1$. What does this mean for Theorem 8.11.1?

8.56. Assume that the scaling function for a certain wavelet satisfies the following:

(i) There are exactly four nonzero coefficients h_0, h_1, h_2, and h_3.

(ii) The first moment (8.101) of the mother wavelet ψ is zero, that is,

$$\sum_{k\in\mathbb{Z}} (-1)^k k h_k = 0.$$

(iii) The three scaling identities in Exercise 8.51(i)–(ii) hold.

This gives a system of four equations and four unknowns. Solve these to get (8.91).

8.57. Plot the approximation and detail of the function f in (8.99) with the Daubechies *db*2 wavelet $T_j[f]$ for varying depths.

Notes

Much of our treatment of wavelets and Fourier series was inspired by [Str93, BN09, GV15]. For more on the FWT see also [BCR91]. For the proof that both φ_n and ψ_n (in the construction of the Daubechies wavelets) converge uniformly to their limiting functions φ and ψ see [DD10].

9 Polynomial Approximation and Interpolation

There's no sense being precise when you don't know what you're talking about.
—John von Neumann

This chapter is about approximating continuous functions on bounded intervals with polynomials. This is useful because polynomials are relatively simple functions that can be evaluated rapidly. They can also be differentiated and integrated easily. Thus we can often approximate the integral of a function, the zeros of a function, or the extrema of a function very well by approximating it closely with a polynomial and then computing the desired operation for the polynomial approximation.

We begin by proving the *Weierstrass approximation theorem*, which guarantees that every continuous function can be approximated arbitrarily closely in the uniform norm. To prove this we use Bernstein polynomials (see Volume 1, Section 2.6). The more closely these polynomials approximate a (nonpolynomial) function, the higher degree the polynomials must be. Unfortunately to approximate an arbitrary continuous function very closely it often takes a very high degree polynomial.

But for functions that have some level of regularity, or smoothness (that is, functions that are C^k for some $k \geq 1$), we can approximate much more efficiently using a special collection of orthogonal polynomials called *Chebyshev polynomials*. In this case the function is *interpolated* by the polynomial; that is, the polynomials are required to agree with the values of the function at certain predetermined points called *nodes*. We first discuss existence and basic properties of interpolation, and then we discuss the use of orthogonal polynomials, especially Chebyshev polynomials in interpolation. Chebyshev polynomials have a strong connection to Fourier series, and that connection leads to a very fast method for computing highly accurate Chebyshev interpolations using the FFT.

Finally we discuss several methods of numerical integration based on polynomial approximations. These include Newton–Cotes methods (like the trapezoid rule and Simpson's rule), Clenshaw–Curtis quadrature, which uses Chebyshev polynomials, and Gaussian quadrature, which uses Legendre orthogonal polynomials.

9.1 Polynomial Approximation

In this section we prove that any continuous function on a compact interval $[a, b]$ can be approximated arbitrarily closely (in the uniform norm) by a polynomial. In other words, the vector space of polynomials is dense in the space $C([a, b]; \mathbb{R})$ of continuous functions under the uniform norm $\|\cdot\|_{L^\infty}$). This is called the *Weierstrass approximation theorem* and is a fundamental theorem in mathematical analysis.

9.1.1 The Bernstein Transformation

To prove the Weierstrass approximation theorem we use the Bernstein polynomials, described in Volume 1, Section 2.6, and the *Bernstein transformation*, described below. This transformation takes a function f and returns a sum of Bernstein polynomials of a given degree that approximates f on the interval $[0, 1]$. The approximation becomes increasingly accurate as the degree of the polynomial gets large.

Recall that for each $n \in \mathbb{N}$ the degree-n Bernstein polynomials

$$B_j^n(x) = \binom{n}{j} x^j (1-x)^{n-j} \quad \text{for } j \in \{0, \dots, n\} \tag{9.1}$$

are nonnegative on the interval $[0, 1]$ and have their extrema on $[0, 1]$ occurring at the points $\frac{0}{n}, \frac{1}{n}, \dots, \frac{n}{n}$. Moreover, the set of all Bernstein polynomials of a given degree sum to one (they form what is known as a *partition of unity*).

Definition 9.1.1. *Denote the vector space of polynomials in x with coefficients in $\mathbb{R}$ by $\mathbb{R}[x]$ and the subspace of polynomials of degree at most n by $\mathbb{R}[x; n]$. For each $n \in \mathbb{Z}^+$ the* Bernstein transformation $B_n : C([0, 1]; \mathbb{R}) \to \mathbb{R}[x; n]$ *is given by*

$$B_n[f](x) = \sum_{k=0}^{n} f\left(\tfrac{k}{n}\right) B_k^n(x).$$

Example 9.1.2. Let $g(x) = \sin(\pi x)$. If $n = 3$, the Bernstein transformation $B_3[g](x)$ is

$$\begin{aligned} B_3[g](x) &= g(0)B_0^3(x) + g\left(\frac{1}{3}\right) B_1^3(x) + g\left(\frac{2}{3}\right) B_2^3(x) + g\left(\frac{3}{3}\right) B_3^3(x) \\ &= 0 + \frac{\sqrt{3}}{2}\binom{3}{1} x(1-x)^2 + \frac{\sqrt{3}}{2}\binom{3}{2} x^2(1-x) + 0 \\ &= \frac{3\sqrt{3}}{2}(x - x^2). \end{aligned}$$

Although this is a sum of polynomials of degree 3, the highest-degree terms cancel, yielding a polynomial of degree 2 in x. Note that $B_3[g](0) = 0 = g(0)$ and $B_3[g](1) = 0 = g(1)$. The functions g and $B_3[g]$ are plotted in the left panel of Figure 9.1.

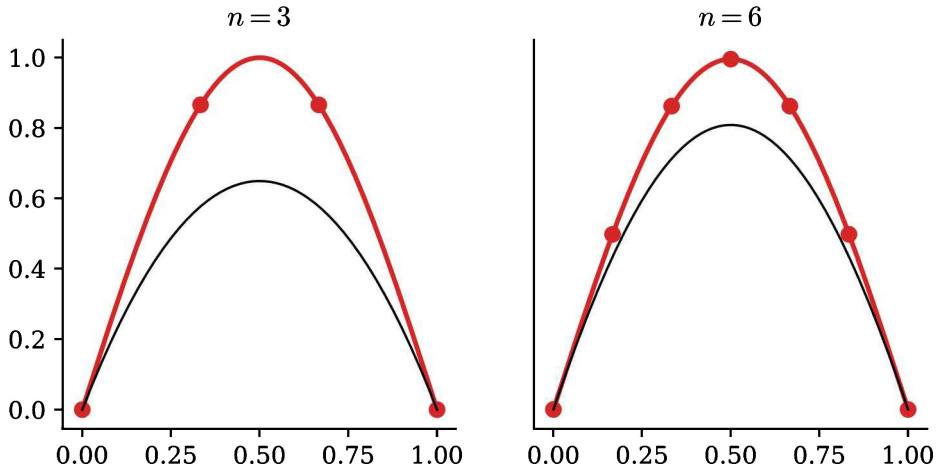


Figure 9.1. *Bernstein polynomial approximation $B_n[g]$ (plotted in black) of the function $g(x) = \sin(\pi x)$ (plotted in red) for $n = 3$ and $n = 6$. Note that the approximations $B_n[g]$ agree with g at $x = 0$ and $x = 1$ but do not intersect the graph of g anywhere on the open interval $(0, 1)$.*

Example 9.1.3. As before, let $g(x) = \sin(\pi x)$, but now take $n = 6$. The Bernstein transformation $B_6[g](x)$ is

$$\begin{aligned} B_6[g](x) &= g(0)B_0^6(x) + g\left(\frac{1}{6}\right)B_1^6(x) + g\left(\frac{2}{6}\right)B_2^6(x) + g\left(\frac{3}{6}\right)B_3^6(x) \\ &\quad + g\left(\frac{4}{6}\right)B_4^6(x) + g\left(\frac{5}{6}\right)B_5^6(x) + g\left(\frac{6}{6}\right)B_6^6(x) \\ &= 3x(1-x)^5 + \frac{15\sqrt{3}}{2}x^2(1-x)^4 + 20x^3(1-x)^3 \\ &\quad + \frac{15\sqrt{3}}{2}x^4(1-x)^2 + 3x^5(1-x) \\ &\approx -0.02x^6 + 0.058x^5 + 0.93x^4 - 1.96x^3 - 2.01x^2 + 3.0x. \end{aligned}$$

The functions g and $B_6[g]$ are plotted in the right panel of Figure 9.1.

The next two lemmata give some of the basic properties of the Bernstein transformation.

Lemma 9.1.4. *For all $n \in \mathbb{Z}^+$ we have*

$$B_n[1] = 1, \quad B_n[x] = x, \quad \text{and} \quad B_n[x^2] = x^2 + \frac{x - x^2}{n}.$$

Proof. The proof is Exercise 9.3. □

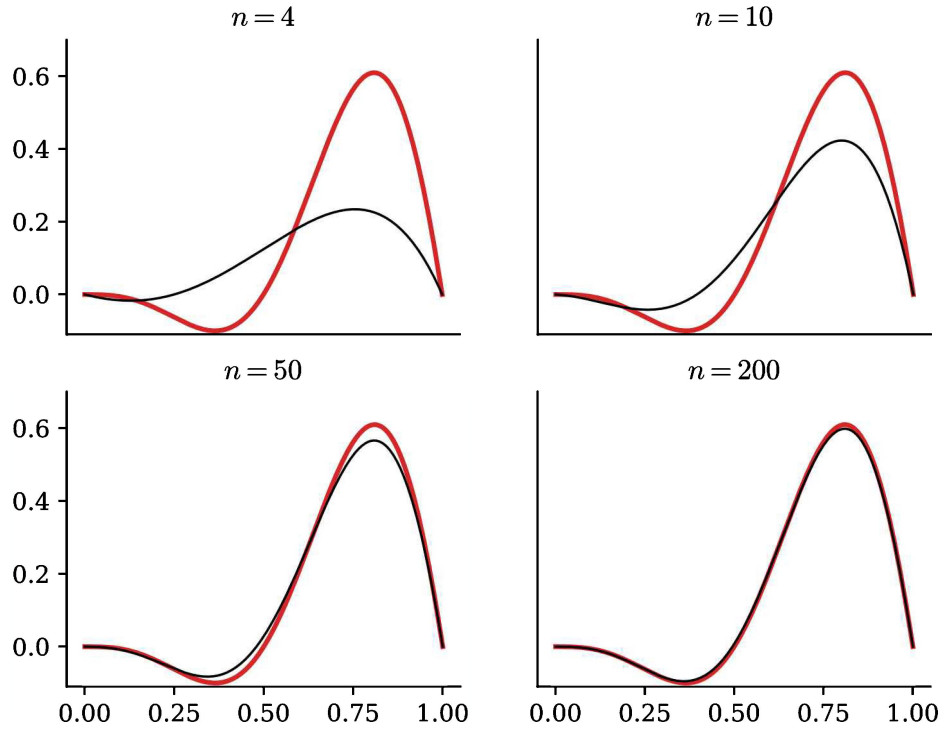


Figure 9.2. *Bernstein polynomial approximations (plotted in black) of the function* $f(x) = x^2 \sin(2\pi x + \pi)$ *(plotted in red), for* $n = 4$, 10, 50, *and* 200, *respectively. This is not a very good approximation unless* n *is large.*

Example 9.1.5. In Figure 9.2 we plot the function

$$f(x) = x^2 \sin(2\pi x + \pi) \tag{9.2}$$

and the Bernstein approximations $B_n[f](x)$ for $n = 4$, 10, 50, and 200. Although these approximations do converge uniformly to f as $n \to \infty$, they are not very good approximations until n is fairly large.

Lemma 9.1.6. *Let* $f, g \in C([0,1]; \mathbb{R})$ *and* $a, b \in \mathbb{R}$. *For all* $n \in \mathbb{N}$, *the Bernstein transformation* B_n *satisfies the following properties:*

(i) *Linearity:* $B_n[af + bg] = aB_n[f] + bB_n[g]$.

(ii) *Weak monotonicity: If* $f \leq g$ *on* $[0,1]$, *then* $B_n[f] \leq B_n[g]$ *on* $[0,1]$.

(iii) *Strong monotonicity: If* $|f| \leq g$ *on* $[0,1]$, *then* $|B_n[f]| \leq B_n[g]$ *on* $[0,1]$.

Proof.

(i) We have

$$\begin{aligned} B_n[af+bg](x) &= \sum_{k=0}^{n} (af+bg)\left(\tfrac{k}{n}\right) B_k^n(x) \\ &= a\sum_{k=0}^{n} f\left(\tfrac{k}{n}\right) B_k^n(x) + b\sum_{k=0}^{n} g\left(\tfrac{k}{n}\right) B_k^n(x) \\ &= aB_n[f](x) + bB_n[g](x). \end{aligned}$$

(ii) The proof is Exercise 9.4.

(iii) If $|f| \leq g$, then $-g \leq f \leq g$, which implies $-B_n[g] \leq B_n[f] \leq B_n[g]$, or equivalently $|B_n[f]| \leq B_n[g]$. □

9.1.2 Weierstrass Approximation

We now have all the tools we need to prove the Weierstrass approximation theorem.

Theorem 9.1.7 (Weierstrass Approximation Theorem). *If $f \in C([a,b];\mathbb{R})$, then there exists a sequence of polynomials $(p_k)_{k=0}^{\infty}$ in $\mathbb{R}[x]$ such that $\|p_n - f\|_{L^\infty} \to 0$ as $n \to \infty$. In other words, $\mathbb{R}[x]$ is dense in $C([a,b];\mathbb{R})$ under the* sup*-norm.*

Proof. Rescale the domain $[a,b]$ to $[0,1]$ via $\gamma(x) = \frac{x-a}{b-a}$, so that we may assume $a = 0$ and $b = 1$. The desired sequence of polynomials that converges to f is given by the Bernstein transformation: given $f \in C([0,1];\mathbb{R})$, we show that for all $\varepsilon > 0$ there exists $N > 0$ such that $\|B_n[f] - f\|_{L^\infty} < \varepsilon$ for all $n \geq N$.

Since $[0,1]$ is compact, f is bounded and uniformly continuous on $[0,1]$. Hence, for every $\varepsilon > 0$ there exists $\delta > 0$ such that $|f(x) - f(y)| < \frac{\varepsilon}{2}$ whenever $|x - y| < \delta$ and $x, y \in [0,1]$. Let $M = \max_{x\in[0,1]} |f(x)|$. We claim that

$$|f(x) - f(c)| < \frac{\varepsilon}{2} + \frac{2M}{\delta^2}(x-c)^2$$

for any $c \in [0,1]$. To see the claim, note that if $|x - c| < \delta$, then $|f(x) - f(c)| < \frac{\varepsilon}{2}$. But if $|x - c| \geq \delta$, then

$$|f(x) - f(c)| \leq 2M \leq 2M\frac{(x-c)^2}{\delta^2} < \frac{\varepsilon}{2} + \frac{2M}{\delta^2}(x-c)^2.$$

Thus the claim holds.

By Lemmata 9.1.4 and 9.1.6, we have

$$\begin{aligned}
|B_n[f](x) - f(c)| &= |B_n[f - f(c)](x)| \\
&\leq B_n\left[\frac{\varepsilon}{2} + \frac{2M}{\delta^2}(x-c)^2\right] \\
&= \frac{\varepsilon}{2} + \frac{2M}{\delta^2}\left(x^2 + \frac{x - x^2}{n} - 2cx + c^2\right) \\
&= \frac{\varepsilon}{2} + \frac{2M}{\delta^2}\left(\frac{x - x^2}{n} + (x-c)^2\right) \\
&\leq \frac{\varepsilon}{2} + \frac{2M}{\delta^2}\left(\frac{1}{4n} + (x-c)^2\right),
\end{aligned}$$

where the last inequality follows from the fact that $\max_{x\in[0,1]}(x - x^2) = \frac{1}{4}$. Substituting c in for x gives $|B_n[f](c) - f(c)| \leq \frac{\varepsilon}{2} + \frac{M}{2n\delta^2}$. Since c is arbitrary, we have $\|B_n[f] - f\|_{L^\infty} < \varepsilon$ whenever $n \geq N > \frac{M}{\delta^2\varepsilon}$. □

Remark 9.1.8. Although useful for proving the theorem, the Bernstein approximation method requires that n be large in order to produce close approximations. A particular weakness of this method is that for a polynomial p of degree $k \geq 2$, we usually have $B_k[p] \neq p$. For example, $B_2[x^2] = \frac{1}{2}(x^2 + x)$. Thus n must often be much larger than k in order for $B_n[p]$ to achieve a good approximation of p.

Remark 9.1.9. If the function to approximate is C^1, then we can approximate it much more efficiently using some powerful methods involving interpolation by Chebyshev polynomials, described in Sections 9.4 and 9.5. The smoother f is (the more derivatives of f that exist), the lower the degree of the polynomial required to approximate f to a given accuracy.

9.2 Interpolation

The previous section focuses on uniform approximation of functions by polynomials, but those polynomials are not required to actually agree with the function at any points—they are just required to be uniformly near the function. However, in many settings we want an approximation that actually agrees with the function at certain points. This is called *interpolation*.

Interpolation plays a key role in applied and computational mathematics; it is particularly important in numerical analysis, where interpolating polynomials are used to approximate continuous functions. It is also pervasive throughout computer-aided design, signal processing, coding theory, and mathematical systems theory.

In this section we prove the *Lagrange interpolation theorem*, which states that if $x_0, x_1, \ldots, x_n$ are distinct, then for any values $y_0, y_1, \ldots, y_n$, there exists a unique polynomial $p(x)$ of degree at most n such that $p(x_i) = y_i$ for each $i = 0, 1, 2, \ldots, n$. Although the Lagrange interpolation theorem does give a constructive proof of the existence of the unique interpolating polynomial, that construction is not generally a good numerical algorithm. But we describe two stable, efficient methods for numerically computing the unique interpolating polynomial, namely, *barycentric Lagrange interpolation* and *Newton interpolation.*

9.2.1 Interpolation

Definition 9.2.1. *Given a collection of points* $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ *in* $\mathbb{R}^2$ *with all the* x_i *distinct, we say that a polynomial* $p \in \mathbb{R}[x]$ interpolates *the collection of points if* $p(x_j) = y_j$ *for each* $j \in \{0, 1, 2, \ldots, n\}$. *If* $I \subset \mathbb{R}$ *is an interval containing the distinct points* $\{x_0, \ldots, x_n\}$, *then a polynomial* p interpolates $f : I \to \mathbb{R}$ *at* $\{x_0, \ldots, x_n\}$ *if* p *interpolates the collection* $\{(x_0, f(x_0)), \ldots, (x_n, f(x_n))\}$.

Remark 9.2.2. Although the definition is given here for points $(x_i, y_i) \in \mathbb{R}^2$, this definition also makes sense for points in $\mathbb{C}^2$, and most of the theorems of this chapter also hold over $\mathbb{C}$.

Example 9.2.3. The rational function $f(x) = 1/(1+25x^2)$ is not polynomial, but $p(x) = 1 - \frac{25}{26}x^2$ is a polynomial interpolation of f at the points $\{-1, 0, 1\}$, because $p(-1) = f(-1) = 1/26$, $p(0) = f(0) = 1$, and $p(1) = f(1) = 1/26$. This is shown in the left panel of Figure 9.3.

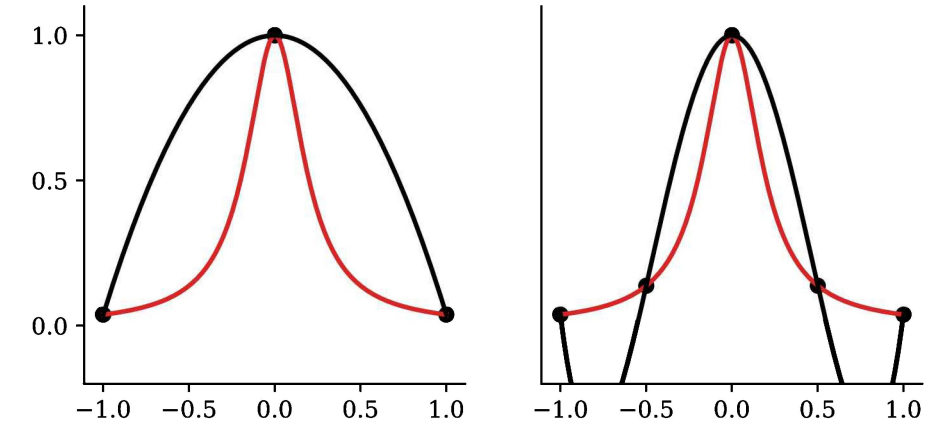


Figure 9.3. *A plot of the rational function* $f(x) = \frac{1}{1+25x^2}$ *(in red) and two interpolating polynomials. In the left panel is the degree-2 polynomial* $1 - \frac{25}{26}x^2$ *(in black) interpolating* f *at the points* $x \in \{-1, 0, 1\}$, *as described in Example 9.2.3. On the right is the degree-4 polynomial (black) interpolating* f *at the points* $x \in \{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}$, *as described in Example 9.2.8. After simplifying, the degree-4 polynomial can be written as* $\frac{1250}{377}x^4 - \frac{3225}{754}x^2 + 1$.

Unexample 9.2.4. The Bernstein polynomial approximation $B_n[f]$ is not (usually) an interpolation of f at the points $\{\frac{0}{n}, \frac{1}{n}, \ldots, \frac{n}{n}\}$. This is because even though $B_n[f]$ is a polynomial that is determined by the values of f at the points k/n, it usually does not pass through the points $(k/n, f(k/n))$ for $k \in \{1, \ldots, n-1\}$. See also Figure 9.1.

Remark 9.2.5. Any polynomial passing through $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ is an interpolation, but we usually want the interpolating polynomial of lowest degree. For example, any polynomial of the form $p(x) = x^n$ passes through the two points $\{(0,0),(1,1)\}$, but the polynomial of least degree that passes through those two points is the line $f(x) = x$.

Theorem 9.2.6 (Interpolation Theorem). *Given $n+1$ distinct points $x_0, x_1, \ldots, x_n$ and corresponding values $y_0, \ldots, y_n$, there exists a unique polynomial of degree at most n that interpolates the collection $\{(x_0, y_0), \ldots, (x_n, y_n)\}$.*

Proof. Define a family of n-degree polynomials by

$$L_{n,j}(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}. \tag{9.3}$$

These polynomials are called the *Lagrange basis functions*. Evaluating $L_{n,j}$ at each x_k gives $L_{n,j}(x_k) = \delta_{jk}$. Hence, the linear combination

$$p(x) = \sum_{j=0}^{n} y_j L_{n,j}(x) \tag{9.4}$$

is an interpolating polynomial for the given collection. To prove uniqueness, suppose there exists another interpolating polynomial q of degree at most n. The polynomial $p - q$ has degree at most n, yet it has $n+1$ distinct zeros (the points $x_0, \ldots, x_n$). But a nonzero polynomial of degree n can have at most n zeros, by the fundamental theorem of algebra (see Volume 1, Theorem 15.3.15); hence, $p - q = 0$. □

Corollary 9.2.7. *Two distinct polynomials of degree n can intersect in at most n points.*

9.2.2 Lagrange Interpolation

Lagrange interpolation is the method used in the proof of Theorem 9.2.6 for constructing the unique interpolating polynomial of degree at most n. Namely, given $\{x_0, \ldots, x_n\}$ first construct the basis polynomials $L_{n,j}$, and then use (9.4) and the values $y_0, \ldots, y_n$ to assemble the desired polynomial.

Example 9.2.8. To use Lagrange interpolation to interpolate a function in the interval $[-1,1]$ at the five equally spaced points $x_i = \frac{i-2}{2}$ for $i \in \{0, \ldots, 4\}$, construct the Lagrange basis polynomials of degree 4:

$$L_{4,0}(x) = \prod_{k=1}^{4} \frac{x - x_k}{(-1) - x_k} = \frac{2}{3}(x + \tfrac{1}{2})x(x - \tfrac{1}{2})(x - 1) = \frac{1}{6}(4x^4 - 4x^3 - x^2 + x),$$

and similarly

$$L_{4,1}(x) = \prod_{\substack{k=0\\k\neq 1}}^{4} \frac{x-x_k}{(-\frac{1}{2})-x_k} = -\frac{8}{3}(x+1)x(x-\tfrac{1}{2})(x-1)$$
$$= -\frac{4}{3}(2x^4 - x^3 - 2x^2 + x),$$
$$L_{4,2}(x) = \prod_{\substack{k=0\\k\neq 2}}^{4} \frac{x-x_k}{0-x_k} = 4(x+1)(x+\tfrac{1}{2})(x-\tfrac{1}{2})(x-1) = 4x^4 - 5x^2 + 1,$$
$$L_{4,3}(x) = \prod_{\substack{k=0\\k\neq 3}}^{4} \frac{x-x_k}{\frac{1}{2}-x_k} = -\frac{8}{3}(x+1)(x+\tfrac{1}{2})x(x-1) = -\frac{4}{3}(2x^4 + x^3 - 2x^2 - x),$$
$$L_{4,4}(x) = \prod_{\substack{k=0\\k\neq 4}}^{4} \frac{x-x_k}{1-x_k} = \frac{2}{3}(x+1)(x+\tfrac{1}{2})x(x-\tfrac{1}{2}) = \frac{1}{6}(4x^4 + 4x^3 - x^2 - x).$$

To use these to interpolate the rational function $\frac{1}{1+25x^2}$ (see Figure 9.3, right panel), compute the function values $y_j = f(x_j)$ for all $j \in \{0, \ldots, 4\}$ and apply (9.4):

$$p(x) = \tfrac{1}{26}L_{4,0} + \tfrac{4}{29}L_{4,1} + L_{4,2} + \tfrac{4}{29}L_{4,3} + \tfrac{1}{26}L_{4,4} \qquad (9.5)$$
$$= \frac{1250}{377}x^4 - \frac{3225}{754}x^2 + 1.$$

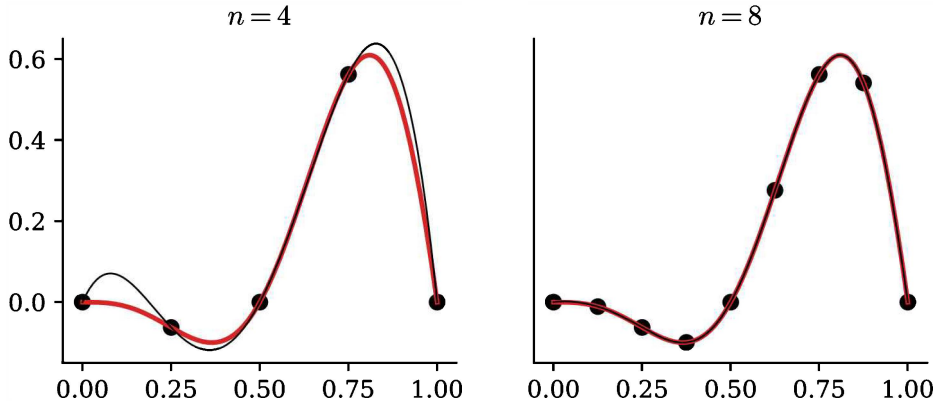


Figure 9.4. *Lagrange polynomial interpolations (in black) of the function* $f(x) = x^2 \sin(2\pi x + \pi)$ *(plotted in red) with equally spaced values in* x *for* $n = 4$ *and* $n = 8$. *The approximation is eyeball perfect for* $n = 8$. *Compare these to the Bernstein approximations in Figure 9.2. Beware, however, that this is a special example—interpolating at uniformly spaced points can sometimes provide very poor approximations; see Section 9.4.1 for more on this.*

Nota Bene 9.2.9. In college algebra classes it is traditional to simplify expressions like (9.3), (9.4), and (9.5) by expanding all the multiplications and gathering all the terms of the same degree. In the previous example, we simplified $L_{4,0}(x)$ from its original form $\frac{2}{3}(x+\frac{1}{2})x(x-\frac{1}{2})(x-1)$ to the form $\frac{1}{6}(4x^4-4x^3-x^2+x)$, and similarly with the other Lagrange basis polynomials $L_{4,j}(x)$. We also simplified (9.5) to $\frac{1250}{377}x^4-\frac{3225}{754}x^2+1$. But these simplifications are not necessary. The expressions (9.3), (9.4), and (9.5) are all polynomials whether they are simplified or not, and they can still be evaluated in their unsimplified form.

When we talk about *Lagrange interpolation* as a method for constructing the unique interpolating polynomial we usually mean the unsimplified form of (9.4) with the $L_{n,j}$ of (9.3) also unsimplified. Moreover, as an algorithm this unsimplified form is easy to code up, whereas methods for automatically simplifying the polynomial are not so easy. In Sections 9.2.3 and 9.2.4 we discuss two other methods for constructing the interpolating polynomial that are algebraically equivalent to Lagrange interpolation (that is, they all simplify to give the same polynomial), but, in their unsimplified form, they yield more efficient algorithms for computation than the naïve Lagrange method.

The interpolation theorem (Theorem 9.2.6) has the following immediate corollary.

Corollary 9.2.10. *If f is a polynomial of degree at most n, then the polynomial constructed by Lagrange interpolation of f at $n+1$ distinct points is f.*

This corollary shows one virtue of Lagrange interpolation over the Bernstein polynomial approximation: even when f is a polynomial, the Bernstein approximation $B_n[f]$, is rarely equal to f; see Remark 9.1.8.

Complexity of Lagrange Interpolation

Assuming the values $y_0,\dots,y_n$ are known and the denominators

$$w_j=\left(\prod_{k\neq j}(x_j-x_k)\right)^{-1} \tag{9.6}$$

are computed in advance (a cost of $\sim 2n$ FLOPs each, for a total complexity of $\sim 2n^2$ FLOPs for the initial startup), evaluating the unsimplified polynomial at a point x involves n multiplications and n subtractions, for each of the $n+1$ basis polynomials $L_{n,j}(x)$, and then an additional $n+1$ multiplications and n additions to compute the expression (9.4), for a total of $\sim 2n^2$ FLOPs for each evaluation of the interpolating polynomial.

Example 9.2.11. In Figure 9.4, we see the unique degree-n interpolation of $f(x) = x^2 \sin(2\pi x + \pi)$ at $n+1$ equally spaced nodes in the domain $[0,1]$ for $n = 4$ (left panel) and $n = 8$ (right panel). In this case, the Lagrange interpolation is more accurate for $n = 8$ than the Bernstein approximation is when $n = 200$. In Section 9.4.1 we show, however, that interpolation at uniformly spaced points is not always so accurate. It can sometimes provide a very poor approximation.

9.2.3 Barycentric Lagrange Interpolation

Lagrange interpolation can be made more stable and more efficient with some simple modifications. The resulting method is called *barycentric Lagrange interpolation*. Barycentric Lagrange interpolation still has a startup cost of $\sim 2n^2$, but evaluating the resulting polynomial at a given value x costs only $O(n)$ FLOPs. Contrast this with naïve Lagrange interpolation which has a cost of $O(n^2)$ for each evaluation of the polynomial.

The Barycentric Lagrange Method

Barycentric Lagrange interpolation begins with the observation that the Lagrange basis function $L_{n,j}$ in (9.3) can be rewritten as

$$L_{n,j}(x) = v(x)\frac{w_j}{(x - x_j)},$$

where the w_j (hereafter called the *barycentric weights*) are given in (9.6), and $v(x) = \prod_{j=0}^{n}(x - x_j)$. Using (9.4) we can write

$$p(x) = v(x)\sum_{j=0}^{n}\frac{w_j}{x - x_j}y_j. \tag{9.7}$$

Applying (9.7) to the degree-0 polynomial 1 (so $y_j = 1$ for all j) gives

$$1 = v(x)\sum_{j=0}^{n}\frac{w_j}{x - x_j},$$

so we can avoid computing $v(x)$ and write

$$p(x) = \frac{\displaystyle\sum_{j=0}^{n}\frac{w_j}{x - x_j}y_j}{\displaystyle\sum_{j=0}^{n}\frac{w_j}{x - x_j}}. \tag{9.8}$$

We call (9.8) the *barycentric form* of the interpolating polynomial p. Although this does not look like a polynomial, the previous discussion shows that it is, and, in

fact, it is the unique polynomial of degree at most n that interpolates the data $\{(x_0, y_0), \ldots, (x_n, y_n)\}$.

Remark 9.2.12. Although the barycentric form (9.8) defines a polynomial function (meaning that there is a polynomial, in the traditional sense, that agrees with it at all the points where (9.8) is defined), the expression (9.8) is not defined at the points x_j for $j \in \{0, \ldots, n\}$. But the value of the polynomial at x_j is already known to be y_j, so (9.8) need never be evaluated at any of the x_j.

Example 9.2.13. To use barycentric Lagrange to interpolate a function in the interval $[-1, 1]$ at the five equally spaced points $x_i = \frac{i-2}{2}$ for $i \in \{0, \ldots, 4\}$, use (9.6) to compute the barycentric weights

$$w_0 = \left(\prod_{k=1}^{4} (-1 - x_k) \right)^{-1} = \left((-1 + \tfrac{1}{2})(-1 - 0)(-1 - \tfrac{1}{2})(-1 - 1) \right)^{-1} = \frac{2}{3}$$

and, similarly,

$$w_1 = -\frac{8}{3}, \qquad w_2 = 4, \qquad w_3 = -\frac{8}{3}, \qquad \text{and} \qquad w_4 = \frac{2}{3}.$$

To use these to interpolate the rational function $f(x) = \frac{1}{1+25x^2}$, use the function values $y_j = f(x_j)$ for all $j \in \{0, \ldots, 4\}$, and then apply (9.8):

$$p(x) = \frac{\left(\frac{\frac{2}{3}}{x+1}\right)\frac{1}{26} - \left(\frac{\frac{8}{3}}{x+\frac{1}{2}}\right)\frac{4}{29} + \left(\frac{4}{x}\right)1 - \left(\frac{\frac{8}{3}}{x-\frac{1}{2}}\right)\frac{4}{29} + \left(\frac{\frac{2}{3}}{x-1}\right)\frac{1}{26}}{\left(\frac{\frac{2}{3}}{x+1}\right) - \left(\frac{\frac{8}{3}}{x+\frac{1}{2}}\right) + \left(\frac{4}{x}\right) - \left(\frac{\frac{8}{3}}{x-\frac{1}{2}}\right) + \left(\frac{\frac{2}{3}}{x-1}\right)}. \tag{9.9}$$

By the interpolation theorem (Theorem 9.2.6), the polynomial (9.9) is unique and thus must equal (9.5), so both of these simplify to $\frac{1250}{377}x^4 - \frac{3225}{754}x^2 + 1$. But one benefit of the barycentric Lagrange method is that one can rapidly compute the value of this polynomial using (9.9), without doing any simplification.

Complexity of Barycentric Lagrange

Every evaluation of $p(x)$ in (9.8) uses the weights w_j, which are constant, so these can be computed in advance. Computing each w_j requires $2n$ FLOPs, for a total startup cost of $\sim 2n^2$ FLOPs. To evaluate the polynomial at a given point x requires computing (9.8). To do this, first compute the expressions $\frac{w_j}{x - x_j}$, which require only 2 FLOPs each, for a total of $2(n+1)$ FLOPs. Computing the sum in the numerator requires an additional $n+1$ multiplications and n additions, and computing the sum in the denominator requires n more additions, for a grand total of $\sim 5n$ FLOPs. Thus the startup cost of barycentric Lagrange interpolation is the same as naïve Lagrange interpolation ($\sim 2n^2$), but evaluation of the polynomial at any point x costs only $\sim 5n$ instead of $\sim 2n^2$.

Remark 9.2.14. When a generic polynomial is fully simplified into the standard form $\sum_{j=0}^{n} a_j x^j$, evaluating it requires at least n multiplications and n additions (using Horner's method), for a minimum number of $\sim 2n$ FLOPs. The barycentric method is not significantly worse than this, and any algorithm to further simplify the polynomial would require additional computation. Thus barycentric Lagrange is not far from being optimal, in terms of temporal complexity.

Remark 9.2.15. The weights w_j depend only on the x_j and are independent of the y_j, so once the w_j are computed, they can be used for interpolation with any collection of y_j, without any additional startup cost. Adding a new point x_{n+1} to the formulas can also be done with an additional temporal cost of $O(n)$ (see Exercise 9.11).

9.2.4 *Newton Interpolation

Newton interpolation is an alternative method of constructing the interpolating polynomial. It also requires only $O(n)$ FLOPs to evaluate the polynomial, once some initial numbers have been computed, and computing those initial numbers requires $O(n^2)$ FLOPs. It has the disadvantage that the initial numbers depend on the values y_j, so changing the y_j means another startup cost of $O(n^2)$. Nevertheless, Newton interpolation is a practical and commonly used method. And it has the advantage of generalizing well to *Lagrange–Hermite interpolation*, a variant of interpolation where not only values of $f(x_j)$ are to be matched by the polynomial, but also derivatives (see Volume 1, Section 15.7.4). Traditionally Newton interpolation was the preferred method for interpolation, but more recently it has become clear that barycentric Lagrange is a better algorithm for many applications (see [BT04]).

Newton interpolation is an iterative method that constructs the interpolating polynomial of degree at most k through the first $k+1$ pairs $\{(x_0, y_0), \ldots, (x_k, y_k)\}$.

Theorem 9.2.16. *For each $k \in \{0, \ldots, n\}$ let $p_k(x)$ be the unique polynomial of degree at most k that interpolates the subset $\{(x_j, y_j)\}_{j=0}^{k}$ with the x_i all distinct. For each $k \in \{1, \ldots, n\}$ the polynomial $p_k(x)$ satisfies the relation*

$$p_k(x) = p_{k-1}(x) + a_k w_k(x), \tag{9.10}$$

where

$$w_k(x) := \prod_{j=0}^{k-1} (x - x_j) \tag{9.11}$$

and

$$a_k = \frac{y_k - p_{k-1}(x_k)}{w_k(x_k)}. \tag{9.12}$$

Proof. The difference $p_k(x) - p_{k-1}(x)$ is a polynomial of degree at most k with all the same zeros as $w_k(x)$; therefore it must equal some scalar multiple of $w_k(x)$. Evaluating (9.10) at the point x_k yields (9.12). □

Corollary 9.2.17. *For $\{a_j\}_{j=0}^n$ and $w_j(x)$ defined iteratively, as above, we have*

$$p_n(x) = \sum_{j=0}^{n} a_j w_j(x), \tag{9.13}$$

where $w_0(x) \equiv 1$ and $a_0 = y_0$. If the coefficients a_j are known in advance, then, for a given value of x, the expression (9.13) can be computed as

$$p_n(x) = (\dots((a_n(x - x_{n-1}) + a_{n-1})(x - x_{n-2}) + a_{n-2})\cdots + a_1)(x - x_0) + a_0. \tag{9.14}$$

The corollary motivates the following proposition and the corresponding algorithm, called *divided differences*, which allows for fast computation.

Proposition 9.2.18. *For each $k \in \{0, \dots, n\}$ define $y[x_k] = y_k$, and for $0 \le j < k$ recursively define*

$$y[x_j, \dots, x_k] = \frac{y[x_{j+1}, \dots, x_k] - y[x_j, \dots, x_{k-1}]}{x_k - x_j}. \tag{9.15}$$

The coefficients a_k in (9.10) satisfy

$$a_k = y[x_0, x_1, \dots, x_k].$$

Proof. Let $p_k(x)$ denote the unique polynomial of degree at most k that interpolates $\{(x_j, y_j)\}_{j=0}^k$. Similarly, let $P_k(x)$ be the unique polynomial of degree at most $k-1$ that interpolates $\{(x_j, y_j)\}_{j=1}^k$. For each $k \in \{1, \dots, n\}$ we have

$$p_k(x) = P_k(x) + \frac{x - x_k}{x_k - x_0}(P_k(x) - p_{k-1}(x)).$$

Matching the terms of degree k gives (9.15). The details are Exercise 9.13. □

Example 9.2.19. To apply the divided differences algorithm to interpolate the set $\{(0,1), (-1,3), (1,1), (2,15)\}$, we compute

$$a_0 = y[x_0] = 1 \qquad \text{and}$$
$$a_1 = y[x_0, x_1] = \frac{y[x_1] - y[x_0]}{x_1 - x_0} = \frac{3-1}{-1-0} = -2.$$

Computing a_2 requires

$$y[x_1, x_2] = \frac{y[x_2] - y[x_1]}{x_2 - x_1} = -1,$$

from which we compute

$$a_2 = y[x_0, x_1, x_2] = \frac{y[x_1, x_2] - y[x_0, x_1]}{x_2 - x_0} = 1.$$

Computing a_3 requires

$$y[x_2, x_3] = \frac{y[x_3 - x_2]}{x_3 - x_2} = 14 \qquad \text{and}$$
$$y[x_1, x_2, x_3] = \frac{y[x_2, x_3] - y[x_1, x_2]}{x_3 - x_1} = 5,$$

yielding $a_3 = 2$. Combining these using (9.14) gives

$$\begin{aligned} p_3(x) &= ((a_3(x - x_2) + a_2)(x - x_1) + a_1)(x - x_0) + a_0 \\ &= ((2(x - 1) + 1)(x + 1) - 2)x + 1. \end{aligned}$$

9.3 Orthogonal Polynomials for Approximation

As discussed in Chapter 8, choosing an orthogonal basis for a space of functions is a very powerful tool for approximating functions. Fourier series and wavelets are important examples of this. Orthogonal bases are also very useful in the space $\mathbb{R}[x]$ of polynomial functions. There are many choices of inner products that we can put on $\mathbb{R}[x]$, and these different choices each lead to a different orthogonal basis.

9.3.1 Legendre Polynomials

The Legendre polynomials arise from considering the vector space $\mathbb{R}[x]$ with the inner product

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\,dx. \tag{9.16}$$

Applying the Gram–Schmidt process (see Volume 1, Section 3.3.1) to the power basis $\{1, x^1, x^2, \ldots\} \subset \mathbb{R}[x]$ gives an orthonormal basis. Recall that this process involves letting

$$q_0 = \frac{1}{\|1\|} = \frac{1}{\sqrt{2}}$$

and defining $q_1, q_2, \ldots$, recursively, by

$$q_k = \frac{x^k - p_{k-1}}{\|x^k - p_{k-1}\|}, \qquad k \in \mathbb{Z}^+,$$

where

$$p_{k-1} = \operatorname{proj}_{Q_{k-1}}(x^k) = \sum_{i=0}^{k-1} \langle q_i, x^k \rangle\, q_i \tag{9.17}$$

is the orthogonal projection of x^k onto $Q_{k-1} = \operatorname{span}(\{q_0, \ldots, q_{k-1}\})$.

We now compute the orthogonal basis

$$p_0 = \operatorname{proj}_{Q_0}(x) = \left\langle \frac{1}{\sqrt{2}}, x \right\rangle \frac{1}{\sqrt{2}} = 0,$$

so x is already orthogonal to q_0. This gives

$$q_1 = \frac{x - p_0}{\|x - p_0\|} = \frac{\sqrt{3}}{\sqrt{2}}x.$$

And next, a straightforward integration shows that $\langle x, x^2 \rangle = 0$, which gives

$$p_1 = \operatorname{proj}_{Q_1}(x^2) = \left\langle \frac{1}{\sqrt{2}}, x^2 \right\rangle \frac{1}{\sqrt{2}} + \left\langle \frac{\sqrt{3}}{\sqrt{2}}x, x^2 \right\rangle \frac{\sqrt{3}}{\sqrt{2}}x = \frac{1}{3},$$

so

$$q_2 = \frac{x^2 - p_1}{\|x^2 - p_1\|} = \frac{\sqrt{5}}{2\sqrt{2}}(3x^2 - 1).$$

Continuing in this manner gives an infinite orthonormal set $\{q_0, q_1, \dots\}$ of polynomials, with each q_n of degree n.

Although orthonormal sets have some advantages over other orthogonal sets, for polynomials it is often more convenient to rescale the polynomials to create an orthogonal set $\{u_0, u_1, \dots\}$ of nonzero polynomials of the form $u_k = x^k - p_{k-1}$, where p_{k-1} is given by

$$p_{k-1} = \sum_{j=0}^{k-1} \frac{\langle u_j, x^k \rangle}{\langle u_j, u_j \rangle} u_j. \tag{9.18}$$

That is, we take each u_k to be a *monic* polynomial of degree k (*monic* means the coefficient of the highest-degree monomial is 1). For example, with the Legendre polynomials this gives $u_0 = 1$, $u_1 = x$, and $u_2 = x^2 - 1/3$.

9.3.2 Monic Chebyshev Polynomials

The Chebyshev polynomials are constructed in a manner similar to the Legendre polynomials but using a different inner product on $\mathbb{R}[x]$, namely

$$\langle f, g \rangle = \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1 - x^2}}\, dx. \tag{9.19}$$

Applying the Gram–Schmidt process to the power basis $\{1, x, x^2, x^3, \dots\}$ using this inner product, but rescaling to make these polynomials monic, gives the *monic Chebyshev polynomials* $\{\widehat{T}_n\}_{n=0}^{\infty}$, which satisfy

$$\begin{aligned}
\widehat{T}_0(x) &= 1, \\
\widehat{T}_1(x) &= x, \\
\widehat{T}_2(x) &= x^2 - \frac{1}{2}, \\
&\vdots \\
\widehat{T}_{n+1}(x) &= x\widehat{T}_n(x) - \frac{1}{4}\widehat{T}_{n-1}(x), \qquad n \geq 2.
\end{aligned} \tag{9.20}$$

This forms an orthogonal sequence of monic polynomials on the interval $(-1, 1)$ with the inner product (9.19).

Proposition 9.3.1. *The monic Chebyshev polynomials satisfy the following relation:*

$$\widehat{T}_n(x) = \frac{1}{2^{n-1}} \cos\left(n \cos^{-1} x\right) \quad \text{for } n \in \mathbb{Z}^+. \tag{9.21}$$

Proof. The proof is Exercise 9.17. □

9.3.3 Chebyshev Polynomials

It is useful to rescale the monic Chebyshev polynomials to get rid of the 2^{n-1} in (9.21), so for $n \geq 1$ we define

$$T_n(x) = 2^{n-1} \widehat{T}_n(x).$$

We call these nonmonic polynomials simply *Chebyshev polynomials*. They are sometimes called Chebyshev polynomials *of the first kind*, since there are other kinds of Chebyshev polynomials, but we don't use that name ("of the first kind") because these are the only kind of Chebyshev polynomials that we really use in this book.

The Chebyshev polynomials satisfy the recurrence

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 - 1, \\ &\vdots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x), \qquad n \geq 2. \end{aligned} \tag{9.22}$$

See Figure 9.5 for a plot of some of these polynomials.

Remark 9.3.2. The Chebyshev polynomials T_n satisfy the relations

$$\langle T_n, T_m \rangle = \begin{cases} 0 & \text{if } n \neq m, \\ \pi & \text{if } n = m = 0, \\ \pi/2 & \text{if } n = m \neq 0, \end{cases} \tag{9.23}$$

so, as with the monic Chebyshev polynomials, these are orthogonal but not orthonormal.

By (9.21), we have

$$T_n(\cos(t)) = \cos(nt) \qquad \forall n \in \mathbb{N}. \tag{9.24}$$

This relation provides an important connection between Chebyshev expansions and Fourier series, which we explore further in Section 9.5.

Proposition 9.3.3. *Let $T_n(x)$ be the Chebyshev polynomial of degree n.*

(i) *The zeros of $T_n(x)$ (and of $\widehat{T}_n(x)$) are given by*

$$z_j = \cos\left(\frac{\pi}{n}\left(j + \frac{1}{2}\right)\right), \qquad j = 0, 1, 2, \ldots, n-1. \tag{9.25}$$

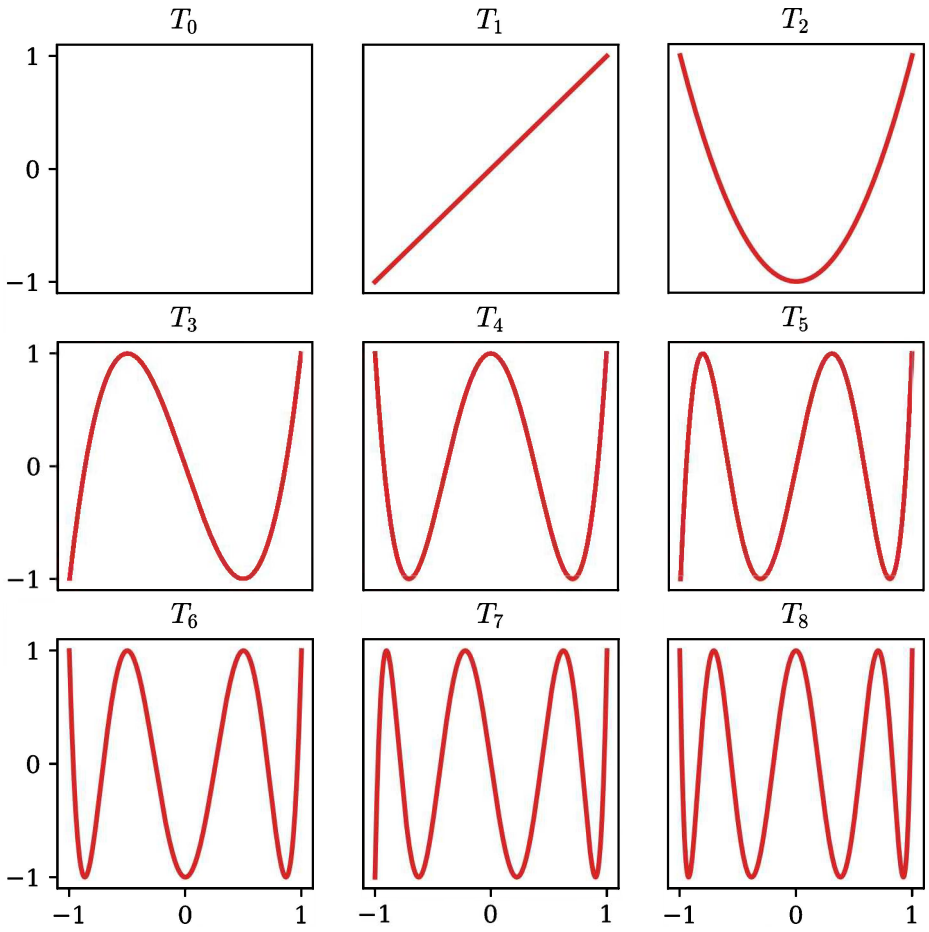


Figure 9.5. *A plot of the first nine Chebyshev polynomials on the interval* $[-1,1]$. *The degree of the polynomial* $T_n(x)$ *is* n, *and its zeros are* $z_j = \cos\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)$ *for* $j \in \{0,1,2,\dots,n-1\}$. *When restricted to the domain* $[-1,1]$, *the range of the Chebyshev polynomials is also contained in* $[-1,1]$.

We call these the Chebyshev zeros. *They are often called* Chebyshev points of the first kind *or* Gauss–Chebyshev points*; see Figure 9.6.*

(ii) *The extrema of* $T_n(x)$ *(and of* $\widehat{T}_n(x)$*) in* $[-1,1]$ *occur at the points*

$$y_j = \cos\left(\frac{j\pi}{n}\right), \qquad j = 0,1,2,\dots,n, \tag{9.26}$$

and yield $T_n(y_j) = (-1)^j$ *and* $\widehat{T}_n(y_j) = \frac{(-1)^j}{2^{n-1}}$. *We call the points* y_j Chebyshev extremizers *or just* Chebyshev points. *These are often called* Chebyshev points of the second kind *or* Chebyshev–Gauss–Lobatto points.

Proof. The proof is Exercise 9.21. □

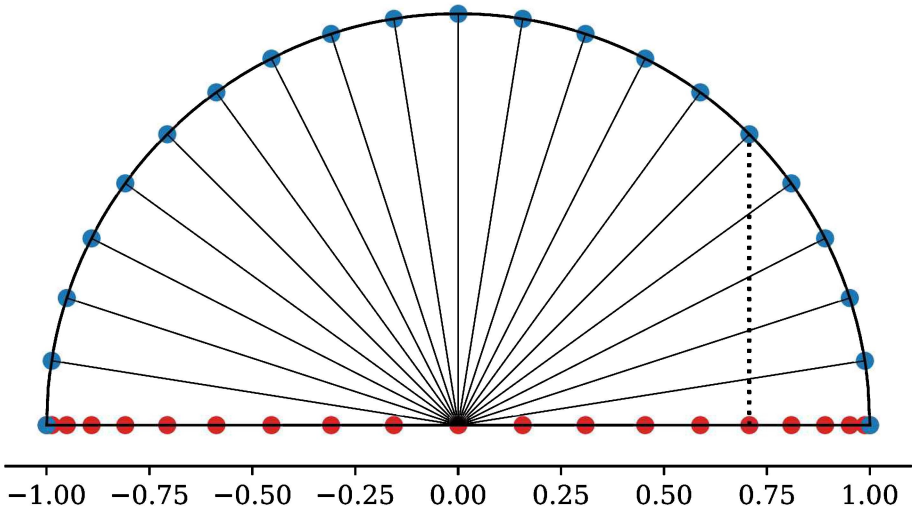


Figure 9.6. *A plot (red) of the Chebyshev points (extremizers) when $n = 20$. These correspond to the x-coordinates of points (blue) uniformly distributed around the upper half of the unit circle. Notice the clustering of the zeros near the two endpoints.*

9.3.4 Other Inner Products and Orthogonal Polynomials

We can generalize the previous process to any interval (a, b) with an inner product of the form

$$\langle f, g \rangle = \int_a^b w(x) f(x) g(x)\, dx, \tag{9.27}$$

where the *weight function* $w(x) > 0$ is an integrable function.

The orthogonal polynomials in Table 9.1 are some of the most widely used examples in applications. They arise frequently in numerical analysis, probability theory, statistics, number theory, and physics.

Performing the Gram–Schmidt process (but scaling to make the polynomials monic, instead of orthonormal) gives a recursive formula for the orthogonal monic polynomials for the corresponding inner product, as described in the next theorem.

Class	Domain	$w(x)$	α_{k+1}	β_{k+1}
Chebyshev	$(-1, 1)$	$(1 - x^2)^{-1/2}$	0	$1/4$ $(1/2$ for $k = 1)$
Hermite	$(-\infty, \infty)$	$\exp(-x^2)$	0	$k/2$
Laguerre	$(0, \infty)$	$\exp(-x)$	$2k + 1$	k^2
Legendre	$(-1, 1)$	1	0	$k^2/(4k^2 - 1)$

Table 9.1. *Commonly used orthogonal polynomials. The constants α_{k+1} and β_{k+1} are the coefficients of the recursion described in Theorem 9.3.4 for $k \in \mathbb{N}$.*

Theorem 9.3.4. *Let I be an interval in $\mathbb{R}$ and let $w : I \to [0, \infty)$ be a nonnegative, integrable function on I. If the integrals $\int_I w(x)x^{2k}\,dx$ are finite and nonzero for all $k \in \mathbb{N}$, then $\langle f, g\rangle = \int_I f(x)g(x)w(x)\,dx$ defines an inner product on $\mathbb{R}[x]$ and there exists a unique basis of monic orthogonal polynomials $\{u_0, u_1, \ldots\}$, with $\deg u_k = k$ for each $k \in \mathbb{N}$. Moreover, the polynomials satisfy the recursive equation*

$$u_{k+1} = (x - \alpha_{k+1})u_k - \beta_{k+1}u_{k-1}, \tag{9.28}$$

where the recursion begins with $u_0 = 1$ and $u_1 = x - \alpha_1$, and the coefficients α_{k+1} and β_{k+1} are given by

$$\alpha_{k+1} = \frac{\langle u_k, xu_k\rangle}{\langle u_k, u_k\rangle} \qquad \text{and} \qquad \beta_{k+1} = \frac{\langle u_{k-1}, xu_k\rangle}{\langle u_{k-1}, u_{k-1}\rangle}. \tag{9.29}$$

Proof. It is straightforward to check that $\langle f, g\rangle = \int_I f(x)g(x)w(x)\,dx$ defines an inner product. We show (by induction) that for any $k \geq 0$ the given equations do, in fact, define an orthogonal set $\{u_\ell\}_{\ell=0}^{k}$ with each u_ℓ monic of degree ℓ for every $\ell \in \{0, \ldots, k\}$. This shows that the set $\{u_\ell\}_{\ell=0}^{k}$ forms a basis of the space $\mathbb{R}[x; k]$ of polynomials of degree at most k and that for each $\ell \leq k$, the polynomial u_ℓ is orthogonal to every polynomial in $\mathbb{R}[x; \ell - 1]$ (that is, every polynomial of degree at most $\ell - 1$).

The initial case of $k = 0$ is immediate. For $k = 1$ we need only check that $\langle 1, x - \alpha_1\rangle = 0$, which follows from the definition of α_1. Assume now that the claim holds for some $k > 1$. Since u_k and u_{k-1} are monic, they must have $\langle u_k, u_k\rangle > 0$ and $\langle u_{k-1}, u_{k-1}\rangle > 0$, which shows that the denominators in (9.29) defining α_{k+1} and β_{k+1} are not zero, so the polynomial u_{k+1}, defined by (9.28), makes sense and is a monic polynomial of degree $k + 1$.

We have

$$\begin{aligned}\langle u_k, u_{k+1}\rangle &= \langle u_k, xu_k - \alpha_{k+1}u_k - \beta_{k+1}u_{k-1}\rangle \\ &= \langle u_k, xu_k\rangle - \alpha_{k+1}\langle u_k, u_k\rangle = 0,\end{aligned}$$

$$\begin{aligned}\langle u_{k-1}, u_{k+1}\rangle &= \langle u_{k-1}, xu_k - \alpha_{k+1}u_k - \beta_{k+1}u_{k-1}\rangle \\ &= \langle u_{k-1}, xu_k\rangle - \beta_{k+1}\langle u_{k-1}, u_{k-1}\rangle = 0,\end{aligned}$$

and

$$\begin{aligned}\langle u_j, u_{k+1}\rangle &= \langle u_j, xu_k - \alpha_{k+1}u_k - \beta_{k+1}u_{k-1}\rangle \\ &= \langle u_j, xu_k\rangle \text{ for each } j < k - 1.\end{aligned}$$

However, $\langle u_j, xu_k\rangle = \langle xu_j, u_k\rangle = 0$, since xu_j is a polynomial of degree at most $k-1$ and hence is always orthogonal to u_k. Therefore, the set $\{u_\ell\}_{\ell=0}^{k+1}$ is orthogonal.

To prove uniqueness, suppose that $\{v_k\}_{k=0}^{\infty}$ is also an orthogonal basis of monic polynomials each satisfying $\deg v_k = k$. Thus $u_k - v_k$ is a degree $k - 1$ polynomial, and hence $\langle u_k - v_k, u_k\rangle = 0$ and $\langle u_k - v_k, v_k\rangle = 0$, since u_k and v_k are orthogonal to all lower-degree polynomials. Therefore $\|u_k - v_k\|^2 = \langle u_k - v_k, u_k - v_k\rangle = 0$, and thus $u_k = v_k$. ∎

Remark 9.3.5. The converse also holds: If a sequence of monic polynomials satisfies the recurrence in (9.28), then there exists a domain I and weight function $w(x)$ such that the polynomials are orthogonal with respect to the inner product defined by I and w; see [Fav35] for details.

9.3.5 *Further Analysis of Legendre Polynomials

We conclude this section by applying Theorem 9.3.4 to the monic Legendre polynomials. Assume the domain is $(-1, 1)$, and the weight function is $w(x) = 1$. We show that $\alpha_{k+1} = 0$ and $\beta_{k+1} = k^2/(4k^2 - 1)$, for each $k \geq 0$. As a first step, we prove *Rodrigues' formula*, which is very useful in its own right.

Theorem 9.3.6 (Rodrigues' Formula). *For each $k \in \mathbb{N}$ the monic Legendre polynomial u_k satisfies*

$$u_k(x) = \frac{k!}{(2k)!} \frac{d^k}{dx^k}(x^2 - 1)^k. \tag{9.30}$$

Proof. We first show that the right-hand side of (9.30) is orthogonal to the set $\mathbb{R}[x; k-1]$. Any basis for $\mathbb{R}[x; k-1]$ can be used to check this orthogonality, so we use the power basis $\{1, x, \ldots, x^{k-1}\}$. Repeated integration by parts gives

$$\int_{-1}^{1} x^l \frac{d^k}{dx^k}(x^2 - 1)^k \, dx = \sum_{j=0}^{l} (-1)^j l(l-1)\cdots(l-j+1)x^{l-j} \frac{d^{k-j-1}}{dx^{k-j-1}}(x^2-1)^k \Big|_{-1}^{1} = 0$$

for $l = 0, 1, 2, \ldots, k-1$. Since the right-hand side of (9.30) is a degree k polynomial that is orthogonal to all lower-degree polynomials, it must be a scalar multiple of $u_k(x)$. Thus we need only choose a scaling to make it monic.

Note that

$$\frac{d^k}{dx^k}(x^2-1)^k = \frac{d^k}{dx^k} \sum_{j=0}^{k} \binom{k}{j} x^{2j}(-1)^{k-j} = \frac{(2k)!}{k!}x^k - \frac{(2k-2)!}{(k-2)!}kx^{k-2} + \cdots.$$

This implies

$$\frac{k!}{(2k)!}\frac{d^k}{dx^k}(x^2-1)^k = x^k - \frac{k^2(k-1)}{2k(2k-1)}x^{k-2} + \cdots, \tag{9.31}$$

which gives the desired monic scaling. □

Remark 9.3.7. Each of the orthogonal polynomials described in Table 9.1 has a similar formula and the term Rodrigues' formula is used to describe them.

The recursion formula (9.28) gives an interesting way to compute the Legendre polynomials, computing the coefficients α_k and β_k using (9.29). But Rodrigues' formula gives a way to compute the Legendre polynomials directly, as described in the following corollary.

Corollary 9.3.8. *For each $n \in \mathbb{N}$, the monic Legendre polynomial u_n is given by*

$$u_n(x) = \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \frac{(n!)^2(2n-2k)!}{k!(n-k)!(n-2k)!(2n)!} x^{n-2k}. \tag{9.32}$$

Proof. By the binomial theorem we have

$$\frac{n!}{(2n)!}(x^2-1)^n = \frac{n!}{(2n)!} \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} x^{2(n-k)}(-1)^k.$$

Differentiating n times gives (9.32). □

Theorem 9.3.9. *The recursion* (9.28) *for Legendre polynomials satisfies $\alpha_{k+1} = 0$ and $\beta_{k+1} = k^2/(4k^2-1)$ for each $k \geq 0$.*

Proof. Corollary 9.3.8 shows that u_k is an odd function when k is odd, and it is an even function when k is even. Combining this with (9.28) implies that $\alpha_{k+1} = 0$.

To prove that $\beta_{k+1} = k^2/(4k^2-1)$, we write $u_k(x) = x^k + \mu_k x^{k-2} + \cdots$, where μ_k is given in (9.31). Thus

$$\begin{aligned} \beta_{k+1} &= \frac{x u_k(x) - u_{k+1}(x)}{u_{k-1}(x)} = \mu_k - \mu_{k+1} = \frac{-k(k-1)}{2(2k-1)} + \frac{(k+1)k}{2(2k+1)} \\ &= -k\frac{(k-1)(2k+1) - (k+1)(2k-1)}{2(2k-1)(2k+1)} = \frac{k^2}{4k^2-1}. \end{aligned}$$ □

Remark 9.3.10. For each k, the monic Legendre polynomial u_k satisfies the second-order ordinary differential equation

$$(x^2-1)\frac{d^2u_k}{dx^2} + 2x\frac{du_k}{dx} - k(k+1)u_k = 0. \tag{9.33}$$

Indeed each of the orthogonal polynomials described in Table 9.1 can be generated by a sequence of differential equations in this manner.

Another way to construct the Legendre polynomials is to start with the linear operator $D : \mathbb{R}[x] \to \mathbb{R}[x]$ given by $D = (x^2-1)\frac{d^2}{dx^2} + 2x\frac{d}{dx}$. Relation (9.33) shows that each Legendre polynomial u_k is an eigenvector (also called an eigenfunction) of D with eigenvalue $k(k+1)$. It can be verified that D is self-adjoint (Hermitian) with respect to the inner product (9.16), which means that the eigenvectors must be orthogonal.

9.4 Interpolation and Approximation Error

Interpolating a function does not always give a good approximation of that function, and, contrary to intuition, adding more points does not always improve the quality of the approximation. The key is how the nodes are distributed. If the nodes are uniformly spaced, interpolation at those points can give terrible approximations, but if the nodes are spaced like the Chebyshev zeros (9.25) or the Chebyshev extremizers (9.25), then interpolating gives a good approximation, as measured by the sup-norm, provided the original function is well behaved. The quality of the approximation improves for smoother functions and for higher-degree polynomials.

9.4.1 Interpolation Error

We begin this section with a theorem that describes the interpolation error for a degree-n polynomial interpolation of a smooth function in terms of the $(n+1)$th derivative of the function and the polynomial $v(x) = \prod_{i=0}^{n}(x - x_i)$, where the x_i are the nodes of the interpolation.

Theorem 9.4.1. *Let $p \in \mathbb{R}[x; n]$ be the unique interpolating polynomial of degree at most n for a function $f \in C^{n+1}([a,b];\mathbb{R})$ at $n+1$ distinct points $x_0, x_1, \ldots, x_n$ on the interval $[a,b]$. For each $x \in [a,b]$ there exists $\xi \in (a,b)$ such that*

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) v(x), \tag{9.34}$$

where

$$v(x) = \prod_{k=0}^{n}(x - x_k). \tag{9.35}$$

Proof. Fix $x \in [a,b]$. Assume $x \notin \{x_0, x_1, x_2, \ldots, x_n\}$, since otherwise the result holds trivially. Let

$$g(y) = f(y) - p(y) - v(y)\frac{f(x) - p(x)}{v(x)}.$$

Note that $g(x) = 0$ and $g(x_j) = 0$ for all $j = 0, 1, 2, \ldots, n$. By Rolle's theorem, $g'(y)$ has $n+1$ distinct zeros in (a,b). Similarly, $g''(y)$ has n distinct zeros in (a,b). Repeating, $g^{(n+1)}(y)$ has at least one zero in (a,b); call it ξ. Thus we have

$$\begin{aligned} 0 = g^{(n+1)}(\xi) &= f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - v^{(n+1)}(\xi)\frac{f(x) - p(x)}{v(x)} \\ &= f^{(n+1)}(\xi) - \frac{f(x) - p(x)}{v(x)}(n+1)!. \end{aligned}$$

Simplifying gives (9.34). ∎

Remark 9.4.2. You might wonder why we used Bernstein polynomials to prove the Weierstrass approximation theorem (Theorem 9.1.7) when the previous theorem seems to suggest that polynomial interpolation can also give good approximations. There are at least two reasons why Theorem 9.4.1, as it stands here, is not strong enough to prove the Weierstrass approximation theorem. First, to prove convergence, we need a bound on the polynomial $v(x)$. Without some restrictions on the interpolation nodes $x_0, \ldots, x_n$, the polynomial $v(x)$ can grow very large (see immediately below), but this problem can be circumvented: If the nodes are carefully chosen, then $|v(x)|$ can be bounded.

Second, a more fundamental problem is the fact that Theorem 9.4.1, and most other theorems about convergence of polynomial interpolation, requires the function f being interpolated to lie in C^{n+1}. Therefore, a convergence result as $n \to \infty$ would require $f \in C^{\infty}$, whereas Weierstrass approximation holds for any continuous function, with no differentiability requirement at all.

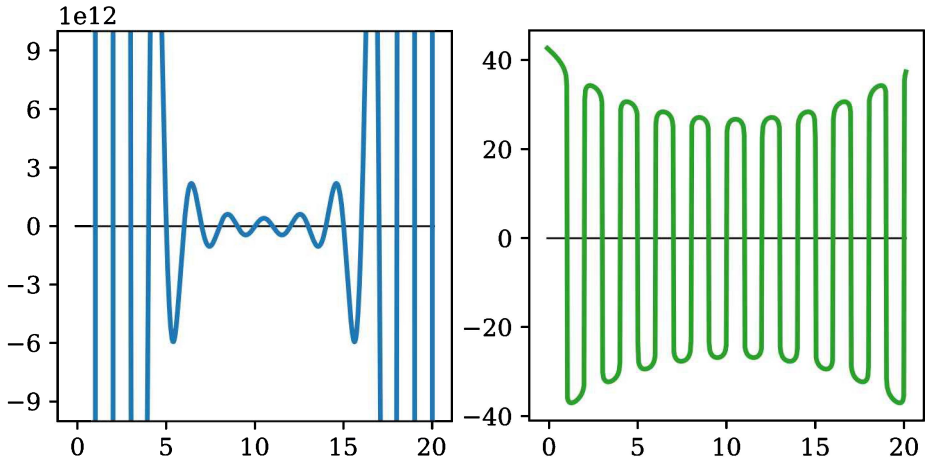


Figure 9.7. *Plots of the Wilkinson polynomial $W(x)$ (left panel, blue) and the signed-log of the Wilkinson polynomial, that is,* $\operatorname{sign}(W(x)) \cdot \log(|W(x)| + 1)$ *(right panel, green). Note that the y-axis in the left panel is measured in multiples of* 10^{12} (1e12). *To get a sense of the scale of the plot, notice that* $W(0) = 2.43 \times 10^{18} = e^{42.34}$, *corresponding to a value of* 42.34 *on the log plot.*

Uniformly Spaced Points

If the $(n+1)$th derivative of f is bounded by M on an interval $I = [a, b]$, so $|f^{(n+1)}(x)| \leq M$ for all $x \in I$, then Theorem 9.4.1 shows the error $\|f - p\|_{L^\infty} = \sup_{x \in I} |f(x) - p(x)|$ is controlled by the polynomial $v(x)$. But, unfortunately, if the nodes are uniformly spaced, then this polynomial can behave very badly. In the special case where the nodes are the integers $1, 2, 3, \ldots, 20$, this polynomial is called the *Wilkinson polynomial* (see Figure 9.7):

$$
\begin{aligned}
W(x) &= \prod_{i=1}^{20} (x - x_i) = (x-1)(x-2)\cdots(x-20) \\
&= x^{20} - 210x^{19} + 20615x^{18} - 1256850x^{17} + 53327946x^{16} - 1672280820x^{15} \\
&\quad + 40171771630x^{14} - 756111184500x^{13} + 11310276995381x^{12} \\
&\quad - 135585182899530x^{11} + 1307535010540395x^{10} - 10142299865511450x^{9} \\
&\quad + 63030812099294896x^{8} - 311333643161390640x^{7} + 1206647803780373360x^{6} \\
&\quad - 3599979517947607200x^{5} + 8037811822645051776x^{4} - 12870931245150988800x^{3} \\
&\quad + 13803759753640704000x^{2} - 8752948036761600000x + 2432902008176640000.
\end{aligned}
$$

Although Wilkinson's polynomial takes on the value zero at each of the nodes $1, 2, \ldots, 20$, it gets very far away from zero between those points, especially near the endpoints of the interval $[1, 20]$. This illustrates that equally spaced points in high-degree polynomials can cause serious problems.

The fact that $v(x)$ (or $W(x)$) can become so absurdly large has led many students, and some professors, of numerical analysis to the mistaken conclusion that

polynomial interpolation is useless. This is not true. With a good choice of interpolation points, polynomial approximation can be very well behaved and extremely useful. The key is to choose the interpolation points judiciously. We see this more in the next subsection and throughout the rest of this chapter.

Nota Bene 9.4.3. One might expect that increasing the number of interpolation points would give a better approximation of the functions, but this is not always the case. It turns out that adding more interpolating points can make the oscillation much worse, rather than better. This is called *Runge's phenomenon*. See Figure 9.8 for an example. But again, choosing the interpolation nodes more judiciously solves this problem. We treat this more in Section 9.4.2.

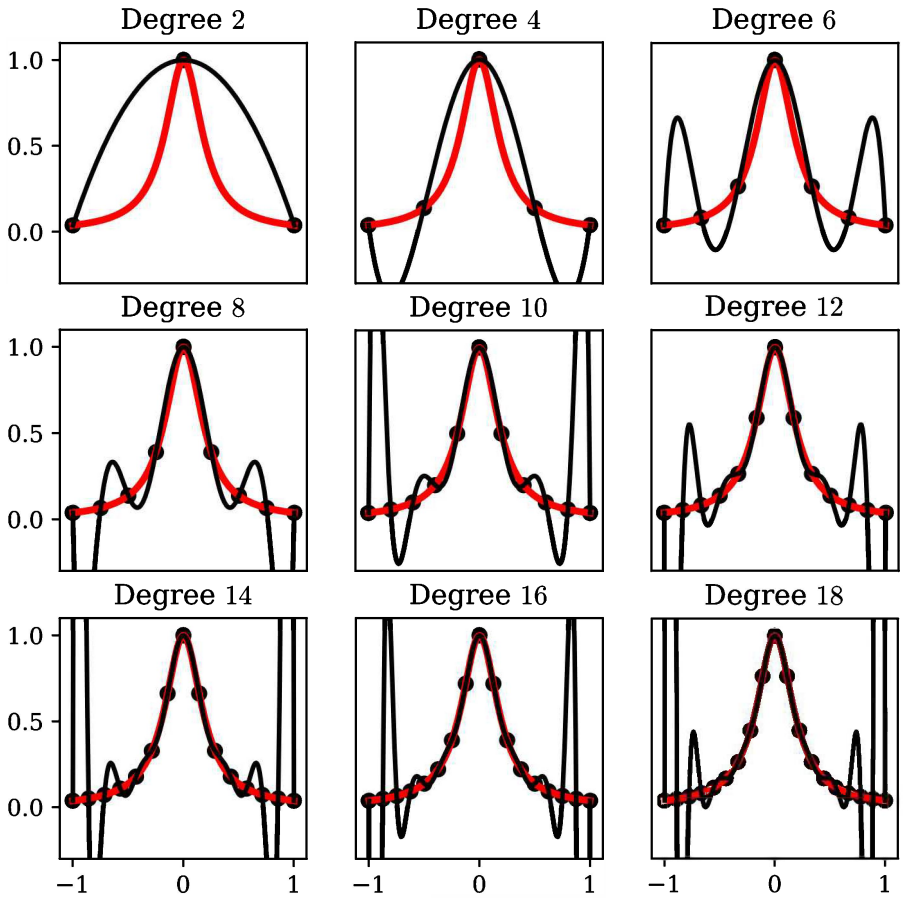


Figure 9.8. *Runge's phenomenon: Interpolating the function* $1/(1+25x^2)$ *(red) at uniformly spaced points. As the number of points increases the interpolation polynomial (black) oscillates more and more wildly near the endpoints.*

9.4.2 Monic Polynomial Approximation

Consider the question of finding the degree-n monic polynomial that best approximates a continuous function f in the uniform norm; that is, given a continuous function f on some interval $I \subset \mathbb{R}$, we seek the polynomial $p \in \mathbb{R}[x;n]$ such that $\|f-p\|_{L^\infty} \leq \|f-q\|_{L^\infty}$ for all $q \in \mathbb{R}[x;n]$, that is,

$$\sup_{x\in I} |f(x)-p(x)| \leq \sup_{x\in I} |f(x)-q(x)|$$

for all $q \in \mathbb{R}[x;n]$. Solving for the absolute best solution to this problem is usually difficult, but we can get very near optimal with Chebyshev polynomials.

In fact, the monic Chebyshev polynomial of degree n is the best degree-n monic polynomial approximation of 0 on the interval $[-1,1]$. This fact is sometimes called the *minimax theorem* because it states that the monic Chebyshev polynomial has the minimal maximum (that is, the smallest L^∞-norm) on $[-1,1]$ of any monic polynomial of degree n.

Theorem 9.4.4 (Minimax Theorem). *If $p(x)$ is a monic polynomial of degree n, then*

$$\frac{1}{2^{n-1}} = \sup_{x\in[-1,1]} |\widehat{T}_n(x)| \leq \sup_{x\in[-1,1]} |p(x)|,$$

where $\widehat{T}_n(x)$ is the monic Chebyshev polynomial of degree n.

Proof. Suppose that $|p(x)| < \frac{1}{2^{n-1}}$ for all $x \in [-1,1]$. If $y_j = \cos\left(\frac{j\pi}{n}\right)$, then for each $j = 0,1,2,\ldots,n$, the monic Chebyshev polynomial satisfies $\widehat{T}_n(y_j) = \frac{(-1)^j}{2^{n-1}}$. Thus

$$(-1)^j \left[\widehat{T}_n(y_j) - p(y_j)\right] > 0 \qquad \forall j = 0,1,2,\ldots,n. \tag{9.36}$$

This means that $\widehat{T}_n(x) - p(x)$ crosses the x-axis n times (has n zeros) in the interval $(-1,1)$. But $\widehat{T}_n(x) - p(x)$ is a polynomial of degree $n-1$, which implies that $\widehat{T}_n(x) = p(x)$, but that contradicts (9.36). □

The monic Chebyshev polynomials satisfy $\widehat{T}_{n+1} = \prod_{j=0}^{n}(x - z_j)$, where the $z_j = \cos\left(\frac{\pi}{n+1}\left(j+\frac{1}{2}\right)\right)$ are the Chebyshev zeros. Because of this, we can give the previous proposition another formulation.

Corollary 9.4.5. *For any collection of points $x_0,\ldots,x_n$, consider the monic polynomial $\prod_{k=0}^{n}(x-x_k)$ of degree $n+1$. The L^∞-norm of this polynomial on the interval $[-1,1]$ is minimized when $\{x_0,\ldots,x_n\} = \{z_0,\ldots,z_n\}$ is the set of degree-$(n+1)$ Chebyshev zeros.*

9.4.3 Error for Interpolation at Chebyshev Roots

The results of the previous subsection, combined with Theorem 9.4.1, give some control of the interpolation error when interpolating at the Chebyshev zeros. Recall

that Theorem 9.4.1 guarantees that for any $x \in [-1,1]$, the interpolating polynomial $p(x)$ of the function $f(x)$ at the points $x_0, \dots, x_n \in [-1,1]$ has the error (9.34)

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) v(x)$$

for some value $\xi \in (-1,1)$, where $v(x) = \prod_{k=0}^{n}(x - x_k)$. In the most general setting, the value $f^{(n+1)}(\xi)$ may be hard to control, since the exact dependence of ξ on x is not specified. But, if $f \in C^{n+1}([-1,1];\mathbb{R})$, then $f^{(n+1)}(\xi)$ is bounded on the compact interval $[-1,1]$. It is reasonable, therefore, to focus on minimizing the product $|v(x)|$ as a proxy for the whole error term. Choosing the interpolating points to be Chebyshev zeros minimizes this product, which gives the following bound on the interpolation error.

Proposition 9.4.6. *If $|f^{(n+1)}(x)|$ is bounded by M on $[-1,1]$, and if $p(x)$ is the degree-n interpolating polynomial of the function $f(x)$ at the Chebyshev zeros $\{z_0, \dots, z_n\}$, given in (9.25), then*

$$\sup_{x \in [-1,1]} |f(x) - p(x)| \le \frac{M}{2^n (n+1)!}. \tag{9.37}$$

Proof. By Theorem 9.4.1 we have

$$\sup_{x \in [-1,1]} |f(x) - p(x)| \le \frac{M}{(n+1)!} \sup_{x \in [-1,1]} \left| \prod_{k=0}^{n} (x - x_k) \right|.$$

By the previous corollary, this bound is minimized when the interpolation points $\{x_0, \dots, x_n\}$ are the Chebyshev zeros $\{z_0, \dots, z_n\}$. And for all $x \in [-1,1]$ we have

$$\left| \prod_{k=0}^{n} (x - z_k) \right| = |\widehat{T}_{n+1}(x)| \le \frac{1}{2^n}$$

by Proposition 9.3.3(ii). □

9.4.4 Interpolation at Chebyshev Extremizers

So far we have focused on interpolation at Chebyshev zeros, but another set of interpolation points with very good behavior is the collection of Chebyshev extremizers $\{y_k = \cos(\pi k/n)\}_{k=0}^{n}$. These have a few advantages over the Chebyshev zeros: first, they include the endpoints of the interval, so we can force the interpolating polynomial to take a certain value at the boundary, and second, they are somewhat simpler to compute with, especially with the methods of the next section.

It should not be too surprising that interpolation at the extremizers gives an approximation that is almost as good as interpolation at the zeros of T_{n+1}, since both of them are the real part of a set of points that are uniformly distributed around the unit circle, and consequently, both of them cluster near the endpoints of the interval.

As with interpolation at the Chebyshev zeros, we can bound the error for interpolation at the Chebyshev extremizers by bounding the product $v(x) = \prod_{j=0}^{n}(x - y_j)$ and using (9.34) in Theorem 9.4.1. First we prove some basic properties of the partial product $\prod_{j=1}^{n-1}(x - y_j)$, which is often denoted $\widehat{U}_{n-1}(x)$ and is called the *monic Chebyshev polynomial of the second kind* of degree $n-1$. Note that $y_0 = -1$ and $y_n = 1$, so

$$v(x) = \prod_{j=0}^{n}(x - y_j) = (x^2 - 1)\widehat{U}_{n-1}(x). \tag{9.38}$$

Proposition 9.4.7. *If $\{y_k = \cos(\pi k/n)\}_{k=0}^{n}$ are the extremizers of $T_n(x)$ on $[-1,1]$, then the polynomials $v(x) = \prod_{j=0}^{n}(x - y_j)$ and $\widehat{U}_{n-1}(x) = \prod_{j=1}^{n-1}(x - y_j)$ satisfy the following:*

(i) $\widehat{U}_{n-1}(x) = \frac{1}{n}\frac{d}{dx}\widehat{T}_n(x)$.

(ii) *If $x = \cos(t)$ with $t \in [0, \pi]$, then $\widehat{U}_{n-1}(x) = \frac{1}{2^{n-1}}\frac{\sin(nt)}{\sin(t)}$.*

(iii) *For all $x \in [-1,1]$ we have $|v(x)| \le \frac{1}{2^{n-1}}$.*

Proof. The proofs of (i) and (ii) are Exercise 9.24. Property (iii) follows from (ii) and (9.38), since

$$\begin{aligned} |v(x)| &= \left|\prod_{j=0}^{n}(x - y_j)\right| = |(x^2-1)\widehat{U}_{n-1}(x)| = \frac{1}{2^{n-1}}\left|\sin^2(t)\frac{\sin(nt)}{\sin(t)}\right| \\ &= \frac{1}{2^{n-1}}|\sin(t)\sin(nt)| \le \frac{1}{2^{n-1}}. \quad \square \end{aligned}$$

Corollary 9.4.8. *If $|f^{(n+1)}(x)|$ is bounded by M on $[-1,1]$, and if $p(x)$ is the interpolating polynomial of the function $f(x)$ at the Chebyshev extremizers $\{y_0, \ldots, y_n\}$, then*

$$\sup_{x\in[-1,1]} |f(x) - p(x)| \le \frac{M}{2^{n-1}(n+1)!}. \tag{9.39}$$

Proof. By Theorem 9.4.1 we have

$$\sup_{x\in[-1,1]} |f(x) - p(x)| \le \frac{M}{(n+1)!} \sup_{x\in[-1,1]} \prod_{k=0}^{n} |x - y_k| \le \frac{M}{2^{n-1}(n+1)!},$$

where the last inequality follows from Proposition 9.4.7(iii). $\square$

This gives a uniform bound on the error for interpolation at the Chebyshev extremizers that is twice the size of the bound (9.37) for interpolation at the Chebyshev zeros. But both of these go to zero rapidly as $n \to \infty$, assuming that f is in $C^\infty([-1,1];\mathbb{R})$ with all derivatives bounded by some fixed $M < \infty$.

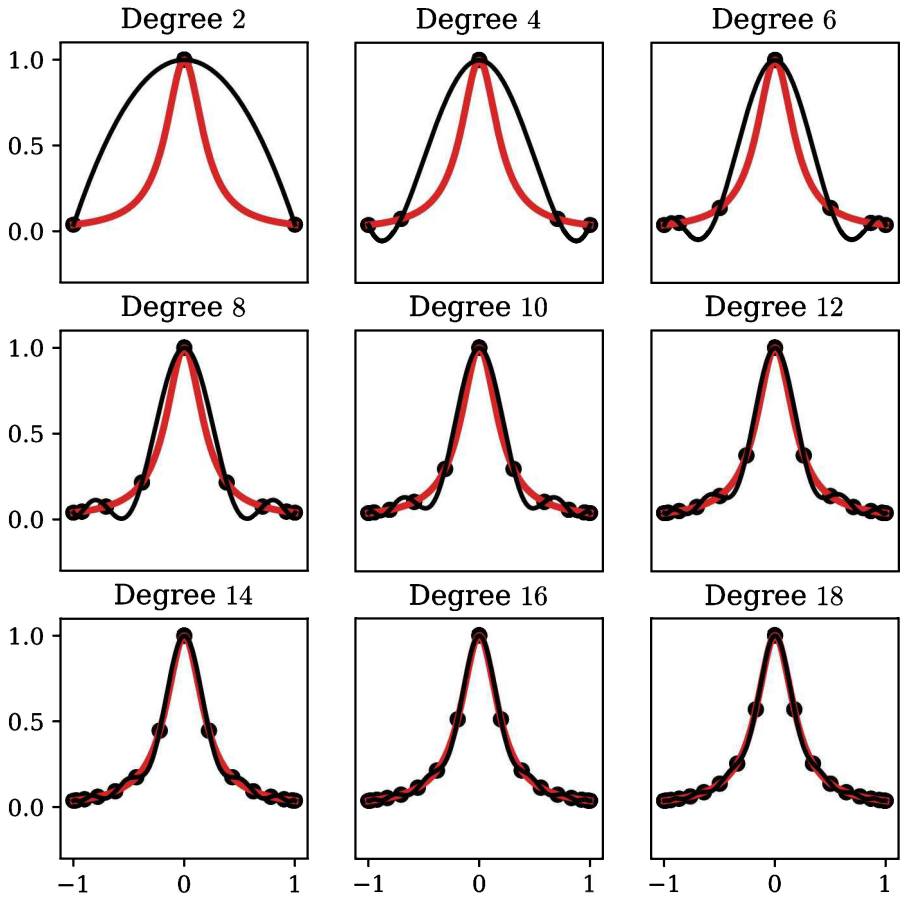


Figure 9.9. *Polynomial interpolation (black) of the function* $1/(1+25x^2)$ *(red) at Chebyshev extremizers. Notice that the interpolation improves as the number of points increases, and Runge's phenomenon does not occur. Contrast this to the interpolation at uniformly spaced points in Figure 9.8.*

Remark 9.4.9. The upshot of this corollary and Proposition 9.4.6 is that interpolation at Chebyshev extremizers and zeros both converge for sufficiently smooth functions. This and the results of the previous section show that the Runge phenomenon is not a problem when the interpolation points are the Chebyshev zeros or extremizers. Even if the interpolating polynomial oscillates around the function f, the maximum total error is bounded and becomes smaller as n grows; for example, see Figure 9.9.

Remark 9.4.10. The Chebyshev polynomials and Chebyshev interpolation can be adapted to any interval $[a,b]$. Specifically, if we want to interpolate f on the interval $[a,b]$, we can make a linear change of variables of the form $g(x) = rx + s$ for some $r, s \in \mathbb{R}$ to get a new function $\tilde{f}(x) : [-1,1] \to \mathbb{R}$ given by $\tilde{f}(x) = f(g(x))$. Interpolation of $\tilde{f}$ on $[-1,1]$ then corresponds to interpolation of f on $[a,b]$. The details are Exercise 9.27.

9.5 Fast Chebyshev Interpolation

Recall that the Chebyshev polynomials $T_0, T_1, \ldots, T_n$ form an orthonormal basis for the vector space $\mathbb{R}[x; n]$ of polynomials of degree at most n. Thus for any $p \in \mathbb{R}[x; n]$ there exists a unique set of coefficients $a_0, \ldots, a_n$ such that

$$p = \sum_{k=0}^{n} a_k T_k. \tag{9.40}$$

The Chebyshev basis has many advantages over the standard monomial basis $\{x^k\}_{k=0}^n$. Among the advantages is the fact that a small change in the coefficients a_k of the Chebyshev basis produces only a small change in the location of the zeros of p inside $[-1, 1]$, but if p is expressed in terms of the standard monomial basis, then a small change in the coefficients can change the zeros of p a lot. We say the rootfinding problem for the Chebyshev basis is *well conditioned*, whereas the rootfinding problem for the monomial basis is *ill conditioned*. The conditioning of a problem describes how much or how little the solution of the problem changes when the inputs to the problem are slightly changed; see Section 11.2 for more on conditioning.

In this section we describe a fast algorithm to express the interpolation of a function f at the Chebyshev extremizers in terms of the Chebyshev basis. As a function, this polynomial is the same as the one constructed by Lagrange interpolation, but it is written in terms of the Chebyshev basis instead of in terms of the Lagrange basis or the standard monomial basis. Since any degree-n polynomial is its own interpolation, this algorithm will also express any degree-n polynomial in terms of the Chebyshev basis.

9.5.1 Fast Chebyshev Interpolation

If the interpolating polynomial p has already been computed in terms of the monomial basis, then the naïve way to change from the monomial basis to the Chebyshev basis is to multiply by the $(n+1) \times (n+1)$ transition matrix (see Volume 1, Section 2.4). The temporal complexity of this algorithm is dominated by the matrix multiplication, which is typically $O(n^3)$.

Alternatively, if we have not yet computed the interpolating polynomial in terms of another basis, we could compute the values of $f(x)$ at the $n+1$ points $x_0, \ldots, x_n$ and compute the values of the Chebyshev basis functions at those points to get a system of $(n+1)$ linear equations in the coefficients a_k. Solving this system also has temporal complexity of $O(n^3)$. But in the case that the interpolating points are the Chebyshev extremizers,[40] there is a much more efficient way to express the interpolating polynomial in terms of the Chebyshev basis. This is based on the relation (9.24)

$$T_n(\cos(t)) = \cos(nt) \qquad \forall n \in \mathbb{N}.$$

This relation reveals a deep and important connection to Fourier series, which permits the use of the FFT to compute the Chebyshev interpolation in terms of the Chebyshev basis very rapidly, in $O(n \log n)$ time. The following theorem is the main result of this section.

[40] Similar methods also work for the Chebyshev zeros, but everything turns out a little messier.

Theorem 9.5.1. *For each $k \in \mathbb{Z}$, let $y_k = \cos(\pi k/n)$. The coefficients $a_0, \dots, a_n \in \mathbb{R}$ of (9.40) are given by a certain multiple of the real part of the DFT of the $2n$-dimensional vector of samples $\{p(y_k)\}_{k=0}^{2n-1}$. Specifically, we have*

$$a_k = \gamma_k \, \Re\left(\mathrm{DFT}(p(y_0), p(y_1), \dots, p(y_{2n-1}))\right)_k, \tag{9.41}$$

where $\Re$ denotes the real part, and the coefficient γ_k is

$$\gamma_k = \begin{cases} 1 & \text{if } k \in \{0, n\}, \\ 2 & \text{otherwise.} \end{cases} \tag{9.42}$$

Proof. Let $\omega_{2n} = e^{2\pi i/2n}$. For the Chebyshev extremizers y_j we have

$$p(y_j) = \sum_{k=0}^{n} a_k T_k(y_j) = \sum_{k=0}^{n} a_k \cos\left(\frac{\pi jk}{n}\right) = \sum_{k=0}^{n} a_k \Re(\omega_{2n}^{jk}) = \Re\left(\sum_{k=0}^{n} a_k \omega_{2n}^{jk}\right).$$

The right side resembles the real part of the inverse discrete Fourier transform (IDFT), except that the upper limit of the summation does not go to $2n-1$, which is what it would need to be for the IDFT, given the way we have indexed things. But we may extend the coefficients to $\{a_k\}_{k=0}^{2n-1}$, by defining $a_{n+j} = a_{n-j}$, $j = 1, 2, \dots, n$. Exercise 9.29 shows that for any $j, k \in \mathbb{Z}$ we have $\Re\left(\omega^{k(n+j)}\right) = \Re\left(\omega^{k(n-j)}\right)$, which gives

$$\sum_{k=n}^{2n} a_k \Re(\omega_{2n}^{jk}) = \sum_{k=0}^{n} a_{n+k} \Re(\omega_{2n}^{j(n+k)}) = \sum_{k=0}^{n} a_{n-k} \Re(\omega_{2n}^{j(n-k)}) = \sum_{k=0}^{n} a_k \Re(\omega_{2n}^{jk}).$$

Moreover, since $a_{2n}\omega_{2n}^{2nj} = a_0$, it follows for $j = 0, 1, 2, \dots, 2n-1$ that

$$\begin{aligned} p(y_j) = \sum_{k=0}^{n} a_k \Re(\omega_{2n}^{jk}) &= \frac{1}{2}\left(\sum_{k=0}^{n} a_k \Re(\omega_{2n}^{jk}) + \sum_{k=0}^{n} a_k \Re(\omega_{2n}^{jk})\right) \\ &= \frac{1}{2}\left(\sum_{k=0}^{n} a_k \Re(\omega_{2n}^{jk}) + \sum_{k=n}^{2n} a_k \Re(\omega_{2n}^{jk})\right) \\ &= \Re\left(\sum_{k=0}^{2n-1} \frac{a_k}{\gamma_k} \omega_{2n}^{jk}\right), \end{aligned}$$

where the last equality follows from the definition of γ_k, which accounts for the double counting of $a_n\omega_{2n}^{jn}$ and of $a_0 = a_{2n}$.

Thus the sample values are the real part of the IDFT of $(\frac{a_0}{\gamma_0}, \frac{a_1}{\gamma_1}, \dots, \frac{a_{2n-1}}{\gamma_{2n-1}})$. Taking the DFT of both sides and rescaling componentwise gives (9.41). □

Remark 9.5.2. Taking the real part of the DFT in the formula above is actually unnecessary, since the DFT of the vector

$$(p(y_0), p(y_1), \dots, p(y_{n-1}), p(y_n), p(y_{n-1}), \dots, p(y_1))$$

is always real; see Exercise 9.30 for details. But we usually take the real part in numerical computations in order to eliminate any imaginary error terms that arise.

Corollary 9.5.3. *If $f : [-1,1] \to \mathbb{R}$ is any function, the degree-n interpolating polynomial p_n of f through the $n+1$ Chebyshev extremizers $\{y_k = \cos(\pi k/n)\}_{k=0}^n$ can be written as*

$$p_n(x) = \sum_{k=0}^{n} a_k T_k.$$

where each a_k is given by

$$\begin{aligned} a_k &= \gamma_k \, \Re\left(\mathrm{DFT}(f(y_0), f(y_1), \ldots, f(y_{2n-1}))\right)_k \\ &= \gamma_k \, \Re\left(\mathrm{DFT}(f(y_0), \ldots, f(y_{n-1}), f(y_n), f(y_{n-1}), \ldots, f(y_1))\right)_k, \end{aligned} \tag{9.43}$$

and γ_k is given in (9.42).

Proof. By definition of the interpolating polynomial, we have

$$p_n(y_k) = f(y_k) \tag{9.44}$$

for each y_k with $k \in \{0, \ldots, n\}$. Moreover, $\cos(\pi + x) = \cos(\pi - x)$ for any $x \in \mathbb{R}$, which gives $y_{n+j} = y_{n-j}$ for each $j \in \{0, \ldots, n\}$; thus (9.44) holds for all $k \in \{0, \ldots, 2n-1\}$. The corollary now follows immediately from Theorem 9.5.1. ∎

Remark 9.5.4. Since the DFT of any vector may be computed using the FFT, the previous corollary gives a method for computing the degree-n interpolation (in the Chebyshev basis) of any function f at the $n+1$ Chebyshev extremizers in $O(n \log n)$ time. Compare this to barycentric Lagrange and Newton interpolation that cost $O(n^2)$ for their initial setup.

This algorithm gives the resulting polynomial as a linear combination of the Chebyshev polynomials rather than as a linear combination of the standard monomial basis, but, as mentioned above, for many applications the Chebyshev basis is actually preferable. Moreover, Clenshaw's algorithm (Exercise 9.20) allows the polynomial expressed in the Chebyshev basis to be evaluated in $O(n)$ time, which is as efficient as evaluating a polynomial in the standard monomial basis.

Finally, by the results in the previous section, if f is sufficiently smooth and n is large enough, then this Chebyshev interpolation is a close approximation to f.

Example 9.5.5. We interpolate the function $f(x) = e^x \sin(2\pi x) + x + 6$ using the method given above. Algorithm 9.1 gives the code that performs the interpolation. A plot of the resulting approximation in degree 29 is given in Figure 9.10. The plot is eyeball perfect. The sup-norm error in this example is 5.3×10^{-15}—that is, essentially perfect, to machine accuracy (for more about the limits of machine accuracy in floating point, see Section 11.1). In other words, with 30 numbers we can store the whole function f to within machine accuracy. This is remarkable.

```python
import numpy as np
from numpy.fft import fft

def cheb_interp(f,n):
    """ Compute the coefficients of the degree-n Chebyshev
    interpolation of f at the extremizers y_k=cos(k pi/n).
    """

    y = np.cos((np.pi * np.arange(2*n)) / n)
    samples = f(y)

    coeffs =  np.real(fft(samples))[:n+1] / n
    coeffs[0] = coeffs[0]/2
    coeffs[n] = coeffs[n]/2

    return coeffs
```

Algorithm 9.1. *Code to produce the degree-n Chebyshev coefficients for any function $f(x)$, as described in Corollary 9.5.3. Note that NumPy's* `fft` *is missing the scaling factor of $1/2n$, so we must put it in explicitly. Combining the scaling factor with γ_k means that the 0th and nth coefficients are scaled by $1/2n$, and the rest of the coefficients are scaled by $1/n$. Note that this algorithm could have used the FFT algorithm in Algorithm 8.2, but the reader should become used to using professional grade packages that have been carefully optimized for speed and usability.*

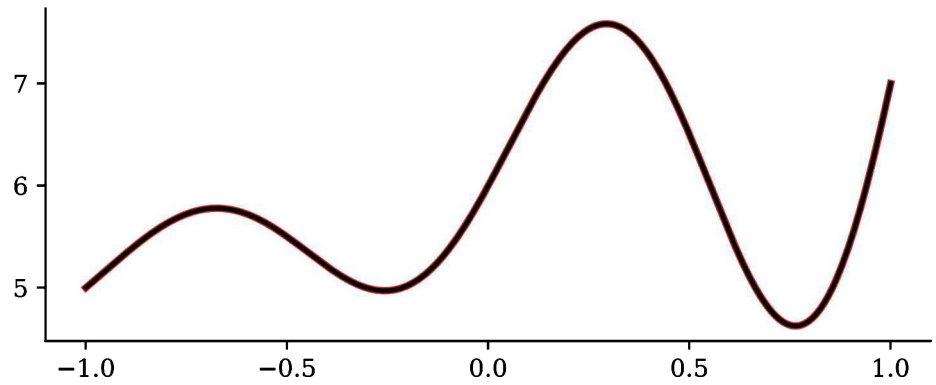


Figure 9.10. *The function in Example 9.5.5 and its degree-29 interpolation using 30 Chebyshev points. The actual function is plotted in red, but it is directly beneath the interpolation (the black curve is plotted with a slightly thinner width so the red can still be seen beneath).*

Example 9.5.6. We interpolate the function $f(x) = x^4$ at the five Chebyshev points $y_0, \ldots, y_4$. Since f is a polynomial of degree 4, we should recover x^4 exactly but expressed in terms of the Chebyshev basis; that is, we should find the coefficients a_k such that $x^4 = \sum_{k=0}^{4} a_k T_k(x)$. Computing the coefficients a_k numerically, as before, we get

$$\begin{aligned} a_0 &= 0.375000000000000, \\ a_1 &= 0.000000000000000, \\ a_2 &= 0.500000000000000, \\ a_3 &= 0.000000000000000, \\ a_4 &= 0.125000000000000. \end{aligned}$$

We can verify this answer by hand:

$$\begin{aligned} \frac{1}{8}T_4(x) + \frac{1}{2}T_2(x) + \frac{3}{8}T_0(x) &= \frac{1}{8}(8x^4 - 8x^2 + 1) + \frac{1}{2}(2x^2 - 1) + \frac{3}{8} \\ &= x^4 - x^2 + \frac{1}{8} + x^2 - \frac{1}{2} + \frac{3}{8} \\ &= x^4. \end{aligned}$$

So the numerical solution using the FFT is the same as (or rather, within machine accuracy of) the exact solution.

Example 9.5.7. The function $g(x) = |x|$ is not smooth at 0, so we cannot expect the Chebyshev interpolation of g to be as good as the interpolation for the smooth functions in Examples 9.5.5 and 9.5.6. In Figure 9.11 we plot the degree-28 Chebyshev interpolation of g. You can see that the approximation is very good away from 0, and it is less accurate near the singular point at 0. The sup-norm error for this approximation is less than 0.0058—far from the extremely small errors of the previous examples, but still a very good low-degree polynomial approximation near this singular point.

Example 9.5.8. Even functions that fail to be differentiable at many points, like the one depicted in Figure 9.12, can still be approximated fairly well with Chebyshev interpolation if the number of interpolation points is sufficiently large. The use of the FFT in the computation of the coefficients means that finding these approximations is still very efficient even when interpolating at many points.

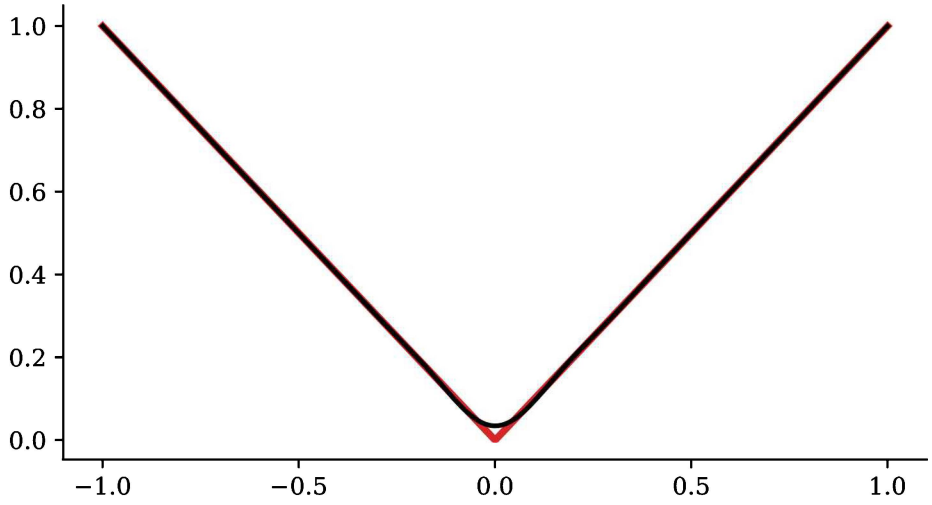


Figure 9.11. *Degree-28 interpolation of the absolute value function, using Chebyshev points for $n = 29$. The actual function is plotted in red and the interpolation in black. The approximation is still very good away from the singularity at 0 and not really terrible even at 0—with an error no worse than 0.0053.*

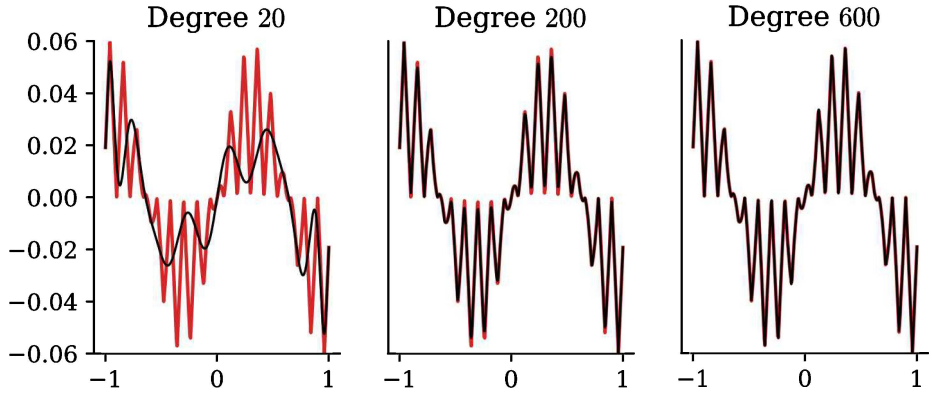


Figure 9.12. *Interpolation of a very singular function using Chebyshev extremizers. The actual function is plotted in red and the interpolation in black. By degree 200, the approximation is visually pretty good, and by degree 600 it has error less than 0.0006.*

9.5.2 *Chebyshev Projections

There is another way to assign a polynomial of degree n to a function $f : [-1, 1] \to \mathbb{R}$, provided that $f \in L^2([-1, 1]; \mathbb{R})$. In this space, endowed with the Chebyshev inner product (9.19), the set of Chebyshev polynomials $T_0, T_1, \ldots, T_n$ of degree at most n is orthogonal, and thus we may project f orthogonally onto the subspace

$\mathbb{R}[x;n] \subset L^2([-1,1];\mathbb{R})$ spanned by these polynomials to get

$$\begin{aligned} f_n &= \operatorname{proj}_{\mathbb{R}[x;n]} f \\ &= \sum_{k=0}^{n} b_k T_k \\ &= \frac{1}{\pi} \langle f, T_0 \rangle T_0 + \frac{2}{\pi} \sum_{k=1}^{n} \langle f, T_k \rangle T_k. \end{aligned}$$

The coefficients $\frac{1}{\pi}$ and $\frac{2}{\pi}$ in the last line are the inverses of $\langle T_0, T_0 \rangle = \pi$ and $\langle T_k, T_k \rangle = \frac{\pi}{2}$ (see (9.23)). In general this projection is not the same as the interpolating polynomial p_n of the previous section, but it is fairly close, as we show below.

The main problem with computing this projection is that the inner products

$$\langle f, T_k \rangle = \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} \, dx$$

are usually not easy to compute, but we can approximate them numerically. One natural way to approximate them is by subdividing the interval $[-1,1]$ at the Chebyshev extremizers y_k and then using the trapezoid rule (see Section 9.6.3). Remarkably, this gives almost the same result as interpolation at the Chebyshev extremizers, as we now show.

For $k > 0$ the coefficient b_k is $\frac{2}{\pi} \langle f, T_k \rangle$. Making the substitution $x = \cos(\theta)$, for $k > 0$ we have

$$\begin{aligned} b_k &= \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} \, dx \\ &= \frac{2}{\pi} \int_{\pi}^{0} \frac{f(\cos(\theta)) \cos(k\theta)}{\sqrt{1-\cos^2(\theta)}} (-\sin(\theta)) \, d\theta \\ &= \frac{2}{\pi} \int_{0}^{\pi} f(\cos(\theta)) \cos(k\theta) \, d\theta. \end{aligned}$$

Dividing the interval $[0,\pi]$ uniformly into n pieces gives a grid of $n+1$ points $\theta_j = \pi j/n$ for $j \in \{0,\dots,n\}$. Using this grid for the trapezoid rule (see Example 9.6.3) to approximate the integral gives

$$\begin{aligned} \frac{2}{\pi} \langle f, T_k \rangle &= \frac{2}{\pi} \int_{0}^{\pi} f(\cos(\theta)) \cos(k\theta) \, d\theta \\ &\approx \frac{2}{\pi} \sum_{j=0}^{n-1} \frac{1}{2} \left(f(\cos(\theta_j)) \cos(k\theta_j) + f(\cos(\theta_{j+1})) \cos(k\theta_{j+1}) \right) \frac{\pi}{n} \\ &= \frac{1}{n} \left(f(y_0) \cos(k\theta_0) + 2 \sum_{\ell=1}^{n-1} f(y_\ell) \cos(k\theta_\ell) + f(y_n) \cos(k\theta_n) \right). \end{aligned}$$

Using the same argument given in the proof of Theorem 9.5.1 shows that this last line is the same as the kth term in the expression

$$2\Re \left(\mathrm{DFT}(f(y_0), f(y_1), \dots, f(y_{2n-1}) \right)_k.$$

Dividing everything by 2 also gives a trapezoid-rule approximation for

$$b_0 \approx \Re\left(\mathrm{DFT}(f(y_0), f(y_1), \dots, f(y_{2n-1}))_0\right).$$

The final result is closely related to the result of Theorem 9.5.1. Indeed, we have shown that

$$b_k \approx \begin{cases} a_k & \text{if } k \neq n, \\ 2a_k & \text{if } k = n, \end{cases} \tag{9.45}$$

so this approximation of the projection f_n is almost identical to the interpolation p_n at the Chebyshev extremizers.

In general, the projection f_n has a somewhat smaller uniform error than the interpolation p_n. But the approximated projection given by (9.45) does not, because it is, after all, just an approximation of the projection f_n. Moreover, the interpolation is guaranteed to pass through the extremizers, including the endpoints $(-1, f(-1))$ and $(1, f(1))$, whereas the projection and its approximation are not.

9.6 Integration by Interpolation

Polynomial approximation is a powerful tool for computing integrals. The main idea of this and the next section is simple: integrating polynomials is easy, so approximate the integrand with a polynomial and integrate the polynomial instead. The methods we consider in the rest of this chapter differ mostly in the choice of how to approximate the integrand. In this section we consider Newton–Cotes, which corresponds to interpolating at uniformly spaced points. To avoid problems arising from the Runge phenomenon, it is generally better to use low-degree interpolations on many subintervals, rather than using a single high-degree interpolation on the entire domain of integration. This is called *composite Newton–Cotes quadrature.*

9.6.1 Numerical Quadrature

Numerical quadrature is a method of choosing points x_i and weights w_i so that for any sufficiently well-behaved functions f, the integral $\int_a^b f(x)\,dx$ is closely approximated by the sum $\sum_i w_i f(x_i)$.

Definition 9.6.1. *A* numerical quadrature *for continuous functions on a bounded interval $[a,b]$ is a set of $n+1$ points (called* nodes*) $x_0 < x_1 < \cdots < x_n$ in $[a,b]$ and a set of corresponding* weights *$w_0, w_1, \dots, w_n \in \mathbb{R}$ for every $n \in \mathbb{Z}^+$ such that for all functions $f \in C([a,b];\mathbb{R})$ we have*

$$\sum_{i=0}^{n} f(x_i) w_i \to \int_a^b f(x)\,dx$$

as $n \to \infty$.

Example 9.6.2. Right-hand Riemann sums are a form of numerical quadrature with equally spaced nodes $a = x_0 < x_1 < \cdots < x_n = b$ and with weights $w_0 = 0$ and $w_i = x_i - x_{i-1}$ for all $i \in \{1, \ldots, n\}$, giving

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_i)(x_i - x_{i-1}).$$

Since all continuous functions are integrable, their Riemann sums converge to the integral as $n \to \infty$, so this simple quadrature can approximate the integral as closely as desired by making n large enough. But the convergence is not very rapid—we usually need to take n to be large in order to get a good approximation; see Figure 9.13.

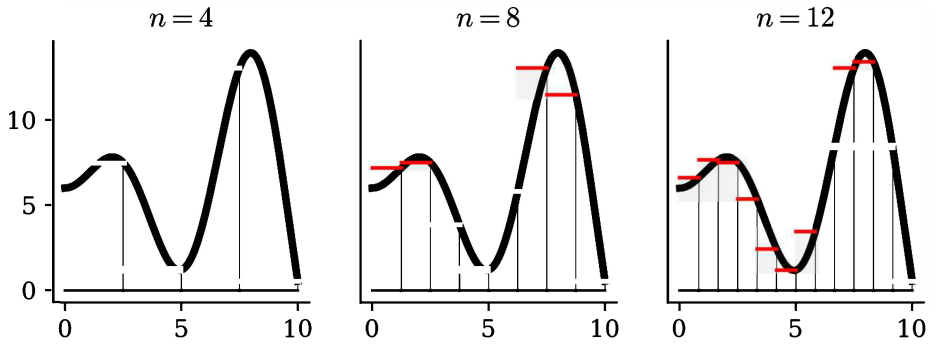


Figure 9.13. *Quadrature by Riemann sums. Here we are taking right-hand sums, as in Example 9.6.2, with $n = 4$, 8, and 12, respectively. The approximation is still relatively poor with $n = 12$ (right panel).*

9.6.2 Quadrature by Polynomial Interpolation

Many quadrature rules arise from integrating a polynomial interpolation. The idea is to choose $n+1$ distinct nodes $x_0, \ldots, x_n$ in the interval $[a, b]$, compute the unique interpolating polynomial $p_n \in \mathbb{R}[x; n]$ for f at those nodes, and then integrate that polynomial as an approximation of the integral of f. This gives

$$\int_a^b f(x)\,dx \approx \int_a^b p_n(x)\,dx = \int_a^b \sum_{j=0}^{n} f(x_j) L_{n,j}(x)\,dx = \sum_{j=0}^{n} f(x_j) w_j, \tag{9.46}$$

where

$$w_j = \int_a^b L_{n,j}(x)\,dx \quad \text{for } j \in \{0, 1, 2, \ldots, n\}, \tag{9.47}$$

and the $L_{n,j}$ are the Lagrange basis functions (9.3).

9.6.3 Newton–Cotes Quadrature

Interpolation quadrature with $n+1$ evenly spaced nodes $a = x_0 < x_1 < \cdots < x_n = b$ in $[a, b]$ is called Newton–Cotes quadrature of order n. Newton–Cotes performs poorly for large values of n because the high-order polynomial approximations of the integrand can suffer from Runge's phenomenon. One remedy, which we explore in the next section, is to choose the nodes more judiciously—either at the Chebyshev extremizers (this is called *Clenshaw–Curtis quadrature*) or at zeros of the Legendre polynomials (called *Gaussian quadrature*).

Another remedy is to chop the domain into subintervals and use lower-order Newton–Cotes quadrature on each subinterval. Assuming that $n = d\ell$, with $d, \ell \in \mathbb{Z}^+$, we can split up the domain as

$$[a, b] = [x_0, x_n] = [x_0, x_d] \cup [x_d, x_{2d}] \cup \cdots \cup [x_{d(\ell-1)}, x_{d\ell}].$$

Since each subinterval has $d+1$ nodes, we can use a degree-d Newton–Cotes quadrature on each subinterval. Thus we have

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{\ell-1} \int_{x_{id}}^{x_{(i+1)d}} p_i(x)\,dx, \tag{9.48}$$

where $p_i(x)$ is a degree-d polynomial. We call (9.48) the *composite Newton–Cotes rule*.

Example 9.6.3. First-order ($d = 1$) composite Newton–Cotes quadrature is better known as the *trapezoid rule*. First consider the contribution of the interval $[x_0, x_1]$, where $h = x_1 - x_0$ to the approximation (9.48). The first-order Newton–Cotes quadrature weights are given by the integrals of the first-order Lagrange basis functions

$$w_0 = \int_{x_0}^{x_1} \frac{x - x_1}{x_0 - x_1}\,dx = \frac{1}{2}(x_1 - x_0) = \frac{h}{2},$$
$$w_1 = \int_{x_0}^{x_1} \frac{x - x_0}{x_1 - x_0}\,dx = \frac{1}{2}(x_1 - x_0) = \frac{h}{2}.$$

Thus we have

$$\int_{x_0}^{x_1} f(x)\,dx \approx f(x_0)w_0 + f(x_1)w_1 = \frac{h}{2}\left(f(x_0) + f(x_1)\right).$$

If we choose the points $x_0, \ldots, x_n$ to be equally spaced with a gap of $h = x_{i+1} - x_i = (b-a)/n$, then these weights are the same for all the subintervals. Thus (9.48) becomes

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} p_i(x)\,dx = \sum_{i=0}^{n-1} \frac{f(x_{i+1}) + f(x_i)}{2} h. \tag{9.49}$$

Remark 9.6.4. One can show that the error in the trapezoid-rule estimate shrinks like $O(n^{-2})$, which means that as n grows, the error of the integral is bounded by a constant times n^{-2}, whereas the errors in the right- or left-hand rule with Riemann sums are in $O(n^{-1})$. It's worth noting that the computational complexity of the trapezoid method is no greater than that of computing a Riemann sum, and so the trapezoid rule gives improved accuracy without a higher computational cost. See also Figure 9.14.

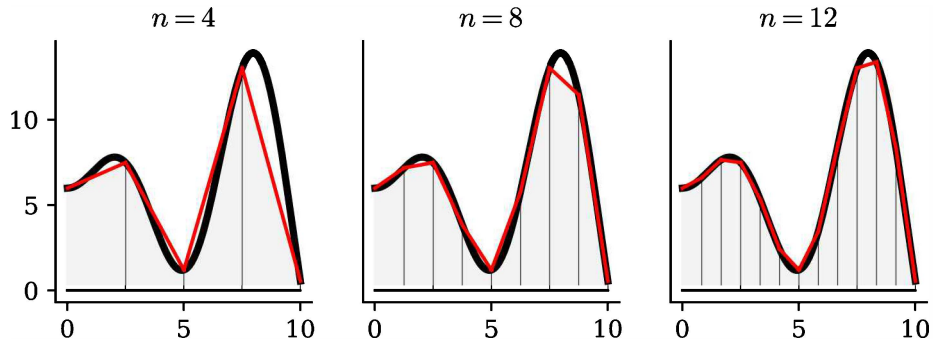


Figure 9.14. *Quadrature by the trapezoid rule with $n = 4$, 8, and 12, respectively. Compare the quality of the approximations here with those for Riemann sums in Figure 9.13.*

Example 9.6.5. The second-order Newton–Cotes quadrature rule is better known as *Simpson's rule*. As with (9.48), begin by approximating the integral of the function $f(x)$ on the interval $[x_0, x_2]$. The second-order Newton–Cotes quadrature weights are given by the integrals of the second-order Lagrange basis functions

$$L_{2,0}(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)},$$
$$L_{2,1}(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)},$$
$$L_{2,2}(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}.$$

It follows that

$$w_0 = \int_{x_0}^{x_2} L_{2,0}(x)\,dx = \int_{x_0}^{x_2} \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}\,dx, \tag{9.50a}$$
$$w_1 = \int_{x_0}^{x_2} L_{2,1}(x)\,dx = \int_{x_0}^{x_2} \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}\,dx, \tag{9.50b}$$
$$w_2 = \int_{x_0}^{x_2} L_{2,2}(x)\,dx = \int_{x_0}^{x_2} \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}\,dx. \tag{9.50c}$$

These integrals are tedious to compute, but some work shows that

$$w_0 = w_2 = \frac{x_2 - x_0}{6} = \frac{h}{3} \qquad \text{and} \qquad w_1 = \frac{2(x_2 - x_0)}{3} = \frac{4h}{3}, \tag{9.51}$$

where $h = x_{i+1} - x_i = \frac{b-a}{n}$ for all i; see Exercise 9.34 for details. Again, the weights do not depend on the actual values of x_2 and x_0—just the difference between them—and so this rule works for all of the subintervals. Thus if there are an even number $n = 2\ell$ of nodes, the degree-2 composite Newton–Cotes rule (Simpson's rule) is

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{\ell} \int_{2i}^{2(i+1)} p_i(x)\,dx = \sum_{i=0}^{\ell-1} \frac{h}{3}\left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right].$$

This simplifies to

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} f(x_i)w_i, \qquad w_i = \frac{h}{3} \times \begin{cases} 1 & \text{if } i = 0, n, \\ 2 & \text{if } i \text{ is even (but not 0 or } n), \\ 4 & \text{if } i \text{ is odd.} \end{cases} \tag{9.52}$$

Remark 9.6.6. Now that we have (9.52), estimating the integral using Simpson's rule is no harder to compute than the trapezoid rule or Riemann sums, but it gives greater accuracy; see also Figure 9.15. From Theorem 9.4.1, it is straightforward to show that the error in Simpson's rule is in $O(n^{-3})$ (see Exercise 9.36), and a more sophisticated argument shows that the error is actually in $O(n^{-4})$.

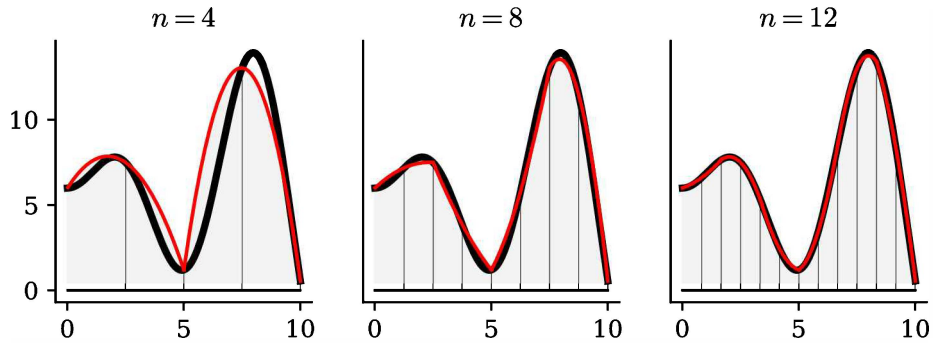


Figure 9.15. *Quadrature by Simpson's rule with $n = 4$, 8, and 12, respectively. Compare the quality of the approximations here with those for Riemann sums in Figure 9.13 and the trapezoid rule in Figure 9.14. The Simpson approximation here for $n = 12$ is visually almost perfect, while the trapezoid rule still has visible flaws at $n = 12$, and Riemann sums are obviously very far from accurate at $n = 12$.*

Example 9.6.7. Table 9.2 shows the result of Simpson's rule and the trapezoid rule applied to the following integral for various values of n:

$$\int_1^{20} \log(x)\,dx = 20\log(20) - 19 \approx 40.914645471079815. \tag{9.53}$$

	Trapezoid Rule		Simpson's Rule	
n	Approximation	Error	Approximation	Error
4	39.5343	1.3803e-00	40.5232	0.3914e-00
16	40.8073	0.1074e-00	40.9047	0.0100e-00
64	40.9077	6.9562e-03	40.9146	7.7541e-05
256	40.9142	4.3600e-04	40.9146	3.3449e-07
1024	40.9146	2.7255e-05	40.9146	1.3162e-09
4096	40.9146	1.7034e-06	40.9146	5.1585e-12

Table 9.2. *Performance of trapezoid rule and Simpson's rule on the integral* (9.53). *For both methods, the results become more accurate with larger values of n. Estimating the big-O rate by hand from the table suggests (correctly) that the trapezoid rule has error roughly $O(n^{-2})$, whereas Simpson's rule has error roughly $O(n^{-4})$.*

Remark 9.6.8. A composite Newton–Cotes quadrature with a larger (but fixed) value of d converges even faster as n goes to infinity, but it is not a good idea to let d grow with n because this would be similar to high-degree, naïve (noncomposite) Newton–Cotes quadrature—using a single polynomial to interpolate through many equally spaced points—and this approximation can be adversely affected by Runge's phenomenon.

9.6.4 Method of Undetermined Coefficients

Naïve computation of the integrals in (9.47) to find the quadrature weights can be painful, but there is another way, based on the observation that $n+1$ distinct points uniquely determine a polynomial of degree at most n through those points. The next proposition is immediate.

Proposition 9.6.9. *Any quadrature rule on $[a,b]$ arising from polynomial interpolation in $n+1$ distinct nodes in $[a,b]$, as in* (9.46), is exact *on $\mathbb{R}[x;n]$, meaning that it computes the value of the integral $\int_a^b f(x)\,dx$ exactly for any $f \in \mathbb{R}[x;n]$.*

Since quadrature by interpolation at $n+1$ points yields exactly the correct answer for any polynomial of degree n or less, for any such quadrature rule we have

$$\frac{b^{k+1}-a^{k+1}}{k+1} = \int_a^b x^k\,dx = \sum_{i=0}^{n} x_i^k w_i \quad \forall k \in \{0,1,\ldots,n\},$$

which gives a linear system of $n+1$ equations and $n+1$ unknowns (the weights). In matrix form, this is written as

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_n \\ x_0^2 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^n & x_1^n & x_2^n & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} b-a \\ (b^2-a^2)/2 \\ (b^3-a^3)/3 \\ \vdots \\ (b^{n+1}-a^{n+1})/(n+1) \end{bmatrix}. \tag{9.54}$$

The matrix on the left-hand side is the transpose of the *Vandermonde matrix*, which is known to be invertible if the x_i are distinct, so a unique solution exists and can be found by solving the linear system. This approach to finding the weights is called the *method of undetermined coefficients*.

Example 9.6.10. If $n = 2$ and the nodes $\{x_0, x_1, x_2\}$ are evenly spaced with $x_0 = a$, $x_1 = \frac{b+a}{2}$ and $x_2 = b$ (Newton–Cotes of order 2), then the corresponding quadrature weights (9.47) are given by the integrals of the second-order Lagrange basis functions

$$\begin{aligned} w_0 &= \int_{x_0}^{x_2} L_{2,0}(x)\,dx = \int_{x_0}^{x_2} \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\,dx, \\ w_1 &= \int_{x_0}^{x_2} L_{2,1}(x)\,dx = \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\,dx, \\ w_2 &= \int_{x_0}^{x_2} L_{2,2}(x)\,dx = \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\,dx. \end{aligned}$$

The method of undetermined coefficients computes these by solving the system (9.54), which, in this case, reduces to

$$\begin{bmatrix} 1 & 1 & 1 \\ 2x_0 & 2x_1 & 2x_2 \\ 3x_0^2 & 3x_1^2 & 3x_2^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} x_2 - x_0 \\ x_2^2 - x_0^2 \\ x_2^3 - x_0^3 \end{bmatrix}.$$

A little work (basic Gaussian elimination) gives the same result for the w_j as (9.51) in Example 9.6.5.

9.7 Clenshaw–Curtis and Gaussian Quadrature

Newton–Cotes quadrature corresponds to interpolating at equally spaced nodes. In this section we consider quadrature methods where the nodes are chosen more judiciously. Throughout this section we work on the interval $[-1, 1]$ rather than on an arbitrary interval $[a, b]$, but we can always rescale so that any integral $\int_a^b f(x)\,dx$ on a compact interval $[a, b]$ can be rewritten as an integral on the interval $[-1, 1]$.

9.7.1 Clenshaw–Curtis Quadrature

Instead of integrating by interpolating at uniformly spaced points, we can interpolate at the extremizers of the Chebyshev polynomials and then integrate the resulting polynomial. This is called *Clenshaw–Curtis quadrature*.

Expressing the interpolating polynomial p_n in terms of the Chebyshev basis $p_n(x) = \sum_{k=0}^{n} a_k T_k(x)$, and then using the fact (see Exercise 9.41) that

$$\int_{-1}^{1} T_k(x)\,dx = \begin{cases} 0 & \text{if } k \text{ is odd,} \\ \frac{2}{1-k^2} & \text{if } k \text{ is even,} \end{cases} \tag{9.55}$$

we obtain

$$\int_{-1}^{1} f(x)\,dx \approx \int_{-1}^{1} p_n(x)\,dx = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{2a_{2k}}{1-4k^2}. \tag{9.56}$$

This gives a simple and efficient method for computing the Clenshaw–Curtis approximation of $\int_{-1}^{1} f(x)\,dx$, namely, compute the interpolating polynomial p_n in terms of the Chebyshev basis $p_n(x) = \sum_{k=0}^{n} a_k T_k(x)$ using fast Chebyshev interpolation, and then apply (9.56).

The standard form for a quadrature rule is

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{j=0}^{n} f(y_j) w_j. \tag{9.57}$$

In this case the $y_j = \cos(\frac{j\pi}{n})$ are the Chebyshev extremizers, and the w_j are the quadrature weights given by (9.47). We can compute the weights w_j by using (9.43) to express the coefficients a_{2k} in (9.56) in terms of the samples $f(y_\ell)$:

$$\begin{aligned} \int_{-1}^{1} p_n(x)\,dx &= \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{2}{1-4k^2} \left(\frac{1}{2n} \sum_{\ell=0}^{2n-1} \gamma_{2k} \cos(-2k\ell\pi/n) f(y_\ell) \right) \\ &= \frac{1}{n} \sum_{\ell=0}^{2n-1} f(y_\ell) \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{\gamma_{2k}}{1-4k^2} \cos(-2k\ell\pi/n) \\ &= \frac{1}{n} \sum_{\ell=0}^{n} \gamma_\ell f(y_\ell) \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{\gamma_{2k}}{1-4k^2} \cos(-2k\ell\pi/n), \end{aligned}$$

where the last line follows from the fact that $y_{n+j} = y_{n-j}$ and $\cos(-2k\ell\pi/n) = \cos(-2k(\ell \pm n)\pi/n)$ (recall from (9.42) that $\gamma_0 = \gamma_n = 1$ and $\gamma_j = 2$ otherwise). Thus we have

$$w_j = \frac{\gamma_j}{n} \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{\gamma_{2k}}{1-4k^2} \cos(-2kj\pi/n). \tag{9.58}$$

Remark 9.7.1. Not surprisingly, there is an efficient algorithm for computing the weights in $O(n \log(n))$ time, using the FFT. This algorithm is due to Waldvogel and is especially advantageous if multiple functions are to be integrated with the same order-n quadrature rule.

9.7.2 Gaussian Quadrature

Gaussian quadrature uses a different set of points for its interpolation, namely the zeros of the Legendre polynomials. Surprisingly, if f is a polynomial of degree at most $2n+1$, then Gaussian quadrature with only $n+1$ points gives precisely the right answer, with no error whatsoever; that is, Gaussian quadrature with $n+1$ points is exact on $\mathbb{R}[x; 2n+1]$. This suggests that whenever a function can be well approximated with an interpolating polynomial through the $n+1$ Legendre zeros, then Gaussian quadrature should give a very good approximation of the integral of that function.

Theorem 9.7.2. *If $\{x_i\}_{i=0}^n \subset [-1,1]$ are the zeros of the $(n+1)$th Legendre polynomial, then for all $f \in \mathbb{R}[x; 2n+1]$ we have*

$$\int_{-1}^{1} f(x)\,dx = \sum_{i=0}^{n} f(x_i) w_i, \tag{9.59}$$

where

$$w_i = \int_{-1}^{1} L_{n,i}(x)\,dx, \quad i = 0, 1, 2, \ldots, n, \tag{9.60}$$

are the integrals of the corresponding Lagrange basis polynomials.

Proof. Let p_{n+1} denote the $(n+1)$th Legendre polynomial. The division algorithm for single-variable polynomials says that for any $f \in \mathbb{R}[x; 2n+1]$ there exist unique polynomials $q \in \mathbb{R}[x; n]$ and $r \in \mathbb{R}[x; n]$ such that $f = qp_{n+1} + r$ (see Volume 1, Section 15.2). Since p_{n+1} is orthogonal to all polynomials of lower degree, it must be orthogonal to q. It follows that

$$\int_{-1}^{1} f(x)\,dx = \int_{-1}^{1} p_{n+1}(x)q(x) + r(x)\,dx = \int_{-1}^{1} r(x)\,dx.$$

Moreover, we have $f(x_i) = r(x_i)$, since $p_{n+1}(x_i) = 0$ for all $i \in \{0, 1, \ldots, n\}$. Thus

$$\begin{aligned}\sum_{i=0}^{n} f(x_i) w_i = \sum_{i=0}^{n} r(x_i) w_i &= \int_{-1}^{1} \sum_{i=0}^{n} r(x_i) L_{n,i}(x)\,dx \\ &= \int_{-1}^{1} r(x)\,dx = \int_{-1}^{1} f(x)\,dx. \quad \square\end{aligned}$$

This theorem shows that only $n+1$ points are needed to evaluate the integral of a polynomial of degree $2n+1$ exactly. Thus any integrable function $g : [-1,1] \to \mathbb{R}$ can be integrated fairly accurately if it is closely approximated by a polynomial f of degree $2n+1$ or less that agrees with g at the Legendre zeros $x_0, \ldots, x_n$. For small n, the zeros (x_j) and weights (w_j) can be precomputed and stored to be used for quadrature of any function. These zeros and weights for Gaussian quadrature are built in to many computational systems. For large n there is a fast algorithm for computing the zeros and weights in $O(n)$ time [Bog14].

Example 9.7.3. The monic Legendre polynomial of degree 2 is $x^2 - \frac{1}{3}$, so its zeros are $\pm\frac{1}{\sqrt{3}}$. Theorem 9.7.2 guarantees that

$$\int_{-1}^{1} p(x)\,dx = w_0\,p\left(\frac{-1}{\sqrt{3}}\right) + w_1\,p\left(\frac{1}{\sqrt{3}}\right)$$

for all $p(x) \in \mathbb{R}[x;3]$, with $w_i = \int_{-1}^{1} L_{1,i}(x)\,dx$. A straightforward calculation shows that $L_{1,0}(x) = -\frac{\sqrt{3}}{2}(x - \frac{1}{\sqrt{3}})$ and $L_{1,1}(x) = \frac{\sqrt{3}}{2}(x + \frac{1}{\sqrt{3}})$, and integrating these gives $w_0 = w_1 = 1$.

This implies that

$$\begin{aligned}\int_{-1}^{1} \left(ax^3 + bx^2 + cx + d\right)\,dx &= a\left(\frac{-1}{\sqrt{3}}\right)^3 + b\left(\frac{-1}{\sqrt{3}}\right)^2 + c\left(\frac{-1}{\sqrt{3}}\right) + d \\ &\quad + a\left(\frac{1}{\sqrt{3}}\right)^3 + b\left(\frac{1}{\sqrt{3}}\right)^2 + c\left(\frac{1}{\sqrt{3}}\right) + d \\ &= \frac{2b}{3} + 2d\end{aligned}$$

for all $a, b, c, d, \in \mathbb{R}$. The coefficients of terms of odd degree do not contribute in the computation because all odd functions integrate to 0 on $[-1, 1]$.

Example 9.7.4. Given any function f for which there is a good degree-3 polynomial approximation of f on $[-1, 1]$ that agrees with f at the points $\pm\frac{1}{\sqrt{3}}$, we have the approximation

$$\int_{-1}^{1} f(x)\,dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right). \tag{9.61}$$

For example, the function $f(x) = \sqrt{\cos(x)}$ on the interval $[-1, 1]$ has

$$\int_{-1}^{1} \sqrt{\cos(x)}\,dx = 1.82796941\ldots,$$

while the approximation (9.61) gives

$$\int_{-1}^{1} \sqrt{\cos(x)}\,dx \approx \sqrt{\cos(-1/\sqrt{3})} + \sqrt{\cos(1/\sqrt{3})} = 1.83075048,$$

so with only two evaluations of the function we have an approximation of the integral that is accurate within 3×10^{-3}.

Example 9.7.5. In the case of $n = 4$, using the five zeros $x_0, \dots, x_4$ of the degree-5 Legendre polynomial and the five weights $w_0, \dots, w_4$ gives

$$\int_{-1}^{1} f(x)\,dx \approx w_0 f(x_0) + w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3) + w_4 f(x_4).$$

Using this approximation for the integral in Example 9.7.4 gives

$$\int_{-1}^{1} \sqrt{\cos(x)}\,dx \approx 1.82797138,$$

which has an error of 1.97×10^{-6}.

9.7.3 Convergence

As n gets large, naïve Newton–Cotes quadrature (not the composite form) often fails to converge to the integral it is intended to approximate because of Runge's phenomenon—as the degree increases the interpolating polynomial oscillates more and more wildly, and the weights w_i can grow very large. But the next theorem shows that Clenshaw–Curtis and Gaussian quadrature don't have this problem, since both converge for any continuous function. First we need the following lemma.

Lemma 9.7.6. *The weights w_j in Clenshaw–Curtis quadrature and Gaussian quadrature are positive for all $j \in \{0, \dots, n\}$, and they satisfy $\sum_{j=0}^{n} w_j = 2$.*

Proof. By (9.58) the Clenshaw–Curtis weights satisfy

$$w_j = \frac{\gamma_j}{n}\left(1 - \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\gamma_{2k}}{4k^2 - 1} \cos(-2k\ell\pi/n)\right).$$

Since $1 \le \gamma_k \le 2$ for all k (see (9.42)) this gives

$$\begin{aligned} w_j &\ge \frac{1}{n}\left(1 - \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} \frac{2}{4k^2 - 1}\right) \\ &= \frac{1}{n}\left(1 - \frac{2n}{2n+1}\right) > 0, \end{aligned}$$

where the last equality follows from Exercise 1.17(ii).

For positivity of the Gaussian quadrature weights, consider the Lagrange basis polynomials $L_{n,i}(x) \in \mathbb{R}[x; n]$. Recall that $L_{n,j}(x_i) = \delta_{i,j}$. For each $j \in \{0, 1, 2, \dots, n\}$ we have $L_{n,j}(x)^2 \in \mathbb{R}[x; 2n]$, and, by Theorem 9.7.2, this gives

$$w_j = L_{n,j}(x_j)^2 w_j = \sum_{i=0}^{n} L_{n,j}(x_i)^2 w_i = \int_a^b L_{n,j}(x)^2\,dx > 0.$$

Finally, each quadrature method is exactly correct for polynomials of degree at most n, and so for the polynomial $1 \in \mathbb{R}[x; 0]$ we get $\sum_{k=0}^{n} w_k = \int_{-1}^{1} 1\,dx = 2$. ∎

Theorem 9.7.7. *If f is continuous on $[-1,1]$, then Clenshaw–Curtis and Gaussian quadrature both converge to $\int_{-1}^{1} f(x)\,dx$ as $n \to \infty$.*

Proof. Given $\varepsilon > 0$, the Weierstrass approximation theorem (Theorem 9.1.7), guarantees there exists $n > 0$ such that

$$\|f - p\|_{L^\infty} < \frac{\varepsilon}{4}$$

for some $p \in \mathbb{R}[x; n]$. This implies

$$\begin{aligned}
\left| \int_{-1}^{1} f(x)\,dx - \sum_{i=0}^{n} f(x_i) w_i \right| &\leq \left| \int_{-1}^{1} f(x)\,dx - \int_{-1}^{1} p(x)\,dx \right| \\
&\quad + \left| \sum_{i=0}^{n} p(x_i) w_i - \sum_{i=0}^{n} f(x_i) w_i \right| \\
&\leq \int_{-1}^{1} |f(x) - p(x)|\,dx + \sum_{i=0}^{n} |f(x_i) - p(x_i)||w_i| \\
&\leq 2\|f - p\|_{L^\infty} + \|f - p\|_{L^\infty} \left(\sum_{i=0}^{n} |w_i| \right) \\
&\leq 4\|f - p\|_{L^\infty} < \varepsilon. \quad \square
\end{aligned}$$

Convergence for Analytic Functions

For a special class of functions called *analytic functions*, Gaussian and Clenshaw–Curtis quadrature converge very rapidly. We say that f is *analytic* at a point x if f has a convergent Taylor series expansion in an open neighborhood of x. All polynomials and polynomial combinations of e^x, $\sin(x)$, $\cos(x)$, as well as many other common functions, are analytic at all points of $\mathbb{C}$; see Volume 1, Chapter 11.

Given $\rho > 1$, if a function is analytic on an ellipse in $\mathbb{C}$ with foci at ± 1, and with major and minor semiaxes $\cosh(\rho)$ and $\sinh(\rho)$, respectively, then the error in Clenshaw–Curtis quadrature applied to f is in $O(\rho^{-n})$ as $n \to \infty$, and the error in Gaussian quadrature is in $O(\rho^{-2n})$; see [Tre13] for details. Contrast these with Simpson's rule, which is in $O(n^{-4})$ or with naïve (noncomposite) Newton–Cotes, which does not necessarily converge at all.

Remark 9.7.8. Because the exponent of ρ in the convergence bound for Gaussian quadrature is $-2n$, while the corresponding coefficient for Clenshaw–Curtis is $-n$, you might think that Gaussian quadrature should converge significantly faster than Clenshaw–Curtis quadrature. But in practice that is not usually the case. Clenshaw–Curtis quadrature converges about as rapidly as Gaussian quadrature, and both converge very rapidly when the integrand is analytic.

9.7.4 *Gaussian Quadrature with Other Orthogonal Polynomials

Theorem 9.7.2 can be extended to orthogonal polynomials with other domains and weights. Any class of orthogonal polynomials corresponding to an inner product

$\langle f, g\rangle = \int_a^b f(x)g(x)w(x)\,dx$ with weight function $w(x)$ satisfies

$$\int_a^b p(x)w(x)\,dx = \sum_{i=0}^{n} p(x_i)w_i \tag{9.62}$$

for any $p \in \mathbb{R}[x; 2n+1]$, where

$$w_i = \int_a^b L_{n,i}(x)w(x)\,dx, \quad i = 0, 1, 2, \ldots, n,$$

are the quadrature weights, and $\{x_i\}_{i=0}^n \subset [a, b]$ are the zeros of the degree-$(n+1)$ polynomial $p_{n+1}(x)$ in the w-orthogonal polynomial basis. The proof is like that of Theorem 9.7.2, but we must show that the zeros of the corresponding w-orthogonal basis functions do, in fact, all lie in the interval (a, b).

Lemma 9.7.9. *Assume $w(x) > 0$ is continuous on $[a, b]$. If a nontrivial function $f \in C([a, b]; \mathbb{R})$ is w-orthogonal to $\mathbb{R}[x; n]$, that is,*

$$\int_a^b f(x)p(x)w(x)\,dx = 0 \quad \forall p \in \mathbb{R}[x; n],$$

then f changes sign at least $n+2$ times on (a, b).

Proof. By hypothesis, f is w-orthogonal to $1 \in \mathbb{R}[x; n]$. Thus f must change sign at least once in (a, b); otherwise the integral $\int_a^b f(x)w(x)\,dx$ would be nonzero, which is a contradiction. Suppose that f changes sign exactly $r+1 \leq n+1$ times in (a, b). Since f is continuous, it must, therefore, have at least $r \leq n$ distinct zeros $\{t_i\}_{i=1}^r$ in the interval (a, b). The polynomial

$$p(x) = \prod_{i=1}^{r}(x - t_i) \in \mathbb{R}[x; r]$$

has degree $r \leq n$ and changes signs precisely when f does, and thus

$$\int_a^b f(x)p(x)w(x)\,dx \neq 0,$$

which contradicts the w-orthogonality hypothesis. Therefore f changes sign at least $n+2$ times in (a, b). □

Corollary 9.7.10. *The degree-n w-orthogonal polynomial on the domain $[a, b]$ has exactly n distinct real zeros in the interior of its domain (a, b). Moreover, all of the zeros are simple (multiplicity one).*

Corollary 9.7.11. *Assume $w(x) > 0$ is in $C([a, b]; \mathbb{R})$. The class of orthogonal polynomials corresponding to the inner product $\langle f, g\rangle = \int_a^b f(x)g(x)w(x)\,dx$ with weight function $w(x)$ satisfies*

$$\int_a^b p(x)w(x)\,dx = \sum_{i=0}^{n} p(x_i)w_i$$

for any $p \in \mathbb{R}[x; 2n+1]$, *where*

$$w_i = \int_a^b L_{n,i}(x) w(x)\, dx, \quad i = 0, 1, 2, \ldots, n,$$

are the quadrature weights, and $\{x_i\}_{i=0}^n \subset [a, b]$ *are the zeros of the* $(n+1)$*th polynomial* $p_{n+1}(x)$ *in the* w*-orthogonal polynomial basis.*

Proof. The proof is essentially the same as that for Theorem 9.7.2. □

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

9.1. Show that the kth Bernstein polynomial $B_k^n(x)$ of degree n has a local maximum at $x = \frac{k}{n}$.

9.2. Prove that for any $n \in \mathbb{Z}^+$ and any $f : [0, 1] \to \mathbb{R}$ the polynomial $B_n[f]$ agrees with f at the endpoints: $f(0) = B_n[f](0)$ and $f(1) = B_n[f](1)$.

9.3. Prove Lemma 9.1.4. Hint: It may be useful to employ the results or techniques of Exercise 1.42.

9.4. Prove weak monotonicity of the Bernstein operator (Lemma 9.1.6(ii)). Hint: Consider $B_n[g - f]$ and recall that $B_k^n(x) \geq 0$ for all $x \in [0, 1]$.

9.5. Code up a method to compute the polynomial $B_n[f]$ for any $n \in \mathbb{N}$ and any callable function f. Use your code to reproduce the plots in Figure 9.2.

9.6. Another way to interpolate is to solve the linear system

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

The $(n+1) \times (n+1)$ matrix is called the *Vandermonde matrix*.

(i) Prove the Vandermonde matrix is nonsingular when the x_i are distinct.

(ii) Use the Vandermonde matrix to find the unique cubic polynomial that interpolates the set

$$A = \{(-1, 2), (0, -4), (1, -6), (2, -16)\}.$$

9.7. Use Lagrange interpolation to find the unique cubic polynomial that interpolates the set A from Exercise 9.6. Your answer need not be written in the form $a_3x^3 + a_2x^2 + a_1x + a_0$.

9.8. Use barycentric Lagrange interpolation to find the unique cubic polynomial that interpolates the set A from Exercise 9.6. Your answer need not be written in the form $a_3x^3 + a_2x^2 + a_1x + a_0$.

9.9. Code up a method to compute the barycentric weights for a given set of distinct points $\{x_0, \ldots, x_n\}$. Using your weights method (and the barycentric construction), code up a method to evaluate the unique interpolating polynomial of $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ at any point x.

9.10. Using your method from the previous problem, for each of $n = 2, 3, \ldots, 20$ compute and plot the interpolation polynomial for the function $f(x) = |x|$ at $n+1$ evenly spaced points on the interval $[-1, 1]$ and graph f, for comparison, on the same plot. Which of these interpolating polynomials has the smallest error (measured in terms of the L^∞-norm) on the interval $[-1, 1]$?

9.11.* Prove the claim made in Remark 9.2.15: If a new point x_{n+1} is added to the set of points to interpolate, then the new barycentric weights for the set $\{x_0, \ldots, x_n, x_{n+1}\}$ can be computed in $O(n)$ time by giving an algorithm for computing the new w_j using the old w_j, and then show that the algorithm has temporal complexity in $O(n)$.

9.12.* Use Newton interpolation to find the unique cubic polynomial that interpolates the set A from Exercise 9.6. Your answer need not be written in the form $a_3x^3 + a_2x^2 + a_1x + a_0$.

9.13.* Write up the details of the proof of Proposition 9.2.18.

9.14.* Prove that the computation of the a_k in Newton iteration has temporal complexity $O(n^2)$, while the computation of $p(x)$ for any x is $O(n)$, once the a_k are known.

9.15. Compute the first three monic Legendre polynomials.

9.16. Compute the first five monic Chebyshev polynomials.

9.17.† ⚠ Using the recursive relation (9.20), prove Proposition 9.3.1; that is, show that the monic Chebyshev polynomials satisfy the relation

$$\widehat{T}_k(x) = \frac{1}{2^{k-1}} \cos\left(k \cos^{-1}(x)\right) \quad \text{for } k \in \mathbb{Z}^+. \tag{9.63}$$

Hint: Let $\theta = \cos^{-1} x$ and use the trigonometric identity:

$$\cos\left[(k \pm 1)\theta\right] = \cos(\theta)\cos(k\theta) \mp \sin(\theta)\sin(k\theta).$$

9.18. Use (9.63), above, to show that the monic Chebyshev polynomials $\{\widehat{T}_k(x)\}_{k=0}^{\infty}$ are orthogonal with respect to the inner product

$$\langle f, g \rangle = \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}}\, dx.$$

Hint: Recall the trigonometric identity

$$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta) + \cos(\alpha-\beta).$$

9.19. Prove that the nonmonic Chebyshev polynomials satisfy

$$\langle T_n, T_n \rangle = \begin{cases} \pi & \text{if } n = 0, \\ \pi/2 & \text{if } n \neq 0. \end{cases}$$

9.20. **Clenshaw's Algorithm:** The Chebyshev polynomials $T_0, \ldots, T_n$ form a basis of $\mathbb{R}[x;n]$ over $\mathbb{R}$, so any polynomial in $\mathbb{R}[x;n]$ can be written uniquely in the form $p = \sum_{k=0}^{n} a_k T_k$ with $a_k \in \mathbb{R}$ for all $k \in \{0, \ldots, n\}$. Consider the following algorithm: For any fixed $x \in [-1,1]$, set $u_{n+1} = 0$ and $u_n = a_n$, and recursively compute

$$u_k = 2xu_{k+1} - u_{k+2} + a_k \qquad \text{for } k = n-1, n-2, \ldots, 0.$$

(i) Prove, using the recurrence (9.22), that $p(x) = \frac{1}{2}(a_0 + u_0 - u_2)$. This is called *Clenshaw's algorithm* for evaluating polynomials in the Chebyshev basis at points in x.

(ii) Compute the leading-order (both temporal and spatial) complexity of Clenshaw's algorithm.

9.21.* Prove Proposition 9.3.3.

9.22.* Use equation (9.63) to show that the monic Chebyshev polynomials satisfy the ordinary differential equation

$$(1-x^2)\widehat{T}_k''(x) - x\widehat{T}_k'(x) + k^2\widehat{T}_k(x) = 0 \qquad \forall k \in \mathbb{Z}^+. \tag{9.64}$$

The nonmonic polynomials also satisfy this ordinary differential equation, since it is a linear equation.

9.23.* Prove that the monic Chebyshev polynomials satisfy

$$\widehat{T}_k(x) = \frac{2(-1)^k k!}{(2k)!}\sqrt{1-x^2}\,\frac{d^k}{dx^k}(1-x^2)^{k-1/2} \qquad \forall k \in \mathbb{Z}^+. \tag{9.65}$$

Hint: Follow the same approach as in the proof of Theorem 9.3.6.

9.24. Prove parts (i) and (ii) of Proposition 9.4.7.

9.25. Use barycentric Lagrange interpolation to find the degree-3 interpolating polynomial for the data $(-1, \sin(-\pi))$, $(-1/3, \sin(-\pi/3))$, $(1/3, \sin(\pi/3))$, and $(1, \sin(\pi))$; that is, interpolate through the points $\{(x_j, \sin(\pi x_j)\}_{j=0}^{3}$ for $x_0 = -1$, $x_1 = -1/3$, $x_2 = 1/3$, and $x_3 = 1$. Plot your answer and the function $\sin(\pi x)$ on the interval $[-1,1]$.

9.26. Repeat the previous problem, but instead use the Chebyshev extremizers $y_j = \cos(j\pi/3)$; that is, interpolate through the points $\{(y_j, \sin(\pi y_j))\}_{j=0}^{3}$. On the interval $[-1,1]$ plot your answer along with the function $\sin(\pi x)$ and your answer to the previous problem.

9.27. Work out the details for Remark 9.4.10; that is, give a formula for the linear change of variables $\tilde{x}$ that maps $[-1,1]$ to $[a,b]$, and give an explicit formula for the points of $[a,b]$ corresponding to the Chebyshev zeros under this map. Give the analogue of Proposition 9.4.6 for interpolation at these points on the interval $[a,b]$.

9.28. Let $z_1', \ldots, z_{20}'$ be the points in $[1,20]$ corresponding to the degree-20 Chebyshev zeros, as computed in the previous problem. Plot the polynomial $q(x) = \prod_{i=1}^{20}(x - z_i')$, on the interval $[1,20]$, and plot the Wilkinson polynomial $W(x)$ on the same graph. Compute $\sup_{x\in[1,20]} q(x)$, and compare this to $\sup_{x\in[1,20]} W(x)$.

9.29. Let $j,k,n \in \mathbb{Z}$ with $n \neq 0$. Prove that if $\omega = e^{i\pi/n}$, then $\Re\left(\omega^{k(n+j)}\right) = \Re\left(\omega^{k(n-j)}\right)$.

9.30. Let $n \in \mathbb{Z}^+$ be given. Assuming the vector $(a_0, a_1, \ldots, a_{2n-1}) \in \mathbb{R}^{2n}$ satisfies $a_{n+j} = a_{n-j}$ for $j = 1,2,\ldots,n-1$, prove that the Fourier coefficients

$$c_k = \frac{1}{2n}\sum_{j=0}^{2n-1} a_j \omega_{2n}^{-jk}$$

are all real.

9.31. The polynomial approximation $p(x)$ computed in Exercise 9.26 can be written as $p(x) = \sum_{k=0}^{3} a_k T_k$. Use the DFT, as described in the text, to compute the coefficients a_k by hand. Expand the sum and collect like terms to prove that your answer gives the same polynomial as $p(x)$.

9.32. Using Algorithm 9.1, plot the Chebyshev interpolating polynomials of degree 2^k for $k = 1,\ldots,7$ for the function

$$f(x) = \begin{cases} 1+x & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases}$$

on the interval $[-1,1]$. Also graph f for comparison in each of the plots.

9.33.* Code up the algorithm for computing the trapezoid-rule approximation of the Chebyshev projection method, as described in Section 9.5.2. Repeat Exercise 9.32 using your projection code instead of Algorithm 9.1, and compare the results to the results of Exercise 9.32.

9.34. Compute the integral of one of the three Lagrange basis functions for the second-order Newton–Cotes quadrature and verify that it equals the corresponding weight.

9.35. Show that Simpson's rule is exact for the integral $\int_a^b x^k\,dx$ for any $k \in \{0,1,2,3\}$. Use this to prove that Simpson's rule is exact for any cubic polynomial. Hint: First prove the results just for three points x_0, x_1, x_2, and then

apply that result to the triple of points x_2, x_3, x_4, and then again for each triple of the form $x_{2k}, x_{2k+1}, x_{2k+2}$, which then gives the result for Simpson's rule for any even $n \geq 2$.

9.36. Assume that $f \in C^3([a,b];\mathbb{R})$.

(i) Show that $|f^{(3)}(x)|$ is bounded by some $M < \infty$.

(ii) Show that if p is the degree-2 interpolating polynomial for f at uniformly spaced points $x_0 < x_1 < x_2$, then

$$\sup_{x\in[x_0,x_2]} |f(x) - p(x)| \leq \frac{M}{6} \sup_{x\in[x_0,x_2]} \prod_{i=0}^{2} (x - x_i).$$

Hint: Consider using Theorem 9.4.1.

(iii) Under the assumptions of the previous step, show that

$$\sup_{x\in[x_0,x_2]} |f(x) - p(x)| \leq \frac{M}{3} h^3,$$

where $h = x_1 - x_0 = x_2 - x_1$.

(iv) Show that the degree-2 Newton–Cotes approximation I_2 of $I = \int_{x_0}^{x_2} f(x)\,dx$ has error $|I - I_2|$ bounded by $\frac{2M}{3}h^4$.

(v) Show that the error arising from using Simpson's rule to approximate $\int_a^b f(x)\,dx$ lies in $O(n^{-3})$ as $n \to \infty$.

9.37. Using the data in Table 9.2, show empirically that the error for the trapezoid rule is roughly $O(n^{-2})$ and the error for Simpson's rule is roughly $O(n^{-4})$. Specifically, assume that the errors are of the form $t = kn^a$ and then estimate a using the data for various values of n.

9.38. Find the weights w_0, w_1, w_2 so that

$$\int_0^1 p(x)\,dx = p(0)w_0 + p(1/2)w_1 + p(1)w_2 \quad \forall p \in \mathbb{R}[x;3].$$

9.39.* Determine the values of the nodes x_0, x_1, x_2 and the weights w_0, w_1, w_2 so that

$$\int_0^1 p(x)x^2\,dx = p(x_0)w_0 + p(x_1)w_1 + p(x_2)w_2 \quad \forall p \in \mathbb{R}[x;3].$$

9.40.* For what value of $c \in (0,2)$ is the following quadrature rule exact for all $p \in \mathbb{R}[x;2]$?

$$\int_0^2 p(x)\,dx = p(c) + p(2-c)$$

9.41. Prove that $\int_{-1}^{1} T_k(x)\,dx$ is zero if k is odd and is $\frac{2}{1-k^2}$ if k is even; see (9.55).

9.42. Prove that Gaussian quadrature with $n+1$ nodes cannot be exact for all $f \in \mathbb{R}[x;2n+2]$.

9.43. Compute the Taylor polynomial of degree 3 for $\sin(x+3)$ around 0. Use this and the results of Example 9.7.3 to estimate the integral

$$\int_{-1}^{1} \sin(x+3)\,dx.$$

Compare your computation to the value of the integral computed symbolically.

9.44. Generalize Theorem 9.7.2 from the interval $[-1,1]$ to an arbitrary interval $[a,b]$. Give explicit formulas for the appropriate sample points in terms of the zeros of the Legendre polynomials, and give explicit formulas for the appropriate weights in terms of integrals of the Lagrange basis polynomials. Prove your generalized theorem is correct.

9.45. (i) Using built-in methods for finding the zeros and weights for Gaussian quadrature, code up a method that accepts any callable function f on $[-1,1]$ and any integer $n > 1$ and uses Gaussian quadrature at $n+1$ points to approximate the integral $\int_{-1}^{1} f(x)\,dx$.

(ii) Using your method, compute the Gaussian quadrature estimate of the integral $\int_{-1}^{1} |x|\,dx$ for $n = 10, 20, 30, \ldots, 100$. Compare your results to the true answer (which is 1).

(iii) Repeat the computation for $\int_{-1}^{1} \cos(x)\,dx$. Compare your results to the true answer of $2\sin(1)$.

(iv) Explain why the computations for one of these integrals are so much more accurate than for the other integral.

9.46.* Prove that if

$$\int_a^b p(x)w(x)\,dx = \sum_{i=0}^{n} p(x_i)w_i \quad \forall p \in \mathbb{R}[x; 2n+1],$$

then the polynomial $\prod_{i=0}^{n-1}(x - x_i)$ is w-orthogonal to $\mathbb{R}[x;n]$ on $[a,b]$.

Notes

Much of the material discussed in this chapter is inspired by [Tre13]. Our treatment of barycentric Lagrange interpolation is based on [BT04]. Lagrange interpolation is also discussed in more detail, and from a different perspective, in Volume 1, Section 15.7.1.

For a detailed discussion of Chebyshev polynomials and many of their properties, we recommend [MH03]. When f is analytic in a Bernstein ellipse around the line segment $[-1,1]$, then much stronger results than (9.37) and (9.39) can be proved about the convergence of the interpolation at the Chebyshev zeros and extremizers. This is discussed in detail in [Tre13, Chapter 7].

For more information about conditioning of the problem of finding zeros of a polynomial as a function of the coefficients, see Volume 1, Section 7.5.2, and [Tre13, Appendix: Myth 6]). The fast algorithm for computing the weights in Clenshaw–Curtis quadrature is due to Waldvogel [Wal06]. For more about the convergence of Clenshaw–Curtis quadrature versus Gaussian quadrature see [Tre08].

Part III

Interlude

10 Review of Multivariate Differentiation

Nothing ruins your day more than getting a bad review.
—Taylor Swift

In this chapter we briefly review derivatives in multiple dimensions. A solid working understanding of multivariate differentiation is essential for the remainder of the text, most notably for the theory of optimization. We cover differentiation only briefly here, and mostly without proofs. For a rigorous treatment of the topic, see Volume 1, Chapter 6.

10.1 Directional, Partial, and Total Derivatives

In single-variable calculus the derivative of a well-behaved function $f : (a, b) \to \mathbb{R}$ at a point $t \in (a, b)$ is the rate of change (or slope of the tangent) of the function f at the point $t \in (a, b)$. For the multivariate case, given an open set $U \in \mathbb{R}^n$ and a well-behaved function $f : U \to \mathbb{R}$, the derivative at a point $\mathbf{x} \in U$ allows us to find the tangent plane (or hyperplane) to the graph of f at $\mathbf{x}$, as well as the slope in any direction along the plane. In this section and the next we review the basic theory of derivatives and how to write the tangent plane of a function at a point.

10.1.1 Curves and Tangent Vectors

Before we treat derivatives with a multivariate domain, we first consider the case where the domain is one dimensional, but the codomain is higher dimensional.

Definition 10.1.1. *A curve is a map $\gamma : (a, b) \to \mathbb{R}^n$. We say that the curve γ is* differentiable at $t \in (a, b)$ *with* derivative

$$\gamma'(t) = \lim_{h \to 0} \frac{\gamma(t + h) - \gamma(t)}{h}, \tag{10.1}$$

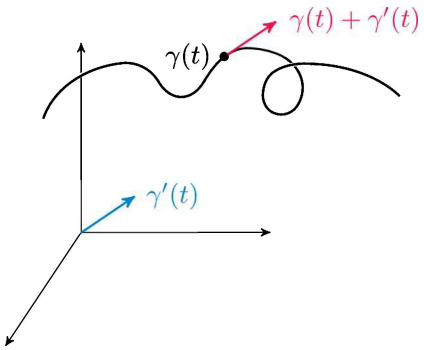


Figure 10.1. *The derivative $\gamma'(t)$ of a differentiable curve $\gamma : (a, b) \to \mathbb{R}^n$ points in the direction of the line tangent to the curve at $\gamma(t)$. Note that the tangent vector $\gamma'(t)$ (blue) represents the instantaneous velocity of the curve at $\gamma(t)$, whereas the line segment (red) from $\gamma(t)$ to $\gamma(t) + \gamma'(t)$ is what is often informally called the "tangent" to the curve.*

if the limit exists. The derivative is a vector, commonly called the tangent vector *or the* velocity*; see Figure 10.1 for an illustration. If γ is differentiable at every point of (a, b), we say that the curve γ is* differentiable on (a, b)*. In this case, γ' defines a curve as well, often called the* tangent curve *of γ.*

Throughout this chapter we will assume that all curves are differentiable.

Remark 10.1.2. If the curve γ is given by

$$\gamma(t) = (\gamma_1(t), \ldots, \gamma_n(t)),$$

then the derivative gives the componentwise representation

$$\gamma'(t) = (\gamma_1'(t), \ldots, \gamma_n'(t)).$$

Example 10.1.3. The differentiable curve $\gamma : \mathbb{R} \to \mathbb{R}^2$ given by

$$\gamma(t) = (\cos t, \sin t)$$

traces out a circle of radius one, centered at the origin, going counterclockwise. The tangent vector at $t = \pi/2$ is $\gamma'(\pi/2) = (-1, 0)$. The tangent curve is $\gamma'(t) = (-\sin t, \cos t)$. Note that $\gamma(t)$ and $\gamma'(t)$ are orthogonal for each $t \in \mathbb{R}$. This is treated further in Example 10.1.5.

By looking at the individual coordinates of the differentiable curves, we can prove the following results.

Proposition 10.1.4. *Let $f, g : \mathbb{R} \to \mathbb{R}^n$ be differentiable curves, and let $\varphi : \mathbb{R} \to \mathbb{R}$ be differentiable. The following hold:*

(i) $(f(t) + g(t))' = f'(t) + g'(t)$.

(ii) $(\varphi(t)f(t))' = \varphi'(t)f(t) + \varphi(t)f'(t)$.

(iii) $\langle f(t), g(t)\rangle' = \langle f'(t), g(t)\rangle + \langle f(t), g'(t)\rangle$.

(iv) $(f \circ \varphi)'(t) = f'(\varphi(t))\varphi'(t)$.

Example 10.1.5. Consider again the differentiable curve of Example 10.1.3. It is straightforward to verify that $\langle \gamma(t), \gamma(t)\rangle = \|\gamma(t)\|^2 = 1$ for each $t \in \mathbb{R}$. By Proposition 10.1.4(iii) the derivative is $\langle \gamma'(t), \gamma(t)\rangle + \langle \gamma(t), \gamma'(t)\rangle = 0$, which implies $2\langle \gamma'(t), \gamma(t)\rangle = 0$. This shows that the tangent vector $\gamma'(t)$ is orthogonal to the curve $\gamma(t)$.

10.1.2 Directional Derivatives

Let $U \subset \mathbb{R}^n$ be an open set containing $\mathbf{x}$ and $f : U \to \mathbb{R}$ a well-behaved[41] function. Given a vector $\mathbf{v} \in \mathbb{R}^n$, the rate of change of the function f at the point $\mathbf{x}$, moving in the direction $\mathbf{v} \in \mathbb{R}^n$, is the *directional derivative* of f at $\mathbf{x}$ with respect to $\mathbf{v}$.

Definition 10.1.6. *Let $U \in \mathbb{R}^n$ be an open set containing $\mathbf{x}$ and $f : U \to \mathbb{R}$. Given $\mathbf{v} \in \mathbb{R}^n$, the* directional derivative *of f at $\mathbf{x}$ with respect to $\mathbf{v}$ is the limit (if it exists)*

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t}. \tag{10.2}$$

Remark 10.1.7. The curve $\gamma(t) = \mathbf{x} + t\mathbf{v}$ is the line in $\mathbb{R}^n$ passing through $\mathbf{x}$ and pointing in direction $\mathbf{v}$. The directional derivative is simply the derivative of the composition $f \circ \gamma$ at the point $t = 0$, that is,

$$\frac{d}{dt}(f \circ \gamma)\Big|_{t=0} = \lim_{t \to 0} \frac{f(\gamma(t)) - f(\gamma(0))}{t} = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t} = D_{\mathbf{v}} f(\mathbf{x}).$$

Remark 10.1.8. If $\mathbf{u}$ is a unit vector, the directional derivative of f at $\mathbf{x}$ in the direction of $\mathbf{u}$ is the slope of the tangent line along the direction $\mathbf{u}$. If $\mathbf{u}$ is not a unit vector, then the directional derivative is the product of the slope and the magnitude $\|\mathbf{u}\|$ of the vector $\mathbf{u}$.

[41] By *well-behaved* we mean that the limit in (10.2) exists.

Example 10.1.9. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = xy^2 + x^3y$. The directional derivative at $\mathbf{x} = (x, y)$ in the direction $\mathbf{v} = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ is found by computing the derivative of $f(\mathbf{x} + t\mathbf{v})$ with respect to t at $t = 0$:

$$\begin{aligned} D_{\mathbf{v}} f(x, y) &= \frac{d}{dt} f\left(x + \frac{t}{\sqrt{2}}, y + \frac{t}{\sqrt{2}}\right)\Big|_{t=0} \\ &= \frac{d}{dt}\left(\left(x + \frac{t}{\sqrt{2}}\right)\left(y + \frac{t}{\sqrt{2}}\right)^2 + \left(x + \frac{t}{\sqrt{2}}\right)^3\left(y + \frac{t}{\sqrt{2}}\right)\right)\Big|_{t=0} \\ &= \frac{1}{\sqrt{2}}(y^2 + 2xy + 3x^2y + x^3). \end{aligned}$$

10.1.3 Partial Derivatives

Taking directional derivatives along the standard basis vectors $\mathbf{e}_i$ for each i gives *partial derivatives*. In other words, the partial derivatives are simply the directional derivatives $D_{\mathbf{e}_i} f(\mathbf{x})$, which are often written as $D_i f(\mathbf{x})$ or $\frac{\partial f}{\partial x_i}$.

Definition 10.1.10. *Let $U \in \mathbb{R}^n$ be an open set and $f : U \to \mathbb{R}^m$. The ith* partial derivative *of f at the point $\mathbf{x} \in U$ is given by the limit (if it exists)*

$$D_i f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{e}_i) - f(\mathbf{x})}{t}.$$

Remark 10.1.11. In the previous definition the ith coordinate is the only one that varies in the limit. Thus we can think of the ith partial derivative as the derivative of a single-variable function with the only variable being the ith coordinate; the other coordinates are treated as constants.

Example 10.1.12. The partial derivative $D_1 f(x, y)$ of the function $f(x, y) = xy^2 + x^3y$ can be computed by treating f as a function of x only, holding y as a constant. Thus, this partial derivative is

$$D_1 f(x, y) = y^2 + 3x^2y.$$

Similarly, the partial derivative $D_2 f(x, y)$ can be computed by treating f as a function of y only, holding x as a constant. Thus, this partial derivative is

$$D_2 f(x, y) = 2xy + x^3.$$

10.1.4 The Derivative

We now review the idea of the total derivative of a function $f : U \to \mathbb{R}^m$, where $U \in \mathbb{R}^n$ is an open set.

Definition 10.1.13. *Let $U \subset \mathbb{R}^n$ be an open set. A function $f : U \to \mathbb{R}^m$ is* differentiable *at $\mathbf{x} \in U$ if there exists a matrix $Df(\mathbf{x}) \in M_{m\times n}(\mathbb{R})$ such that*

$$\lim_{\mathbf{h}\to\mathbf{0}} \frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - Df(\mathbf{x})\mathbf{h}\|}{\|\mathbf{h}\|} = 0. \tag{10.3}$$

The matrix $Df(\mathbf{x})$ is called the derivative *of f at $\mathbf{x}$. If f is differentiable for each $\mathbf{x} \in U$, we say that f is* differentiable *on U. If f is differentiable on U and the function $g : U \to M_{m\times n}(\mathbb{R})$ given by $g(\mathbf{x}) = Df(\mathbf{x})$ is continuous on U, then f is* continuously differentiable *on U. In this case we write $f \in C^1(U;\mathbb{R})$ or say f is C^1.*

Remark 10.1.14. The derivative is sometimes called the *total derivative* as a way to distinguish it from the directional and partial derivatives.

Nota Bene 10.1.15. An element of $\mathbb{R}^n$ is an n-tuple $\mathbf{a} = (a_1, \dots, a_n)$, but when written in terms of the standard basis it corresponds to a column vector $\begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}^\mathsf{T}$. Unless otherwise indicated, we always use the standard basis $\mathbf{e}_1, \dots, \mathbf{e}_n$, where the ith entry of $\mathbf{e}_i$ is 1 and all other entries are 0, so $\mathbf{e}_i = (0, \dots, 1, \dots, 0)$ and $\begin{bmatrix} a_1, \dots, a_n \end{bmatrix}^\mathsf{T}$ denotes the element $\mathbf{a} = \sum_{i=1}^n \mathbf{e}_i a_i$. We always use parentheses ( ) to indicate an n-tuple in $\mathbb{R}^n$, and we use square brackets [ ] to indicate the representation of that tuple in a given basis (always the standard basis, unless otherwise indicated).

The derivative $Df(\mathbf{x})$ of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ at $\mathbf{x}$ is actually a linear operator mapping $\mathbb{R}^n$ to $\mathbb{R}^m$, which means that in standard coordinates it is given by an $m \times n$ matrix. In particular, for a function $f : \mathbb{R}^n \to \mathbb{R}$, the derivative maps $\mathbb{R}^n$ to $\mathbb{R}$ and hence is expressed in standard coordinates as a $1 \times n$ matrix, that is, as a row vector. If we need to use $Df(\mathbf{x})$ as a vector in $\mathbb{R}^n$ instead of as a linear operator, we take its transpose $Df(\mathbf{x})^\mathsf{T}$ (often called the *gradient*), which is represented in the standard basis as a column vector. For more details on the derivative as a linear operator, see Volume 1, Chapter 6.

Theorem 10.1.16. *Let $U \subset \mathbb{R}^n$ be an open set and $f : U \to \mathbb{R}^m$ be given by $f = (f_1, f_2, \dots, f_m)$. If f is differentiable on U, then the partial derivatives $D_i f_j(\mathbf{x})$ exist for each $i, j \in \{1, \dots, m\}$ and $\mathbf{x} \in U$, and the matrix $Df(\mathbf{x})$ satisfies*

$$Df(\mathbf{x}) = \begin{bmatrix} D_1 f_1(\mathbf{x}) & D_2 f_1(\mathbf{x}) & \cdots & D_n f_1(\mathbf{x}) \\ D_1 f_2(\mathbf{x}) & D_2 f_2(\mathbf{x}) & \cdots & D_n f_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ D_1 f_m(\mathbf{x}) & D_2 f_m(\mathbf{x}) & \cdots & D_n f_m(\mathbf{x}) \end{bmatrix}. \tag{10.4}$$

Example 10.1.17. Let $f : \mathbb{R}^3 \to \mathbb{R}^2$ be given by

$$f(x, y, z) = (xy + x^2z^2, y^3z^5 + x).$$

The derivative $Df(\mathbf{x})$ of f, written in standard coordinates, is the matrix

$$Df(x, y, z) = \begin{bmatrix} y + 2xz^2 & x & 2x^2z \\ 1 & 3y^2z^5 & 5y^3z^4 \end{bmatrix}.$$

Example 10.1.18. Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be the *softmax function*, which is given by

$$f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})),$$

where for $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, each $f_i(\mathbf{x})$ satisfies

$$f_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{k=1}^n e^{x_k}}. \tag{10.5}$$

The partial derivatives of (10.5) are

$$\begin{aligned} D_j f_i(\mathbf{x}) = \frac{\partial f_i(\mathbf{x})}{\partial x_j} &= \frac{e^{x_i}\delta_{ij}\left(\sum_{k=1}^n e^{x_k}\right) - e^{x_i}e^{x_j}}{\left(\sum_{k=1}^n e^{x_k}\right)^2} \\ &= \frac{e^{x_i}}{\left(\sum_{k=1}^n e^{x_k}\right)} \frac{\delta_{ij}\left(\sum_{k=1}^n e^{x_k}\right) - e^{x_j}}{\left(\sum_{k=1}^n e^{x_k}\right)} \\ &= f_i(\mathbf{x})(\delta_{ij} - f_j(\mathbf{x})). \end{aligned}$$

This shows that $Df(\mathbf{x})$ can be written as the following symmetric matrix:

$$Df(\mathbf{x}) = \operatorname{diag}(f(\mathbf{x})) - f(\mathbf{x})f(\mathbf{x})^{\mathsf{T}}, \tag{10.6}$$

where $\operatorname{diag}(f(\mathbf{x}))$ denotes the diagonal matrix whose (i, i) entry is $f_i(\mathbf{x})$.

10.2 Properties of Derivatives

In this section we review three important properties of the derivative, namely linearity, the product rule, and the chain rule.

10.2.1 Linearity

Derivatives are linear maps on the space of differentiable functions.

Theorem 10.2.1 (Linearity). *Let $U \subset \mathbb{R}^n$ be an open set. If $f : U \to \mathbb{R}^m$ and $g : U \to \mathbb{R}^m$ are differentiable on U and $a, b \in \mathbb{R}$, then $af + bg$ is also differentiable*

on U and satisfies the rule

$$D(af(\mathbf{x}) + bg(\mathbf{x})) = aDf(\mathbf{x}) + bDg(\mathbf{x}) \tag{10.7}$$

for each $\mathbf{x} \in U$.

Remark 10.2.2. An immediate consequence of the theorem is that if $f : \mathbb{R}^n \to \mathbb{R}$ is given by $f(\mathbf{x}) = \sum_{k=1}^{m} a_k f_k(\mathbf{x})$, then $Df(\mathbf{x}) = \sum_{k=1}^{m} a_k Df_k(\mathbf{x})$.

Nota Bene 10.2.3. Beware that the derivative at a point $\mathbf{x}$ is not a linear function in $\mathbf{x}$. This can be seen in Examples 10.1.17 and 10.1.18 by comparing $Df(a\mathbf{x})$ to $aDf(\mathbf{x})$.

10.2.2 Product Rule

The product rule holds for the total derivative of real-valued functions.

Theorem 10.2.4 (Product Rule). *Let $U \subset \mathbb{R}^n$ be an open set. If $f : U \to \mathbb{R}$ and $g : U \to \mathbb{R}$ are differentiable on U, then the product map $h = fg$ is also differentiable on U and satisfies the* product rule

$$Dh(\mathbf{x}) = g(\mathbf{x})Df(\mathbf{x}) + f(\mathbf{x})Dg(\mathbf{x}) \tag{10.8}$$

for each $\mathbf{x} \in U$.

Example 10.2.5. Let $f : \mathbb{R}^3 \to \mathbb{R}$ and $g : \mathbb{R}^3 \to \mathbb{R}$ be defined by

$$f(x, y, z) = x^5 y + xy^2 + z^7 \qquad \text{and} \qquad g(x, y, z) = x^3 + z^{11}.$$

By the product rule we have

$$\begin{aligned} D(fg)(x, y, z) &= g(x, y, z)Df(x, y, z) + f(x, y, z)Dg(x, y, z) \\ &= (x^3 + z^{11}) \begin{bmatrix} 5x^4 y + y^2 & x^5 + 2xy & 7z^6 \end{bmatrix} \\ &\quad + (x^5 y + xy^2 + z^7) \begin{bmatrix} 3x^2 & 0 & 11z^{10} \end{bmatrix}. \end{aligned}$$

For example, the derivative at the point $(0, 1, -1) \in \mathbb{R}^3$ is given by

$$D(fg)(0, 1, -1) = -\begin{bmatrix} 1 & 0 & 7 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 11 \end{bmatrix} = \begin{bmatrix} -1 & 0 & -18 \end{bmatrix}.$$

Proposition 10.2.6. *For any $\mathbf{a} \in \mathbb{R}^n$, if $g(\mathbf{x}) = \mathbf{a}^\mathsf{T}\mathbf{x}$ (and note that $\mathbf{a}^\mathsf{T}\mathbf{x} = \mathbf{x}^\mathsf{T}\mathbf{a}$ always holds), then*

$$Dg(\mathbf{x}) = \mathbf{a}^\mathsf{T}.$$

Proof. This follows because (10.3) holds for all $\mathbf{x}$ with $A = \mathbf{a}^\mathsf{T}$ because

$$\|g(\mathbf{x} + \mathbf{h}) - g(\mathbf{x}) - A\mathbf{h}\| = \|\mathbf{a}^\mathsf{T}(\mathbf{x} + \mathbf{h}) - \mathbf{a}^\mathsf{T}\mathbf{x} - \mathbf{a}^\mathsf{T}\mathbf{h}\| = 0. \quad \square$$

Corollary 10.2.7. *Let $A = [a_{ij}]$ be an $m \times n$ matrix. If $g : \mathbb{R}^n \to \mathbb{R}^m$ is given by $g(\mathbf{x}) = A\mathbf{x}$, then $Dg(\mathbf{x}) = A$.*

Proof. Write g in coordinates $g(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) & g_2(\mathbf{x}) & \cdots & g_m(\mathbf{x}) \end{bmatrix}^\mathsf{T}$, with each $g_i(\mathbf{x}) = \mathbf{a}_i^\mathsf{T}\mathbf{x}$, where each $\mathbf{a}_i = (a_{i1}, \ldots, a_{in})$ is the transpose of the ith row of A. This gives

$$Dg(\mathbf{x}) = \begin{bmatrix} Dg_1(\mathbf{x}) \\ Dg_2(\mathbf{x}) \\ \vdots \\ Dg_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\mathsf{T} \\ \vdots \\ \mathbf{a}_m^\mathsf{T} \end{bmatrix} = A. \quad \square$$

Example 10.2.8. If $g : \mathbb{R}^n \to \mathbb{R}^n$ is given by $g(\mathbf{x}) = \mathbf{x}$, then $Dg(x) = I$.

Proposition 10.2.9 (Inner Product Rule). *Let $U \subset \mathbb{R}^n$ be an open set. If $\mathbf{u}, \mathbf{v}$ are differentiable functions from U into $\mathbb{R}^m$ and $f(\mathbf{x}) = \langle \mathbf{u}(\mathbf{x}), \mathbf{v}(\mathbf{x}) \rangle = \mathbf{u}(\mathbf{x})^\mathsf{T}\mathbf{v}(\mathbf{x})$, then*

$$Df(\mathbf{x}) = \mathbf{u}(\mathbf{x})^\mathsf{T} D\mathbf{v}(\mathbf{x}) + \mathbf{v}(\mathbf{x})^\mathsf{T} D\mathbf{u}(\mathbf{x}).$$

Proof. Write $\mathbf{u}(\mathbf{x}) = (u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_m(\mathbf{x}))$ and $\mathbf{v}(\mathbf{x}) = (v_1(\mathbf{x}), v_2(\mathbf{x}), \ldots, v_m(\mathbf{x}))$, so that $f(\mathbf{x}) = \sum_{i=1}^m u_i(\mathbf{x}) v_i(\mathbf{x})$. The product rule (Theorem 10.2.4) gives

$$Df(\mathbf{x}) = \sum_{i=1}^m \left(u_i(\mathbf{x}) Dv_i(\mathbf{x}) + v_i(\mathbf{x}) Du_i(\mathbf{x}) \right) = \mathbf{u}(\mathbf{x})^\mathsf{T} D\mathbf{v}(\mathbf{x}) + \mathbf{v}(\mathbf{x})^\mathsf{T} D\mathbf{u}(\mathbf{x}). \quad \square$$

Example 10.2.10. If a vector-valued function $r : U \to \mathbb{R}^m$ is differentiable on the open set $U \subset \mathbb{R}^n$, then the function $f(\mathbf{x}) = \|\mathbf{r}(\mathbf{x})\|^2 = \mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{r}(\mathbf{x})$ is also differentiable on U and satisfies $Df(\mathbf{x}) = 2\mathbf{r}(\mathbf{x})^\mathsf{T} D\mathbf{r}(\mathbf{x})$.

Corollary 10.2.11. *If $g : \mathbb{R}^n \to \mathbb{R}$ is given by $g(\mathbf{x}) = \mathbf{x}^\mathsf{T} A\mathbf{x}$ for some $A \in M_n(\mathbb{R})$, then*

$$Dg(\mathbf{x}) = \mathbf{x}^\mathsf{T}(A + A^\mathsf{T}).$$

Proof. Let $\mathbf{u}(\mathbf{x}) = \mathbf{x}$ and $\mathbf{v}(\mathbf{x}) = A\mathbf{x}$, so $g(\mathbf{x}) = \mathbf{u}(\mathbf{x})^\mathsf{T}\mathbf{v}(\mathbf{x})$. The inner product rule, combined with Example 10.2.8 and Corollary 10.2.7, gives

$$Dg(\mathbf{x}) = \mathbf{u}(\mathbf{x})^\mathsf{T} D\mathbf{v}(\mathbf{x}) + \mathbf{v}(\mathbf{x})^\mathsf{T} D\mathbf{u}(\mathbf{x}) = \mathbf{x}^\mathsf{T} A + (A\mathbf{x})^\mathsf{T} I = \mathbf{x}^\mathsf{T}(A + A^\mathsf{T}). \quad \square$$

Example 10.2.12. If $A \in M_n(\mathbb{R})$ is symmetric and $f : \mathbb{R}^n \to \mathbb{R}$ is given by $f(\mathbf{x}) = \mathbf{x}^\mathsf{T} A\mathbf{x}$, then $Df(\mathbf{x}) = 2\mathbf{x}^\mathsf{T} A$.

The following is a slight generalization of the product rule.

Proposition 10.2.13. *Let* $\mathbf{w} : \mathbb{R}^n \to \mathbb{R}^m$ *and* $B : \mathbb{R}^n \to M_{k\times m}(\mathbb{R})$ *be differentiable functions given by* $\mathbf{w}(\mathbf{x}) = (w_1(\mathbf{x}), \dots, w_m(\mathbf{x}))$ *and*

$$B(\mathbf{x}) = \begin{bmatrix} b_{1,1}(\mathbf{x}) & b_{1,2}(\mathbf{x}) & \dots & b_{1,m}(\mathbf{x}) \\ b_{2,1}(\mathbf{x}) & b_{2,2}(\mathbf{x}) & \dots & b_{2,m}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1}(\mathbf{x}) & b_{k,2}(\mathbf{x}) & \dots & b_{k,m}(\mathbf{x}) \end{bmatrix}.$$

If $H : \mathbb{R}^n \to \mathbb{R}^k$ *is given by* $H(\mathbf{x}) = B(\mathbf{x})\mathbf{w}(\mathbf{x})$, *then*

$$DH(\mathbf{x}) = B(\mathbf{x})D\mathbf{w}(\mathbf{x}) + \begin{bmatrix} \mathbf{w}(\mathbf{x})^\mathsf{T} D\mathbf{b}_1(\mathbf{x})^\mathsf{T} \\ \vdots \\ \mathbf{w}(\mathbf{x})^\mathsf{T} D\mathbf{b}_k(\mathbf{x})^\mathsf{T} \end{bmatrix}, \tag{10.9}$$

where $\mathbf{b}_i$ *is the* i*th row of* B.

Proof. Let $H_i(\mathbf{x}) = \mathbf{b}_i(\mathbf{x})\mathbf{w}(\mathbf{x})$ be the ith coordinate of $H(\mathbf{x})$. The inner product rule (Proposition 10.2.9) gives $DH_i(\mathbf{x}) = \mathbf{b}_i(\mathbf{x})D\mathbf{w}(\mathbf{x}) + \mathbf{w}(\mathbf{x})^\mathsf{T} D\mathbf{b}_i(\mathbf{x})^\mathsf{T}$ for each i. Stacking these vertically gives (10.9). □

10.2.3 Chain Rule

The chain rule also holds for total derivatives.

Theorem 10.2.14 (Chain Rule). *Assume that* $U \subset \mathbb{R}^\ell$ *and* $V \subset \mathbb{R}^m$ *are open sets and that* $g : U \to V$ *and* $f : V \to \mathbb{R}^n$ *with* $g(U) \subset V$. *If* g *is differentiable on* U *and* f *is differentiable on* V, *then the composite map* $h = f \circ g$ *is also differentiable on* U *and satisfies the* chain rule

$$Dh(\mathbf{x}) = Df(g(\mathbf{x}))Dg(\mathbf{x}) \tag{10.10}$$

for each $\mathbf{x} \in U$.

Example 10.2.15. The function $h(x, y) = (\sin(xy), xy(x-y)^2)$ can be written as $h = f \circ g$, where $f(p, q) = (\sin p, pq^2)$ and $g(x, y) = (xy, x - y)$. We can write $p(x, y) = xy$ and $q(x, y) = x - y$. The chain rule gives

$$\begin{aligned} Dh(x,y) = Df(p,q)Dg(x,y) &= \begin{bmatrix} \cos p & 0 \\ q^2 & 2pq \end{bmatrix} \begin{bmatrix} y & x \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(xy) & 0 \\ (x-y)^2 & 2xy(x-y) \end{bmatrix} \begin{bmatrix} y & x \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} y\cos(xy) & x\cos(xy) \\ y(x-y)^2 + 2xy(x-y) & x(x-y)^2 - 2xy(x-y) \end{bmatrix}. \end{aligned}$$

The total derivative may be used to compute directional derivatives.

Theorem 10.2.16. *Let $U \subset \mathbb{R}^n$ be an open set. If $f : U \to \mathbb{R}^m$ is differentiable at $\mathbf{x} \in U$, then the directional derivative $D_{\mathbf{v}} f(\mathbf{x})$ along $\mathbf{v} \in \mathbb{R}^n$ at $\mathbf{x}$ exists and is the product of the derivative $Df(\mathbf{x})$ and the tangent vector $\mathbf{v}$:*

$$D_{\mathbf{v}} f(\mathbf{x}) = Df(\mathbf{x})\mathbf{v}. \tag{10.11}$$

Proof. Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be arbitrary and $\gamma : \mathbb{R} \to \mathbb{R}^n$ be defined by the differentiable curve $\gamma(t) = \mathbf{x} + t\mathbf{v}$. The chain rule gives

$$D_{\mathbf{v}} f(\mathbf{x}) = \frac{d}{dt} f(\gamma(t))\Big|_{t=0} = Df(\gamma(0))\gamma'(0) = Df(\mathbf{x})\mathbf{v}. \quad \square$$

Example 10.2.17. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = xy^2 + x^3 y$, as in Example 10.1.9. We have

$$Df(x, y) = \begin{bmatrix} D_1 f(x, y) & D_2 f(x, y) \end{bmatrix} = \begin{bmatrix} y^2 + 3x^2 y & 2xy + x^3 \end{bmatrix}.$$

Thus, the directional derivative of f in the direction $\mathbf{v} = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ is

$$D_{\mathbf{v}} f(x, y) = \begin{bmatrix} y^2 + 3x^2 y & 2xy + x^3 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}}(y^2 + 3x^2 y + 2xy + x^3),$$

which agrees with the (more laborious) calculation in Example 10.1.9.

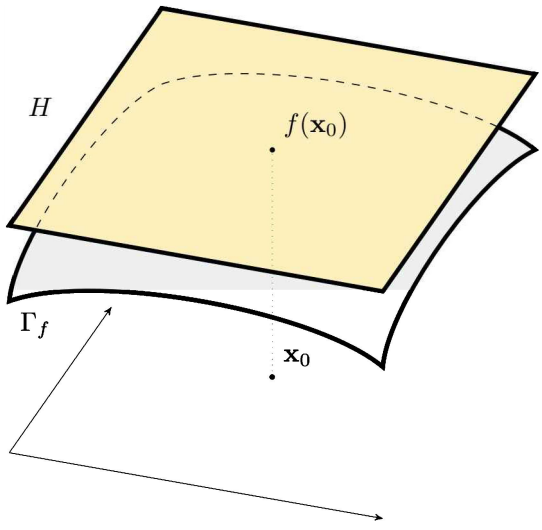


Figure 10.2. *Depiction (yellow) of the tangent plane H to the graph (gray) $\Gamma_f = \{(\mathbf{x}, z) \in \mathbb{R}^2 \times \mathbb{R} \mid z = f(\mathbf{x})\}$ of a function f at the point $(\mathbf{x}_0, f(\mathbf{x}_0))$ as given in (10.14).*

10.2.4 Tangent Planes

We conclude this section by describing the tangent plane to the graph of a function at a point. For a function $f : U \to \mathbb{R}$, the *graph of* f is the set

$$\Gamma_f = \{(\mathbf{x}, z) \in U \times \mathbb{R} \mid z = f(\mathbf{x})\} \subset \mathbb{R}^n \times \mathbb{R} = \mathbb{R}^{n+1}; \tag{10.12}$$

see Appendix A.2 of Volume 1 for more on graphs of functions. If U is open and f is differentiable on U, then the *tangent plane* to the graph of f at $\mathbf{x}_0 \in U$ is the graph of the function

$$L(\mathbf{x}) = f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0). \tag{10.13}$$

Thus, the tangent plane is the set

$$H = \{(\mathbf{x}, z) \in \mathbb{R}^{n+1} \mid z = L(\mathbf{x})\}, \tag{10.14}$$

which is a *hyperplane* in $\mathbb{R}^{n+1}$; that is, a set of the form $\{\mathbf{v} \in \mathbb{R}^{n+1} \mid \mathbf{a}^\mathsf{T}\mathbf{v} = b\}$ for some $\mathbf{a} \in \mathbb{R}^{n+1}$ and $b \in \mathbb{R}$. We may write the tangent plane as a hyperplane by setting $\mathbf{a} = (Df(\mathbf{x}_0), -1)$ and $b = Df(\mathbf{x}_0)\mathbf{x}_0 - f(\mathbf{x}_0)$. See Figure 10.2 for an illustration.

Remark 10.2.18. This is a generalization of the formula for the line tangent to the graph in single-variable calculus given by the equation

$$y = f(x_0) + f'(x_0)(x - x_0).$$

Example 10.2.19. Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ in Example 10.1.9 given by $f(x, y) = xy^2 + x^3y$. The tangent plane at the point (x_0, y_0) is the graph of the function

$$\begin{aligned} L(x, y) &= f(x_0, y_0) + Df(x_0, y_0) \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} \\ &= f(x_0, y_0) + \begin{bmatrix} y_0^2 + 3x_0^2 y_0 & 2x_0 y_0 + x_0^3 \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}. \end{aligned}$$

Nota Bene 10.2.20. The tangent plane $H = \{(\mathbf{x}, L(\mathbf{x})) \mid \mathbf{x} \in \mathbb{R}^n\}$ in $\mathbb{R}^{n+1}$ is not a vector subspace because it does not (usually) pass through the origin in $\mathbb{R}^{n+1}$. Instead it's an *affine* set, which is a translate of a vector subspace; for more on affine sets, see Section 13.1.8. Indeed, $T = \{(\mathbf{v}, Df(\mathbf{x}_0)\mathbf{v}) \mid \mathbf{v} \in \mathbb{R}^n\}$ is a vector subspace of $\mathbb{R}^{n+1}$, called the *tangent space* of the graph of f at $(\mathbf{x}_0, f(\mathbf{x}_0))$, and the tangent plane H is the translate of T given by $H = T + (\mathbf{x}_0, f(\mathbf{x}_0))$.

This is similar to the situation for derivatives of curves, where the tangent vector $\gamma'(t_0)$ at t_0 does not necessarily touch the curve $\gamma(t)$ at all, but the tangent line $\gamma(t_0) + t\gamma'(t_0)$ is actually tangent to the curve at the point $\gamma(t_0)$; see Figure 10.1.

10.3 Implicit Function Theorem and Taylor's Theorem

In this section we review the implicit function theorem, higher-order derivatives, and Taylor's theorem. These are essential for understanding smooth, nonlinear functions and will be used throughout the rest of the book.

10.3.1 Implicit Function Theorem

Let $I, J \subset \mathbb{R}$ be open intervals (possibly infinite). Given a function of two variables $F : I \times J \to \mathbb{R}$, each constant $c \in \mathbb{R}$ defines a *level set* $\{(x, y) \in I \times J \mid F(x, y) = c\}$. The *implicit function theorem* gives conditions that guarantee the level set is *locally* the graph of a function $f : I' \to J$, where $I' \subset I$ is also an open interval. This new function f is the *implicit function* satisfying the relation $F(x, f(x)) = c$ for all $x \in I'$. The implicit function theorem also generalizes to higher dimensions.

Example 10.3.1. Let $F(x, y) = x^2 + y^2$, and consider the circle of radius 3 defined by the level set $\{(x, y) \subset \mathbb{R} \times \mathbb{R} \mid F(x, y) = 9\}$. In a neighborhood of the point $(x_0, y_0) = (0, 3)$ we can define y as a function of x on the interval $(-3, 3)$; that is, $y(x) = \sqrt{9 - x^2}$. We say that F *implicitly defines* y as a function of x in a neighborhood of $(0, 3)$.

However, we cannot define y as a function of x in a neighborhood around the point $(3, 0)$ since in any neighborhood of $(3, 0)$ there are two points of the level set of the form $(x, \pm\sqrt{9 - x^2})$ with the same x coordinate. This is depicted in Figure 10.3. The implicit function theorem gives general conditions for when one or more variables can be defined implicitly as functions of other variables.

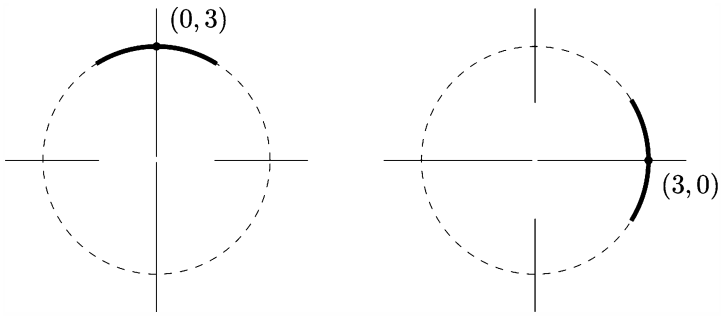


Figure 10.3. *An illustration of Example 10.3.1. In a neighborhood around the point* $(0, 3)$*, the points on the circle* $F(x, y) = 9$ *(black arc on the left) can be written as* $(x, f(x))$*, provided* x *remains in a small enough neighborhood (blue line) of* 0*. But near the point* $(3, 0)$ *there is no function of* x *defining* y*. Instead, there is a function* g *so that we can write points of the circle near* $(3, 0)$ *as* $(g(y), y)$*, provided* y *remains in a small enough neighborhood (red line) of* 0*. Thus,* x *is implicitly defined as a function* g *of* y*.*

In the previous example, we were able to solve explicitly for y as a function of x, but in many cases solving for one variable in terms of the others is very hard or even impossible. Yet, for many problems just knowing it exists and knowing its derivative is enough. The implicit function theorem tells us not only when the function exists but also how to compute its derivative without computing the function itself.

Theorem 10.3.2 (Implicit Function Theorem). *Assume that $U \subset \mathbb{R}^m$ and $V \subset \mathbb{R}^n$ are open sets containing $\mathbf{x}_0$ and $\mathbf{y}_0$, respectively, and $F : U \times V \to \mathbb{R}^n$ is a continuously differentiable map. Let $D_1F(\mathbf{x}_0, \mathbf{y}_0)$ denote the derivative of F with $\mathbf{y} = \mathbf{y}_0$ held constant, and let $D_2F(\mathbf{x}_0, \mathbf{y}_0)$ denote the derivative of F with $\mathbf{x} = \mathbf{x}_0$ held constant. If $D_2F(\mathbf{x}_0, \mathbf{y}_0)$ is nonsingular, then there exists an open neighborhood $U_0 \times V_0 \subset U \times V$ of $(\mathbf{x}_0, \mathbf{y}_0)$ and a unique continuously differentiable function $f : U_0 \to V_0$ such that $f(\mathbf{x}_0) = \mathbf{y}_0$ and*

$$\{(\mathbf{x}, \mathbf{y}) \in U_0 \times V_0 \mid F(\mathbf{x}, \mathbf{y}) = F(\mathbf{x}_0, \mathbf{y}_0)\} = \{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in U_0\}. \tag{10.15}$$

Moreover, for each $\mathbf{x} \in U_0$, the derivative of f satisfies

$$Df(\mathbf{x}) = -D_2F(\mathbf{x}, f(\mathbf{x}))^{-1} D_1F(\mathbf{x}, f(\mathbf{x})). \tag{10.16}$$

Example 10.3.3. In Example 10.3.1, the hypothesis of the implicit function theorem is satisfied since $D_2F(x_0, y_0) = 2y_0 \neq 0$. Hence, there exists a unique continuously differentiable function $f(x)$ in a neighborhood of the point (x_0, y_0) satisfying $F(x, f(x)) = 0$. Setting $y = f(x)$ and differentiating the equation $F(x, y) = 0$ with respect to x gives

$$0 = D_1F(x, f(x)) + D_2F(x, f(x))f'(x) = 2x + 2y(x)y'(x).$$

Solving for $y' = f'(x)$ yields

$$y'(x) = f'(x) = -\frac{D_1F(x, y)}{D_2F(x, y)} = -\frac{2x}{2y} = \frac{-x}{y},$$

which agrees with (10.16).

Remark 10.3.4. The previous example is a special case of a claim often seen in a multivariable calculus class. For any function $F : \mathbb{R}^2 \to \mathbb{R}$, if the equation $F(x, y) = c$ defines y implicitly as a function of x, then the derivative $\frac{dy}{dx}$ is given by

$$\frac{dy}{dx} = -\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}}. \tag{10.17}$$

The implicit function theorem guarantees that y is a function of x whenever $\frac{\partial F}{\partial y} \neq 0$, and (10.17) is a special case of (10.16).

Example 10.3.5. Consider the two-dimensional surface S defined implicitly by the equation $F(x, y, z) = 0$, where

$$F(x, y, z) = z^3 + 3xyz^2 - 5x^2y^2z + 14.$$

If $(x_0, y_0, z_0) = (1, -1, 2) \in S$, then $D_3F(x_0, y_0, z_0) = -5 \neq 0$. By the implicit function theorem, the surface S can be written explicitly as the graph of a function $z = z(x, y)$ in a neighborhood of (x_0, y_0, z_0).

Furthermore, the partial derivatives of $z(x, y)$ can be computed by differentiating $F(x, y, z(x, y)) = 0$, which gives

$$\begin{aligned} 0 &= D_1F(x, y, z) + D_3F(x, y, z)D_1z(x, y) \\ &= (3yz^2 - 10xy^2z) + (3z^2 + 6xyz - 5x^2y^2)\frac{\partial z}{\partial x}, \\ 0 &= D_2F(x, y, z) + D_3F(x, y, z)D_2z(x, y) \\ &= (3xz^2 - 10x^2yz) + (3z^2 + 6xyz - 5x^2y^2)\frac{\partial z}{\partial y}. \end{aligned}$$

Substituting x_0, y_0, z_0 and solving for the partial derivatives of z gives

$$\begin{aligned} \frac{\partial z}{\partial x} &= D_1z(x_0, y_0) = -\frac{D_1F(x_0, y_0, z_0)}{D_3F(x_0, y_0, z_0)} = -\frac{32}{5}, \\ \frac{\partial z}{\partial y} &= D_2z(x_0, y_0) = -\frac{D_2F(x_0, y_0, z_0)}{D_3F(x_0, y_0, z_0)} = \frac{32}{5}. \end{aligned}$$

Thus, the tangent plane of the surface S at (x_0, y_0, z_0) is

$$32(x - 1) - 32(y + 1) + 5(z - 2) = 0.$$

10.3.2 Higher-Order Derivatives

Let $U \subset \mathbb{R}^n$ be an open set. If the function $f : U \to \mathbb{R}$ is continuously differentiable on U, then the function $g : U \to \mathbb{R}^n$ given by $g(\mathbf{x}) = Df(\mathbf{x})^\mathsf{T}$ is continuous on U. This derivative is transposed because $Df(\mathbf{x})$ is a row vector, and we want to treat elements of $\mathbb{R}^n$ as column vectors; see Nota Bene 10.1.15. If g is differentiable on U, then we can take its derivative $Dg(\mathbf{x})$ for each $\mathbf{x} \in U$. The $n \times n$ derivative matrix $Dg(\mathbf{x})$ can be considered the second derivative of f at $\mathbf{x} \in U$, hereafter denoted $D^2f(\mathbf{x})$ and called the *Hessian of* f at $\mathbf{x}$. If g is continuously differentiable, we say that f is *twice continuously differentiable* and write $f \in C^2(U; \mathbb{R})$ or just say f *is* C^2.

Write

$$g(\mathbf{x}) = Df(\mathbf{x})^\mathsf{T} = \begin{bmatrix} D_1f(\mathbf{x}) \\ \vdots \\ D_nf(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_n(\mathbf{x}) \end{bmatrix},$$

where each $g_i(\mathbf{x}) = D_i f(\mathbf{x})$. Differentiating gives

$$D^2 f(\mathbf{x}) = \begin{bmatrix} D_1 g_1(\mathbf{x}) & D_2 g_1(\mathbf{x}) & \cdots & D_n g_1(\mathbf{x}) \\ D_1 g_2(\mathbf{x}) & D_2 g_2(\mathbf{x}) & \cdots & D_n g_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ D_1 g_n(\mathbf{x}) & D_2 g_n(\mathbf{x}) & \cdots & D_n g_n(\mathbf{x}) \end{bmatrix}$$
$$= \begin{bmatrix} D_1 D_1 f(\mathbf{x}) & D_2 D_1 f(\mathbf{x}) & \cdots & D_n D_1 f(\mathbf{x}) \\ D_1 D_2 f(\mathbf{x}) & D_2 D_2 f(\mathbf{x}) & \cdots & D_n D_2 f(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ D_1 D_n f(\mathbf{x}) & D_2 D_n f(\mathbf{x}) & \cdots & D_n D_n f(\mathbf{x}) \end{bmatrix}.$$

Proposition 10.3.6. *Let $U \subset \mathbb{R}^n$ be an open set and assume $f : U \to \mathbb{R}$ is twice continuously differentiable on U. For any $\mathbf{x} \in U$, the matrix $D^2 f(\mathbf{x})$ is symmetric, and hence*

$$D_i D_j f(\mathbf{x}) = D_j D_i f(\mathbf{x}) \tag{10.18}$$

for every i and j.

Taking things one step further, let $U \subset \mathbb{R}^n$ be an open set and let $h : U \to \mathbb{R}^{n^2}$ be the function that maps $\mathbf{x}$ to the flattened Hessian $D^2 f(\mathbf{x})$, where the elements of the matrix are just listed in a single column vector of length n^2. If h is continuously differentiable, then we say that f is *thrice continuously differentiable* and write $f \in C^3(U; \mathbb{R})$ or just say f *is* C^3.

10.3.3 Taylor's Theorem

Taylor's theorem is one of the most powerful tools in analysis. It allows us to approximate smooth (differentiable) functions in a small neighborhood to arbitrary precision using polynomials. This allows us to approximate functions that are otherwise difficult to analyze and can give a lot of insight into the behavior of a function.

The univariate version of Taylor's theorem holds both in $\mathbb{R}$ and in $\mathbb{C}$, and we need the complex version in the next chapter. We use $\mathbb{F}$ to denote either $\mathbb{R}$ or $\mathbb{C}$.

Theorem 10.3.7. *Let $U \subset \mathbb{F}$ be an open set with $f : U \to \mathbb{F}$ in $C^k(U; \mathbb{F})$ (meaning that $\frac{d^k f}{dz^k}$ exists and is continuous on all of U). If $z \in U$ and $h \in \mathbb{F}$ are such that the line segment $\ell(z, z+h) = \{z + th \mid 0 \le t \le 1\}$ is contained in U, then the kth derivative $f^{(k)}$ must be bounded by some $M < \infty$ on the line segment. For any such M we have*

$$f(z+h) = f(z) + f'(z)h + \frac{1}{2} f''(z)h^2 + \cdots + R_k, \tag{10.19}$$

where $|R_k| \le \frac{M}{k!}|h|^k$.

There is also a multivariate version of Taylor's theorem, but talking about multivariate higher-order derivatives is a little messy, so here we only give quadratic and cubic versions of Taylor's theorem.

Theorem 10.3.8. *Let $U \subset \mathbb{R}^n$ be an open set. Assume that $\mathbf{x} \in U$ and $\mathbf{h} \in \mathbb{R}^n$ are such that the line segment $\ell(\mathbf{x}, \mathbf{x}+\mathbf{h}) = \{\mathbf{x}+t\mathbf{h} \mid 0 \le t \le 1\}$ is contained in U. If $f \in C^2(U;\mathbb{R})$ and $\|D^2 f(\mathbf{x}+t\mathbf{h})\| \le L$ on all of $\ell(\mathbf{x}, \mathbf{x}+\mathbf{h})$, then*

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + Df(\mathbf{x})\mathbf{h} + R_2, \tag{10.20}$$

where

$$R_2 = \int_0^1 (1-t)\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{x}+t\mathbf{h})\mathbf{h}\, dt \qquad \text{and} \qquad |R_2| \le \frac{L}{2}\|\mathbf{h}\|^2. \tag{10.21}$$

Similarly, if $f \in C^3(U;\mathbb{R})$, and $\|D^3 f\| \le M$ on $\ell(\mathbf{x}, \mathbf{x}+\mathbf{h})$, then

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + Df(\mathbf{x})\mathbf{h} + \frac{1}{2}\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{x})\mathbf{h} + R_3, \tag{10.22}$$

where

$$|R_3| \le \frac{M}{3!}\|\mathbf{h}\|^3. \tag{10.23}$$

Corollary 10.3.9. *Let $U \subset \mathbb{R}^n$ be an open set and $f \in C^2(U;\mathbb{R})$. If $\|D^2 f(\mathbf{x})\| \le L$ for each $\mathbf{x} \in U$ and $\|\mathbf{h}\|$ is sufficiently small, then*

$$\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - Df(\mathbf{x})\mathbf{h}\| \le \frac{1}{2}L\|\mathbf{h}\|^2. \tag{10.24}$$

Example 10.3.10. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be given by $f(x,y) = e^{x+y}$. To find the second-order Taylor polynomial of f at $(0,0)$ compute the derivative

$$Df(0,0) = \begin{bmatrix} e^{x+y} & e^{x+y} \end{bmatrix}\Big|_{0,0} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

and the Hessian

$$D^2 f(0,0) = \begin{bmatrix} e^{x+y} & e^{x+y} \\ e^{x+y} & e^{x+y} \end{bmatrix}\Big|_{0,0} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

With $\mathbf{x} = (0,0)$ and $\mathbf{h} = (x,y)$, the second-order approximation of f at $\mathbf{x}$ is

$$\begin{aligned} f(x,y) &\approx f(\mathbf{0}) + Df(\mathbf{0})\mathbf{h} + \frac{1}{2}\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{0})\mathbf{h}, \\ &= 1 + x + y + \frac{1}{2}(x^2 + 2xy + y^2). \end{aligned}$$

Example 10.3.11. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be given by $f(x, y) = \cos(x)e^{3y}$. To find the second-order Taylor polynomial of f at $(0, 0)$, compute the derivative

$$Df(0,0) = \begin{bmatrix} -\sin(x)e^{3y} & 3\cos(x)e^{3y} \end{bmatrix}\Big|_{0,0} = \begin{bmatrix} 0 & 3 \end{bmatrix}$$

and the Hessian

$$D^2 f(0,0) = \begin{bmatrix} -\cos(x)e^{3y} & -3\sin(x)e^{3y} \\ -3\sin(x)e^{3y} & 9\cos(x)e^{3y} \end{bmatrix}\Big|_{0,0} = \begin{bmatrix} -1 & 0 \\ 0 & 9 \end{bmatrix}.$$

If $\mathbf{x} = (0, 0)$ and $\mathbf{h} = (x, y)$, then the second-order approximation of f at $\mathbf{x}$ is

$$\begin{aligned} f(x,y) &\approx f(\mathbf{0}) + Df(\mathbf{0})\mathbf{h} + \frac{1}{2}\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{0})\mathbf{h}, \\ &= 1 + 3y - \frac{1}{2}x^2 + \frac{9}{2}y^2. \end{aligned}$$

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

10.1. Generalize the result in Example 10.1.5: Prove that if the differentiable curve $\gamma : (a, b) \to \mathbb{R}^n$ has constant norm for all $t \in (a, b)$, that is, if $\|\gamma(t)\| = C$ for some constant C, then the tangent vector $\gamma'(t)$ is orthogonal to $\gamma(t)$ for each $t \in (a, b)$.

10.2. Show that the function f in (10.5) of Example 10.1.18 has derivative $Df(\mathbf{x}) = \operatorname{diag}(f(\mathbf{x})) - f(\mathbf{x})f(\mathbf{x})^{\mathsf{T}}$.

10.3. Let $A \in M_n(\mathbb{R})$ and consider the differentiable curve $f : \mathbb{R} \to M_n(\mathbb{R})$ given by

$$f(t) = e^{At} := \sum_{k=0}^{\infty} \frac{(At)^k}{k!} = \sum_{k=0}^{\infty} \frac{A^k t^k}{k!}.$$

This is well defined, since the sum converges absolutely, that is, for any matrix norm[42] $\|\cdot\|$ we have

$$\|e^{At}\| \le \sum_{k=0}^{\infty} \frac{\|A\|^k |t|^k}{k!} = e^{\|A\||t|}.$$

Use the definition of the derivative to prove that

$$Df(t) = Ae^{At}$$

for every $t \in \mathbb{R}$. You may assume without proof that

(i) $f(t+s) = f(t)f(s)$ and

(ii) $e^{At} = I + At + O(t^2)$ for small $|t|$.

10.4. Let $A = [a_{ij}]$ be an $n \times n$ matrix. As $|t| \to 0$, we have

$$\begin{aligned} \det(I + tA) &= \begin{vmatrix} 1+ta_{11} & ta_{12} & \cdots & ta_{1n} \\ ta_{21} & 1+ta_{22} & \cdots & ta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ta_{n1} & ta_{n2} & \cdots & 1+ta_{nn} \end{vmatrix} \\ &= (1+ta_{11})(1+ta_{22})\cdots(1+ta_{nn}) + O(t^2) \\ &= 1 + t\,\mathrm{tr}(A) + O(t^2). \end{aligned}$$

(i) Use this to prove that $\frac{d}{dt}\det(I+tA)\big|_{t=0} = \mathrm{tr}(A)$.

(ii) Let $f : M_n(\mathbb{R}) \to \mathbb{R}$ by given by $f(X) = \det(X)$. What is the directional derivative of f at the identity in the direction A?

10.5. The coordinate transformation $f : \mathbb{R}^3 \to \mathbb{R}^3$ from spherical coordinates to rectangular coordinates is given by

$$f(\rho, \theta, \phi) = (\rho\sin(\phi)\cos(\theta), \rho\sin(\phi)\sin(\theta), \rho\cos(\phi)).$$

Compute $Df(\mathbf{x})$ and show that $\det Df(\mathbf{x}) = -\rho^2 \sin\phi$.

10.6. Given $A \in M_n(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^n$, let $f(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|^2$. Prove that

$$Df(\mathbf{x}) = 2(\mathbf{x}^{\mathsf{T}}A^{\mathsf{T}} - \mathbf{b}^{\mathsf{T}})A.$$

Hint: Write f as $(A\mathbf{x} - \mathbf{b})^{\mathsf{T}}(A\mathbf{x} - \mathbf{b})$, then expand and differentiate.

[42] For more about matrix norms, see Volume 1, Section 3.5. For more about the absolute convergence of sums, including this one, see Volume 1, Section 5.6.3.

10.7. Let $U \subset \mathbb{R}^n$ be an open set. Prove that if $f : U \to \mathbb{R}$ and $g : U \to \mathbb{R}$ are differentiable on U, and g is nonzero on U, then the map $h = f/g$ is also differentiable on U and satisfies the *quotient rule*

$$Dh(\mathbf{x}) = \frac{g(\mathbf{x})Df(\mathbf{x}) - f(\mathbf{x})Dg(\mathbf{x})}{g(\mathbf{x})^2} \tag{10.25}$$

for each $\mathbf{x} \in U$.

10.8. Let $U \subset \mathbb{R}^n$ be an open set containing $\mathbf{x}$. Given a differentiable function $f : U \to \mathbb{R}$, find the unit vector $\mathbf{v}$ that maximizes the directional derivative $D_{\mathbf{v}} f(\mathbf{x}) = Df(\mathbf{x})\mathbf{v}$.

10.9. Given a vector $(y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$, let $g : \mathbb{R}^n \to \mathbb{R}$ be the function $g(\mathbf{x}) = -\sum_{i=1}^n y_k \log(f_i(\mathbf{x}))$, where each $f_i : \mathbb{R}^n \to \mathbb{R}$ is given by (10.5). Use the chain rule to show that $D_j g(\mathbf{x}) = f_j(\mathbf{x}) \left(\sum_{i=1}^n y_i\right) - y_j$.

10.10. Given a sequence of real numbers $b_1, b_2, \ldots, b_m$ and an $m \times n$ matrix $A = [a_{ij}]$ with rows $\mathbf{a}_1^\mathsf{T}, \mathbf{a}_2^\mathsf{T}, \ldots, \mathbf{a}_m^\mathsf{T}$ in $\mathbb{R}^n$, define the *LogSumExp* function $f : \mathbb{R}^n \to \mathbb{R}$ as

$$f(\mathbf{x}) = \log\left(\sum_{j=1}^m z_j(\mathbf{x})\right), \tag{10.26}$$

where each $z_j(\mathbf{x}) = \exp(\mathbf{a}_j^\mathsf{T}\mathbf{x} + b_j)$. Show that

$$Df(\mathbf{x}) = \frac{\sum_{j=1}^m a_j^\mathsf{T} z_j(\mathbf{x})}{\sum_{j=1}^m z_j(\mathbf{x})} = \frac{\mathbf{z}(\mathbf{x})^\mathsf{T} A}{\mathbb{1}^\mathsf{T}\mathbf{z}(\mathbf{x})}, \tag{10.27}$$

where $\mathbb{1} \in \mathbb{R}^m$ is the ones vector and $\mathbf{z}(\mathbf{x}) = (z_1(\mathbf{x}), z_2(\mathbf{x}), \ldots, z_m(\mathbf{x}))$.

10.11. Show that the equations

$$\begin{aligned} \sin(x + z) + \ln(yz^2) &= 0, \\ e^{x+z} + yz &= 0 \end{aligned}$$

implicitly define continuously differentiable functions $x(z)$ and $y(z)$ in an open neighborhood of the point $(1, 1, -1)$. Find $x'(z)$ and $y'(z)$.

10.12. Let $A \in M_n(\mathbb{R})$ be symmetric and $\mathbf{b} \in \mathbb{R}^n$. Show that if $f(\mathbf{x}) = \mathbf{x}^\mathsf{T} A\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x}$, then $D^2 f(\mathbf{x}) = 2A$.

10.13. Let $U \subset \mathbb{R}^n$ be an open set and assume $\mathbf{r} : U \to \mathbb{R}^m$ is twice continuously differentiable on U. In Example 10.2.10 we showed that the "norm squared" function $f(\mathbf{x}) = \|\mathbf{r}(\mathbf{x})\|^2 = \mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{r}(\mathbf{x})$ satisfies $Df(\mathbf{x}) = 2\mathbf{r}(\mathbf{x})^\mathsf{T} D\mathbf{r}(\mathbf{x})$. Now show that the Hessian of f satisfies

$$D^2 f(\mathbf{x}) = 2\left(D\mathbf{r}(\mathbf{x})^\mathsf{T} D\mathbf{r}(\mathbf{x}) + \sum_{i=1}^m r_i(\mathbf{x}) D^2 r_i(\mathbf{x})\right), \tag{10.28}$$

where r_i is the ith coordinate function of $\mathbf{r}$.

10.14. Show that the Hessian of the *LogSumExp* function (10.26) is given by

$$D^2 f(\mathbf{x}) = A^\mathsf{T} \left(\frac{1}{\mathbb{1}^\mathsf{T} \mathbf{z}(\mathbf{x})} \operatorname{diag}(\mathbf{z}(\mathbf{x})) - \frac{1}{(\mathbb{1}^\mathsf{T} \mathbf{z}(\mathbf{x}))^2} \mathbf{z}(\mathbf{x}) \mathbf{z}(\mathbf{x})^\mathsf{T} \right) A, \qquad (10.29)$$

where z is defined as in Exercise 10.10.

10.15. Find the second-order Taylor polynomial for $g(x, y, z) = e^{2x + yz}$ at the point $(0, 0, 0)$.

Notes

For a deeper and more thorough exploration into differentiation, see Volume 1, Chapter 6.

11 Fundamentals of Numerical Computation

Nothing brings fear to my heart more than a floating point number.
—Gerald Jay Sussman

11.1 Floating-Point Arithmetic

Essentially every algorithm involving real numbers is implemented using floating-point numbers. Many basic floating-point operations are built into CPUs at the lowest level, so they can be executed very rapidly. These operations include arithmetic operations (addition, subtraction, multiplication, division, and square root), as well as other important functions such as round. More complicated operations, and operations on more general data types, such as arbitrarily long integers, are slower because they process more data and require software function calls instead of hardware calls that are built in to the processor's design.[43] For this reason the temporal complexity of many algorithms is reported in terms of the number of floating-point operations (FLOPs) used by the algorithm; see also Section 1.5.1.

11.1.1 Fundamentals of Floating Point

The default number in most floating point systems is the 8-byte (64-bit) double-precision floating-point number. Floating-point numbers are maintained in binary scientific notation, like $-1.01111010100_2 \times 2^{101101_2}$. These numbers are broken into three parts: the *sign*, the *significand*, and the *exponent*:

$$\underbrace{-}_{\text{sign}} \underbrace{1.01111010100_2}_{\text{significand}} \times 2^{\overbrace{101101_2}^{\text{exponent}}}.$$

[43] With software function calls, the computer has to find instructions in memory for how to compute the desired function.

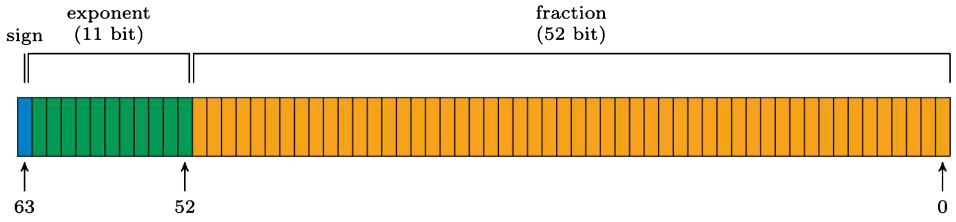


Figure 11.1. *Double-precision bit map for IEEE 754 double-precision binary floating-point format. Bits are, from left to right,* $sgn, e_{62}, e_{61}, \ldots, e_{52}, b_{51}, \ldots, b_0$.

Except for special numbers like 0 we can *normalize* each number so that the leading digit of the significand is 1, so we need not record that digit. Instead we only record the digits after the decimal point, which we call the *fraction*.[44] Almost all computers currently follow the IEEE 754 standard for double-precision floating-point numbers. In this standard, the first bit is the sign of the number. The next 11 bits correspond to the exponent and the last 52 bits represent the fraction; see Figure 11.1 for an illustration. All together, we have

$$\text{value} = (-1)^{\text{sgn}}(1.b_{51}b_{50}\cdots b_0)_2 \times 2^{e-1023} = (-1)^{\text{sgn}}\left(1 + \sum_{i=1}^{52} b_{52-i}2^{-i}\right) \times 2^{e-1023},$$

where the exponent is represented as the bits $e_{62}e_{61}\ldots e_{53}e_{52}$ with $e = \sum_{i=0}^{10} e_{52+i}2^i$. If e is the stored value of the exponent, the actual exponent of the floating-point number is shifted to $e - 1023$; this allows us to represent both positive and negative exponents without tracking another sign. The exponents of -1023 (all zeros) and $+1024$ (all ones) are reserved for special numbers, so the range of possible exponents is all the integers from -1022 up to $+1023$.

Example 11.1.1. The number

$$5 = 101_2 = 1.01_2 \times 2^2$$

is positive, so it has a sign bit of 0, an exponent of $2 = 00000000010_2$, which is shifted to a stored exponent of $2 + 1023 = 10000000001_2$, and a fraction of $.01_2$, so 5 is represented in floating point as

0100000000010100000000000000000000000000000000000000000000000000.

[44] Beware that some people use the words *fraction* and *significand* interchangeably. Also, some people use the word *mantissa* instead of *significand*, but that term is misleading because it also means the fractional part of a logarithm, which is similar to, but not quite the same as, the significand.

Example 11.1.2. The number

$$-0.25 = -0.01_2 = -1.0_2 \times 2^{-2}$$

is negative, so it has a sign bit of 1, an exponent of $-2 = -00000000010_2$, which is shifted to a stored exponent of $-2 + 1023 = 0111111101_2$, and a fraction of all zeros, so -0.25 is represented in floating point as

1011111111010000000000000000000000000000000000000000000000000000.

Special Numbers

There are two special exponents—all zeros and all ones—that allow representation of the following special numbers:

- Denormalized numbers.
- $+0$ and -0.
- $+\text{INF}$ and $-\text{INF}$.
- NaN.

The special exponent -1023 (all zeros) indicates that the exponent should be treated as -1022, and the leading bit, which is usually taken to be 1, is now 0. This allows us to represent additional numbers like 1.0×2^{-1025} as 0.001×2^{-1022}. These are called *denormalized numbers* and they range in size all the way down to $2^{-52-1022}$.

Zero is the special denormalized number represented by setting both the exponent and the fraction to be all zeros. The sign bit is still used, however, so floating point has both a $+0$ and a -0. These usually print as the same number, and an equality check `-0 == +0` returns `True`, yet they are different because $1/(+0)$ returns $+\text{INF}$ and $1/(-0)$ returns $-\text{INF}$.

The special exponent of $+1024$ (all ones) indicates INF (∞) or $-\text{INF}$ ($-\infty$) if the fraction is all zeros. The exponent of $+1024$ indicates NaN (Not a Number) if the fraction is nonzero. IEEE 754 arithmetic yields the following results with computations involving INF:

$$\begin{aligned}
\text{any nonnegative number}/\pm\text{INF} &= \pm 0, & \pm\text{INF} \times \text{INF} &= \pm\text{INF},\\
\text{any positive number}/\pm 0 &= \pm\text{INF}, & \text{INF} + \text{INF} &= \text{INF},\\
\text{any positive number} \times \pm\text{INF} &= \pm\text{INF}, & \pm 0/\pm 0 &= \text{NaN},\\
\pm\text{INF} \times \pm 0 &= \text{NaN}, & \pm\text{INF}/\pm\text{INF} &= \text{NaN},\\
\text{INF} - \text{INF} &= \text{NaN}.
\end{aligned}$$

Any operation involving NaN returns NaN.

Gaps

Since the significand can only take on 2^{52} values, not every number can be represented. There are always gaps between the values we can represent with finite-precision numbers. There is a less popular system, called *fixed-point arithmetic*, in which all of the gaps are the same width along the number line. But this has the disadvantage that you can't represent both very small and very large numbers in the same system.

In floating point the size of a gap is determined by the size of the numbers near the gap. The floating-point numbers that lie in the range $[1, 2]$ are of the form

$$1, 1 + 2^{-52}, 1 + 2 \times 2^{-52}, 1 + 3 \times 2^{-52}, \ldots, 1 + (2^{52} - 1) \times 2^{-52}, 2$$

and the floating-point numbers that lie in the range $[2^k, 2^{k+1}]$ are just the numbers in $[1, 2]$ multiplied by 2^k. So the gaps between very small numbers are tiny, but the gaps between very large numbers are large.

Except for those numbers that are too large or too small to be represented, the distance from any real number x to its nearest floating-point representation is bounded by $2^{-53}|x|$. So, if $\mathrm{fl}(x)$ is the floating-point representation of x (that is, $\mathrm{fl}(x)$ is the representable number closest to x), then

$$|x - \mathrm{fl}(x)| \leq 2^{-53}|x|. \tag{11.1}$$

The advantage of floating-point numbers is that they allow us to represent very small numbers very finely and still represent enormous numbers in the same system. The disadvantage, as shown in Example 1.0.1, is that in some algorithms the relatively small errors caused by these gaps can compound into very large errors.

11.1.2 A Model of Floating-Point Arithmetic

Basic floating-point operations like addition, subtraction, multiplication, division, and square root are hard-coded into most computers. Generally addition, subtraction, and multiplication are several times faster than division and square root, but this depends on the specific implementation. Also, the error in various basic floating-point operations may depend on the particular numbers being operated on and on the specific implementation. Of course we want our analysis of algorithms to apply to all sorts of machines and not depend on the specific implementation. So, rather than go into the detailed specifics of these operations and the errors they produce for each implementation, we set some basic assumptions that we expect all floating-point systems to use and then make all our analyses using those assumptions.

The standard model for floating-point arithmetic consists of a set $\mathbf{F} \subset \mathbb{R}$ consisting of 0 and all numbers of the form

$$\pm \frac{s}{b^{p-1}} b^e,$$

where b (the base, which is usually 2) and p (the precision, which is the number of digits in the significand) are fixed integers, whereas s and e are variable integers with $b^{p-1} \leq s < b^p$ and $-M \leq e \leq N$ for some choice of $M, N \in \mathbb{N}$. Here s/b^{p-1} is the significand, and e is the exponent. Depending on the situation, we may ignore the bounds on s and e to simplify the analysis.

Example 11.1.3. For IEEE 754 double-precision binary floating point, the base b is 2, the precision p is 53, and the exponent e is bounded by $-1022 \leq e \leq 1023$ (ignoring denormalized numbers). The significand is $s/2^{52} = 1 + b_{51}/2 + b_{50}/4 + \cdots + b_0/2^{52}$. Clearing denominators gives

$$2^{52} \leq s = 2^{52} + b_{51}2^{51} + \cdots + b_1 2^1 + b_0 2^0 \leq 2^{53} - 1.$$

So s is any integer in the range from 2^{52} up to $2^{53} - 1$.

We require the system to satisfy two key axioms:

(i) There exists a number $\varepsilon_{\text{machine}}$, called *machine epsilon* or *unit round-off*,[45] such that for any x in the representable range, there exists δ with $|\delta| \leq \varepsilon_{\text{machine}}$ such that

$$\text{fl}(x) = (1 + \delta)x. \tag{11.2}$$

That is, the relative error $|\text{fl}(x) - x|/|x|$ is no more than $\varepsilon_{\text{machine}}$.

(ii) If $*$ denotes any of the standard operations $+$, $-$, $\times$, or $/$, and if $\circledast$ denotes its floating-point counterpart, then for $x, y, \in \mathbf{F}$ with $x * y$ in the representable range we have

$$x \circledast y = (1 + \delta)(x * y), \tag{11.3}$$

where $|\delta| \leq \varepsilon_{\text{machine}}$. Or, equivalently, the relative error

$$\left| \frac{(x \circledast y) - (x * y)}{x * y} \right|$$

is at most $\varepsilon_{\text{machine}}$.

We use this model whenever we need to analyze errors or evaluate the stability of algorithms.

Example 11.1.4. Equation (11.1) shows that $|\text{fl}(x) - x| \leq 2^{-53}|x|$ for IEEE 754 standard, double-precision. In fact, IEEE 754 requires that $\varepsilon_{\text{machine}} = 2^{-53}$.

Remark 11.1.5. The key thing to note in both of these axioms is that the error is relative. If we were interested in absolute error, we would require the differences $|\text{fl}(x) - x|$ or $|(x \circledast y) - (x * y)|$ to be less than $\varepsilon_{\text{machine}}$, but instead we require the ratios $|\text{fl}(x) - x|/|x|$ or $|(x \circledast y) - (x * y)|/|x * y|$ to be less than $\varepsilon_{\text{machine}}$.

The relative error is generally more meaningful than the absolute error. It is probably OK to be off by a mile when measuring the distance to the sun, but it's definitely not OK when measuring the distance to the bathroom. In order to know whether an error is significant, it needs to be measured and reported in proportion to the thing being approximated. Relative error does that.

[45] Some authors use the term *machine epsilon* to mean the distance from 1 to the next largest floating-point number (that is, b^{1-p}), and they reserve *unit round-off* for the quantity we call $\varepsilon_{\text{machine}}$. For example, [Hig96] and NumPy use that convention, while [Dem97] and [TB97] use our convention.

Remark 11.1.6. It is common to treat the square root as a basic floating-point operation and assume that the floating-point implementation `sqrt` of square root satisfies the analogous property: if x and $\sqrt{x}$ are in the representable range, then $\texttt{sqrt}(x) = (1+\delta)\sqrt{x}$ for $|\delta| \leq \varepsilon_{\text{machine}}$.

Remark 11.1.7. Complex floating-point arithmetic is usually done by simply reducing complex numbers to their real and imaginary parts. We may still use our model of floating-point arithmetic, but the value of $\varepsilon_{\text{machine}}$ is changed by a factor of approximately $2^{\frac{5}{2}}$.

11.1.3 Practical Considerations for Using Floating Point

Programmers and algorithm designers who forget or ignore the basic properties of floating-point arithmetic can get into trouble. We discuss these issues more in Section 11.3, but here are a few practical considerations to keep in mind.

Test for Relative Nearness—Not Equality

Essentially every operation has some round-off error, so two floating-point numbers that should be equal will almost never be identical—they will only be close. For example, the command `sqrt(x)**2 == x` returns `False` for most values of x. A slightly better, but still naïve, way to try to identify when two floating-point numbers x and y are equal is to choose a small error tolerance δ and then check whether $|x-y| \leq \delta$. The problem with this is that it ignores the fact that the gaps between floating-point numbers are proportional to the size of the numbers; so, for example, even when x and y are adjacent floating-point numbers, if they are much larger than δ, this test for nearness fails.

A better solution is to check for relative nearness: choose some $\delta > 0$ and then test whether $|x-y| \leq \delta \max(|x|,|y|)$. If δ is not much bigger than $\varepsilon_{\text{machine}}$, then this test returns `True` only for x and y that are nearly adjacent.

Be Aware of Relative Size in Addition and Subtraction

Adding a relatively small number to a relatively large one returns the larger one unchanged. If $|y|$ is much smaller than the gap between x and its nearest floating-point neighbor, then $x \oplus y = x$. Here $\oplus$ is floating-point addition, as described in axiom (ii).

Example 11.1.8. In double-precision floating-point arithmetic we have $2^{53} + 1 = 2^{53}$ and $1 + 2^{-53} = 1$.

Similarly, subtracting two numbers that are different, but whose difference is relatively small, results in a substantial loss of precision. This is a problem, for example, when trying to compute derivatives from their definition as a limit of difference quotients.

Example 11.1.9. You might try to compute the derivative of $\cos(x)$ at $x = 1$ as

$$\lim_{h \to 0} \frac{\cos(1+h) - \cos(1)}{h}.$$

This estimate gradually improves until h gets to be about 2^{-26}, after which the approximation degrades. So while the true value of the derivative of cos at 1 is $-\sin(1) = -0.841470984808$, in Python with the default double-precision arithmetic, the command

```
for n in range(20,54):
    print(n, (cos(1+2**(-n)) - cos(1)) / (2**(-n)))
```

yields the following:

20	-0.841471242486	37	-0.84147644043
21	-0.84147111373	38	-0.841461181641
22	-0.841471049003	39	-0.841491699219
23	-0.841471017338	40	-0.841430664062
24	-0.841471001506	41	-0.841552734375
25	-0.841470994055	42	-0.84130859375
26	-0.841470986605	43	-0.841796875
27	-0.841470986605	44	-0.841796875
28	-0.841470986605	45	-0.83984375
29	-0.841471016407	46	-0.84375
30	-0.841470956802	47	-0.84375
31	-0.841470956802	48	-0.84375
32	-0.841471195221	49	-0.875
33	-0.841470718384	50	-0.875
34	-0.841470718384	51	-0.75
35	-0.841472625732	52	-1.0
36	-0.841468811035	53	0.0

For more about numerical computation of derivatives, see Section 11.4.

Keep Intermediate Steps at Reasonable Size

Despite the fact that floating-point arithmetic can represent some very large and very small numbers, it is still easy to exceed the largest possible value. This is called *overflow*. Overflow is a risk when working with exponentials, factorials, and other fast-growing functions, including many of the functions discussed in the first half of this book related to counting and probability. It is also easy to produce calculations whose results have absolute value smaller than the smallest positive representable value. This is called *underflow*.

Generally the intermediate steps in a computation are more susceptible to overflow or underflow than the final result is. Moreover, even if the intermediate steps do not actually overflow or underflow, if they are much larger than the final answer, you are at risk of losing precision in your computation.

Example 11.1.10. For many values of k the binomial coefficients $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ are not impossibly big, even if n is large. But computing them by first computing $n!$ and then dividing by the product $(n-k)!k!$ causes the computation to overflow once $n > 170$, regardless of k, because $171! > 2^{1024}$.

This problem can be solved in at least two ways. The first is to simplify the expression algebraically before constructing the algorithm: $n!/(n-k)! = n \cdot (n-1) \cdots (n-k+1)$, which is much smaller than $n!$ if k is small. A second way to deal with this is to use logarithms. We have

$$\begin{aligned}\frac{n!}{k!(n-k)!} &= \exp\left(\log\left(\frac{n!}{k!(n-k)!}\right)\right) \\ &= \exp\left(\sum_{j=1}^{n}\log(j) - \sum_{j=1}^{k}\log(j) - \sum_{j=1}^{n-k}\log(j)\right).\end{aligned}$$

This expression is much less likely to overflow than the original algorithm because the logarithms are much smaller than their inputs, they are summed rather than multiplied, and they are exponentiated only at the end of the computation.

11.1.4 *Financial Computations

Financial computations are problematic in base-2 floating point because a high degree of accuracy is required and because rounding is usually done in base 10. The primary problem with using base-2 floating point is that negative powers of 10 have a nonterminating representation in base 2, for example,

$$0.10 = 0.000\overline{1100}_2 \qquad \text{and} \qquad 0.01 = 0.0000001010001111010111000\overline{0}_2,$$

where the part that is overlined is repeated infinitely. Since 0.01 does not have an exact representation as a base-2 floating-point number, it is impossible to round to the nearest hundredth in base 2—instead the machine rounds to the appropriate binary approximation, depending on the binary precision being used. Financial numbers usually must be recorded, rounded, and reported to the nearest hundredth or some other power of 10, and there are strict laws regarding this rounding. Therefore, any system that relies on binary arithmetic will introduce errors in the continual conversion back and forth between decimal and binary.

Many programming languages have a software implementation of base-10 floating-point arithmetic that helps reduce these problems. But if the decimal arithmetic is not implemented directly in the hardware, there is a substantial loss of speed using decimal arithmetic compared to binary floating point (a factor of roughly 100 in Python).

Of course, switching to base 10 does not prevent all round-off errors. As shown earlier, floating point in any base presents problems with loss of precision. This can be a special problem in financial transactions because even minor errors have real costs and create significant opportunities for malicious exploitation.

It is tempting to try to solve this by performing all computations with exact (integer or rational) arithmetic to avoid errors. But there are at least two problems with this idea: first, some important financial computations, like continuous compounding, involve transcendental numbers that cannot be represented as rational numbers without round-off; and second, exact arithmetic is inefficient and slows down rapidly as the number of computations grows. A system based on exact arithmetic could rapidly become unusable in settings involving many operations, like a record of many savings accounts, with interest, deposits, and withdrawals over many days.

The numbers occurring in financial applications usually lie within a well-defined range and require a well-defined level of precision. In those situations fixed-point decimal arithmetic may be useful.

11.2 A Brief Review of Conditioning

This section is an abbreviated version of Section 7.5 of Volume 1.

The solution to nearly every problem in applied mathematics can ultimately be expressed as an algorithm. Executing these algorithms amounts to evaluating functions. But evaluating functions numerically has several potential sources of error. Two of the most important of these are errors in the inputs and errors in the intermediate computations. Since every measurement is inherently imprecise, and most numbers cannot be represented exactly as a floating-point number, inputs almost always have some error. Moreover, floating-point arithmetic almost always introduces additional error at each intermediate computation, and depending on the algorithm, these can accumulate to produce significant errors in the output.

For each problem we must ask (i) how sensitive is the function to small changes in the inputs? And (ii) how much error can accumulate from round-off (floating-point) error in the algorithm? The answer to the second question is measured by the *stability* of the algorithm. Stability is treated in Section 11.3. The answer to the first question is captured in the *conditioning* of the problem. If a small change to the input only results in a small change to the output of the function, we say that the function is *well conditioned*. But if small changes to the input result in large changes to the output, we say the function is *ill conditioned*. Not surprisingly, a function can be ill conditioned for some inputs and well conditioned for other inputs.

Example 11.2.1. Consider the function $f(x) = x/(1-x)$. For values of x close to 1, a small change in x produces a large change in $f(x)$. For example, if the correct input is $x_* = 1.001$, and if that is approximated by $\tilde{x} = 1.002$, the actual output of $f(\tilde{x}) = 1/(1-1.002) = -500$ is very different from the desired output of $f(x_*) = 1/(1-1.001) = -1000$. So this problem is ill conditioned near $x = 1$. Note that this error has nothing to do with round-off errors in the algorithm for computing the values—it is entirely a property of the function itself.

But if the desired input is $x_0 = 35$, then the correct output is $f(x_*) = -1.0294$, and even a bad approximation to the input like $\tilde{x} = 36$ gives a good approximate output $f(\tilde{x}) = -1.0286$. So this problem is well conditioned near $x = 35$.

11.2.1 Condition Number of a Function

The condition number of a function at a point measures how sensitive the function is to changes in input values. Throughout this section we assume that norms have been fixed on $\mathbb{F}^n$ and $\mathbb{F}^m$ (we use $\|\cdot\|$ to denote both of these norms).

Definition 11.2.2. *Let $U \subset \mathbb{F}^n$ and $f : U \to \mathbb{F}^m$ be given. The* absolute condition number *of f at $\mathbf{x} \in U$ is*

$$\hat{\kappa}(\mathbf{x}) = \lim_{\delta \to 0^+} \sup_{\|\mathbf{h}\| < \delta} \frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})\|}{\|\mathbf{h}\|}.$$

Proposition 11.2.3. *Let $U \subset \mathbb{F}^n$ be an open set containing $\mathbf{x}$. If $f : U \to \mathbb{F}^m$ is differentiable at $\mathbf{x}$, then*

$$\hat{\kappa}(\mathbf{x}) = \|Df(\mathbf{x})\|, \tag{11.4}$$

where $\|Df(\mathbf{x})\|$ is the induced norm of the linear transformation $Df(\mathbf{x})$; see Volume 1, Section 3.5.2, for more on induced norms.

Remark 11.2.4. The condition number depends on the choice of norm, but all norms on $\mathbb{F}^n$ are topologically equivalent (Volume 1, Theorem 5.8.7), which means that for any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ there exist constants $0 < m \leq M$ such that

$$m\|\mathbf{x}\|_a \leq \|\mathbf{x}\|_b \leq M\|\mathbf{x}\|_a \tag{11.5}$$

for all $\mathbf{x} \in \mathbb{F}^n$. Therefore, the condition number of f with respect to the norm $\|\cdot\|_b$ is bounded by a fixed multiple of the condition number of f for the norm $\|\cdot\|_a$, as follows:

$$\frac{m}{M}\frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})\|_a}{\|\mathbf{h}\|_a} \leq \frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})\|_b}{\|\mathbf{h}\|_b} \leq \frac{M}{m}\frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})\|_a}{\|\mathbf{h}\|_a},$$

and thus

$$\frac{m}{M}\hat{\kappa}_a(\mathbf{x}) \leq \hat{\kappa}_b(\mathbf{x}) \leq \frac{M}{m}\hat{\kappa}_a(\mathbf{x})$$

for all $\mathbf{x}$.

In most settings, relative error is more useful than absolute error. An error of 1 is tiny if the true answer is 10^{20}, but it is huge if the true answer is 10^{-20}. Relative error accounts for this difference. Since the condition number is really about the size of errors in the output, the relative condition number is usually a better measure of conditioning than the absolute condition number.

Definition 11.2.5. *Let $U \subset \mathbb{F}^n$ be an open set, and let $f : U \to \mathbb{F}^m$ be a function. The* relative condition number *of f at $\mathbf{x} \in U$ is*

$$\kappa(\mathbf{x}) = \lim_{\delta \to 0^+} \sup_{\|\mathbf{h}\| < \delta} \left(\frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})\|}{\|f(\mathbf{x})\|} \Big/ \frac{\|\mathbf{h}\|}{\|\mathbf{x}\|} \right) = \frac{\hat{\kappa}(\mathbf{x})}{\|f(\mathbf{x})\|/\|\mathbf{x}\|}. \tag{11.6}$$

Remark 11.2.6. The relative condition number depends on the choice of norm, but, as in the case of the absolute condition number (see Remark 11.2.4), the value of $\kappa(\mathbf{x})$ relative to a norm $\|\cdot\|_a$ is bounded by a constant times the value of $\kappa(\mathbf{x})$ relative to the norm $\|\cdot\|_b$.

Remark 11.2.7. A problem is *well conditioned at* $\mathbf{x}$ if the relative condition number is small. Similarly, the problem is *ill conditioned* if the relative condition number is large. Of course, what we mean by "small" or "large" depends on the problem.

Nota Bene 11.2.8. Roughly speaking, we have

$$\begin{pmatrix}\text{relative}\\ \text{change}\\ \text{in output}\end{pmatrix} = \begin{pmatrix}\text{relative}\\ \text{condition}\\ \text{number}\end{pmatrix} \times \begin{pmatrix}\text{relative}\\ \text{change}\\ \text{in input}\end{pmatrix}.$$

This leads to a general rule of thumb that, *without any error in the algorithm itself*, we should expect to lose k digits of accuracy if the relative condition number is 10^k.

If f is differentiable, then Proposition 11.2.3 gives a formula for the relative condition number in terms of the derivative.

Corollary 11.2.9. *Let $U \subset \mathbb{F}^n$ be an open set containing $\mathbf{x}$. If $f : U \to \mathbb{F}^m$ is differentiable at $\mathbf{x}$, then*

$$\kappa(\mathbf{x}) = \frac{\|Df(\mathbf{x})\|}{\|f(\mathbf{x})\|/\|\mathbf{x}\|} = \frac{\|\mathbf{x}\|\|Df(\mathbf{x})\|}{\|f(\mathbf{x})\|}. \tag{11.7}$$

Example 11.2.10. Consider the function $f(x) = \frac{x}{1-x}$ of Example 11.2.1. We have $Df(x) = (1-x)^{-2}$, and hence, by (11.7), we have

$$\kappa = \frac{\|Df(\mathbf{x})\|}{\|f(\mathbf{x})\|/\|\mathbf{x}\|} = \frac{\left|\frac{1}{(1-x)^2}\right|}{\left|\frac{x}{1-x}\right|/|x|} = \left|\frac{1}{1-x}\right|.$$

This problem has a small relative condition number when x is far from 1 and a large relative condition number when $|1-x|$ is small.

Example 11.2.11. Given y, consider the problem of finding x on the curve $x^3 - x = y^2$. Setting $F(x,y) = x^3 - x - y^2$, we can rewrite this as the problem of finding x to satisfy $F(x,y) = 0$. Note that $D_x F(x,y) = 3x^2 - 1$, so, provided that $x \neq \pm\sqrt{1/3}$, the implicit function theorem (Theorem 10.3.2) applies and guarantees that there is (locally) a function $x(y)$ such that $F(x(y), y) = 0$.

Moreover, $Dx(y) = dx/dy = 2y/(3x^2 - 1)$. Therefore, near a point (x, y) on the curve with $x \neq \pm\sqrt{1/3}$, the relative condition number of this function is

$$\kappa(x, y) = \frac{|2y/(3x^2 - 1)|}{|x|/|y|} = \frac{2y^2}{|3x^3 - x|} = \frac{2|x^3 - x|}{|x||3x^2 - 1|} = \frac{2|x^2 - 1|}{|3x^2 - 1|}.$$

This problem is ill conditioned when x is close to $\pm\sqrt{1/3}$ and well conditioned elsewhere.

11.2.2 Condition of Finding a Simple Root of a Polynomial

The implicit function theorem can be used to show that varying the coefficients of a single-variable polynomial p causes the simple roots (those of multiplicity 1) of p to vary as a continuous function of the coefficients. The next proposition gives the condition number of that function.

Proposition 11.2.12. *Define $P : \mathbb{F}^{n+1} \times \mathbb{F} \to \mathbb{F}$ by $P(\mathbf{a}, x) = \sum_{i=0}^{n} a_i x^i$. For any given $\mathbf{a}^* \in \mathbb{F}^{n+1}$ and any simple root $x^* \in \mathbb{F}$ of the polynomial $p(x) = P(\mathbf{a}^*, x)$, there is a neighborhood U of $\mathbf{a}^*$ in $\mathbb{F}^{n+1}$ and a continuously differentiable function $r : U \to \mathbb{F}$ with $r(\mathbf{a}^*) = x^*$ such that $P(\mathbf{a}, r(\mathbf{a})) = 0$ for all $\mathbf{a} \in U$. Moreover, the relative condition number of r as a function of the ith coefficient a_i at the point $(\mathbf{a}^*, x^*)$ is*

$$\kappa = \left| \frac{(x^*)^{i-1} a_i^*}{p'(x^*)} \right|. \tag{11.8}$$

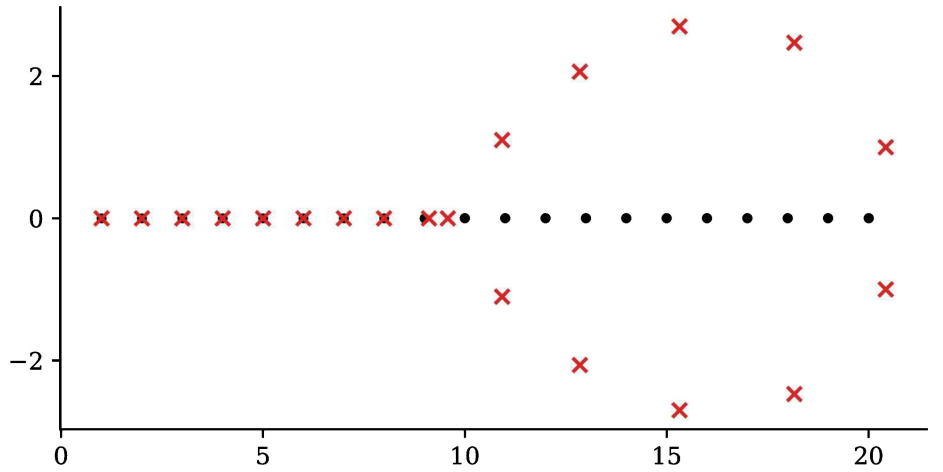


Figure 11.2. *The black dots are the true roots of the Wilkinson polynomial $w(x)$ plotted in the complex plane. The red crosses are the roots of the polynomial obtained by perturbing $w(x)$ by 10^{-7} in the coefficient of x^{19}. As described in Example 11.2.13, the roots are very sensitive to tiny variations in this coefficient because the relative condition number is very large.*

Proof. A root x^* of a polynomial p is simple if and only if $p'(x^*) \neq 0$. Differentiating P at $(\mathbf{a}^*, x^*)$ with respect to x gives $D_x P(\mathbf{a}^*, x^*) = \sum_{i=1}^{n} i a_i^* (x^*)^{i-1} = p'(x^*)$. Because $p'(x^*)$ is invertible, the implicit function theorem guarantees the existence of a neighborhood U of $\mathbf{a}$ and a unique continuous function $r : U \to \mathbb{F}$ such that $r(\mathbf{a}^*) = x^*$ and such that $P(\mathbf{a}, r(\mathbf{a})) = 0$ for all $\mathbf{a} \in U$. Moreover, we have

$$Dr(\mathbf{a}^*) = -D_x P(\mathbf{a}^*, x^*)^{-1} D_{\mathbf{a}} P(\mathbf{a}^*, x^*) = -\frac{1}{p'(x^*)} \begin{bmatrix} 1 & x^* & (x^*)^2 & \cdots & (x^*)^n \end{bmatrix}.$$

Combining this with (11.7) shows that the relative condition number of r as a function of the ith coefficient a_i is given by (11.8). □

Example 11.2.13. If the derivative $p'(x^*)$ is small, relative to the coefficient a_i^*, then the rootfinding problem is ill conditioned near $(\mathbf{a}^*, x^*)$. A classic example of this is the Wilkinson polynomial

$$w(x) = \prod_{r=1}^{20} (x - r) = x^{20} - 210x^{19} + 20615x^{18} - \cdots + 2432902008176640000.$$

Perturbing the polynomial by changing the coefficient of x^{19} from -210 to -210.0000001 changes the roots substantially. Specifically, the 10 largest (in modulus) roots become $20.4 \pm 0.99i$, $18.16 \pm 2.47i$, $15.31 \pm 2.70i$, $12.85 \pm 2.06i$, and $10.92 \pm 1.10i$, while the 10 smaller roots remain real and are closer to their original integer values. This is plotted in Figure 11.2. The big change in the value of the roots is because the derivative $p'(x^*)$ is small for roots like $x^* = 15$, relative to $(x^*)^{18} a_{19}^*$, where a_{19}^* is the coefficient of x^{19}. Specifically, at $x^* = 15$ we have

$$\kappa = \frac{15^{18}(-210)}{p'(15)} \approx 3.0 \times 10^{10}.$$

11.2.3 Condition Number of a Matrix

We conclude this section by discussing the condition number for problems of the form $A\mathbf{x} = \mathbf{b}$, where $A \in M_n(\mathbb{F})$ is nonsingular. There are several cases to consider:

(i) Given A, what is the relative condition number of $f(\mathbf{x}) = A\mathbf{x}$?

(ii) Given $\mathbf{x} \in \mathbb{F}^n$, what is the relative condition number of $g(A) = A\mathbf{x}$?

(iii) Given A, what is the relative condition number of $h(\mathbf{b}) = A^{-1}\mathbf{b}$?

Although the relative condition numbers of these three cases are not identical, they are all bounded by the number $\|A\|\|A^{-1}\|$, and this is the best uniform bound.

Theorem 11.2.14.

(i) *The relative condition number of* $f(\mathbf{x}) = A\mathbf{x}$ *satisfies*

$$\kappa = \|A\| \frac{\|\mathbf{x}\|}{\|A\mathbf{x}\|} \leq \|A\|\|A^{-1}\|. \tag{11.9}$$

Moreover, if the norm $\|\cdot\|$ *is the 2-norm, then equality holds when* $\mathbf{x}$ *is a right singular vector of* A *corresponding to the minimal singular value.*

(ii) *Given* $\mathbf{x} \in \mathbb{F}^n$*, the relative condition number of* $g(A) = A\mathbf{x}$ *satisfies*

$$\kappa = \|\mathbf{x}\| \frac{\|A\|}{\|A\mathbf{x}\|} \leq \|A\|\|A^{-1}\|. \tag{11.10}$$

Moreover, for the 2-norm, equality holds when $\mathbf{x}$ *is a right singular vector of* A *corresponding to the minimal singular value.*

(iii) *The relative condition number of* $h(\mathbf{b}) = A^{-1}\mathbf{b}$ *satisfies*

$$\kappa = \|A^{-1}\| \frac{\|\mathbf{b}\|}{\|A^{-1}\mathbf{b}\|} \leq \|A\|\|A^{-1}\|. \tag{11.11}$$

Moreover, for the 2-norm, equality holds when $\mathbf{b}$ *is a left singular vector of* A *corresponding to the maximal singular value.*

Proof. For the proof of (11.9), note that Corollary 10.2.7 gives $Df(\mathbf{x}) = A$, so Corollary 11.2.9 gives the first equality of (11.9). To get the upper bound, substitute $\mathbf{x} = A\mathbf{y}$ into the definition of $\|A^{-1}\|$ to get

$$\|A^{-1}\| = \sup_{\mathbf{x}} \frac{\|A^{-1}\mathbf{x}\|}{\|\mathbf{x}\|} = \sup_{\mathbf{y}} \frac{\|A^{-1}A\mathbf{y}\|}{\|A\mathbf{y}\|} = \sup_{\mathbf{y}} \frac{\|\mathbf{y}\|}{\|A\mathbf{y}\|}.$$

This gives $\|A^{-1}\| \geq \|\mathbf{y}\|/\|A\mathbf{y}\|$ for all $\mathbf{y}$, from which we get (11.9).

The proof of (iii) is similar. Again Corollary 10.2.7 gives $Dh(\mathbf{b}) = A^{-1}$. The proof of (ii) is a little trickier. The details are given in Volume 1, Theorem 7.5.11. □

The previous theorem inspires the following definition.

Definition 11.2.15. *Let* $A \in M_n(\mathbb{F})$*. The* condition number of A *is*

$$\kappa(A) = \|A\|\|A^{-1}\|.$$

Nota Bene 11.2.16. Although $\kappa(A)$ is called the *condition number of the matrix* A, it is not the condition number (as given in Definition 11.2.5) of most problems associated to A. Rather, it is the supremum of the condition numbers of each of the various problems in Theorem 11.2.14; in other words, it is a sharp uniform bound for each of those condition numbers. Also the problem of finding eigenvalues of A has an entirely different condition number (see Volume 1, Section 7.5.4).

11.3 Stability of Numerical Algorithms

As described in Section 11.2 the two main sources of significant error in a numerical computation are poor stability and ill conditioning. If a problem is ill conditioned, then no algorithm will give good results—the problem itself is not well suited to numerical computation. If the problem is well conditioned, then there is hope that it is amenable to computation, but one still needs an algorithm to compute the solution to the problem. It is possible that the algorithm can produce a large error from round-off and truncation, rendering the solution unreliable. Such an algorithm is a bad algorithm, even if the problem itself is well conditioned. It is important to remember that even when an algorithm has been proved correct when implemented in exact arithmetic, it can still give very bad answers when implemented in finite-precision arithmetic on a computer. The study of the stability of algorithms is about quantifying the severity of the accumulated errors produced by round-off.

Throughout this section, assume that a variable wearing a tilde denotes a computed quantity; thus, $\tilde{\mathbf{x}}$ denotes the computed value of $\mathbf{x}$.

11.3.1 Forward Error

When evaluating the errors in an algorithm, it might seem natural to think about the *forward error*, which is the (relative) difference between the computed value and the exact "true value."

Definition 11.3.1. *For a function $f : \mathbb{F}^n \to \mathbb{F}^m$, let $\tilde{f}(\mathbf{x})$ represent the computed value of $f(\mathbf{x})$ (that is, the result at $\mathbf{x}$ of some algorithm to compute f). Given a choice of norm on $\mathbb{F}^m$, the* relative forward error *is given by*

$$\varepsilon_{\tilde{f}} = \frac{\|f(\mathbf{x}) - \tilde{f}(\mathbf{x})\|}{\|f(\mathbf{x})\|}. \tag{11.12}$$

Remark 11.3.2. It is common to call the relative forward error of an algorithm the *accuracy* of the computation. This is different from, but depends very much upon, the *precision*, of the machine, which is determined by $\varepsilon_{\text{machine}}$.

Remark 11.3.3. As in the case of condition numbers (see Remark 11.2.4) the relative forward error depends on the choice of norm, but the relative forward error with respect to the norm $\|\cdot\|_b$ is bounded by a fixed multiple of the relative forward error for the norm $\|\cdot\|_a$, as follows:

$$\frac{m}{M}\frac{\|f(\mathbf{x}) - \tilde{f}(\mathbf{x})\|_a}{\|f(\mathbf{x})\|_a} \leq \frac{\|f(\mathbf{x}) - \tilde{f}(\mathbf{x})\|_b}{\|f(\mathbf{x})\|_b} \leq \frac{M}{m}\frac{\|f(\mathbf{x}) - \tilde{f}(\mathbf{x})\|_a}{\|f(\mathbf{x})\|_a}.$$

There are at least two problems with using forward error to evaluate the quality of an algorithm. The first is that it does not account for the fact that the output of the algorithm might be fundamentally wrong or nonsensical, even if the forward error is small.

Unexample 11.3.4. Consider a decay function $f(t) = e^{-t}$, being used in a physical problem to represent the mass of some object at time t. Suppose we have two algorithms for computing $f(t)$, and suppose that on the input $t = 10$, the first algorithm produces the result $\tilde{f}(t) = -4.5 \times 10^{-5}$ and the second produces the output $\breve{f}(t) = 4.5 \times 10^{-4}$. The correct answer is approximately 4.5×10^{-5}; therefore, the relative forward error of the first algorithm is $\varepsilon_{\tilde{f}} = (4.5 - (-4.5))/4.5 = 2$, whereas the second algorithm has a relative forward error of $\varepsilon_{\breve{f}} = (4.5 - (4.5 \times 10))/4.5 = 9$.

However, the quantity $\tilde{f}(10) = -4.5 \times 10^{-5}$ does not make sense for the function $f(t) = e^{-t}$, because e^{-t} is always positive, and the concept of negative mass is not physically meaningful. Meanwhile, $4.5 \times 10^{-4} \approx e^{-7.7}$ is at least the correct value of e^{-t} for a value of t close to 10.

The second problem with using forward error to evaluate an algorithm is that we cannot expect the input itself to be exact—in reality one computes with approximate inputs, and so we are really considering the error produced by $\tilde{f}(\mathbf{x} + \boldsymbol{\varepsilon})$ for some small $\boldsymbol{\varepsilon}$.

11.3.2 Backward Error

To separate the conditioning of the problem from the stability of the algorithm, we instead think about the *relative backward error* of the algorithm.

Definition 11.3.5. *Let $\|\cdot\|$ denote a norm on $\mathbb{F}^m$. Given a function $f : \mathbb{F}^n \to \mathbb{F}^m$, the* absolute backward error *of an algorithm $\tilde{f}$ for f is the smallest $\|\boldsymbol{\delta}\|$ for all $\boldsymbol{\delta} \in \mathbb{F}^n$ such that $\tilde{f}(\mathbf{x}) = f(\mathbf{x} + \boldsymbol{\delta})$; that is, if*

$$\widehat{\beta}_{\tilde{f}} = \min\{\|\boldsymbol{\delta}\| : \tilde{f}(\mathbf{x}) = f(\mathbf{x} + \boldsymbol{\delta})\},$$

then the relative backward error *of $\tilde{f}$ is $\beta_{\tilde{f}} = \dfrac{\widehat{\beta}}{\|\mathbf{x}\|}$.*

If the relative backward error is small, then the algorithm gives exactly the right answer to nearly the right problem. Since our inputs are almost never exact, this is a reasonable measure of the quality of the algorithm. Indeed, as pointed out in [Hig96, Section 1.5], if the backward error is no larger than the uncertainties in the inputs, the quantity $\mathbf{x} + \boldsymbol{\delta}$ could even *be* the correct input, and $\tilde{f}(\mathbf{x})$ could well be the exact answer to our question. So it is reasonable to say that the algorithm performs well when the relative backward error is small.

Example 11.3.6. Consider the two algorithms in Unexample 11.3.4. The first algorithm has no meaningful finite value for the backward error, since the output is not equal to the exact answer for any input. It might make sense in this setting to say that the backward error is infinite $\beta_{\tilde{f}} = \infty$.

The second algorithm produces $\breve{f}(10) = 4.5 \times 10^{-4} = f(7.7)$, so the relative backward error of the second algorithm is $\beta_{\breve{f}} = (10 - 7.7)/10 = 0.23$.

Example 11.3.7. Let $\tilde{f}$ be the algorithm $\tilde{f}(x,y) = \mathrm{fl}(x) \oplus \mathrm{fl}(y)$ for computing the sum $f(x,y) = x + y$. By the axioms of floating-point arithmetic, we have

$$\begin{aligned}\tilde{f}(x,y) &= \mathrm{fl}(x) \oplus \mathrm{fl}(y) \\ &= (\mathrm{fl}(x) + \mathrm{fl}(y))(1+\delta_1) \\ &= (x(1+\delta_2) + y(1+\delta_3))(1+\delta_1) \\ &= x(1+\delta_2)(1+\delta_1) + y(1+\delta_3)(1+\delta_1) \\ &\leq x + y + x(\delta_2+\delta_1) + y(\delta_3+\delta_1) + (x+y)\varepsilon_{\text{machine}}^2,\end{aligned}$$

where $|\delta_1|, |\delta_2|, |\delta_3| \leq \varepsilon_{\text{machine}}$. Letting

$$\boldsymbol{\delta} = (x(1+\delta_2)(1+\delta_1) - x,\ y(1+\delta_3)(1+\delta_1) - y)$$

gives $\tilde{f}(x,y) = f((x,y) + \boldsymbol{\delta})$, so the relative backward error of this algorithm is bounded by

$$\begin{aligned}\frac{\|\boldsymbol{\delta}\|}{\|(x,y)\|} &\leq \frac{(2\varepsilon_{\text{machine}} + \varepsilon_{\text{machine}}^2)\|(x,y)\|}{\|(x,y)\|} \\ &= 2\varepsilon_{\text{machine}} + \varepsilon_{\text{machine}}^2.\end{aligned}$$

11.3.3 Backward Stability

We would like to say that an algorithm is backward stable if the relative backward error is always small. Of course the meaning of "small" might depend on the situation, but the following definition gives a uniform meaning to the idea of backward stability as precision increases.

Definition 11.3.8. *An algorithm $\tilde{f}$ for f is* backward stable *if there exists a constant $C \geq 0$ such that for each $\mathbf{x}$ in the domain of f and for each $\varepsilon_{\text{machine}}$ (determining the precision of all the arithmetic used in the algorithm) there exists $\boldsymbol{\delta} \in \mathbb{F}^n$ such that $\tilde{f}(\mathbf{x}) = f(\mathbf{x} + \boldsymbol{\delta})$ with*

$$\frac{\|\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \leq C\varepsilon_{\text{machine}}. \tag{11.13}$$

If an algorithm is backward stable, we can make the relative backward error as small as desired by using sufficiently high precision (a sufficiently small $\varepsilon_{\text{machine}}$).

Remark 11.3.9. It is common to write (11.13) as

$$\frac{\|\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \in O(\varepsilon_{\text{machine}}) \qquad \text{as } \varepsilon_{\text{machine}} \to 0.$$

Notice here that $\boldsymbol{\delta}$ is a function of $\varepsilon_{\text{machine}}$ and $\mathbf{x}$, while C must be independent of both $\mathbf{x}$ and $\varepsilon_{\text{machine}}$.

Example 11.3.10. If $\tilde{f}$ is the algorithm $\tilde{f}(x,y) = \text{fl}(x) \oplus \text{fl}(y)$ for computing the sum $f(x,y) = x + y$, then, as shown in Example 11.3.7, the relative backward error of the algorithm is bounded by $\beta_{\tilde{f}} \leq 2\varepsilon_{\text{machine}} + \varepsilon_{\text{machine}}^2$, which is in $O(\varepsilon_{\text{machine}})$ as $\varepsilon_{\text{machine}} \to 0$. Therefore floating-point addition is backward stable. The proofs that the other basic floating-point operations are backward stable are similar.

Example 11.3.11. For $x \in \mathbb{R}$ we show that the algorithm $\tilde{g}(x) = \text{fl}(x) \otimes \text{fl}(x)$ for computing $g(x) = x^2$ is backward stable. The axioms of floating-point arithmetic give

$$\tilde{g}(x) = x^2(1+\delta_1)^2(1+\delta_2) = x^2(1 + 2\delta_1 + \delta_1^2 + \delta_2 + 2\delta_1\delta_2 + \delta_1^2\delta_2)$$

or

$$\tilde{g}(x) = x^2(1 + 2\delta_1 + \delta_2 + \delta_3)$$

for some δ_3 with $|\delta_3| < 4\varepsilon_{\text{machine}}^2$. We must find δ such that $|\delta|/|x| \in O(\varepsilon_{\text{machine}})$ and $g(x+\delta) = \tilde{g}(x)$. That is, we need

$$(x+\delta)^2 = x^2 + 2x\delta + \delta^2 = x^2 + x^2(2\delta_1 + \delta_2) + x^2\delta_3.$$

This is quadratic in δ, and the smaller of the two solutions is

$$\delta = x(-1 + \sqrt{1 - (2\delta_1 + \delta_2 + \delta_3)}).$$

Using the Taylor expansion $\sqrt{1-a} = 1 - \frac{1}{2}a + O(a^2)$ shows that

$$|\delta| \leq |x| \left(\frac{7}{2}\varepsilon_{\text{machine}} + O(\varepsilon_{\text{machine}}^2) \right)$$

and thus

$$\frac{|\delta|}{|x|} \in O(\varepsilon_{\text{machine}}),$$

as required.

Example 11.3.12. Consider the algorithm $\tilde{g}(x) = \text{fl}(x) \oplus 1$ for computing $g(x) = x + 1$. We have

$$\begin{aligned} \tilde{g}(x) &= \text{fl}(x) \oplus 1 = (x(1+\delta_2) + 1)(1+\delta_1) \\ &= (x(1+\delta_2)(1+\delta_1) + \delta_1) + 1 \\ &= g(x+\delta), \end{aligned}$$

where $\delta_1, \delta_2 \leq \varepsilon_{\text{machine}}$ and

$$\delta = x(1+\delta_2)(1+\delta_1) + \delta_1 - x = x(\delta_1 + \delta_2 + \delta_1\delta_2) + \delta_1.$$

Thus the relative backward error of $\tilde{g}$ is $\beta_{\tilde{g}} = |(\delta_1 + \delta_2 + \delta_1\delta_2) + \delta_1/x|$, which could become arbitrarily large as $x \to 0$. Thus $\tilde{g}$ is likely not backward stable near $x = 0$ unless δ_1 also gets very small as $x \to 0$ for a fixed $\varepsilon_{\text{machine}}$. But since all we know is that $\delta_1 \leq \varepsilon_{\text{machine}}$, we have no reason to believe that this algorithm is backward stable near $x = 0$.

However, for any $\alpha > 0$, if x is restricted to lie in (α, ∞), the relative backward error is

$$\beta_{\tilde{g}} = |(\delta_1 + \delta_2 + \delta_1\delta_2) + \delta_1/x| \leq \frac{3\varepsilon_{\text{machine}} + \varepsilon_{\text{machine}}^2}{\alpha} \in O(\varepsilon_{\text{machine}}).$$

Therefore, this algorithm is backward stable if x is restricted to lie in (α, ∞).

11.3.4 Numerical Stability

Backward stability is a strong condition. As shown in Example 11.3.12, even a very simple, fundamental algorithm might not be backward stable. But often we can get by with a weaker condition called *numerical stability*, often just called *stability*. For stability, we allow not only the input to be approximated but also the output. That is, rather than requiring the exact answer to nearly the right problem (backward stable), we require only nearly the right answer to nearly the right problem.

Definition 11.3.13. *An algorithm $\tilde{f}$ to compute f is* stable *if there exist positive constants C and D such that for all $\mathbf{x}$ in the domain of f and for any sufficiently small $\varepsilon_{\text{machine}}$ there is a $\boldsymbol{\delta}$ (depending on $\mathbf{x}$ and $\varepsilon_{\text{machine}}$) satisfying*

$$\frac{\|\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \leq C\varepsilon_{\text{machine}}$$

and

$$\frac{\|f(\mathbf{x}+\boldsymbol{\delta}) - \tilde{f}(\mathbf{x})\|}{\|f(\mathbf{x}+\boldsymbol{\delta})\|} \leq D\varepsilon_{\text{machine}}.$$

We write these conditions as $\frac{\|\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \in O(\varepsilon_{\text{machine}})$ and $\frac{\|f(\mathbf{x}+\boldsymbol{\delta}) - \tilde{f}(\mathbf{x})\|}{\|f(\mathbf{x}+\boldsymbol{\delta})\|} \in O(\varepsilon_{\text{machine}})$ as $\varepsilon_{\text{machine}} \to 0$.

Theorem 11.3.16, below, shows that if the relative condition number $\kappa(\mathbf{x})$ of a function f is bounded, then any backward stable algorithm is stable.

Example 11.3.14. Consider again the algorithm in Example 11.3.12. The backward error is well behaved away from $x = 0$ but is problematic near $x = 0$. For $|x| < \frac{1}{2}$ if we let $\delta = x(\delta_1 + \delta_2 + \delta_1\delta_2)$, then $\tilde{g}(x) = g(x + \delta) + \delta_1$, so

$$\frac{|g(x+\delta) - \tilde{g}(x)|}{|g(x+\delta)|} = \frac{|\delta_1|}{|x+1+\delta|} \leq \frac{\varepsilon_{\text{machine}}}{\frac{1}{2}+\delta} \in O(\varepsilon_{\text{machine}}),$$

and $|\delta|/|x| \in O(\varepsilon_{\text{machine}})$, so $\tilde{g}$ is stable in the open interval $|x| < \frac{1}{2}$. Moreover, the algorithm $\tilde{g}$ is stable when $|x| \geq \frac{1}{2}$ because it is backward stable there. Therefore $\tilde{g}$ is stable for all x.

Example 11.3.15. For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, computing the outer product $f(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}^\mathsf{T}$ using the obvious algorithm

$$\tilde{f}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \text{fl}(x_1) \otimes \text{fl}(y_1) & \text{fl}(x_1) \otimes \text{fl}(y_2) & \cdots & \text{fl}(x_1) \otimes \text{fl}(y_n) \\ \text{fl}(x_2) \otimes \text{fl}(y_1) & \text{fl}(x_2) \otimes \text{fl}(y_2) & \cdots & \text{fl}(x_2) \otimes \text{fl}(y_n) \\ \vdots & \vdots & & \vdots \\ \text{fl}(x_n) \otimes \text{fl}(y_1) & \text{fl}(x_n) \otimes \text{fl}(y_2) & \cdots & \text{fl}(x_n) \otimes \text{fl}(y_n) \end{bmatrix}$$

is stable, but not backward stable. If the algorithm were backward stable, there would be a small $\boldsymbol{\delta} = (\boldsymbol{\delta}_1, \boldsymbol{\delta}_2) \in \mathbb{R}^{2n}$ such that $\tilde{f}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} + \boldsymbol{\delta}_1)(\mathbf{y} + \boldsymbol{\delta}_2)^\mathsf{T}$, but the rank of $(\mathbf{x} + \boldsymbol{\delta}_1)(\mathbf{y} + \boldsymbol{\delta}_2)^\mathsf{T}$ is always 1, regardless of the value of $\mathbf{x}, \mathbf{y}, \boldsymbol{\delta}_1, \boldsymbol{\delta}_2$, whereas the rank of the matrix $\tilde{f}(\mathbf{x}, \mathbf{y})$ is almost never 1. Therefore, $\tilde{f}$ cannot be backward stable. The proof that $\tilde{f}(\mathbf{x}, \mathbf{y})$ is stable is Exercise 11.14.

11.3.5 *Conditioning and Stability

Roughly speaking, one can expect that the condition number κ, the forward error, and the backward error are approximately related by

$$\text{relative forward error} \leq \kappa \times \text{relative backward error}$$

if they are all computed using the same norm. So an algorithm with a small backward error for solving a well-conditioned problem also has a small forward error, but if the problem is ill conditioned, the forward error could be large, even when the backward error is small.

The following theorem makes this relationship more precise.

Theorem 11.3.16. *Suppose the relative condition number of a function f at $\mathbf{x}$ is $\kappa(\mathbf{x})$ and that an algorithm $\tilde{f}$ for f is backward stable. Then the relative forward error of $\tilde{f}$ satisfies*

$$\frac{\|\tilde{f}(\mathbf{x}) - f(\mathbf{x})\|}{\|f(\mathbf{x})\|} \in O(\kappa(\mathbf{x})\varepsilon_{\text{machine}}). \tag{11.14}$$

Proof. By the definition of $\kappa(\mathbf{x})$ we have

$$\kappa = \lim_{h \to 0^+} \sup_{\|\boldsymbol{\delta}\| \leq h} \left(\frac{\|f(\mathbf{x}+\boldsymbol{\delta}) - f(\mathbf{x})\|\|\mathbf{x}\|}{\|f(\mathbf{x})\|\|\boldsymbol{\delta}\|} \right).$$

By backward stability, for each $\varepsilon_{\text{machine}}$ there is a $\boldsymbol{\delta}$ such that

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}+\boldsymbol{\delta}) \qquad \text{and} \qquad \frac{\|\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \in O(\varepsilon_{\text{machine}}).$$

Thus as $\varepsilon_{\text{machine}} \to 0$ we have

$$\frac{\|\tilde{f}(\mathbf{x}) - f(\mathbf{x})\|}{\|f(\mathbf{x})\|} = \frac{\|f(\mathbf{x}+\boldsymbol{\delta}) - f(\mathbf{x})\|}{\|f(\mathbf{x})\|} \leq (\kappa(\mathbf{x}) + o(1)) \frac{\|\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \in O(\kappa(\mathbf{x})\varepsilon_{\text{machine}}),$$

where $o(1)$ indicates a quantity that converges to zero as $\varepsilon_{\text{machine}} \to 0$. □

The theorem indicates that the accuracy of a backward stable algorithm is as good as the precision of the arithmetic system and the conditioning of the problem allow.

11.4 Computing Derivatives

Many applications, including optimization, which is a focus of the rest of this book, require approximating first or second derivatives of functions. In this section we discuss some of the main methods for computing derivatives. These include symbolic differentiation (done either by hand or with the help of a computer algebra system), finite differences, complex step differentiation, and algorithmic (or automatic) differentiation.

11.4.1 Symbolic Differentiation

In some situations, one can compute a derivative symbolically, that is, analytically compute a closed-form expression for the derivative, and then use the resulting expression to compute the value of the derivative. Although some simpler expressions can be calculated by hand, computer algebra systems with symbolic differentiation tools, like those in the Python module SymPy, are usually needed to compute more complicated derivatives symbolically.

Example 11.4.1. The symbolic method works very well for functions with known, simple, closed-form expressions for their derivatives. For example, to compute $\frac{d}{dx}\cos(x)$ at $x = 1$, we can use the standard formula from calculus $\frac{d}{dx}\cos(x) = -\sin(x)$. Evaluating at $x = 1$ gives -0.8414709848078965.

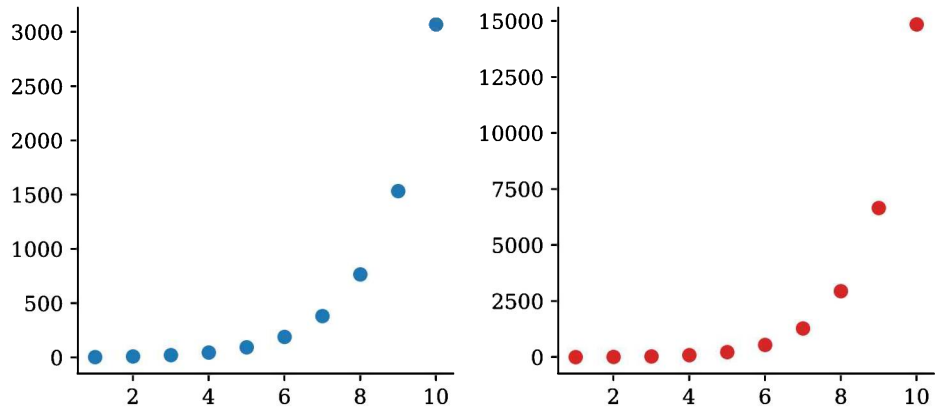


Figure 11.3. *Plot of the number of terms in the expansion of the function $h_k(x)$ (left panel, blue) of Example 11.4.2 and the symbolic derivative $\frac{d}{dx}h_k(x)$ (right panel, red) as functions of k. This rapid growth is an example of expression swell, which can make it costly to compute symbolic derivatives.*

In some applications, it is necessary to compute the derivative of functions that are compositions of many simpler functions. Such functions can be evaluated rapidly at any particular input value, but symbolic differentiation usually involves expanding out the compositions and differentiating. The number of terms in these expansions and their derivatives can rapidly grow to be very large. This growth is called *expression swell*, and it can make the computation of a symbolic derivative prohibitively expensive.

Example 11.4.2. Let $f(x) = x - 7x^2$, and consider the function

$$h_k(x) = \underbrace{(f \circ f \circ \cdots \circ f)}_{k}(x)$$

for various values of $k \in \mathbb{Z}^+$. Here we expand out the composition and compute the derivative symbolically for the cases of $k = 2$ and $k = 3$:

$$\begin{aligned}
h_2(x) &= -7x^2 + x - 7(-7x^2 + x)^2, \\
h_2'(x) &= -14x + 1 - 7(-28x + 2)(-7x^2 + x), \\
h_3(x) &= -7x^2 + x - 7(-7x^2 + x)^2 - 7(-7x^2 + x - 7(-7x^2 + x)^2)^2, \\
h_3'(x) &= -14x + 1 - 7(-28x + 2)(-7x^2 + x) \\
&\quad - 7(-28x - 14(-28x + 2)(-7x^2 + x) + 2)(-7x^2 + x - 7(-7x^2 + x)^2).
\end{aligned}$$

In Figure 11.3 we plot the number of terms in the expansion of h_k and the derivative of h_k for k up to 10.

Of course, if a function is not defined by a standard, closed-form formula but rather by the output of a more complicated algorithm or some opaque process,

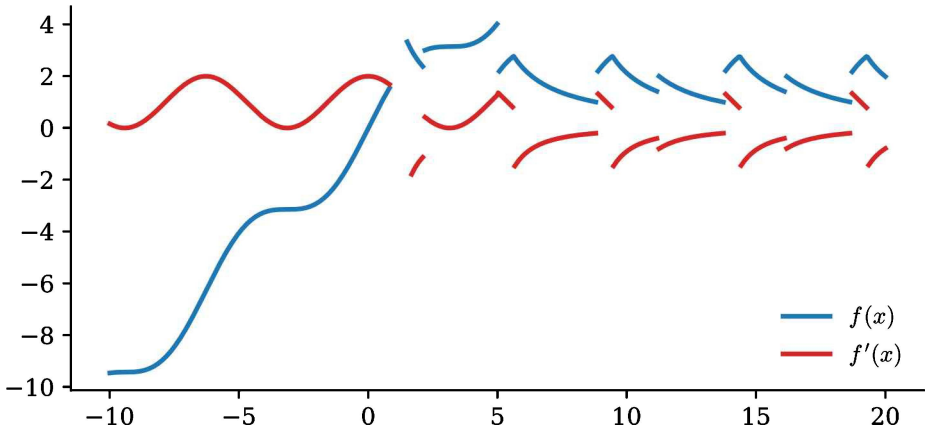


Figure 11.4. *Plot of the algorithmic function f (blue) defined in Unexample 11.4.3 and its derivative (red). Because this function does not have a simple closed form, its derivative is not easy to compute symbolically. But it can be approximated with numerical difference quotients or, more efficiently and accurately, with algorithmic differentiation. See Section 11.4.4 for more about algorithmic differentiation.*

like the result of a physical measurement, then the derivative usually cannot be computed symbolically.

Unexample 11.4.3. Functions without a simple, closed-form expression, like the following Python function, are not well suited to symbolic differentiation.

```
import numpy as np

def f(x):
    z = max(x,1.5)
    while x>5:
        z = int(np.round(np.log(np.abs(x) + 1),0))
        for k in range(z):
            x -= z/2 + np.sin(z)
    if 3*x + 2 < np.exp(z):
            return np.sin(x) + x
    else:
        return 5/x
```

See Figure 11.4 for a plot of this function and its derivative.

11.4.2 Numerical Difference Quotients

In many situations we may only have access to the values of the function as the output of a complicated subroutine or other "black box," so the best we can hope

for is an approximation to the first and second derivatives using specific values of the function. In this case one strategy is to use the difference quotient

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

for small values of h. When $h > 0$, this is called a *forward difference*. In the absence of round-off error, the forward difference would approximate the derivative more and more closely as $h \to 0^+$, and the size of the error in this estimate would lie in $O(h)$. Specifically, if $f \in C^2((a,b);\mathbb{R})$ for some neighborhood (a,b) around x_0 and $|f''(x)| < L$ on (a,b) for some constant L, then Taylor's theorem (Theorem 10.3.7) gives

$$|f(x_0+h) - f(x_0) - hf'(x_0)| \leq \frac{1}{2}h^2 L,$$

so the error from using the forward difference $\frac{f(x_0+h)-f(x_0)}{h}$ to compute $f'(x_0)$ is bounded by

$$\left|\frac{f(x_0+h) - f(x_0)}{h} - f'(x_0)\right| \leq \frac{1}{2}hL \in O(h).$$

The quality of the approximation can be improved to $O(h^2)$ by taking a *centered difference*

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{11.15}$$

If $f \in C^3$ with $|f^{(3)}(x)| < M$ near x_0, Taylor's theorem (Theorem 10.3.7) gives

$$|f(x_0+h) - f(x_0) - hf'(x_0) - \frac{h^2}{2}f''(x_0)| \leq \frac{Mh^3}{3!}. \tag{11.16}$$

Applying (11.16) twice, once with h and once with $-h$, gives

$$\begin{aligned}
&\left|2hf'(x_0) - (f(x_0+h) - f(x_0-h))\right| && (11.17)\\
&\quad= \left|2hf'(x_0) - \left(f(x_0+h) - f(x_0) - \frac{h^2}{2}f''(x_0)\right)\right. && (11.18)\\
&\qquad\qquad \left. - \left(f(x_0-h) - f(x_0) - \frac{h^2}{2}f''(x_0)\right)\right| \\
&\quad\leq \left|hf'(x_0) - \left(f(x_0+h) - f(x_0) - \frac{h^2}{2}f''(x_0)\right)\right| && (11.19)\\
&\qquad\qquad + \left|hf'(x_0) - \left(f(x_0-h) - f(x_0) - \frac{h^2}{2}f''(x_0)\right)\right| \\
&\quad\leq \frac{2Mh^3}{3!}.
\end{aligned}$$

Dividing this inequality by h gives (11.15).

Unfortunately, approximations of the derivative using difference quotients in floating-point arithmetic usually develop large errors when h is too small. One example of this is given in Example 11.1.9. Therefore, minimizing the total error requires finding a value of h that keeps

$$\left|f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h}\right|$$

small enough but that also avoids making the round-off error too large. The optimal answer depends on which difference quotient method is being used and the size of the relative forward error in f, as well as $\varepsilon_{\text{machine}}$. As a rough ballpark estimate, if ϱ is a close upper bound on the relative forward error in f for all x near x_0, then

$$h \approx 2\sqrt{\varrho}$$

is a reasonable choice for the step size with forward differences. For more on the choice of step size, see Section 11.4.5.

11.4.3 Complex Step Differentiation

Computing derivatives by numerical differences is fundamentally limited in its accuracy by the trade-off between the need to make h small to get a reasonable approximation and the need to keep h large enough to avoid loss of precision from subtraction. But if the function f to be differentiated is analytic (its Taylor series converges at every point)[46] in a small neighborhood of the point x_0 in $\mathbb{C}$, then we can compute a numerical derivative $f'(x_0)$ at x_0 without subtraction by using the following proposition.

Proposition 11.4.4. *If $x_0 \in \mathbb{R}$ with f analytic in a small neighborhood of x_0 in $\mathbb{C}$, and $f(x)$ is real for all x in a small real interval around x_0, then for sufficiently small real $h > 0$ we have*

$$f'(x_0) = \Im\left(\frac{f(x_0 + ih)}{h}\right) + O(h^2), \tag{11.20}$$

where $\Im(z)$ is the imaginary part of z.

Proof. If f is analytic near x_0, then there exists $\delta > 0$ such that $f \in C^\infty(B(x_0, \delta); \mathbb{C})$ on $B(x_0, \delta)$ and there exists $M > 0$ with $|f^{(3)}(x)| \leq M$ for all $x \in B(x_0, \delta)$. For $0 < h < \delta$ Taylor's theorem gives

$$\left| f(x_0 + ih) - f(x_0) - ihf'(x_0) + \frac{h^2}{2!} f^{(2)}(x_0) \right| \leq \frac{Mh^3}{3!}.$$

Taking the imaginary part shows that for all real h with $|h| < \delta$ we have

$$|\Im(f(x_0 + ih)) - hf'(x_0)| \leq \frac{Mh^3}{3!}.$$

Dividing by h gives (11.20). □

This shows that to approximate $f'(x_0)$ we may compute $\frac{1}{h}\Im(f(x_0 + ih))$. This is called *complex step differentiation*. Since this method does not involve a subtraction in the numerator, it is resistant to the sorts of errors that arise in the numerical differences methods. Since most numerical computing libraries handle complex computations seamlessly, this gives a practical method for numerically computing many derivatives accurately. This can be a very robust way to accurately approximate the derivative, but it does require the function f to be defined as a composition of analytic functions.

[46] For more on analytic functions, see Volume 1, Chapter 11.

Example 11.4.5. Applying complex step differentiation to compute $\cos(1)$ behaves much better than the forward difference method in Example 11.1.9. In Python with the default double-precision arithmetic, calculating $\Im\left(\frac{\cos(1+2^{-n}i}{2^{-n}}\right)$ for $n \in \{20, \ldots, 500\}$ yields the following:

```
20 -0.8414709848080241      27 -0.8414709848078965
21 -0.8414709848079285      28 -0.8414709848078965
22 -0.8414709848079045      29 -0.8414709848078965
23 -0.8414709848078986      30 -0.8414709848078965
24 -0.8414709848078971      31 -0.8414709848078965
25 -0.8414709848078967      32 -0.8414709848078965
26 -0.8414709848078965      33 -0.8414709848078965
```

So this computes the correct answer, to machine precision, by $n = 26$ (corresponding to $h = 2^{-26}$), and then continues to give that same, correct answer for all n up to 500—it never degrades the way that the forward differences did in Example 11.1.9.

Remark 11.4.6. If the operations involved in computing f use complex numbers other than those in the argument, then round-off error can affect the imaginary parts of this computation and degrade the result.

11.4.4 Brief Overview of Algorithmic Differentiation

Algorithmic differentiation is a powerful method for computing derivatives. The method called *back propagation*, which is one of the fundamental tools for training neural networks, is a special case of algorithmic differentiation.

The rough idea is to define a class of primitive functions (for example, linear operators, arithmetic operations, trigonometric functions, and so forth) with known derivatives, and then construct derivatives of arbitrary compositions of these functions using the chain rule. These methods begin by explicitly coding up algorithms for computing the derivative of each primitive function. These derivatives are usually calculated by a formula (like $\frac{d}{dx}\sin(x) = \cos(x)$), instead of numerically. This helps prevent the loss of precision that typically occurs in numerical difference quotients.

Having the derivatives of the primitive functions coded explicitly means that for any primitive function b and any valid input $\mathbf{x}$ we can rapidly compute both $b(\mathbf{x})$ and $Db(\mathbf{x})$. For any function f that is built from a composition of the primitive functions $f = b_1 \circ b_2 \circ \cdots \circ b_k$, the chain rule gives

$$Df(\mathbf{x}) = Db_1(b_2 \circ \cdots \circ b_k(\mathbf{x}))Db_2(b_3 \circ \cdots \circ b_k(\mathbf{x})) \cdots Db_k(\mathbf{x}). \tag{11.21}$$

Thus, $Df(\mathbf{x})$ can be computed by first computing $b_k(\mathbf{x})$ and $Db_k(\mathbf{x})$, then computing $b_{k-1}(b_k(\mathbf{x}))$ and $Db_{k-1}((b_k(\mathbf{x})))$, and so forth until computing $Db_1(b_2 \circ \cdots \circ b_k(\mathbf{x}))$, and then multiplying together all the terms appearing in (11.21). Moreover, these matrix multiplications can be performed relatively efficiently by judicious choices of the order of multiplication.

The idea of algorithmic differentiation generalizes to more complicated compositions of functions (for example, to functions of the form $b_1(b_2(\mathbf{x}), b_3(\mathbf{x}))$) by carefully tracking the effect of the chain rule through all the compositions. This allows for efficient differentiation of very complicated functions, including functions whose definition involves loops, conditional statements, and other computer code, provided the actual computations themselves consist of compositions of the primitive functions.

Example 11.4.7. The Python function defined in Unexample 11.4.3 can be algorithmically differentiated using the `autograd` module. This module adds explicitly coded derivatives to most NumPy functions and then differentiates any function constructed from those functions by repeatedly applying the chain rule.

```
import autograd.numpy as np    # NumPy with derivatives
from autograd import grad      # Algorithmic differentiator

def f(x):                      # Same function as before
    z = max(x,1.5)
    while x>5:
        z = int(np.round(np.log(np.abs(x) + 1),0))
        for k in range(z):
            x -= z/2 + np.sin(z)
    if 3*x + 2 < np.exp(z):
            return np.sin(x) + x
    else:
        return 5/x

df = grad(f)                   # Derivative of f
```

See Figure 11.4 for a plot of this function and its derivative, computed with `autograd`.

11.4.5 *More on Forward and Centered Differences

In this subsection we discuss the temporal complexity of difference quotient methods and also the problem of choosing the optimal step size.

Temporal Complexity of Difference Quotients

To compute the approximate derivative $\widetilde{Df}(\mathbf{x}_0) = \begin{bmatrix}\widetilde{D_1 f}(\mathbf{x}_0) & \cdots & \widetilde{D_n f}(\mathbf{x}_0)\end{bmatrix}$ of a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ by one-sided difference quotients requires $n + 1$ function evaluations: $\tilde{f}(\mathbf{x}_0)$ and $\tilde{f}(\mathbf{x}_0 + h\mathbf{e}_i)$ for each i. The centered difference requires $2n$ function evaluations and is somewhat more accurate. The decision of whether to use the one-sided or centered difference will depend heavily on the nature

of the problem, the level of accuracy needed, and the overall cost of the additional $n-1$ function evaluations.

To approximate the Hessian $D^2 f(\mathbf{x}_0)$ when the gradient $Df(\mathbf{x})$ is available, we can use the same techniques that we used to calculate the gradient from the original function. This requires $n+1$ evaluations of the gradient.

If the gradient is not available, then we can use the approximation

$$D_{ij} f(\mathbf{x}_0) \approx \frac{f(\mathbf{x}_0 + h\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x}_0 + h\mathbf{e}_i) - f(\mathbf{x}_0 + h\mathbf{e}_j) + f(\mathbf{x}_0)}{h^2},$$

but this, of course, suffers from an additional loss of accuracy and requires $O(n^2)$ evaluations of f. The error on this approximation can be computed in a manner similar to the first derivative, using the higher-order Taylor approximation.

Optimal Step Size

Let $\widetilde{f}'(x_0)$ denote the computed value of $f'(x_0)$, using the centered difference with step size h. The total error $|f'(x_0) - \widetilde{f}'(x_0)|$ can be split into two parts:

$$\begin{aligned} |f'(x_0) - \widetilde{f}'(x_0)| \le &\left| f'(x_0) - \frac{f(x_0+h) - f(x_0-h)}{2h} \right| \\ &+ \left| \frac{f(x_0+h) - f(x_0-h)}{2h} - \widetilde{f}'(x_0) \right|. \end{aligned} \tag{11.22}$$

In the case of centered differences, (11.17) shows that the term

$$\left| f'(x_0) - \frac{f(x_0+h) - f(x_0-h)}{2h} \right|$$

on the right side of (11.22) is bounded by $\frac{h^2 M}{3!}$, where $|f^{(3)}(x)| \le M$ for all x near x_0. To bound the second term, assume that $|f(x)| \le C$ for all x near x_0 and that ϱ is an upper bound on the relative forward error for $f(x)$ near x_0, so that $|f(x) - \tilde{f}(x)| \le C\varrho$ for all x near x_0. We write

$$\widetilde{f}'(x_0) = \Big(\tilde{f}(x_0+h) \ominus \tilde{f}(x_0-h) \Big) \oslash 2h.$$

This gives

$$\begin{aligned} &\left| \frac{f(x_0+h) - f(x_0-h)}{2h} - \widetilde{f}'(x_0) \right| \\ &\quad = \left| \frac{f(x_0+h) - f(x_0-h)}{2h} - \Big(\Big(\tilde{f}(x_0+h) \ominus \tilde{f}(x_0-h) \Big) \oslash 2h \Big) \right| \\ &\quad \le \left| \frac{f(x_0+h) - f(x_0-h)}{2h} - \left(\frac{\tilde{f}(x_0+h) \ominus \tilde{f}(x_0-h)}{2h} \right) \right| + \varepsilon_{\text{machine}} \\ &\quad = \frac{2C\varrho + \varepsilon_{\text{machine}}}{2h} + \varepsilon_{\text{machine}}. \end{aligned}$$

Thus the total error using this method with step size h is bounded by

$$|f'(x_0) - \widetilde{f}'(x_0)| \le \frac{h^2 M}{3!} + \frac{2C\varrho + \varepsilon_{\text{machine}}}{2h} + \varepsilon_{\text{machine}}.$$

Differentiating shows that this bound is minimized at

$$h_* = \sqrt[3]{\frac{3}{M}(C\varrho + \frac{1}{2}\varepsilon_{\text{machine}})}.$$

If $\varrho \gg \varepsilon_{\text{machine}}$ (meaning that ϱ is much larger than $\varepsilon_{\text{machine}}$), then this is approximately

$$h_* \approx \sqrt[3]{\frac{3C}{M}\varrho} \approx 1.4\sqrt[3]{\varrho},$$

provided $C \approx M \gg \varepsilon_{\text{machine}}$.

A similar analysis shows that a good choice of h for forward differences is

$$h_+ \approx \sqrt{\frac{2C}{L}(2\varrho + \varepsilon_{\text{machine}})},$$

where $|f^{(2)}(x)| \leq L$ for all x near x_0. If $C \approx L \gg \varepsilon_{\text{machine}}$, then h_+ is approximately given by

$$h_+ \approx 2\sqrt{\varrho},$$

which is a common choice for approximating derivatives with a forward difference.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

11.1. Express the following numbers in IEEE 754 floating point:

(i) 7.

(ii) 77/64.

(iii) 22/7.

11.2. Not every integer can be represented as a floating-point number. Give a formula in terms of b and p for the smallest positive integer that cannot be represented in the system $\mathbf{F}$ of Section 11.1.2. What is the smallest positive

integer n that cannot be represented as an IEEE 754 double? Write and execute a program to verify that $n-3$, $n-2$, and $n-1$ are all represented as floating-point numbers on your computer, but n is not. What about $n+1, n+2, n+3, \ldots$?

11.3. Floating-point arithmetic does not behave the way regular arithmetic does. Many of the "laws" of regular arithmetic fail to hold. Here are a few:

(i) The set $\mathbf{F}$ is not closed under (true) addition. Give an example of two numbers $x, y \in \mathbf{F}$ such that $x + y$ is in the representable range but $x+y \notin \mathbf{F}$. Explain. For your example, evaluate $x \oplus y$ on your computer. What is the relative error $\left|\frac{(x+y)-(x\oplus y)}{x+y}\right|$?

(ii) The set $\mathbf{F}$ is not closed under (true) division. Give an example of a number in $\mathbf{F}$ such that $1/x$ is in the representable range but $1/x \notin \mathbf{F}$. Explain. For your example, what is $1 \oslash x$? What is the relative error $\left|\frac{(1/x)-(1\oslash x)}{1/x}\right|$?

(iii) Floating-point addition is not associative. Find floating-point numbers x, y, and z such that $(x \oplus y) \oplus z \neq x \oplus (y \oplus z)$. Explain.

(iv) Floating-point arithmetic does not satisfy the distributive law. Find floating-point numbers x, y, and z such that $(x \oplus y) \otimes z \neq (x \otimes z) \oplus (y \otimes z)$. Explain.

11.4. Consider the series $\sum_{n=1}^{10^7} 1000/n$.

(i) Compare the result when you sum the series forward, from 1 to 10^7, with the result when you sum the series backward, from 10^7 down to 1.

(ii) Floating-point addition is commutative, so why does changing the order of this summation change the result?

11.5. Plot the function $f(x) = (1-x)-1$ and the function x at a thousand points along the interval $(-3 \times 10^{-15}, +3 \times 10^{-15})$. Note that $(1-x)-1 = -x$ in exact arithmetic. Explain why the graph of x looks smooth but the graph of $(1-x)-1$ is jagged. Now plot the ratio $((1-x)-1)/x$. Why does the error (the distance from the correct value of -1) get larger as $x \to 0$?

11.6. Find the relative condition number at $x_0 \in \mathbb{R}$ of the following functions:

(i) e^x.

(ii) $\ln(x)$.

(iii) $\cos(x)$.

(iv) $\tan(x)$.

11.7. Given $(x, y) \in \mathbb{C}^2$, consider the problem of finding z such that $x^2+y^2-z^3+z = 0$. Find all the points (x, y) for which z is locally a function of (x, y). For a fixed value of y, find the relative condition number of z as a function of x. What is this relative condition number near the point $(x, y) = (0, 0)$?

11.8. Give an example of a matrix A with condition number $\kappa(A) > 1000$ (assuming the 2-norm). Give an example of a matrix B with condition number $\kappa(B) = 1$. Are there any matrices with condition number less than 1? If so, give an example. If not, prove it.

11.9. Let $\{g_0(x), \ldots, g_n(x)\}$ be a basis of the space $\mathbb{F}[x; n]$ of polynomials of degree no more than n, with $\deg(g_i) = i$. Let $P(\mathbf{a}, x) = \sum_{i=0}^{n} a_i g_i(x)$ be a polynomial expressed in this basis. Just as in the case of Proposition 11.2.12, for any given $\mathbf{a}^* \in \mathbb{F}^{n+1}$ and any simple root x^* of the polynomial $p(x) = P(\mathbf{a}^*, x)$, there is a neighborhood U of $\mathbf{a}^*$ in $\mathbb{F}^{n+1}$ and a continuously differentiable function $r : U \to \mathbb{F}$ with $r(\mathbf{a}^*) = x^*$ such that $P(\mathbf{a}, r(\mathbf{a})) = 0$ for all $\mathbf{a} \in U$.

(i) Show that the relative condition number of r as a function of the ith coefficient a_i at the point $(\mathbf{a}^*, x^*)$ is

$$\kappa(x^*) = \left| \frac{a_i^* g_i(x^*)}{z p'(x^*)} \right|. \tag{11.23}$$

(ii) Prove that for any nonzero scalar $c \in \mathbb{F}$, the condition number (11.23) remains the same if any basis element g_i is replaced by cg_i.

(iii) For a given $p(x)$ with simple root x^*, prove that choosing a monic basis $\{g_0, \ldots, g_n\}$ that minimizes the condition number (11.23) is equivalent to choosing a monic basis that minimizes the numerator $|a_i^* g_i(z)|$.

(iv) Recall that the minimax theorem (Theorem 9.4.4) guarantees that the monic Chebyshev basis ($g_i(x) = \hat{T}_i(x)$ for every i, where $\hat{T}_i$ is the monic Chebyshev polynomial of degree i) minimizes $\sup_{z \in [-1,1]} g_i(z)$ among all monic bases. This does not guarantee that the Chebyshev basis necessarily minimizes the numerator of (11.23) for each i; nevertheless, it suggests that the Chebyshev basis should be among the better-conditioned choices of basis for real roots in $[-1, 1]$. Prove that if $x^* \in (-1, 1)$ is real, then the condition number of the root x^* as a function of the coefficients of the Chebyshev basis is

$$\kappa(x^*) \leq \left| \frac{a_i}{2^{i-1} x^* p'(x^*)} \right|.$$

11.10. Prove that floating-point subtraction is backward stable, as a function from $\mathbb{R}^2$ to $\mathbb{R}$.

11.11. Any reasonable algorithm involves more than one single operation. Let

$$\gamma_k = \frac{k\varepsilon_{\text{machine}}}{1 - k\varepsilon_{\text{machine}}}.$$

Prove the following result, which shows how errors from multiple operations combine. If $|\delta| \leq \gamma_k$ and $|\phi| \leq \gamma_j$, then $(1 + \delta)(1 + \phi) = (1 + \xi)$, where $|\xi| \leq \gamma_{k+j}$.

11.12. Find a good upper bound on the relative forward error that results when computing $(a + b)/(c + d)$ for $a, b, c, d \in \mathbf{F}$.

11.13. For each of the following, decide whether the algorithm is backward stable, stable but not backward stable, or unstable. Sketch a proof.

(i) For $x \in \mathbb{R}$ the algorithm $\tilde{k}(x) = \text{fl}(x) \otimes \text{fl}(x) \otimes \cdots \otimes \text{fl}(x)$ for computing $k(x) = x^n$.

(ii) For $x \in \mathbb{R} \setminus \{0\}$ the algorithm $\tilde{u}(x) = \text{fl}(x) \oslash \text{fl}(x)$ for computing the value of $u(x) = 1$.

(iii) The algorithm for computing e as $\tilde{e} = (1 \oplus 1 \oslash n)^n$ with n some fixed large value, and where the nth power is computed with the algorithm in (i).

(iv) The previous algorithm for $\tilde{e}$, but with $n = \mathrm{fl}(k/\varepsilon_{\text{machine}})$ for some fixed $k > 0$.

(v) The algorithm for computing e as $\hat{e} = \sum_{k=0}^{\infty} \frac{1}{k!}$, where all the operations are the usual floating-point operations and the sum is added starting from $k = 1$ upward, and terminated once $\frac{1}{k!} < \varepsilon_{\text{machine}}$.

(vi) The algorithm for computing e as $\bar{e} = \sum_{k=0}^{\infty} \frac{1}{k!}$, where all the operations are the usual floating-point operations and the sum is added backward (starting from the smallest positive integer k such that $1/k! < \varepsilon_{\text{machine}}$ and proceeding down to $k = 1$).

11.14.* Prove that the algorithm given in Example 11.3.15 for computing the outer product is stable.

11.15. Show that, in the absence of round-off error, if $f \in C^4((a,b);\mathbb{F})$ in a neighborhood of $x_0 \in (a,b)$, then for h sufficiently close to 0 we have

$$f'(x) = \frac{2f(x+h) + 3f(x) - 6f(x-h) + f(x-2h)}{6h} + O(h^4).$$

11.16. Computing gradients with forward differences:

(i) Write code for computing the gradient Df of a function $f : \mathbb{R}^n \to \mathbb{R}$ using forward differences and a given step size. Your code should accept an integer $n > 0$, a callable function $f : \mathbb{R}^n \to \mathbb{R}$, a point $\mathbf{x} \in \mathbb{R}^n$, and a step size h, which should default to $h = 2\sqrt{\varepsilon_{\text{machine}}}$. Your code should return an estimate for $Df(\mathbf{x})$ computed using forward differences.

(ii) Compute the gradient of $f(x,y) = (\frac{x}{x+2+y^2}, \frac{y}{x^2+y^2})$ symbolically and evaluate the symbolic expression numerically at the point $(2,3)$.

(iii) For each value $h = 2^{-k}$ with $k \in \{2, 3, \ldots, 53\}$, use your code to compute the gradient of $f(x,y) = (\frac{x}{x+2+y^2}, \frac{y}{x^2+y^2})$ at the point $(2,3)$.

(iv) Time each of the previous computations, and identify the value of h that gives the optimal accuracy.

11.17. Computing gradients with centered differences:

(i) Write code for computing the gradient Df of a function $f : \mathbb{R}^n \to \mathbb{R}$ using centered differences and a given step size. Your code should accept an integer $n > 0$, a callable function $f : \mathbb{R}^n \to \mathbb{R}$, a point $\mathbf{x} \in \mathbb{R}^n$, and a step size h, which should default to $h = 1.4\sqrt[3]{\varepsilon_{\text{machine}}}$. Your code should return an estimate for $Df(\mathbf{x})$ computed using centered differences.

(ii) For each value $h = 2^{-k}$ with $k \in \{2, 3, \ldots, 53\}$, use your code to compute the gradient of $f(x,y) = (\frac{x}{x+2+y^2}, \frac{y}{x^2+y^2})$ at the point $(2,3)$.

(iii) Time each of the previous computations, and identify the value of h that gives the optimal accuracy. Compare your results to those of the previous problem.

11.18. Compute the derivative of $f(x) = \frac{\sin^3(x)+\cos(x)}{e^x}$ at $x_0 = 1.5$ in each of the following ways. For the difference quotients and complex step methods, compute with the value of $h = 2^{-n}$ for each $n \in \{1, \ldots, 53\}$. Compare the convergence rate, overall accuracy, and computation time for each method.

(i) Compute the derivative $f'(x)$ symbolically (either by hand or with a computer algebra system) and evaluate at x_0.

(ii) Compute the forward difference $\frac{f(x_0+h)-f(x_0)}{h}$.

(iii) Compute the centered difference $\frac{f(x_0+h)-f(x_0-h)}{2h}$.

(iv) Compute the complex step approximation $\frac{\Im(f(x_0+ih))}{h}$.

(v) Use an algorithmic differentiation package to compute $f'(x_0)$.

11.19. Let $\mathrm{ReLu} : \mathbb{R} \to \mathbb{R}$ be the function[47]

$$\mathrm{ReLu}(x) = \max(0, x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

(i) Symbolically compute the derivative with respect to $\mathbf{w}$ of $\mathrm{ReLu}(\mathbf{w}^\mathsf{T}\mathbf{x}+b)$.

(ii) Consider the function $F : \mathbb{R}^2 \times (\mathbb{R}^2)^3 \to \mathbb{R}^2$ given by

$$F(\mathbf{x}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{b}_1) = (\mathrm{ReLu}(\mathbf{w}_1^\mathsf{T}\mathbf{x} + b_{11}), \mathrm{ReLu}(\mathbf{w}_2^\mathsf{T}\mathbf{x} + b_{12})).$$

The composition

$$N_1(\mathbf{x}, \mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, b_0, \mathbf{b}_1) = \mathrm{ReLu}(\mathbf{w}_0^\mathsf{T} F(\mathbf{x}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{b}_1) + b_0)$$

corresponds to a very simple single-layer neural network, while

$$\begin{aligned} N_2(\mathbf{x}, W, B) &= N_1(F(\mathbf{x}, \mathbf{w}_3, \mathbf{w}_4, \mathbf{b}_2), \mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, b_0, \mathbf{b}_1) \\ &= \mathrm{ReLu}(\mathbf{w}_0^\mathsf{T} F(F(\mathbf{x}, \mathbf{w}_3, \mathbf{w}_4, \mathbf{b}_2), \mathbf{w}_1, \mathbf{w}_2, \mathbf{b}_1) + b_0) \end{aligned}$$

corresponds to a two-layer neural network (here $W = (\mathbf{w}_0, \ldots, \mathbf{w}_4)$, and $B = (b_0, \mathbf{b}_1, \mathbf{b}_2)$). Symbolically compute the partial derivative with respect to w_{11} of N_1 (here $\mathbf{w}_1 = (w_{11}, w_{12})$).

(iii) Repeat the following 10 times:

(a) Draw each of $x_1, x_2, w_{01}, w_{02}, \ldots, w_{42}, b_0, b_{10}, \ldots, b_{22}$ from a standard normal $\mathcal{N}(0, 1)$.

(b) Use an algorithmic differentiation package to compute the partial derivative with respect to w_{11} at the point $\mathbf{x}$, $\mathbf{w}_0, \ldots, \mathbf{w}_4$, $b_0, \mathbf{b}_1, \mathbf{b}_2$ of the simple one- and two-layer neural networks N_1 and N_2, respectively.

(c) Evaluate your symbolic solution at the chosen values for $\mathbf{x}$, $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2$, b_0, and $\mathbf{b}_1$ and compare to the solution for $k = 1$ found by algorithmic differentiation.

[47] The name ReLu is short for *rectified linear unit*, because a *rectifier* is an electrical device that lets current pass in the positive direction only. This function is a common component in deep neural networks.

Notes

Sources for floating-point arithmetic include [Dem97, Hig96, TB97, IEE08]. For more on Remark 11.1.7 about how machine epsilon changes in complex arithmetic, see [TB97, pg. 100]. Example 11.1.8 is inspired by a similar example given in [Coo14], and Exercise 11.2 is from [TB97, Exercise 13.2].

Our treatment of stability is inspired by [TB97] and [Dem97]. Exercise 11.5 is due to [Dem96] and Exercise 11.13 is based on [TB97, Example 15.1]. Our proof of Theorem 11.14 is based on [TB97, Section 15].

For more examples and discussion of higher-order difference methods for numerical differentiation see [LeV07].

We first learned of complex step differentiation from [Hig18], but it apparently originated with [ST98] and was foreshadowed by ideas from [LM67]. This method can also be applied to compute derivatives of matrix functions; see [AMH10].

Part IV

Optimization

12 Unconstrained Optimization

Premature optimization is the root of all evil.
—Don Knuth

Nearly every problem in the world is, or can be formulated as, an optimization problem. How do I maximize profits? What's the best way to study for the test? How can I minimize my risks? How can I carry out a task with the fewest resources? How can I do the most good? All of these are optimization problems.

Chapter 4 gives many examples of discrete optimization problems, including graph problems such as finding a minimum spanning tree and computing the shortest path between two vertices. In the remainder of the book we focus on optimization problems for which the function to be optimized (called the *objective function*) is a differentiable function of its inputs, and the algorithms for solving these optimization problems are derived from principles of differential calculus.

In single-variable calculus we find the minimum of a function by first locating all its critical points (that is, the points for which the derivative is zero or undefined) and all the boundary points for the domain in question. Critical points that fail the second-derivative test can be discarded, and then we evaluate the function at each of the remaining critical points and all boundary points and compare their values.

This chapter generalizes these ideas to multiple dimensions. We focus here on problems for which there are no imposed constraints or boundaries. These problems are called *unconstrained optimization problems*. In later chapters, we consider optimization problems where constraints are added to force the solution to have certain properties. The addition of constraints adds significant complexity to the problem, and so it takes several chapters to sort through all of the nuances involved. But in this chapter we avoid those challenges and focus only on problems that have no constraints (and therefore no boundaries).

12.1 Fundamentals of Unconstrained Optimization

Unconstrained optimization is the problem of finding the minimizer or maximizer of a function $f : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^n$ is an open set. In this section we give some basic definitions and discuss the *first-* and *second-order necessary conditions* and

the *second-order sufficient condition* for finding optimizers. These are fundamental tools of optimization.

12.1.1 Mathematical Descriptions

We begin with some basic definitions.

Definition 12.1.1. *Assume $\Omega \subset \mathbb{R}^n$ and let $f : \Omega \to \mathbb{R}$ be given. A point $\mathbf{x}^* \in \Omega$ is a* minimizer *of f on Ω if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \Omega$. In this case $f(\mathbf{x}^*)$ is called a* minimum value *of f on Ω or just a* minimum *of f. If there exists an open neighborhood U of $\mathbf{x}^*$ in $\mathbb{R}^n$ such that $\mathbf{x}^*$ is a minimizer on the set $U \cap \Omega$, then $\mathbf{x}^*$ is a* local minimizer *of f on Ω. A minimizer (or local minimizer) is* strict *if the corresponding inequality is strict for all $\mathbf{x} \neq \mathbf{x}^*$ in Ω (or in $U \cap \Omega$). If the set Ω is open in $\mathbb{R}^n$ and there are no other restrictions on the set of possible minimizers, then the problem of finding a minimizer is called an* unconstrained *optimization problem.*

Remark 12.1.2. If $\mathbf{x}^*$ is a minimizer on all of Ω it is common to call it a *global* minimizer of f on Ω to distinguish it from a local minimizer. Of course, a global minimizer is also a local minimizer.

Definition 12.1.3. *A point $\mathbf{x}^* \in \Omega$ is called an* optimizer *if it is either a minimizer or a maximizer of f. In optimization problems the function f to be optimized is called the* objective function. *In minimization problems the objective function is sometimes called the* cost function *or* loss function. *In maximization problems the objective function is sometimes called the* utility function *or* payoff function.

Definition 12.1.4. *Assume $\Omega \subset \mathbb{R}^n$ and let $f : \Omega \to \mathbb{R}$ be given. We denote the set of all minimizers of f on Ω by*

$$\operatorname*{argmin}_{\mathbf{x} \in \Omega} f(\mathbf{x}) = \{\mathbf{x} \in \Omega \mid f(\mathbf{x}) \leq f(\mathbf{y}) \quad \forall \mathbf{y} \in \Omega\}.$$

If the global minimizer $\mathbf{x}^*$ of f on Ω is unique, we often write

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x} \in \Omega} f(\mathbf{x}).$$

The argmax *is defined similarly.*

Remark 12.1.5. Maximization problems can easily be recast into minimization problems. In particular, observe that $\mathbf{x}^* \in \Omega$ is a local (resp., global) minimizer of f in Ω if and only if $\mathbf{x}^* \in \Omega$ is a local (resp., global) maximizer of $-f$. In order to simplify notation and our discussions, we always assume from now on that optimization problems are minimization problems, unless otherwise indicated.

Local versus Global Optimization

As with single-variable calculus, we solve global optimization problems by identifying all the local minimizers (and the boundary, if the problem is constrained), and then check to see which gives the smallest value of the objective function.

There are special classes of functions that have only one local minimizer, and thus the local minimizer is also the unique global minimizer. This occurs, for example, with convex objective functions, which are discussed in Chapter 15.

12.1.2 First-Order Necessary Condition

Differential calculus gives some of the most important tools for finding optimizers. In this section we discuss a necessary condition for a minimizer of a differentiable function. These can, of course, be easily modified to give necessary conditions for a maximizer.

Theorem 12.1.6 (First-Order Necessary Condition (FONC)). *Assume that $\Omega \subset \mathbb{R}^n$ is open and $f : \Omega \to \mathbb{R}$ is differentiable at $\mathbf{x}^* \in \Omega$. If $\mathbf{x}^*$ is a local minimizer of f on Ω, then $Df(\mathbf{x}^*) = \mathbf{0}^\mathsf{T}$.*

Proof. Suppose some local minimizer $\mathbf{x}^* \in \Omega$ satisfies $Df(\mathbf{x}^*) \neq \mathbf{0}^\mathsf{T}$. Thus the unit vector $\mathbf{v} = -\frac{Df(\mathbf{x}^*)^\mathsf{T}}{\|Df(\mathbf{x}^*)^\mathsf{T}\|}$ satisfies $Df(\mathbf{x}^*)\mathbf{v} = -\|Df(\mathbf{x}^*)\| < 0$ (assuming the usual Euclidean norm). Since $\mathbf{x}^*$ is a local minimizer, there exists a $\delta > 0$ such that $f(\mathbf{x}^* + t\mathbf{v}) \geq f(\mathbf{x}^*)$ whenever $0 < t < \delta$. Hence,

$$\begin{aligned}\frac{|f(\mathbf{x}^* + t\mathbf{v}) - f(\mathbf{x}^*) - tDf(\mathbf{x}^*)\mathbf{v}|}{\|t\mathbf{v}\|} &= \frac{f(\mathbf{x}^* + t\mathbf{v}) - f(\mathbf{x}^*) + t\|Df(\mathbf{x}^*)\|}{t} \\ &= \frac{f(\mathbf{x}^* + t\mathbf{v}) - f(\mathbf{x}^*)}{t} + \|Df(\mathbf{x}^*)\| \geq \|Df(\mathbf{x}^*)\|.\end{aligned}$$

This is a contradiction, since the left-hand side of the first equality converges to zero by the definition of the derivative; hence, $Df(\mathbf{x}^*) = \mathbf{0}$. □

Example 12.1.7. Consider the function $f(x, y) = x^3 - 3x^2 + y^2$. We compute the derivative $Df(x, y) = (3x^2 - 6x, 2y)$, which vanishes only at the points $(0, 0)$ and $(2, 0)$. We call these the *critical points* of f. Thus by the FONC, no other point $(x, y) \in \mathbb{R}^2$ can be a local minimizer of f. We do not yet know whether either of the points $(0, 0)$ or $(2, 0)$ are local minimizers; we just know that they are the only two feasible candidates.

Remark 12.1.8. While the FONC (Theorem 12.1.6) gives a necessary condition for a local optimizer, it is not always easy to find all the solutions to the equation $Df(\mathbf{x}) = \mathbf{0}$. Unless the derivative $Df(\mathbf{x})$ is very simple, there is usually no nice, closed-form solution. Sometimes the best approach is to use numerical methods to try to solve the system of equations $Df(\mathbf{x}) = \mathbf{0}$, but it is often more efficient to use other (iterative) techniques that do not depend on finding the zeros of the derivative directly. Most of the rest of this chapter is dedicated to such techniques.

Example 12.1.9. We generalize the one-dimensional optimization problems of Examples 6.1.12 and 6.1.13 to two dimensions by computing the MLE of the mean μ and variance σ^2 of a normal distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

As in Examples 6.1.12 and 6.1.13, we begin with an i.i.d. sample $(x_1, \ldots, x_n)$ drawn from a normal distribution with unknown parameters μ and σ^2 and consider the likelihood

$$L(\mu, \sigma^2) = \prod_{i=1}^{n} f(x_i \mid \mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right).$$

The MLE is the maximizer $(\widehat{\mu}, \widehat{\sigma}^2)$ of the likelihood $L(\mu, \sigma^2)$. As in the one-dimensional case, it is easier to work with the log-likelihood $\ell(\mu, \sigma^2)$, which is given by

$$\ell(\mu, \sigma^2) = \log(L(\mu, \sigma^2)) = -\frac{n}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2.$$

Thus, to find the MLE we must solve the unconstrained minimization problem

$$(\widehat{\mu}, \widehat{\sigma}^2) = \operatorname*{argmin}_{(\mu,\sigma^2)} -\ell(\mu, \sigma^2).$$

Any minimizer $(\widehat{\mu}, \widehat{\sigma}^2)$ of $-\ell(\mu, \sigma^2)$ must be a critical point and hence must satisfy the FONC:

$$0 = \left.\frac{\partial \ell}{\partial \mu}\right|_{\widehat{\mu},\widehat{\sigma}^2} = \frac{1}{\widehat{\sigma}^2}\sum_{i=1}^{n}(x_i - \widehat{\mu}) \quad \text{and} \tag{12.1a}$$

$$0 = \left.\frac{\partial \ell}{\partial \sigma^2}\right|_{\widehat{\mu},\widehat{\sigma}^2} = -\frac{n}{2\widehat{\sigma}^2} + \frac{1}{2(\widehat{\sigma}^2)^2}\sum_{i=1}^{n}(x_i - \widehat{\mu})^2. \tag{12.1b}$$

Here, as in Example 6.1.13, we treat σ^2 not as the square of σ but as an awkwardly named variable in its own right. By (12.1a) we have $0 = \sum_{i=1}^{n}(x_i - \widehat{\mu}) = (\sum_{i=1}^{n} x_i) - n\widehat{\mu}$. Thus, the maximum likelihood estimator $\widehat{\mu}$ of μ for an i.i.d. sample $X_1, \ldots, X_n$ satisfies

$$\widehat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

And by (12.1b) we have $n\widehat{\sigma}^2 = \sum_{i=1}^{n}(x_i - \widehat{\mu})^2$, and hence the maximum likelihood estimator $\widehat{\sigma}^2$ satisfies

$$\widehat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \widehat{\mu})^2.$$

Remark 12.1.10. We have already seen the estimators of the previous example in Section 6.1. Recall that $\widehat{\mu}$ is an unbiased estimator, but $\widehat{\sigma}^2$ is biased.

Nota Bene 12.1.11. The FONC is a necessary condition, meaning that every minimizer must satisfy it. But it is not sufficient, because a point could satisfy the FONC without being a minimizer. For example, the point $x = 0$ satisfies the FONC for the function $f(x) = x^3$, but it is neither a minimizer nor a maximizer.

12.1.3 Second-Order Conditions

The second derivative (or Hessian) of a function f can give both necessary and sufficient conditions for a critical point $\mathbf{x}^*$ to be a minimizer of the function f.

Theorem 12.1.12 (Second-Order Necessary Condition (SONC)). *Assume that $\Omega \subset \mathbb{R}^n$ is open and that $f \in C^2(\Omega; \mathbb{R})$. If $\mathbf{x}^* \in \Omega$ is a local minimizer of f on Ω, then $D^2 f(\mathbf{x}^*)$ is positive semidefinite (hereafter denoted $D^2 f(\mathbf{x}^*) \geq 0$).*

Proof. Suppose, by way of contradiction, that there exists $\mathbf{v} \in \mathbb{R}^n$ such that $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^*)\mathbf{v} < 0$. Choose $\delta > 0$ sufficiently small so that $\mathbf{x}^*$ is a minimizer on $B(\mathbf{x}^*, \delta) \subset \Omega$ and $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^* + t\delta\mathbf{v})\mathbf{v} < 0$ for all $t \in [0, 1]$. By Taylor's theorem (Theorem 10.3.8), for any $0 < \varepsilon < \delta$ we have

$$f(\mathbf{x}^* + \varepsilon\mathbf{v}) = f(\mathbf{x}^*) + \varepsilon Df(\mathbf{x}^*)\mathbf{v} + \varepsilon^2 \int_0^1 (1-t)\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^* + t\varepsilon\mathbf{v})\mathbf{v}\, dt.$$

Since $Df(\mathbf{x}^*) = 0$ (by the FONC) and $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^* + t\varepsilon\mathbf{v})\mathbf{v} < 0$ for all $t \in [0, 1]$, we must have $f(\mathbf{x}^* + \varepsilon\mathbf{v}) < f(\mathbf{x}^*)$, which is a contradiction, since $\mathbf{x}^*$ is a local minimizer. Thus $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^*)\mathbf{v} \geq 0$ for every $\mathbf{v} \in \mathbb{R}^n$. ∎

Example 12.1.13. Like the FONC, the SONC guarantees that certain points cannot be minimizers. For example, consider the function $f(x, y) = -x^2 - y^2$. The derivative $Df(x, y)$ is $\begin{bmatrix} -2x & -2y \end{bmatrix}$, so the origin is the only critical point. The second derivative is

$$D^2 f(x, y) = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix},$$

which is negative definite everywhere, since both eigenvalues are always negative. Therefore by the SONC, no point can be a local minimizer.

Example 12.1.14. Consider the function $f(x,y) = x^3 - 3x^2 + y^2$. Example 12.1.7 shows that the critical points of f are $(0,0)$ and $(2,0)$. The Hessian of f is

$$D^2 f(x,y) = \begin{bmatrix} 6x-6 & 0 \\ 0 & 2 \end{bmatrix}.$$

Since the Hessian is diagonal, we can immediately see that the Hessian is positive definite at $(2,0)$ and indefinite (that is, the eigenvalues are of mixed sign) at $(0,0)$. Thus $(0,0)$ is not a local minimizer, but the SONC does not rule out $(2,0)$. But note that the SONC cannot guarantee that a point is a minimizer.

Example 12.1.15. Consider the function $g(x,y) = x^3 - y^3$. The derivative $Df(x,y)$ is $\begin{bmatrix} 3x^2 & -3y^2 \end{bmatrix}$, so the origin is the only critical point. The second derivative is

$$D^2 f(x,y) = \begin{bmatrix} 3x & 0 \\ 0 & -3y \end{bmatrix},$$

which is zero (hence positive semidefinite) at the origin. Therefore the SONC does not rule out the possibility that the origin could be a minimizer.

The second derivative can also be used to guarantee that certain critical points are minimizers, as the next theorem shows.

Theorem 12.1.16 (Second-Order Sufficient Condition (SOSC)). *Assume that $\Omega \subset \mathbb{R}^n$ is open and that $f \in C^2(\Omega;\mathbb{R})$. If $\mathbf{x}^* \in \Omega$ is such that $Df(\mathbf{x}^*) = \mathbf{0}$ and $D^2 f(\mathbf{x}^*)$ is positive definite (hereafter denoted $D^2 f(\mathbf{x}^*) > 0$), then $\mathbf{x}^*$ is a strict local minimizer on Ω.*

Proof. Since $D^2 f(\mathbf{x}^*) > 0$, there exists $\varepsilon > 0$ such that $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^*)\mathbf{v} > 2\varepsilon\|\mathbf{v}\|^2$ for all $\mathbf{v} \neq \mathbf{0}$, and, in particular, the smallest eigenvalue of $D^2 f(\mathbf{x}^*)$ is at least 2ε. (See Volume 1, Section 4.5.1, for more details.) By the implicit function theorem (Theorem 10.3.2) the eigenvalues of $D^2 f(\mathbf{x})$ depend continuously on $\mathbf{x}$, so there exists $\delta > 0$ such that the smallest eigenvalue of $D^2 f(\mathbf{x})$ is at least ε for all $\mathbf{x} \in B(\mathbf{x}^*, \delta) \subset \Omega$. This implies that $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x})\mathbf{v} > \varepsilon\|\mathbf{v}\|^2$ for all $\mathbf{v} \neq \mathbf{0}$ and for all $\mathbf{x} \in B(\mathbf{x}^*, \delta) \subset \Omega$.

For any $\mathbf{v}$ with $\|\mathbf{v}\| < \delta$, Taylor's theorem (Theorem 10.3.8) implies that

$$f(\mathbf{x}^* + \mathbf{v}) = f(\mathbf{x}^*) + Df(\mathbf{x}^*)\mathbf{v} + \int_0^1 (1-t)\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^* + t\mathbf{v})\mathbf{v}\, dt.$$

Since $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}^* + t\mathbf{v})\mathbf{v} > 0$ for all $\mathbf{v} \neq \mathbf{0}$ and $t \in [0,1]$, the integral is always positive. Since $Df(\mathbf{x}^*)\mathbf{v} = 0$, we must have $f(\mathbf{x}^* + \mathbf{v}) > f(\mathbf{x}^*)$. ∎

Example 12.1.17. ⚠ Solving a linear system is closely related to minimizing a *quadratic objective* $f : \mathbb{R}^n \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \mathbf{x}^\mathsf{T} A\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c \tag{12.2}$$

for some square matrix $A \in M_n(\mathbb{R})$, some vector $\mathbf{b} \in \mathbb{R}^n$, and some constant $c \in \mathbb{R}$. Exercise 12.3 shows that the function (12.2) is equal to the function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} Q\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c, \tag{12.3}$$

where $Q = A + A^\mathsf{T}$, so one may always assume that quadratic functions are of the form (12.3) with Q symmetric. Equation (12.3) has a unique critical point $\mathbf{x}^*$, corresponding to the unique solution of

$$Q\mathbf{x}^* = \mathbf{b}, \tag{12.4}$$

and the SONC shows that this can be a minimizer only if $Q \geq 0$. The SOSC shows that it is always a minimizer if $Q > 0$. Thus solving the system (12.4) with $Q > 0$ is equivalent to solving the quadratic optimization problem (12.3).

Remark 12.1.18. Depending on Q, the best way to solve the linear system (12.4) is often to use numerical optimization algorithms on the quadratic problem (12.3) instead of using standard linear solvers on (12.4). Some of these numerical methods, such as gradient descent and the conjugate-gradient method, are described later in this chapter.

Example 12.1.19. Consider again the function $f(x, y) = x^3 - 3x^2 + y^2$ of Examples 12.1.7 and 12.1.14. Since $D^2 f(2, 0) > 0$, the SOSC guarantees that $(2, 0)$ is a local minimizer.

Example 12.1.20. The Rosenbrock function, pictured in Figures 12.1 and 12.2, is

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2. \tag{12.5}$$

The point $(1, 1)$ satisfies the FONC since

$$Df(1, 1) = (-2(1 - x) - 400x(y - x^2), 200(y - x^2))\big|_{(1,1)} = (0, 0).$$

A straightforward computation shows that there are no other critical points, and hence, by Theorem 12.1.6, no other point can be a local minimizer of f.

The Hessian at the point $(1,1)$ is given by

$$D^2 f(1,1) = \begin{bmatrix} 2 - 400y + 1200x^2 & -400x \\ -400x & 200 \end{bmatrix}\Bigg|_{(1,1)} = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}.$$

To see whether this is positive definite, note first that its eigenvalues satisfy the degree-2 equation $\lambda^2 - B\lambda + C = 0$, with $B = \operatorname{tr}(D^2 f(1,1))$ and $C = \det(D^2 f(1,1))$. Since the Hessian is symmetric, its eigenvalues are all real. The quadratic equation gives $\lambda = \frac{1}{2}(B \pm \sqrt{B^2 - 4C})$. It is left to the reader to show that $\operatorname{tr}(D^2 f(1,1)) > 0$ and $\det(D^2 f(1,1)) > 0$, so $B > \sqrt{B^2 - 4C} > 0$, and hence the eigenvalues must both be positive. Therefore, the point $(1,1)$ satisfies the SOSC and is, indeed, a local minimizer.

The SOSC tells us nothing about global minimizers—even though there are no other local minima, the function could decrease below that local minimum without having another local minimizer. However, in this special case $(1,1)$ is also a global minimizer because $f(1,1) = 0$ and $f(x,y)$ is strictly positive for all other points (x,y).

Remark 12.1.21. The Rosenbrock function is often used as a test function for optimization algorithms because, although it is easy to find the minimizer analytically, many algorithms for finding local minimizers get confused in the long, narrow, almost-flat-bottomed valley of the graph of the Rosenbrock function.

Remark 12.1.22.* In the two-variable case we give a general condition to ensure that $D^2 f(\mathbf{x}^*) > 0$. If $U \subset \mathbb{R}^2$ is open and $f \in C^2(U;\mathbb{R})$, then

$$D^2 f(\mathbf{x}^*) = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{bmatrix}.$$

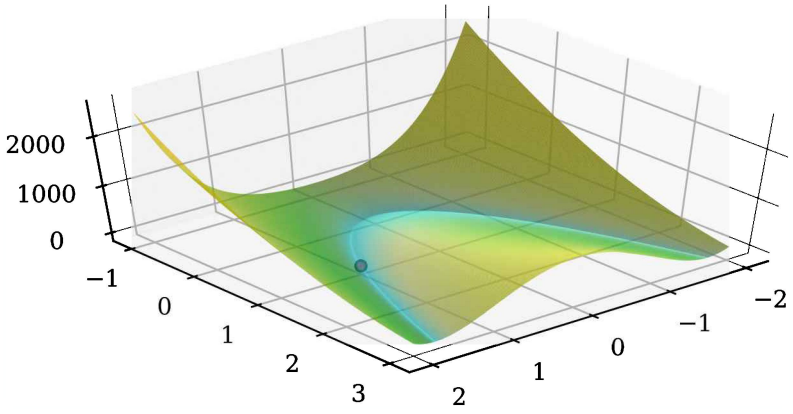


Figure 12.1. *A graph of the Rosenbrock function as defined in* (12.5)*, with its minimizer at* $(1,1)$ *indicated as a darker dot. As an alternative to the three-dimensional plot shown here, the contour plot in Figure 12.2, can also be useful.*

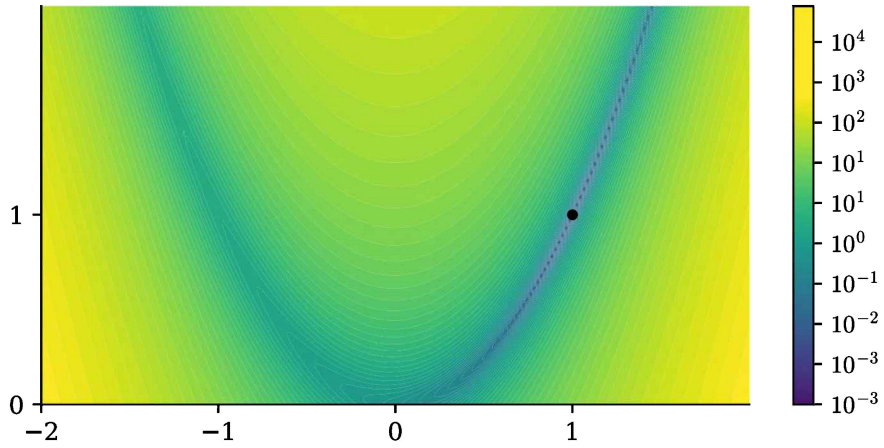


Figure 12.2. *A contour plot of the Rosenbrock function, where the colors represent the values of f, corresponding to the color bar on the right of the figure. The bottom corners are yellow, representing larger values around* 1500*, and the dark blue curves represent small values near zero. The minimizer* $(1, 1)$ *is indicated in black. The light-colored curves in the plot are contour lines, corresponding to* level sets*, which are sets of the form* $\{\mathbf{x} \mid f(\mathbf{x}) = c\}$ *for various values of* c*.*

As observed in the previous example, the eigenvalues of $D^2 f(\mathbf{x}^*)$ are the roots of $\lambda^2 - B\lambda + C$ where $B = \operatorname{tr}(D^2 f(\mathbf{x}^*))$ and $C = \det(D^2 f(\mathbf{x}^*))$. Since $D^2 f(\mathbf{x}^*)$ is symmetric, the eigenvalues are always real. The quadratic formula guarantees that the roots are both positive precisely when $B > \sqrt{B^2 - 4C} > 0$. A little thought shows that this is equivalent to the condition that $B > 0$ and $C > 0$. Expanding out the trace and the determinant, along with a little more work, gives the conditions which are often taught in multivariable calculus classes:

$$f_{xx}|_{\mathbf{x}^*} > 0 \qquad \text{and} \qquad (f_{xx} f_{yy} - f_{xy}^2)|_{\mathbf{x}^*} > 0. \tag{12.6}$$

Similarly, the characteristic polynomial has nonnegative roots precisely when the previous strict inequalities are relaxed to allow equality.

Remark 12.1.23.* Since $D^2 f(\mathbf{x})$ is symmetric, the spectral theorem (Volume 1, Theorem 4.4.7) guarantees that at any point $\mathbf{x} \in \Omega$ there is an orthogonal basis of $\mathbb{R}^n$ consisting of eigenvectors of $D^2 f(\mathbf{x})$ and the eigenvalues are all real. This implies that the directional derivative is increasing in the directions corresponding to the positive eigenvalues and decreasing in the directions corresponding to the negative eigenvalues. In two dimensions, if $Df(\mathbf{x}) = \mathbf{0}$ and $D^2 f(\mathbf{x})$ has both a positive and a negative eigenvalue, then the graph has a saddle shape at $\mathbf{x}$. More generally, we call $\mathbf{x}$ a *saddle point* of f if $D^2 f(\mathbf{x})$ is indefinite (has at least one positive and at least one negative eigenvalue).

Example 12.1.24.* At the point $(0, 0)$, the function $f(x, y) = x^3 - 3x^2 + y^2$ of Examples 12.1.7 and 12.1.14 is a saddle point.

12.2 One-Dimensional Numerical Optimization

For functions with a one-dimensional domain, the tools of the previous section reduce to the usual first- and second-derivative tests of single-variable calculus. However, even in this simple one-dimensional case, finding the points where f' vanishes is not always easy, and most functions have no closed-form expression for the critical points. So we need numerical methods for finding minima of functions of one variable. In this section we discuss several of the most common methods.

All of the optimization methods discussed in this chapter are *iterative*, meaning that they begin with an initial approximation, or guess, and then use that approximation to (hopefully) construct a better approximation. The utility of such algorithms depends on how rapidly they converge. Therefore, we begin the section with a brief discussion of convergence, before moving to the specific algorithms.

12.2.1 Convergence

It only makes sense to discuss the temporal complexity of an algorithm that terminates. Many important algorithms never have a natural termination point—instead they iterate, getting closer to the desired answer with each iteration. When to stop the algorithm depends on how accurate one wants the answer to be. In this situation, there are two important factors that affect the overall complexity of the algorithm:

(i) The number of steps necessary to reach a desired level of accuracy.

(ii) The temporal and spatial complexities of each step.

Both the complexity of each iteration and the convergence rate are important when evaluating iterative algorithms. In some algorithms, each iteration involves only a few simple steps, but many iterations are required to reach the desired accuracy. Other algorithms need only a few iterations to converge to a high level of accuracy, but each iteration is costly to execute.

Definition 12.2.1. *Given a sequence $(x_k)_{k=0}^{\infty} \subset \mathbb{R}$ that converges to $x^* \subset \mathbb{R}$, let $\varepsilon_n = |x_n - x^*|$ for each $n \in \mathbb{N}$. The* convergence of $(x_k)_{k=0}^{\infty}$ to x^* is (at least) linear *of rate μ if there exists a $\mu \in [0,1)$ and an $N > 0$ such that*

$$\varepsilon_n \leq \mu \varepsilon_{n-1} \qquad \text{whenever } n > N. \tag{12.7}$$

If for every *$\mu \in (0,1)$ there exists an N such that (12.7) holds, then we say the convergence is* superlinear. *If $\varepsilon_n \to 0$, but there is no $\mu \in (0,1)$ and $N > 0$ satisfying (12.7), then the convergence is* sublinear. *If there exists an $\alpha > 1$ and numbers $\rho, N > 0$ such that*

$$\varepsilon_n \leq \rho \varepsilon_{n-1}^{\alpha} \qquad \text{whenever } n > N,$$

then we say the convergence of $(x_k)_{k=0}^{\infty}$ to x^* is of order α. *In particular, the convergence is said to be* quadratic convergence *if $\alpha = 2$.*

Roughly speaking, a linear convergence of rate μ adds about $-\log_{10} \mu$ digits of accuracy at each iteration, whereas quadratic convergence ($\alpha = 2$) approximately doubles the number of digits of accuracy with each iteration.

Remark 12.2.2. The number μ must be less than 1 for linear convergence, but the number ρ has no such restriction for higher-order convergence. The reason for the restriction $\mu < 1$ is that we expect the errors to be decreasing at each step, but when μ is 1 or greater, then the relation $\varepsilon_k \leq \mu\varepsilon_{k-1}$ gives no guarantee that the errors decrease. When $\mu < 1$, however, then $\varepsilon_k \leq \mu\varepsilon_{k-1} < \varepsilon_{k-1}$ guarantees that ε_k shrinks, and it gives some information about how quickly ε_k shrinks. For $\alpha > 1$, the errors shrink regardless of the size of ρ.

Example 12.2.3. The sequence $x_k = 1 + (\frac{1}{2})^k$ converges to 1 linearly with rate $\mu = \frac{1}{2}$, because $\varepsilon_k = (\frac{1}{2})^k = \frac{1}{2}\varepsilon_{k-1}$. Since $\mu = \frac{1}{2}$ we expect the number of digits of accuracy to improve by $-\log_{10}(\frac{1}{2}) \approx 0.30$ each iteration—that is, we expect the number of zero digits immediately after the decimal point to increase by about one every $\frac{1}{0.3} \approx 3.3$ iterations. This is clearly visible in the following list of the decimal representations of the first 10 terms of this sequence.

```
k=0   2.0
k=1   1.5                 one digit correct
k=2   1.25
k=3   1.125
k=4   1.0625              two digits correct
k=5   1.03125
k=6   1.015625
k=7   1.0078125           three digits correct
k=8   1.00390625
k=9   1.001953125
k=10  1.0009765625        four digits correct
```

Example 12.2.4. The error in the trapezoid rule for $n+1$ evenly spaced nodes is approximately of the form $\varepsilon_n = cn^{-2}$ for some constant c (see Section 9.6.3). This converges sublinearly, but not linearly, because

$$\varepsilon_n = cn^{-2} = c\frac{(n-1)^2}{n^2}(n-1)^{-2} = \left(1 - \frac{2}{n} + \frac{1}{n^2}\right)\varepsilon_{n-1}.$$

Since the sequence $\left(1 - \frac{2}{n} + \frac{1}{n^2}\right)$ converges to 1 as $n \to \infty$, there is no $\mu < 1$ such that $\varepsilon_n \leq \mu\varepsilon_{n-1}$ for all sufficiently large n.

Simpson's rule has error approximately cn^{-4}, which converges to zero much faster than the trapezoid rule, but it also converges sublinearly.

12.2.2 Bisection Algorithm for Critical Points

The *bisection algorithm* is a method used to find a zero of a function. Applying it to f' on $[a, b]$ for a continuously differentiable function f will find a critical point under mild conditions.

The algorithm begins by computing the value of f' at the endpoints $a_0 = a$ and $b_0 = b$, if those two values have different sign, then the intermediate value theorem (see Volume 1, Corollary 5.9.14) guarantees that there is a zero of f' in the interval (a, b).

Now divide the interval in half to get $c_0 = \frac{a_0+b_0}{2}$ and compute $f'(c_0)$. There are three possibilities: (i) If $f'(c_0) = 0$, the zero has been found, and the algorithm terminates. Otherwise, (ii) $f'(c_0)$ must have sign opposite of $f'(a_0)$ or (iii) $f'(c_0)$ must have sign opposite of $f'(b_0)$. In either of these last two cases, we now have a new interval (a_0, c_0) or (c_0, b_0) that must contain a zero. Call the new interval (a_1, b_1).

At each subsequent step, evaluate f' at $c_k = \frac{a_k+b_k}{2}$, and let the new interval (a_{k+1}, b_{k+1}) be either (a_k, c_k) or (c_k, b_k), depending on which one has a sign change. The process is repeated until the desired accuracy is reached.

At stage k, take the midpoint $c_k = \frac{a_k+b_k}{2}$ of the interval as the approximation of the true zero $x^* \in (a_k, b_k)$. The error $\varepsilon_k = c_k - x^*$ can be no larger than $\frac{1}{2}(b_k - a_k) = 2^{-(k+1)}(b - a)$. In the worst case, where $\varepsilon_k = 2^{-(k+1)}(b - a)$, the sequence ε_k converges linearly at rate $\mu = \frac{1}{2}$, because $\varepsilon_k = \frac{1}{2}\varepsilon_{k-1}$. Just as in Example 12.2.3, this means that the number of correct digits should increase by $-\log_{10}(\frac{1}{2}) \approx 0.30$ for each iteration.

Example 12.2.5. Taking $f(x) = \frac{1}{4}x^4 - 27x$ and $f'(x) = x^3 - 27$ means that the minimizer occurs at $x^* = 3$. Calculating this using the bisection algorithm with an initial interval of $[2.4, 4.4]$ yields the following, for the first 10 iterations:

```
k=0   3.4                one digit correct
k=1   2.9
k=2   3.15
k=3   3.025              two digits correct
k=4   2.9625
k=5   2.99375
k=6   3.009375           three digits correct
k=7   3.0015625
k=8   2.99765625
k=9   2.999609375
k=10  3.0005859375       four digits correct
```

The exact answer, to machine precision, was first found after 48 iterations.

12.2.3 Newton's Method in One Dimension

A much faster method for finding a local minimizer is *Newton's method*. Newton's method, or variants of Newton, like the *secant method* (see Section 12.2.5 below), are almost always the methods of choice whenever they can be used.

To find a critical point of a function f whose second derivative is Lipschitz, we may use Newton's method for finding a zero of $f'(x)$. The method takes an initial

estimate x_k of the critical point and replaces it with

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

If the initial estimate is sufficiently close to the zero, then Newton's method converges quadratically.

Theorem 12.2.6. *Given $f \in C^2((a,b);\mathbb{R})$ with f'' Lipschitz,*[48] *assume that $x^* \in (a,b)$ is a local minimizer of f. If $f''(x^*) \neq 0$, then there exists $\delta > 0$ such that for any initial $x_0 \in B(x^*, \delta)$ Newton's method converges quadratically to x^*; that is, if $\varepsilon_k = |x_k - x^*|$, then $\varepsilon_{k+1} \leq M\varepsilon_k^2$ for some constant $M \geq 0$.*

Proof. This follows by applying the convergence result for the rootfinding Newton's method (Volume 1, Theorem 7.3.4) to the function f'. □

Example 12.2.7. Consider $f(x) = \frac{1}{4}x^4 - 27x$, with $f'(x) = x^3 - 27$, as in Example 12.2.5. Using Newton's method with an initial guess of $x_0 = 3.4$ yields

```
k=0   3.4                  one digit correct
k=1   3.0452133794694354   two digits correct
k=2   3.0006679769830704   four digits correct
k=3   3.0000001486869405   seven digits correct
k=4   3.0000000000000070   fifteen digits correct
k=5   3.0000000000000000   exact, to machine precision
```

In this example the number of correct digits roughly doubles at each iteration, corresponding to the quadratic convergence of Newton's algorithm. The algorithm reaches the exact answer, to machine precision, in five steps. Compare this to bisection, which took 10 iterations just to find the first four digits and 48 iterations to find the exact answer, to machine precision.

Nota Bene 12.2.8. Newton's method is very sensitive to the initial guess. If the value of x_0 is too far from the minimizer, Newton's method may never converge and may even diverge to infinity.

12.2.4 Newton as Quadratic Approximation

An alternative way of thinking about Newton's method for minimization is as a quadratic approximation. This approach begins by fitting a quadratic function $q(x)$

[48] If f'' is differentiable, then f'' is Lipschitz, by Proposition 6.3.7 of Volume 1.

to the function $f(x)$ at x_0 by using the Taylor expansion. Let

$$q(x) := f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2.$$

Note that $q(x_0) = f(x_0)$, $q'(x_0) = f'(x_0)$, and $q''(x_0) = f''(x_0)$. We use the minimizer x_1 of $q(x)$ as a proxy for the minimizer of f (see Figure 12.3). Thus x_1 must satisfy $0 = q'(x_1)$, which gives $0 = f'(x_0) + f''(x_0)(x_1 - x_0)$. Solving this yields the relation

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)},$$

which is exactly the same formula as Newton's method for finding a zero of f'. Therefore, applying one step of Newton's method to the derivative f' is the same as approximating the function f with a quadratic and finding the minimizer of the quadratic.

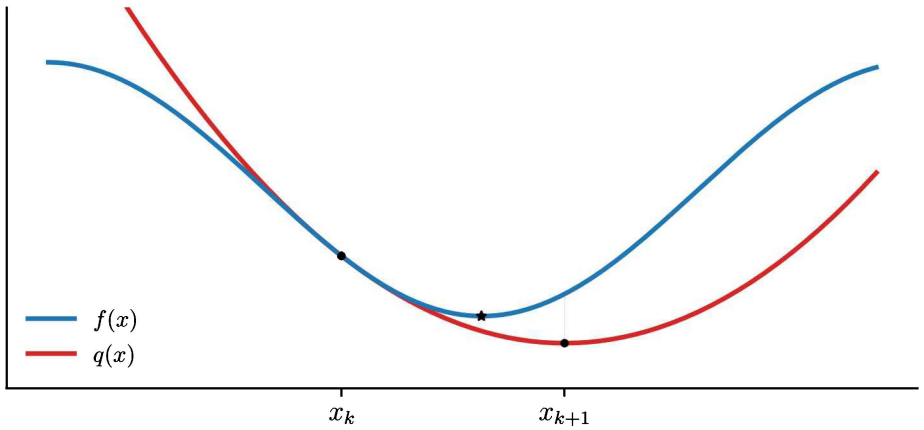


Figure 12.3. *Given an approximation x_k of the minimizer x^* of a function f (blue), Newton's method for minimizing corresponds to approximating f with the quadratic function q (red) that best fits f at x_k and then letting x_{k+1} be the minimizer of q.*

When f is a quadratic function, then the quadratic approximation is actually equal to f. The previous discussion shows that one step of Newton's method gives the minimizer of the quadratic approximation, which is the minimizer of f. Thus, we have the following proposition.

Proposition 12.2.9. *If $f : \mathbb{R} \to \mathbb{R}$ is quadratic (that is, $f(x) = ax^2 + bx + c$, for some $a, b, c, \in \mathbb{R}$, $a > 0$), then Newton's method converges to the unique optimizer of f in a single step, regardless of starting point.*

12.2.5 The Secant Method

The subtraction and division steps of Newton's method are inexpensive, but computing $f''(x)$ may be difficult or expensive. Suppose we cannot or prefer not to

calculate the second derivative of f. We can approximate $f''(x_k)$ by using forward differences of the first derivative

$$f''(x_k) \approx \frac{f'(x_k + h) - f'(x_k)}{h} \qquad \text{for small values of } h.$$

If x_{k-1} is near x_k, then we can take $h = x_{k-1} - x_k$ to obtain

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}. \tag{12.8}$$

Substituting (12.8) into Newton's method, the next approximation is

$$x_{k+1} = x_k - f'(x_k)\frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}. \tag{12.9}$$

This is the *secant method*. At each stage, since $f'(x_{k-1})$ has already been calculated for the previous step, we need only calculate $f'(x_k)$ in order to compute the next approximation x_{k+1}. Of course, this computational savings comes with a cost, namely, the order of convergence is no longer quadratic, as the next theorem shows.

Theorem 12.2.10. *Assume that $f \in C^2((a,b);\mathbb{R})$ and f'' is Lipschitz on (a,b). If $x^* \in (a,b)$ is a local minimizer of f with $f''(x^*) \neq 0$, then there exists $\delta > 0$ such that the secant method converges to x^* with order $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$, provided $x_0, x_1 \in B(x^*, \delta)$ with $x_1 \neq x_0$. In other words, if $\varepsilon_k = |x_k - x^*| < \delta$, then $\varepsilon_{k+1} \leq M\varepsilon_k^{\phi}$ for some constant $M \geq 0$.*

The proof is similar to that of Newton's method (see Volume 1, Theorem 7.3.4) and is given in Section 12.2.7.

Remark 12.2.11. Although Newton's method has a higher order of convergence, the secant method can be faster if computing the second derivative $f''(x_k)$ is much more expensive than computing $f'(x_k)$.

12.2.6 Stopping Criteria

When performing any of the iterative methods in this chapter, we need to know when to stop. There are several standard stopping criteria. These can be based on the norm of the derivative, the change in approximation, or the change of the function. For a prespecified error tolerance $\varepsilon > 0$ we might choose any one of the following conditions to decide when to stop:

(i) $\|Df(x_k)\| < \varepsilon$.

(ii) $\|x_{k+1} - x_k\| < \varepsilon$.

(iii) $|f(x_{k+1}) - f(x_k)| < \varepsilon$.

(iv) The number of iterations is too large.

The first condition is based on the FONC: when the derivative $Df(x)$ is nearly zero, we expect the function to be close to its minimum. The second and third conditions are based on the idea that we should stop when the next iteration does not change enough to matter—either the estimated optimizer x_k is not changing much or the estimated optimal value $f(x_k)$ is not changing much.

These different stopping criteria give different stopping times, and a reasonable argument can be made for each choice. The "correct" choice may depend on the problem being solved. These stopping criteria can also be combined in various ways. Of course, if the method does not converge, then we must stop the method after a fixed number of iterations and consider using a different method.

Remark 12.2.12. This chapter covers some of the more common methods of one-dimensional optimization. But there are many others. Some of these are explored in more depth in the labs for this volume.

12.2.7 *Proof of Theorem 12.2.10

We prove Theorem 12.2.10, which guarantees that the secant method converges superlinearly, with order $\phi = \frac{1+\sqrt{5}}{2}$.

Proof. Assume that f'' is Lipschitz with constant L, so that

$$|f''(c) - f''(d)| \leq L|c - d|$$

for any $c, d \in (a, b)$. Choose $\delta, \eta > 0$ with $B(x^*, \delta) \subset (a, b)$, so that for every $x \in B(x^*, \delta)$ we have $f''(x) \geq \eta > 0$. Let $\lambda = 2\delta L/\eta$, and, if necessary, shrink δ so that $\lambda < 1$. Choose any distinct values of $x_0, x_1 \in B(x^*, \delta)$ to start the iteration. We show inductively that each $x_n \in B(x^*, \delta)$ and that for each $n > 1$ the error $\varepsilon_n = |x_n - x^*|$ is strictly less than $\lambda\varepsilon_{n-1}$ (that is, the convergence is at least linear).

Since $x_0, x_1 \in B(x^*, \delta)$, the initial induction step holds. Now assume the induction hypothesis for all $n < k$ and that $x_n \neq x^*$ for all $n \leq k$. We have

$$\varepsilon_k = |(x_k - x_{k-1}) + (x_{k-1} - x^*)| = \left| -f'(x_{k-1}) \frac{x_{k-1} - x_{k-2}}{f'(x_{k-1}) - f'(x_{k-2})} + (x_{k-1} - x^*) \right|.$$

Since x_{k-1} and x_{k-2} lie in $B(x^*, \delta)$ and $|f''(x)| \geq \eta > 0$ for all $x \in B(x^*, \delta)$, we have $f'(x_{k-1}) \neq f'(x_{k-2})$. The mean value theorem guarantees the existence of some $c, d \in B(x^*, \delta)$ such that $\frac{x_{k-1} - x_{k-2}}{f'(x_{k-1}) - f'(x_{k-2})} = 1/f''(c)$ and $f'(x_{k-1}) = f''(d)(x_{k-1} - x^*)$. Combining these with the previous calculation gives

$$\begin{aligned} \varepsilon_k &= \left| x_{k-1} - x^* - \frac{f'(x_{k-1})}{f''(c)} \right| = \left| (x_{k-1} - x^*) - \frac{f''(d)(x_{k-1} - x^*)}{f''(c)} \right| \\ &= \varepsilon_{k-1} \left| \frac{f''(c) - f''(d)}{f''(c)} \right| \leq \varepsilon_{k-1} L \frac{|c - d|}{|f''(c)|} \leq \varepsilon_{k-1} L \frac{|c - d|}{\eta} \\ &< \lambda \varepsilon_{k-1}. \end{aligned} \tag{12.10}$$

This concludes the induction and shows the secant method converges at least linearly.

To see that the order of convergence is at least ϕ, note that the difference $|c-d|$ in the previous argument can be bounded more tightly. Specifically, c lies in the interval from x_{k-1} to x_{k-2}, and d lies in the interval from x^* to x_{k-1}, so $|c-d| \leq 2|x_{k-2}-x^*| = 2\varepsilon_{k-2}$. Substituting this into (12.10) and letting $C = 2L/\eta$ gives

$$\varepsilon_k \leq \varepsilon_{k-1}\varepsilon_{k-2}2L/\eta = C\varepsilon_{k-1}\varepsilon_{k-2}. \tag{12.11}$$

We wish to find the largest value of α such that

$$\lim_{k\to\infty} \frac{\varepsilon_k}{\varepsilon_{k-1}^{\alpha}} = D \tag{12.12}$$

for some constant D. We have already shown that convergence of the secant method is at least linear, so $\alpha \geq 1$. Let

$$S_k = \frac{\varepsilon_k}{\varepsilon_{k-1}^{\alpha}}.$$

This implies

$$\varepsilon_k = S_k\varepsilon_{k-1}^{\alpha} = S_k(S_{k-1}\varepsilon_{k-2}^{\alpha})^{\alpha} = S_kS_{k-1}^{\alpha}\varepsilon_{k-2}^{\alpha^2},$$

which gives

$$\frac{\varepsilon_k}{\varepsilon_{k-1}\varepsilon_{k-2}} = \frac{S_kS_{k-1}^{\alpha}\varepsilon_{k-2}^{\alpha^2}}{S_{k-1}\varepsilon_{k-2}^{\alpha}\varepsilon_{k-2}} = S_kS_{k-1}^{\alpha-1}\varepsilon_{k-2}^{\alpha^2-\alpha-1}.$$

The first two factors of the right side approach constants, so we must have

$$\alpha^2 - \alpha - 1 = 0,$$

and the quadratic equation, combined with $\alpha \geq 1$, shows that $\alpha = \phi = \frac{1+\sqrt{5}}{2}$. □

12.3 Gradient Descent

For the rest of this chapter we discuss numerical algorithms for solving unconstrained optimization problems in higher dimensions of the form

$$\text{minimize } f : \mathbb{R}^n \to \mathbb{R}.$$

These algorithms can also be used in many cases to solve problems on an open subset $\Omega \subset \mathbb{R}^n$, but there is a possibility that they will step out of Ω. In such situations, one can either just try restarting the algorithm at a different initial point or switch to some of the constrained optimization techniques discussed later in the book.

In this section we discuss a class of algorithms called *gradient descent* methods. These are based on the observation that the negative gradient of the objective function points in the direction of greatest decrease.

12.3.1 Gradient Descent Methods

Exercise 10.8 shows that the gradient $Df(\mathbf{x})^{\mathsf{T}}$ of a function f points in the direction of greatest increase of the function. Here we give the proof again.

Proposition 12.3.1. *Let $\Omega \subset \mathbb{R}^n$ be open and let $f : \Omega \to \mathbb{R}$ be a function that is differentiable at $\mathbf{x} \in \Omega$. Among all unit vectors in $\mathbb{R}^n$, the direction $\mathbf{u} \in \mathbb{R}^n$ with the greatest directional derivative $D_{\mathbf{u}} f(\mathbf{x})$ at $\mathbf{x}$ is the normalization of the gradient $\mathbf{u} = Df(\mathbf{x})^{\mathsf{T}} / \|Df(\mathbf{x})^{\mathsf{T}}\|$.*

Proof. The Cauchy–Schwarz inequality (see Volume 1, Proposition 3.1.17) guarantees that

$$|D_{\mathbf{u}} f(\mathbf{x})| = |Df(\mathbf{x})\mathbf{u}| = |\langle Df(\mathbf{x})^{\mathsf{T}}, \mathbf{u} \rangle| \leq \|Df(\mathbf{x})^{\mathsf{T}}\|$$

for every unit vector $\mathbf{u} \in \mathbb{R}^n$. But letting $\mathbf{u} = Df(\mathbf{x})^{\mathsf{T}} / \|Df(\mathbf{x})^{\mathsf{T}}\|$ gives

$$D_{\mathbf{u}} f(\mathbf{x}) = \langle Df(\mathbf{x})^{\mathsf{T}}, Df(\mathbf{x})^{\mathsf{T}} \rangle / \|Df(\mathbf{x})^{\mathsf{T}}\| = \|Df(\mathbf{x})^{\mathsf{T}}\|.$$

Hence the normalized gradient $\mathbf{u} = Df(\mathbf{x})^{\mathsf{T}} / \|Df(\mathbf{x})^{\mathsf{T}}\|$ maximizes the directional derivative. ∎

To minimize f, move in the opposite direction, which is the direction of greatest decrease. This insight provides an important class of optimization methods called *gradient descent* methods, which are all iterative methods of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k Df(\mathbf{x}_k)^{\mathsf{T}}, \tag{12.13}$$

where the parameter $\alpha_k \in (0, \infty)$, called the *learning rate*, is chosen in various ways.

12.3.2 Exact Gradient Descent

One approach for choosing the learning rate α_k is to minimize f along the ray $\mathbf{x}_k - \alpha Df(\mathbf{x}_k)^{\mathsf{T}}$ for $\alpha > 0$. More precisely, at each step solve the one-dimensional optimization problem

$$\alpha_k = \operatorname*{argmin}_{\alpha > 0} \phi_k(\alpha), \tag{12.14}$$

where $\phi_k(\alpha) = f\left(\mathbf{x}_k - \alpha Df(\mathbf{x}_k)^{\mathsf{T}}\right)$. Finding the optimal α_k can be done using the one-dimensional minimization methods described in Section 12.2. Using the minimizer as α_k gives the maximal decrease in the direction $-Df(\mathbf{x}_k)^{\mathsf{T}}$. We call this the *method of exact gradient descent*.[49]

Example 12.3.2. Recall from Example 12.1.7 that the function $f(x, y) = x^3 - 3x^2 + y^2$ has a local minimizer at $(2, 0)$. Here we take one step of exact gradient descent, starting at the point $\mathbf{x}_0 = (x, y) = (4, 4)$. The gradient is $\begin{bmatrix} 3x^2 - 6x & 2y \end{bmatrix} = \begin{bmatrix} 24 & 8 \end{bmatrix}$, so the next iteration is

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 (24, 8)$$

[49] Some, like [CZ01], call this method *steepest descent*, but others, like [BV04], use that name for an algorithm that we do not discuss in this book.

with α_0 chosen as the minimizer of the function

$$\phi(\alpha) = f((4,4) - \alpha(24,8)) = (4-24\alpha)^3 - 3(4-24\alpha)^2 + (4-8\alpha)^2.$$

Using one of the one-dimensional methods of the previous chapter, we find that $\alpha_0 = 0.10243$ is a minimizer, which gives $\mathbf{x}_1 = (1.5415, 3.1805)$. Note that $f(\mathbf{x}_1) = 0.10244$, whereas $f(\mathbf{x}_0) = 32$, so this first step has already reduced the value of f a lot. Repeating the process continues to descend (see Proposition 12.3.3) and the sequence eventually reaches $(2.0, 4 \times 10^{-8})$ after 35 iterations.

The method of exact gradient descent does indeed descend; that is, the sequence of values $(f(\mathbf{x}_k))_{k=0}^{\infty}$ generated by the method is strictly decreasing, as the next proposition shows.

Proposition 12.3.3. *Let $f \in C^1(\mathbb{R}^n;\mathbb{R})$ with $Df(\mathbf{x}_k)^\mathsf{T} \neq \mathbf{0}$. If α_k is a minimizer of $\phi(\alpha) = f(\mathbf{x}_k - \alpha Df(\mathbf{x}_k)^\mathsf{T})$ on an interval of the form $(0,b)$, then setting $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k Df(\mathbf{x}_k)^\mathsf{T}$ gives*

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k).$$

Proof. Since α_k is a minimizer of ϕ on $(0,b)$, we have $\phi(\alpha_k) \leq \phi(\alpha)$ for all $\alpha \in (0,b)$. The chain rule shows that

$$\phi'(0) = -Df(\mathbf{x}_k)Df(\mathbf{x}_k)^\mathsf{T} = -\|Df(\mathbf{x}_k)\|^2 < 0.$$

Since f is C^1, the function $\phi(\alpha)$ is too, which implies that $\phi'(\alpha)$ is negative on some open interval $(0,\varepsilon)$. It follows that $\phi(\alpha)$ is a decreasing function on the interval $[0,\varepsilon)$. In other words, there exists $\bar{\alpha} > 0$ such that $\phi(\alpha) < \phi(0)$ for all $\alpha \in (0,\bar{\alpha}]$. Therefore, we have

$$f(\mathbf{x}_{k+1}) = \phi(\alpha_k) \leq \phi(\bar{\alpha}) < \phi(0) = f(\mathbf{x}_k). \quad \square$$

12.3.3 Exact Gradient Descent for Quadratics

For a quadratic objective function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}Q\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$ with $Q > 0$, we can find an explicit formula for α_k in the exact gradient descent method. Note that $Df(\mathbf{x}_k)^\mathsf{T} = Q\mathbf{x}_k - \mathbf{b}$. Since α_k minimizes $\phi_k(\alpha)$, the FONC implies that $\phi_k'(\alpha_k) = 0$. Thus,

$$\begin{aligned} 0 = \phi_k'(\alpha_k) &= -Df(\mathbf{x}_k - \alpha_k Df(\mathbf{x}_k)^\mathsf{T})Df(\mathbf{x}_k)^\mathsf{T} \\ &= \mathbf{b}^\mathsf{T}Df(\mathbf{x}_k)^\mathsf{T} + \alpha_k Df(\mathbf{x}_k)QDf(\mathbf{x}_k)^\mathsf{T} - \mathbf{x}_k^\mathsf{T}QDf(\mathbf{x}_k)^\mathsf{T}. \end{aligned}$$

Thus, α_k satisfies

$$\alpha_k = \frac{Df(\mathbf{x}_k)Df(\mathbf{x}_k)^\mathsf{T}}{Df(\mathbf{x}_k)QDf(\mathbf{x}_k)^\mathsf{T}}. \tag{12.15}$$

Recall that the condition number $\kappa(Q)$ of the matrix Q is $\kappa(Q) = \|Q\|\|Q^{-1}\|$; see Definition 11.2.15. Recall also that the 2-norm of a square matrix Q is the

largest singular value of Q; see Volume 1, Exercise 4.21(i). In our current setting with $Q > 0$, the eigenvalues of Q are its singular values, so $\kappa(Q) = \lambda_1/\lambda_n$, where λ_1 is the largest eigenvalue, and λ_n is the smallest. The next proposition shows that the rate of convergence of exact gradient descent is governed by $\kappa(Q)$. Some of the geometric intuition for why this is true is given in Remark 12.3.6.

Proposition 12.3.4. *For a quadratic objective function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}Q\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$ with $Q > 0$, the exact gradient descent method converges linearly with rate no worse than $1 - \kappa(Q)^{-3}$, where $\kappa(Q)$ is the condition number of the matrix Q.*

Proof. To simplify the analysis, first make the change of variables $\mathbf{x} \mapsto \mathbf{x} - \mathbf{x}^*$. This moves the optimizer to the origin and transforms the objective function into the form $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}Q\mathbf{x} + c$. Moreover, since Q is symmetric, it is orthonormally diagonalizable (see Volume 1, Theorem 4.4.7). Thus, there is a linear, orthonormal change of variables (a rigid motion of $\mathbb{R}^n$ about the origin) that makes Q diagonal. Since an orthonormal change of basis does not change the 2-norm, making this change of basis does not change anything about the convergence rate. Therefore, we may assume that $Q = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ with $\lambda_1 \geq \cdots \geq \lambda_n > 0$ (the eigenvalues are all positive because $Q > 0$). Finally, the location of the minimizer and the outcome of each step of the exact gradient descent algorithm are independent of the constant c, so we may assume that the objective $f(\mathbf{x})$ and its derivative $Df(\mathbf{x})$ have the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}Q\mathbf{x} \qquad \text{and} \qquad Df(\mathbf{x})^\mathsf{T} = Q\mathbf{x},$$

with Q diagonal. Thus (12.13) and (12.15) reduce to

$$\alpha_k = \frac{\mathbf{x}_k^\mathsf{T}Q^2\mathbf{x}_k}{\mathbf{x}_k^\mathsf{T}Q^3\mathbf{x}_k} \qquad \text{and} \qquad \mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\mathbf{x}_k^\mathsf{T}Q^2\mathbf{x}_k}{\mathbf{x}_k^\mathsf{T}Q^3\mathbf{x}_k}Q\mathbf{x}_k,$$

respectively.

To simplify notation further, drop the subscript on $\mathbf{x}_k$ and just write $\mathbf{x}_k = \mathbf{x} = (x_1, \dots, x_n)$. At stage $k+1$, the error $\|\mathbf{x}_{k+1}\|_2$ is bounded by

$$\|\mathbf{x}_{k+1}\|_2 = \left\| \left(I - \frac{\mathbf{x}^\mathsf{T}Q^2\mathbf{x}}{\mathbf{x}^\mathsf{T}Q^3\mathbf{x}}Q \right)\mathbf{x} \right\|_2 \leq \left\| I - \frac{\mathbf{x}^\mathsf{T}Q^2\mathbf{x}}{\mathbf{x}^\mathsf{T}Q^3\mathbf{x}}Q \right\|_2 \|\mathbf{x}\|_2.$$

But if a matrix is diagonal, then its 2-norm is the largest element on the diagonal. This implies

$$\begin{aligned}\|\mathbf{x}_{k+1}\|_2 &\leq \max_j \left(1 - \lambda_j \frac{\sum_{i=1}^n \lambda_i^2 x_i^2}{\sum_{i=1}^n \lambda_i^3 x_i^2}\right) \|\mathbf{x}_k\|_2 \\ &= \left(1 - \lambda_n \frac{\sum_{i=1}^n \lambda_i^2 x_i^2}{\sum_{i=1}^n \lambda_i^3 x_i^2}\right) \|\mathbf{x}_k\|_2 \leq \left(1 - \frac{\lambda_n^3}{\lambda_1^3}\right) \|\mathbf{x}_k\|_2.\end{aligned}$$

Therefore the exact gradient descent method converges linearly (or better) with rate $\mu = (1 - (\lambda_n/\lambda_1)^3) = 1 - \kappa(Q)^{-3}$. □

Remark 12.3.5. The proposition shows that when $\kappa(Q)$ is large the convergence rate is poor (close to 1), and when $\kappa(Q)$ is small, the convergence rate is good (close

to 0). In the special case that $\lambda_1 = \cdots = \lambda_n = \lambda$, we have $\kappa(Q) = 1$, and exact gradient descent converges in one step because $\alpha_k = \frac{1}{\lambda}$ and $Q\mathbf{x}_k = \lambda\mathbf{x}_k$, which implies that $\mathbf{x}_{k+1} = \mathbf{0}$.

Remark 12.3.6. The convergence result of Proposition 12.3.4 also has a geometric interpretation. At each stage of the exact gradient descent algorithm, the next direction is orthogonal to the previous one (see Exercise 12.14) and each step stops at a point that is tangent to the *level set* (set of the form $\{\mathbf{x} \mid f(\mathbf{x}) = d\}$) containing the stopping point.[50] An example of this is illustrated in Figure 12.4. If all the eigenvalues of Q are equal, then the level sets are spheres and a normal to a sphere is a radius, so it points directly toward the center of the sphere (the minimizer), and exact gradient descent converges in a single step. As long as the eigenvalues of Q are nearly equal, the level sets are nearly spheres, and each step of the algorithm gets much closer to the minimizer. But if the largest and smallest eigenvalues of Q are very different, the level sets are ellipsoids with high eccentricity, and the normals to these sets do not point toward the center; see Figure 12.4. Instead, the method can bounce back and forth repeatedly, while improving by only a small amount with each iteration.

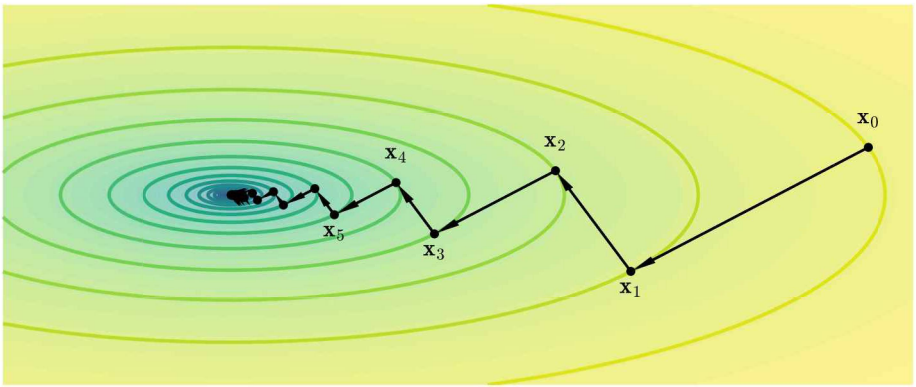


Figure 12.4. *Exact gradient descent at step k always moves in a direction orthogonal to the level set containing $\mathbf{x}_k$, and that direction is always tangent to the level set containing the stopping point $\mathbf{x}_{k+1}$. In this example the level sets are ellipses in the plane.*

12.3.4 Other Gradient Descent Methods

Exact gradient descent is not always the most efficient form of gradient descent. Even if the gradient is easy to calculate, other methods of choosing the learning rate can often give better performance. There are at least two reasons why exact gradient descent is not always the best choice. First, the time spent finding the optimal α_k is often better spent computing the next iteration in a new direction instead. And second, even if it is easy to find the optimal α_k in the line search, that may not

[50] For a function from $\mathbb{R}^2$ to $\mathbb{R}$, the level sets are like contour lines in a topographic map, and plotting them can give a good sense of the shape of the function.

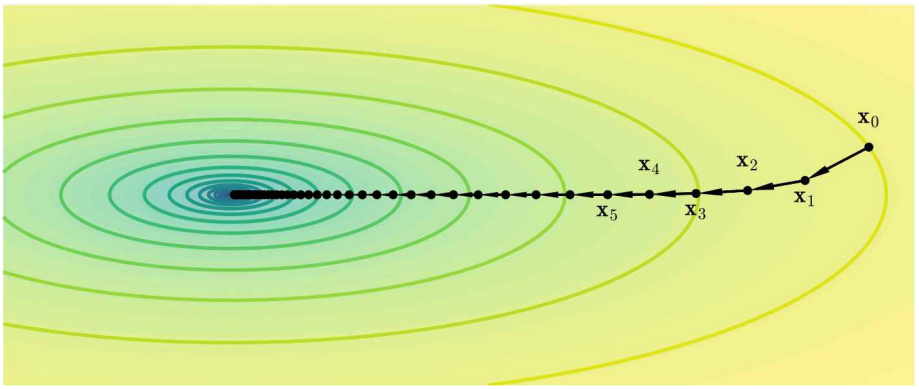


Figure 12.5. *Gradient descent with a relatively small constant learning rate generally moves in a good path toward the minimizer* $\mathbf{x}^*$*, but if the learning rate is too small, it may take many steps to actually reach the minimizer.*

actually be the best choice, because, as described in the previous subsection, if the Hessian of the objective function has a large condition number, then the graph has the shape of a narrow trough, and the method of exact gradient descent can zigzag back and forth, while descending slowly, whereas a smaller learning rate could lead to $\mathbf{x}^*$ in fewer steps.

Constant Learning Rate

One very simple way to choose the learning rate for gradient descent is just to use a small constant value a for α at every step:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - aDf(\mathbf{x}_k)^\mathsf{T}.$$

This can work well in many settings. If the objective function f is continuously differentiable, then, as the approximations $\mathbf{x}_k$ approach a minimizer, the gradient $Df(\mathbf{x}_k)^\mathsf{T}$ approaches $\mathbf{0}$ and the total length of the step $aDf(\mathbf{x}_k)^\mathsf{T}$ also becomes small. Thus, it is possible for gradient descent with constant learning rate to converge to the minimizer; see Figure 12.5 for an illustration.

The challenge in choosing the constant a is that if it is too large, the algorithm can overshoot and not converge (see Figure 12.6); but if a is too small, the algorithm could take many steps to approach the minimizer.

Descent with Backtracking

Backtracking starts with a constant learning rate, but then adjusts it to ensure descent and prevent climbing out away from the minimizer. The algorithm begins by trying the constant learning rate to see if it descends; that is, it tests whether $f(\mathbf{x}_k - aDf(\mathbf{x}_k)^\mathsf{T}) < f(\mathbf{x}_k)$. If not, it replaces a by $a/2$ and tries again, repeating until it finds the largest choice of $2^{-\ell}a$ that gives descent. Of course any value greater than 1 could be used in place of the number 2 here.

It is often beneficial to require additional conditions on the learning rate while backtracking in order to encourage faster descent. One of the best known of these

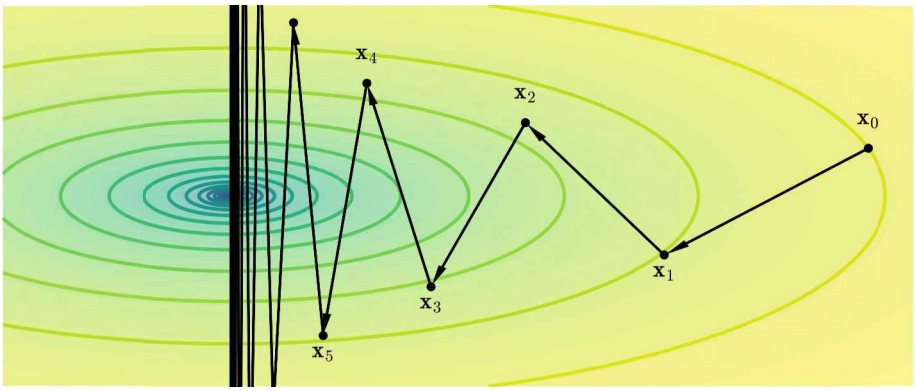


Figure 12.6. *Gradient descent with a constant learning rate that is too large can diverge. In this example, even though the gradient is pointing in the right direction at each step, the algorithm moves too far in that direction. As the iterates get farther away from the minimizer, the norm of the gradient increases, and the distance* $\|\mathbf{x}_k - \mathbf{x}_{k-1}\|$ *grows instead of converging to* 0.

conditions is the *Armijo condition*, which uses the tangent at $\mathbf{x}_k$ to give a bound on how much less $f(\mathbf{x}_{k+1})$ should be than $f(\mathbf{x}_k)$. If the chosen descent direction at $\mathbf{x}_k$ is $\mathbf{d}$, then the Armijo condition is

$$f(\mathbf{x}_k) - f(\mathbf{x}_k + a\mathbf{d}) \geq -\sigma a Df(\mathbf{x}_k)\mathbf{d} \tag{12.16}$$

for some choice of $\sigma \in (0, 1)$. In the case of gradient descent, with $\mathbf{d} = -Df(\mathbf{x}_k)^\mathsf{T}$, the Armijo condition becomes

$$f(\mathbf{x}_k) - f(\mathbf{x}_k - aDf(\mathbf{x}_k)^\mathsf{T}) \geq \sigma a \|Df(\mathbf{x}_k)\|_2^2. \tag{12.17}$$

To understand the Armijo condition geometrically, note that $Df(\mathbf{x}_k)\mathbf{d}$ is the directional derivative at $\mathbf{x}_k$ in the $\mathbf{d}$ direction, so if f were linear, then $-aDf(\mathbf{x}_k)\mathbf{d}$ is the amount that f would decrease when moving from $\mathbf{x}_k$ to $\mathbf{x}_k + a\mathbf{d}$. Of course, near a local minimum, we expect f to curve upward from the tangent plane, so an improvement of $-aDf(\mathbf{x}_k)\mathbf{d}$ is too much to expect. Thus, the Armijo condition requires only that f improve by some fraction $\sigma \in (0, 1)$ of that ideal amount.

Remark 12.3.7. Another condition that is sometimes imposed on the learning rate is the *curvature condition*, which gives a bound for how much less the slope at $\mathbf{x}_{k+1}$ should be than the slope at $\mathbf{x}_k$. Taken together, the Armijo and curvature conditions are called the *Wolfe conditions*.

12.4 Newton and Quasi-Newton Methods

Newton's method for one-dimensional optimization is discussed in Section 12.2, but Newton's method also works for higher dimensions. If the initial starting point is sufficiently close to the minimizer, then Newton's method converges quadratically, which is a much higher order of convergence than most algorithms. This means

it takes very few steps to converge and makes it the algorithm of choice in many settings.

But Newton's method also has some drawbacks. Some of the most notable of these are as follows:

(i) A good initial guess is needed; otherwise it may not converge.

(ii) It requires computing the Hessian, which can contribute significantly to the computational cost.

(iii) It requires $D^2 f(\mathbf{x}_k)$ to be positive definite at each of the points $\mathbf{x}_k$.

(iv) It requires solving a linear system with the Hessian, which has temporal complexity $O(n^3)$, where n is the dimension of the domain of f. In high dimensions this can be prohibitive.

Instead, *quasi-Newton* methods are techniques that are based on, or are similar to, Newton's method but are designed to overcome one or more of these disadvantages. We begin this section with a discussion of multivariate Newton's method and then describe some key quasi-Newton methods.

12.4.1 Newton's Method

If the second derivative $D^2 f$ of $f : \Omega \to \mathbb{R}$ is Lipschitz, then applying the zero-finding version of Newton's method (see Volume 1, Section 7.3.2) to the function $Df^{\mathsf{T}} : \mathbb{R}^n \to \mathbb{R}^n$ gives a method for finding a critical point of f.

Theorem 12.4.1. *Assume $\Omega \subset \mathbb{R}^n$ is an open neighborhood of $\mathbf{x}^*$ and $Df(\mathbf{x}^*) = \mathbf{0}$. If $D^2 f(\mathbf{x}^*) > 0$ and $D^2 f$ is Lipschitz on Ω, then the iterative map*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^{\mathsf{T}} \tag{12.18}$$

converges quadratically to $\mathbf{x}^*$ whenever $\mathbf{x}_0$ is sufficiently close to $\mathbf{x}^*$.

Proof. This follows immediately by applying the convergence result for the zero-finding Newton's method (Volume 1, Theorem 7.3.12) to the function Df^{T}. □

Remark 12.4.2. The temporal complexity of each iteration of Newton's method is $O(d + h + n^3)$, where d is the cost of computing the gradient $Df(\mathbf{x})^{\mathsf{T}}$ and h is the cost of computing the Hessian $D^2 f(\mathbf{x})$.

Nota Bene 12.4.3. Solving a linear system is usually faster and more stable than first computing the matrix inverse and then multiplying by the inverse. It still has complexity $O(n^3)$, but the leading-order complexity is better, as is the stability. We can use this to improve the speed and stability of Newton's method by breaking the computation into two steps:

(i) Solve $D^2 f(\mathbf{x}_k)\mathbf{d}_k = -Df(\mathbf{x}_k)^{\mathsf{T}}$ for $\mathbf{d}_k$.

(ii) Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$.

Example 12.4.4. Consider again the function $f(x,y) = x^3 - 3x^2 + y^2$ of Examples 12.1.7 and 12.3.2, with a local minimizer at $(2,0)$. We take one step of Newton's method starting at the point $\mathbf{x}_0 = (4,4)$. The gradient is

$$Df(\mathbf{x}_0) = \begin{bmatrix} 3x^2 - 6x & 2y \end{bmatrix} = \begin{bmatrix} 24 & 8 \end{bmatrix},$$

and the Hessian is

$$D^2 f(\mathbf{x}_0) = \begin{bmatrix} 6x - 6 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 18 & 0 \\ 0 & 2 \end{bmatrix}.$$

This gives

$$\mathbf{x}_1 = \mathbf{x}_0 - D^2 f(\mathbf{x}_0)^{-1} Df(\mathbf{x}_0)^\mathsf{T} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} - \begin{bmatrix} \frac{1}{18} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 24 \\ 8 \end{bmatrix} = \begin{bmatrix} \frac{8}{3} \\ 0 \end{bmatrix}.$$

Note that $f(\mathbf{x}_1) = -2.37037$, whereas $f(\mathbf{x}_0) = 32$. For comparison, in Example 12.3.2, we computed a single step of exact gradient descent starting at the same point and it yielded $\mathbf{x}_1^{egd} = (1.5415, 3.1805)$ with $f(\mathbf{x}_1^{egd}) = 0.10244$.

Continuing with the Newton algorithm gives $\mathbf{x}_2 = (2.1333, 0)$, and the algorithm reaches the minimizer $(2,0)$ exactly (to machine precision) after just five iterations, which is much faster than exact gradient descent.

12.4.2 Newton as Quadratic Approximation

Just as in the one-dimensional case (see Section 12.2.4 and Figure 12.3), Newton's method can be interpreted as minimizing a quadratic approximation. To see this, assume that $f : \mathbb{R}^n \to \mathbb{R}$ is a C^2 function and that $\mathbf{x}^* \in \mathbb{R}^n$ is a local minimizer of f satisfying $D^2 f(\mathbf{x}^*) > 0$. The degree-2 Taylor polynomial of f at $\mathbf{x}_k$ is

$$q(\mathbf{x}) = f(\mathbf{x}_k) + Df(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\mathsf{T} D^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k). \tag{12.19}$$

Since this is a quadratic function with positive definite Hessian, it is minimized when $Df(\mathbf{x}_k) + (\mathbf{x}_{k+1} - \mathbf{x}_k)^\mathsf{T} D^2 f(\mathbf{x}_k) = \mathbf{0}$, which (after transposing) gives (12.18) again. Hence, minimizing the quadratic approximation gives the same algorithm as Newton's method.

As in the one-dimensional case, if f is itself quadratic, then it is equal to its quadratic approximation, so Newton's method must converge in one step. As a result, we have the following proposition.

Proposition 12.4.5. *If $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} Q \mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c$, for some $Q > 0$, some $\mathbf{b} \in \mathbb{R}^n$, and some $c \in \mathbb{R}$, then Newton's method converges to the unique minimizer of f in a single step, regardless of starting point.*

12.4.3 Descent of Newton

Newton's method does not necessarily descend. But it always moves in a *descent direction*, which is a direction that descends if the step size is small enough.

Proposition 12.4.6. *Let $(\mathbf{x}_k)_{k=0}^{\infty}$ be the sequence generated by Newton's method for minimizing a given objective function f. If the Hessian $D^2 f(\mathbf{x}_k) > 0$ and $Df(\mathbf{x}_k) \neq 0$, then $\mathbf{d}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = -D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^{\mathsf{T}}$ is a* descent direction *for f, that is, there exists an $\bar{\alpha} > 0$ such that for all $\alpha \in (0, \bar{\alpha})$,*

$$f(\mathbf{x}_k + \alpha \mathbf{d}_k) < f(\mathbf{x}_k).$$

Proof. The proof is similar to that of Proposition 12.3.3. The details are Exercise 12.18. □

12.4.4 Newton with Line Search

Even though the direction chosen by Newton's method is a descent direction, the actual method may not descend because the step size chosen by Newton's method could be too large; see Figures 12.7 and 12.8. The fact that $\mathbf{d}_k = -D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)$ is a descent direction means only that f will descend in that direction if the step size is sufficiently small. If the higher derivatives of the objective function f are large, then the function f can move away from its quadratic approximation (12.19) very quickly and the minimizer $\mathbf{x}_k - D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)$ of the approximating quadratic might give a value of f that is larger than $f(\mathbf{x}_k)$. And even if it does descend, this might not be the best choice for the next iteration.

The first of the quasi-Newton methods discussed here is a simple variant on Newton's method that addresses this problem of stepping too far by changing the learning rate and letting

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k),$$

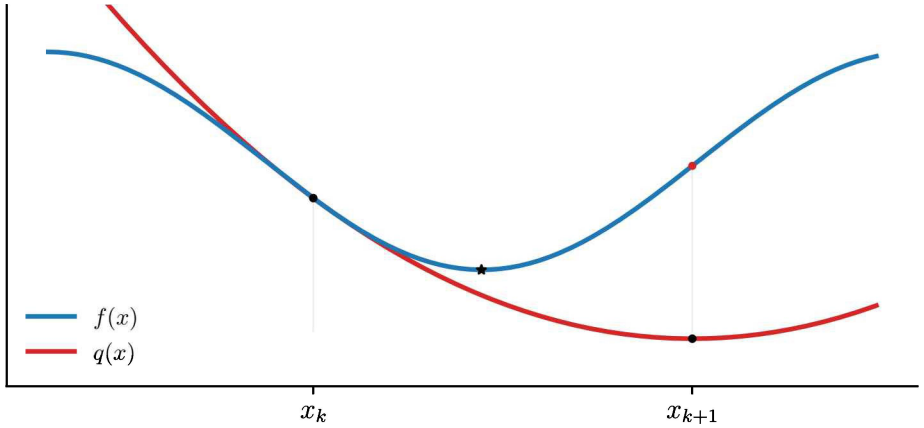


Figure 12.7. *Plot of a function f and a starting point $\mathbf{x}_k$ for which the Newton step does not descend. Instead it produces a point $\mathbf{x}_{k+1}$ with $f(\mathbf{x}_{k+1}) > f(\mathbf{x}_k)$. As long as the Hessian is positive definite at $\mathbf{x}_k$, Proposition 12.4.6 guarantees that Newton's method moves in a descent direction, which means that the objective function decreases if the step size is small enough.*

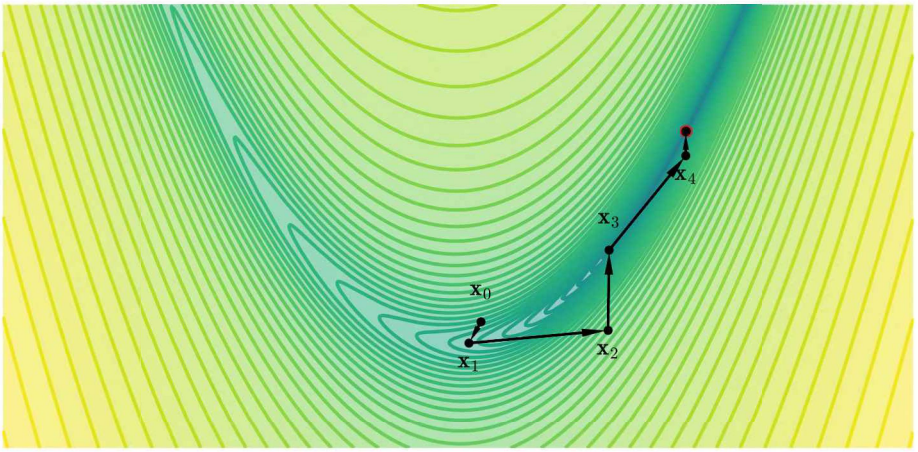


Figure 12.8. *Newton's method applied to the Rosenbrock function. Notice that the* $\mathbf{x}_2$ *and* $\mathbf{x}_4$ *steps here do not descend, but we were lucky with the initial choice* $\mathbf{x}_0$*, and the method still happened to converge.*

where the learning rate $\alpha_k \leq 1$ is chosen by one of the same methods used in gradient descent, for example, by setting $\alpha_k = \operatorname{argmin}_\alpha f(\mathbf{x}_k - \alpha D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k))$ or by simple backtracking until descent occurs.

12.4.5 Levenberg–Marquardt Modification

Newton's method requires the Hessian to be positive definite at each step, but this may not hold if the point $\mathbf{x}_k$ is too far from the minimizer. If $D^2 f(\mathbf{x}_k)$ is not positive definite, then the Newton direction $\mathbf{d}_k = -D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^\mathsf{T}$ may not be a descent direction at all, and the method may not converge. The *Levenberg–Marquardt modification* is a modification to Newton's method (hence, a quasi-Newton method) designed to deal with this problem.

Since $D^2 f(\mathbf{x}_k)$ is symmetric, it has real eigenvalues $\lambda_1, \ldots, \lambda_n$. Denote the corresponding eigenvectors by $\mathbf{v}_1, \ldots, \mathbf{v}_n$. Choose $\mu > 0$ and define the matrix $G = D^2 f(\mathbf{x}) + \mu I$. Exercise 12.22 shows that $D^2 f(\mathbf{x})$ and G share the same eigenvectors, and the eigenvalues of G are $\lambda_1 + \mu, \ldots, \lambda_n + \mu$.

Choosing μ large enough makes the eigenvalues of G all positive and hence makes G positive definite. At each step of Newton's method, replacing the Hessian with $G_k = \frac{1}{\alpha_k}(D^2 f(\mathbf{x}_k) + \mu_k I)$, for suitable positive choices of μ_k and learning rate α_k, defines a new algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k (D^2 f(\mathbf{x}_k) + \mu_k I)^{-1} Df(\mathbf{x}_k)^\mathsf{T},$$

which has the descent property for μ_k sufficiently large and α_k suitably small. This is the *Levenberg–Marquardt modification.*

Remark 12.4.7. The Levenberg–Marquardt modification can be thought of as a weighted combination of Newton's method and gradient descent because setting $\mu = 0$ with $\alpha_k = 1$ just gives Newton's method, whereas if $\mu_k \to \infty$, then this

method becomes a type of gradient descent. As a general strategy, one usually starts with μ small and increases only as needed at each step to keep G positive definite.

Remark 12.4.8. The function $D^2 f(\mathbf{x}) + \mu I$ is the Hessian of the *regularized* function $F(\mathbf{x}) = f(\mathbf{x}) + \mu\|\mathbf{x}\|_2^2$. The function F is less bumpy than f, and as μ increases F becomes more and more like the simple quadratic function $\|\mathbf{x}\|_2^2$. Algorithms minimizing F are less likely to end up in suboptimal local minima than those that minimize f. The Levenberg–Marquardt modification is similar to, but not quite the same as using Newton's method on F, because Newton's method for F would use $\mathbf{x} - D^2F(\mathbf{x})^{-1}DF(\mathbf{x})$ instead of the Levenberg–Marquardt $\mathbf{x} - D^2F(\mathbf{x})^{-1}Df(\mathbf{x})$.

12.4.6 Gauss–Newton for Nonlinear Least Squares

Another weakness of Newton's method is the fact that it requires computing the Hessian at each step, which can be computationally expensive (or unstable). The *Gauss–Newton algorithm* is a quasi-Newton method for avoiding this computation for an important class of optimization problems called *nonlinear least squares (NLS)*.

NLS problems are minimization problems where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ can be written as a sum of squares:

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x})^2 = \frac{1}{2}\mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{r}(\mathbf{x}), \tag{12.20}$$

where each $r_i : \mathbb{R}^n \to \mathbb{R}$ is a smooth function (the r_i are often called *residuals*), and the r_i can be combined to give a vector-valued function $\mathbf{r}(\mathbf{x}) = (r_1(\mathbf{x}), r_2(\mathbf{x}), \ldots, r_m(\mathbf{x}))$.

Example 12.4.9. Assume we are given a family of functions of the form $p(t;\mathbf{x}) = x_0\cos(x_1 t + x_2)$, where $\mathbf{x} = (x_0, x_1, x_2)$ is a vector of parameters. Given a collection of data points $(t_0, p_0), \ldots, (t_{m-1}, p_{m-1})$, we wish to find the parameters $\widehat{\mathbf{x}}$ that best fit the data, so that the MSE (see Definition 6.1.16) $\frac{1}{m}\sum_{i=0}^{m-1}(p(t_i;\mathbf{x}) - p_i)^2$ is minimized. This amounts to solving the optimization problem

$$\widehat{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}} \frac{1}{m}\sum_{i=0}^{m-1}(p(t_i;\mathbf{x}) - p_i)^2 = \operatorname*{argmin}_{\mathbf{x}} \sum_{i=0}^{m-1}(x_0\cos(x_1 t_i + x_2) - p_i)^2.$$

Setting $r_i(\mathbf{x}) = p(t_i;\mathbf{x}) - p_i = x_0\cos(x_1 t_i + x_2) - p_i$ for each i expresses this as an NLS problem.

More generally, for any family of functions $p(t;\mathbf{x})$ parametrized by a vector $\mathbf{x}$, finding $\widehat{\mathbf{x}}$ to fit data points $(t_0, p_0), \ldots, (t_{m-1}, p_{m-1})$ is an NLS problem with $r_i(\mathbf{x}) = p(t_i;\mathbf{x}) - p_i$.

To use Newton's method for (12.20), first compute the derivative and Hessian. The derivative of f is given by $Df(\mathbf{x}) = \mathbf{r}(\mathbf{x})^\mathsf{T} D\mathbf{r}(\mathbf{x})$; see Example 10.2.10. The Hessian of f is given by

$$D^2 f(\mathbf{x}) = D\mathbf{r}(\mathbf{x})^\mathsf{T} D\mathbf{r}(\mathbf{x}) + \underbrace{\sum_{i=1}^{m} r_i(\mathbf{x}) D^2 r_i(\mathbf{x})}_{S(\mathbf{x})}, \tag{12.21}$$

as computed in Exercise 10.13. Thus, Newton's method for this optimization problem is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (D\mathbf{r}(\mathbf{x}_k)^\mathsf{T} D\mathbf{r}(\mathbf{x}_k) + S(\mathbf{x}_k))^{-1} D\mathbf{r}(\mathbf{x}_k)^\mathsf{T} \mathbf{r}(\mathbf{x}_k), \tag{12.22}$$

where $S(\mathbf{x}) = \sum_{i=1}^{m} r_i(\mathbf{x}) D^2 r_i(\mathbf{x})$.

The Gauss–Newton algorithm takes advantage of the special structure of NLS problems to avoid computing the second derivatives. Specifically, the *Gauss–Newton algorithm* method drops the second term $S(\mathbf{x})$ in (12.22) giving

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (D\mathbf{r}(\mathbf{x}_k)^\mathsf{T} D\mathbf{r}(\mathbf{x}_k))^{-1} D\mathbf{r}(\mathbf{x}_k)^\mathsf{T} \mathbf{r}(\mathbf{x}_k). \tag{12.23}$$

At the minimizer $\mathbf{x}^*$, the FONC implies that $Df(\mathbf{x}^*) = \mathbf{r}(\mathbf{x}^*)^\mathsf{T} D\mathbf{r}(\mathbf{x}^*) = \mathbf{0}$. If $\mathbf{r}(\mathbf{x}^*) = \mathbf{0}$, then $S(\mathbf{x}^*) = 0$, and if $\mathbf{x}$ is near $\mathbf{x}^*$, then all entries of $S(\mathbf{x})$ must be small. In either case, dropping S from (12.22) gives a close approximation to Newton's method whenever $S(\mathbf{x})$ is small. In such situations the convergence rate of the Gauss–Newton method is close to quadratic, but without the cost of computing the Hessian. For this reason, the Gauss–Newton algorithm is the standard optimization method for NLS problems. But if $S(\mathbf{x})$ is too large, the method may not converge, so it is still important to have a good initial guess.

Example 12.4.10. Assume that $m+1$ range finders (devices that can measure distance, but no other location information) are placed at points $\mathbf{a}_0, \ldots, \mathbf{a}_m \in \mathbb{R}^n$, and they all record the distance to an object located at point $\mathbf{x}^{\text{true}} \in \mathbb{R}^n$ as $d_i = \|\mathbf{a}_i - \mathbf{x}^{\text{true}}\| + \varepsilon_i$, where the measurement error ε_i has $\mathbb{E}[\varepsilon_i] = 0$. If the true location $\mathbf{x}^{\text{true}} \in \mathbb{R}^n$ of the object is unknown, we can use the measurements of the range finders to try to get an estimate $\widehat{\mathbf{x}}$ for $\mathbf{x}^{\text{true}}$ as

$$\widehat{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}} \sum_{i=0}^{m} \left(d_i - \|\mathbf{x} - \mathbf{a}_i\|\right)^2.$$

Letting $r_i(\mathbf{x}) = d_i - \|\mathbf{x} - \mathbf{a}_i\|$ makes this into an NLS problem.

To make this concrete, consider the case depicted in Figure 12.9, where the object is located at the point $\mathbf{x}^{\text{true}} = (3, 3)$ and there are four range finders, located at points $\mathbf{a}_0 = (0, 0)$, $\mathbf{a}_1 = (1, 1)$, $\mathbf{a}_2 = (2, 0)$, and $\mathbf{a}_3 = (-1, 3)$. Exercise 12.23 is to code up the Gauss-Newton algorithm and apply it to this problem

in the case that the (noisy) measured distances from the range finders to the object are given by $\mathbf{d} = (3.88506517, 2.87540403, 3.10537735, 3.99674185)$. The values of the objective

$$f(\mathbf{x}) = \sum_{i=0}^{3} r_i^2(\mathbf{x}) = \sum_{i=0}^{3} (d_i - \|\mathbf{x} - \mathbf{a}_i\|)^2$$

are plotted in Figure 12.9 as a contour plot. In this example the Gauss–Newton method converges (within machine precision) in eight steps to $\widehat{\mathbf{x}} = (2.9546367, 2.88618843)$, which is not quite equal to the true value because of error in the initial measurements.

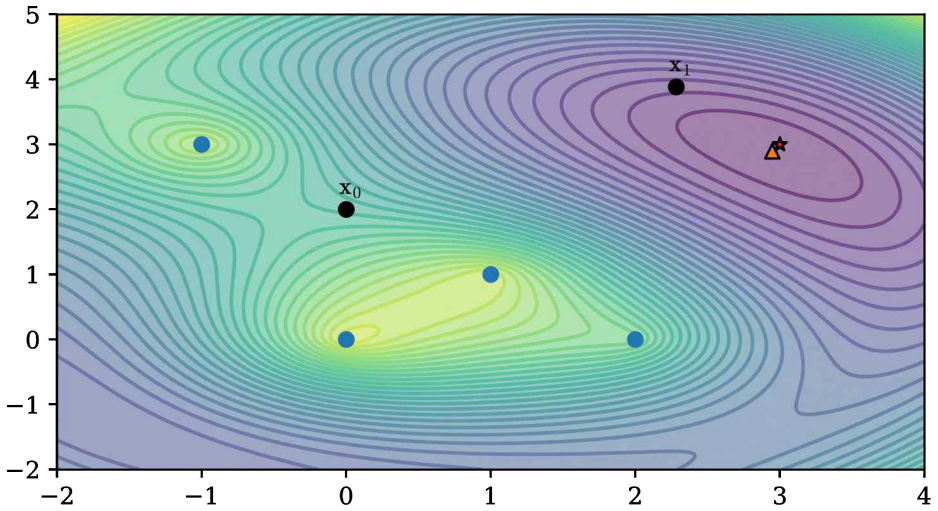


Figure 12.9. *Contour plot of the objective function for the NLS problem of locating* $\mathbf{x}^{\text{true}}$ *(the star) given noisy measurements from four range finders (blue dots), as described in Example 12.4.10. Darker colors correspond to smaller values of the objective. The minimizer* $\widehat{\mathbf{x}}$ *(orange triangle) of the objective function is quickly reached using the Gauss–Newton method, starting at* $\mathbf{x}_0 = (2, 0)$*. The minimizer* $\widehat{\mathbf{x}}$ *does not quite agree with the true location because of the noise in the measurements.*

Remark 12.4.11. Although $D\mathbf{r}(\mathbf{x}_k)^T D\mathbf{r}(\mathbf{x}_k)$ is always positive semidefinite, it may not be full rank and hence not invertible (and therefore not positive definite). To remedy this, we can make a Levenberg–Marquardt modification to Gauss–Newton, replacing $D\mathbf{r}(\mathbf{x}_k)^T D\mathbf{r}(\mathbf{x}_k)$ by $D\mathbf{r}(\mathbf{x}_k)^T D\mathbf{r}(\mathbf{x}_k) + \mu I$. This is essentially the same as approximating $S(\mathbf{x})$ by μI in (12.22). When applied to the Gauss–Newton algorithm, the Levenberg–Marquardt modification is often called the *Levenberg–Marquardt algorithm*.

Example 12.4.12. When applying Gauss–Newton to minimize the quadratic function

$$f(\mathbf{x}) = \|\mathbf{b} - A\mathbf{x}\|_2^2 = \mathbf{x}^\mathsf{T} A^\mathsf{T} A\mathbf{x} - 2\mathbf{x}^\mathsf{T} A\mathbf{b} + \mathbf{b}^\mathsf{T}\mathbf{b},$$

it is straightforward to verify that $S(\mathbf{x}) = 0$, and the Gauss–Newton method is the same as Newton's method. Thus, Gauss–Newton converges for quadratic functions of this form in a single step.

12.5 The BFGS Method

Newton's method converges rapidly, but it requires solving a linear system involving the Hessian matrix at each step; and both computing the Hessian and solving the resulting system can be expensive, especially in high dimensions. The *BFGS method*, named for Broyden, Fletcher, Goldfarb, and Shanno, is a quasi-Newton method that approximates the Hessian in a clever way to allow the inverse of the approximated Hessian to be computed easily. This approximation generally comes at the cost of slower convergence, but the decreased computational complexity of each iteration is often well worth it.

12.5.1 Low-Rank Updating

Newton's method approximates the objective function f with the second-order approximation

$$q_k(\mathbf{x}) = f(\mathbf{x}_k) + Df(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\mathsf{T} D^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$$

and solves for the minimizer, which is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^\mathsf{T}.$$

The BFGS method also approximates the objective function f with a quadratic approximation

$$q_k(\mathbf{x}) = f(\mathbf{x}_k) + Df(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\mathsf{T} A_k (\mathbf{x} - \mathbf{x}_k) \tag{12.24}$$

for a special choice of $A_k > 0$, and the update is similarly given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - A_k^{-1} Df(\mathbf{x}_k)^\mathsf{T}. \tag{12.25}$$

The idea behind BFGS (and similar methods) is to choose A_k to be a low-rank update of A_{k-1} of the form

$$A_k = A_{k-1} + \mathbf{a}_k \mathbf{a}_k^\mathsf{T} \quad \text{or} \quad A_k = A_{k-1} + \mathbf{a}_k \mathbf{a}_k^\mathsf{T} + \mathbf{b}_k \mathbf{b}_k^\mathsf{T}, \tag{12.26}$$

where $\mathbf{a}_k, \mathbf{b}_k \in \mathbb{R}^n$ are chosen to give A_k good properties for optimization. The main advantage of using low-rank updates like these for A_k is that they are easy to invert if the inverse of A_{k-1} is already known. Typically, the process begins either with $A_0 = D^2 f(\mathbf{x}_0)$ or with some easily invertible, positive definite approximation of $D^2 f(\mathbf{x}_0)$, like I. The initial quadratic term A_0 must be positive definite, and this ensures that all the subsequent A_k remain positive definite.

12.5.2 Inverting Low-Rank Updates

If A_{k-1}^{-1} is already known from the previous step, then the following identity, due to Sherman and Morrison, gives an explicit formula for A_k^{-1}, after which the multiplication $A_k^{-1} Df(\mathbf{x}_n)^\mathsf{T}$ only requires $\sim 2n^2$ FLOPs.

Proposition 12.5.1 (Sherman–Morrison Formula). *For any invertible $n \times n$ matrix A, and any vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, the matrix $A + \mathbf{u}\mathbf{v}^\mathsf{T}$ is invertible if and only if $1 + \mathbf{v}^\mathsf{T} A^{-1} \mathbf{u} \neq 0$. If $A + \mathbf{u}\mathbf{v}^\mathsf{T}$ is invertible, then*

$$(A + \mathbf{u}\mathbf{v}^\mathsf{T})^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}A^{-1}}{1 + \mathbf{v}^\mathsf{T}A^{-1}\mathbf{u}}. \tag{12.27}$$

Proof. The proof is Exercise 12.25. □

The benefit of using the Sherman–Morrison formula in this setting is that A^{-1} is already known. The time cost of computing $(A+\mathbf{u}\mathbf{v}^\mathsf{T})^{-1}$ via (12.27) is dominated by the computation $A^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}A^{-1}$. This can be computed as $\left(A^{-1}\mathbf{u}\right)\left(\mathbf{v}^\mathsf{T}A^{-1}\right)$, which consists of two matrix-vector products (with leading-order complexity $\sim 2n^2$ each) followed by a vector-vector outer product, which has leading-order complexity $\sim n^2$. Thus, the total complexity of computing the inverse using Sherman–Morrison is $\sim 5n^2$ FLOPs. Compare this to the cost of inverting directly, or rather solving the corresponding linear system, which is $\sim \frac{2}{3}n^3$ FLOPs.

12.5.3 Two Requirements

Matching Gradient

The quadratic Taylor approximation of f at $\mathbf{x}_k$ matches both the gradient and the Hessian of f at $\mathbf{x}_k$. The main point of the BFGS method is to try to get a good quadratic approximation (12.24) of f without computing the Hessian, so instead of making the Hessian match, a reasonable alternative assumption is that the gradient of the approximation should match the gradient of f at both $\mathbf{x}_k$ and at $\mathbf{x}_{k-1}$, that is, $Dq_k(\mathbf{x}_k) = Df(\mathbf{x}_k)$ and $Dq_k(\mathbf{x}_{k-1}) = Df(\mathbf{x}_{k-1})$. Since the first is automatic, we need only consider the second. Differentiating (12.24) at $\mathbf{x}_{k-1}$ gives

$$Dq_k(\mathbf{x}_{k-1}) = Df(\mathbf{x}_k) + (\mathbf{x}_{k-1} - \mathbf{x}_k)^\mathsf{T} A_k,$$

and so the matching gradient condition becomes

$$Df(\mathbf{x}_k) - Df(\mathbf{x}_{k-1}) = (\mathbf{x}_k - \mathbf{x}_{k-1})^\mathsf{T} A_k. \tag{12.28}$$

We call (12.28) the *matching gradient constraint.*

One possible choice for the matrix A_k is to take the best symmetric rank-one update of A_{k-1} (meaning that $A_k = A_{k-1} + \mathbf{v}\mathbf{v}^\mathsf{T}$ for some $\mathbf{v} \in \mathbb{R}^n$) that satisfies the matching gradient constraint (12.28). This naïve method is called *Broyden's method.* Unfortunately it does not work very well because rank-one updates do not always make A_k positive definite, which means the method does not necessarily descend, as we discuss below.

Positive Definite for Descent

In order to guarantee that each new direction $A_k^{-1}Df(\mathbf{x}_k)^\mathsf{T}$ in (12.25) is a descent direction, the matrix A_k should be positive definite. Specifically, for any optimization method of the form $\mathbf{x}_{k+1} = \mathbf{x}_k - A_k^{-1}Df(\mathbf{x}_k)^\mathsf{T}$, Taylor's formula (Theorem 10.3.8) for $f(\mathbf{x}_{k+1})$ expanded around $\mathbf{x}_k$ gives

$$\begin{aligned} f(\mathbf{x}_{k+1}) &= f(\mathbf{x}_k) + Df(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) + o(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|) \\ &= f(\mathbf{x}_k) - Df(\mathbf{x}_k)(A_k^{-1}Df(\mathbf{x}_k)^\mathsf{T}) + o(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|). \end{aligned}$$

So if $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|$ is sufficiently small and $A_k > 0$, then we have

$$f(\mathbf{x}_k) > f(\mathbf{x}_{k+1}).$$

However, if A_k is not positive definite, then we cannot expect this inequality to hold.

12.5.4 BFGS

The *Broyden–Fletcher–Goldfarb–Shanno (BFGS) method* brings together all the ideas discussed so far in this section. To do this, it makes a positive definite rank-two update instead of the rank-one update of Broyden's method. Specifically, for an objective function f, points $\mathbf{x}_0, \ldots, \mathbf{x}_k$, and matrix $A_{k-1} > 0$, set

$$A_k = A_{k-1} + \frac{\mathbf{y}_k\mathbf{y}_k^\mathsf{T}}{\mathbf{y}_k^\mathsf{T}\mathbf{s}_k} - \frac{A_{k-1}\mathbf{s}_k\mathbf{s}_k^\mathsf{T}A_{k-1}}{\mathbf{s}_k^\mathsf{T}A_{k-1}\mathbf{s}_k}, \tag{12.29}$$

where

$$\mathbf{y}_k = Df(\mathbf{x}_k)^\mathsf{T} - Df(\mathbf{x}_{k-1})^\mathsf{T} \qquad \text{and} \qquad \mathbf{s}_k = \mathbf{x}_k - \mathbf{x}_{k-1}. \tag{12.30}$$

The next point $\mathbf{x}_{k+1}$ is now computed using (12.25).

Theorem 12.5.2. *Let $B(\mathbf{x}^*, \delta)$ be an open ball containing $\mathbf{x}_k$ and $\mathbf{x}_{k-1}$. Assume that $f \in C^2(B(\mathbf{x}^*, \delta); \mathbb{R})$ and $D^2 f(\mathbf{x}) > 0$ for all $\mathbf{x} \in B(\mathbf{x}^*, \delta)$. If $A_{k-1} > 0$ and satisfies the matching gradient constraint* (12.28)*, then the matrix A_k given in* (12.29) *is a positive definite, rank-two (or less) update of A_{k-1} satisfying the matching gradient constraint* (12.28)*.*

Proof. It is straightforward to check that A_k is symmetric whenever A_{k-1} is. It is also straightforward to check that $A_k\mathbf{s}_k = \mathbf{y}_k$, so A_k satisfies the matching gradient constraint (12.28). Moreover, the difference $A_k - A_{k-1}$ is the sum of two rank-one matrices, so it has rank at most two.

To show that $A_k > 0$ we first prove the special case $\mathbf{s}_k^\mathsf{T}A_k\mathbf{s}_k > 0$. Begin by rewriting this as

$$\mathbf{s}_k^\mathsf{T}A_k\mathbf{s}_k = \mathbf{y}_k^\mathsf{T}\mathbf{s}_k = (Df(\mathbf{x}_k) - Df(\mathbf{x}_{k-1}))(\mathbf{x}_k - \mathbf{x}_{k-1}) > 0. \tag{12.31}$$

By Taylor's theorem (Theorem 10.3.8), for each pair $\mathbf{x}, \mathbf{x}' \in B(\mathbf{x}^*, \delta)$, we have

$$f(\mathbf{x}') = f(\mathbf{x}) + Df(\mathbf{x})(\mathbf{x}' - \mathbf{x}) + R_2, \tag{12.32}$$

where

$$R_2 = \int_0^1 (1-t)(\mathbf{x}' - \mathbf{x})^\mathsf{T} D^2 f(\mathbf{x} + t(\mathbf{x}' - \mathbf{x}))(\mathbf{x}' - \mathbf{x})\, dt.$$

Since $D^2 f(\mathbf{x} + t(\mathbf{x}' - \mathbf{x})) > 0$ for all $t \in [0,1]$, we have $R_2 > 0$ and

$$f(\mathbf{x}') - f(\mathbf{x}) > Df(\mathbf{x})(\mathbf{x}' - \mathbf{x})$$

for any $\mathbf{x}, \mathbf{x}' \in B(\mathbf{x}^*, \delta)$. This gives

$$\begin{aligned} f(\mathbf{x}_{k-1}) - f(\mathbf{x}_k) &> Df(\mathbf{x}_k)(\mathbf{x}_{k-1} - \mathbf{x}_k) \qquad \text{and} \\ f(\mathbf{x}_k) - f(\mathbf{x}_{k-1}) &> Df(\mathbf{x}_{k-1})(\mathbf{x}_k - \mathbf{x}_{k-1}), \end{aligned}$$

which combine to give $Df(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}_{k-1}) > Df(\mathbf{x}_{k-1})(\mathbf{x}_k - \mathbf{x}_{k-1})$, and thus (12.31) holds.

Since $A_{k-1} > 0$, it defines an inner product $\langle \mathbf{w}, \mathbf{x} \rangle_* = \mathbf{w}^\mathsf{T} A_{k-1} \mathbf{x}$. To see that A_k is positive definite, compute $\mathbf{z}^\mathsf{T} A_k \mathbf{z}$ for any $\mathbf{z} \in \mathbb{R}^n$ with $\mathbf{z} \neq \mathbf{0}$ as follows:

$$\begin{aligned} \mathbf{z}^\mathsf{T} A_k \mathbf{z} &= \mathbf{z}^\mathsf{T} A_{k-1} \mathbf{z} + \frac{\mathbf{z}^\mathsf{T} \mathbf{y}_k \mathbf{y}_k^\mathsf{T} \mathbf{z}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k} - \frac{\mathbf{z}^\mathsf{T} A_{k-1} \mathbf{s}_k \mathbf{s}_k^\mathsf{T} A_{k-1} \mathbf{z}}{\mathbf{s}_k^\mathsf{T} A_{k-1} \mathbf{s}_k} \\ &= \langle \mathbf{z}, \mathbf{z} \rangle_* + \frac{\mathbf{z}^\mathsf{T} \mathbf{y}_k \mathbf{y}_k^\mathsf{T} \mathbf{z}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k} - \frac{\langle \mathbf{s}_k, \mathbf{z} \rangle_*^2}{\langle \mathbf{s}_k, \mathbf{s}_k \rangle_*}. \end{aligned}$$

By the Cauchy–Schwarz inequality, we have $\langle \mathbf{s}_k, \mathbf{z} \rangle_*^2 \leq \langle \mathbf{s}_k, \mathbf{s}_k \rangle_* \langle \mathbf{z}, \mathbf{z} \rangle_*$, with equality if and only if $\mathbf{z}$ and $\mathbf{s}_k$ are linearly dependent. This gives

$$\mathbf{z}^\mathsf{T} A_k \mathbf{z} \geq \langle \mathbf{z}, \mathbf{z} \rangle_* + \frac{\mathbf{z}^\mathsf{T} \mathbf{y}_k \mathbf{y}_k^\mathsf{T} \mathbf{z}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k} - \frac{\langle \mathbf{s}_k, \mathbf{s}_k \rangle_* \langle \mathbf{z}, \mathbf{z} \rangle_*}{\langle \mathbf{s}_k, \mathbf{s}_k \rangle_*} = \frac{(\mathbf{z}^\mathsf{T} \mathbf{y}_k)^2}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k} \geq 0,$$

where the last inequality follows from (12.31).

If $\mathbf{z}$ and $\mathbf{s}_k$ are linearly independent, then the first inequality above is strict, giving $\mathbf{z}^\mathsf{T} A_k \mathbf{s} > 0$, as desired. If $\mathbf{z}$ and $\mathbf{s}_k$ are linearly dependent, then since $\mathbf{z} \neq 0$ and $\mathbf{s}_k \neq \mathbf{0}$, we must have $\mathbf{z} = a\mathbf{s}_k$ for some $a \neq 0$, and

$$\mathbf{z}^\mathsf{T} A_k \mathbf{z} = a^2 \mathbf{s}_k^\mathsf{T} A_k \mathbf{s}_k > 0,$$

where the inequality again follows from (12.31). □

The last step of the BFGS algorithm is to use the Sherman–Morrison identity twice to invert A_k to get the following (see Exercise 12.28):

$$A_k^{-1} = A_{k-1}^{-1} + \frac{(\mathbf{s}_k^\mathsf{T} \mathbf{y}_k + \mathbf{y}_k^\mathsf{T} A_{k-1}^{-1} \mathbf{y}_k) \mathbf{s}_k \mathbf{s}_k^\mathsf{T}}{(\mathbf{s}_k^\mathsf{T} \mathbf{y}_k)^2} - \frac{A_{k-1}^{-1} \mathbf{y}_k \mathbf{s}_k^\mathsf{T} + \mathbf{s}_k \mathbf{y}_k^\mathsf{T} A_{k-1}^{-1}}{\mathbf{s}_k^\mathsf{T} \mathbf{y}_k}. \tag{12.33}$$

A Python implementation of the complete BFGS algorithm is given in Algorithm 12.1.

```python
import numpy as np
from numpy.linalg import inv, norm

def BFGS(f, df, x0, A0, max_iter=40, tol=1e-8):
    """Minimize f using BFGS, given the derivative df, an
    initial guess x0, and an initial approx A0 of D^2f(x0).
    """

    # Initialize
    done = False
    iters = 0        # Count the number of iterations
    A_inv = inv(A0)  # Initial approximate inverse Hessian
    x = x0 - A_inv @ df(x0)       # x_1
    s = x - x0                    # s_1

    while not done:  # Main BFGS loop
        y = df(x) - df(x0)        # Update y
        sy = s @ y      # This product is used several times
        Ay = A_inv @ y  # This product is used several times
        # Approximiate the new inverse Hessian
        A_inv = (A_inv +((sy + y @ Ay)/sy**2) * np.outer(s,s)
                 - (np.outer(Ay,s) + np.outer(s,Ay))/sy )
        x0 = x
        x = x0 - A_inv @ df(x0)   # Update x.
        s = x - x0                # Update s.
        iters += 1
        # Stopping criteria
        done = ((norm(s) < tol) or
                (norm(df(x)) < tol) or
                (np.abs(f(x) - f(x0)) < tol) or
                (iters >= max_iter))
    return x
```

Algorithm 12.1. *Python implementation of the BFGS algorithm for finding a local minimizer of* `f`*, given the gradient function* `df`*, an initial guess* `x0`*, and an initial approximation* `A0` *of the Hessian. The function* `df` *can usually be constructed easily from* `f` *using algorithmic differentiation (see Section 11.4.4). Note that the Python matrix multiplication operator* `@` *automatically transposes vectors to be the (assumed) correct shape, so a product like* `s @ y` *gives the inner product and* `A_inv @ df(x0)` *computes the usual matrix-vector product even though* `df(x0)` *is a row vector. Computing an outer product like* $\mathbf{s}\mathbf{y}^\mathsf{T}$ *requires* `np.outer(s,y)`*.*

Example 12.5.3. Consider, yet again, the function $f(x,y) = x^3 - 3x^2 + y^2$ of Examples 12.4.4, 12.3.2, and 12.1.7, with a local minimizer at $(2,0)$. We apply the BFGS method, starting at the point $\mathbf{x}_0 = (4,4)$. As computed in Example 12.4.4, the gradient and Hessian are

$$Df(\mathbf{x}_0)^\mathsf{T} = \begin{bmatrix} 24 \\ 8 \end{bmatrix} \qquad \text{and} \qquad D^2 f(\mathbf{x}_0) = \begin{bmatrix} 18 & 0 \\ 0 & 2 \end{bmatrix}.$$

(i) Setting $A_0 = D^2 f(\mathbf{x}_0)$ makes the first step of BFGS the same as Newton, so by Example 12.4.4 we have $\mathbf{x}_1 = (\frac{8}{3}, 0)$.

(ii) Proceeding with the next BFGS step we have

(a) $\mathbf{y}_1 = Df(\mathbf{x}_1)^\mathsf{T} - Df(\mathbf{x}_0)^\mathsf{T} = (-\frac{56}{3}, -8)$ and $\mathbf{s}_1 = \mathbf{x}_1 - \mathbf{x}_0 = (-\frac{4}{3}, -4)$.

(b) Compute $\mathbf{s}_1^\mathsf{T}\mathbf{y}_1 = \frac{512}{9}$, $A_0^{-1}\mathbf{y}_1 = (-\frac{28}{27}, -4)$, and $\mathbf{y}_1^\mathsf{T} A_0^{-1}\mathbf{y}_1 = \frac{4160}{81}$ and plug these into (12.33) to get

$$A_1^{-1} = \frac{1}{256}\begin{bmatrix} 17 & 3 \\ 3 & 121 \end{bmatrix}.$$

(c) This yields $\mathbf{x}_2 = \mathbf{x}_1 - A_1^{-1} Df(\mathbf{x}_1) = (2.3125, -0.0625)$ with $f(\mathbf{x}_2) = -3.67261$.

Compare this result to Newton's method, which yields $\mathbf{x}_2^{\text{Newt}} = (2.1333, 0)$ with $f(\mathbf{x}_2^{\text{Newt}}) = -3.94432$. At this step BFGS does not give quite as good an estimate of $\mathbf{x}^* = (2,0)$ as Newton's method, but it is not much worse.

Continuing with BFGS on this optimization problem yields the solution $\mathbf{x}_9 = (2, -1 \times 10^{-13})$ in nine steps, which is more than Newton's five steps, but much fewer (with a better final result) than exact gradient descent's 35 steps. Since the dimension of this problem is so low, BFGS provides no computational benefit at each iteration over Newton's method in this example. But in higher-dimensional problems, BFGS can be much cheaper per iteration than Newton; see Remark 12.5.4.

Remark 12.5.4. The BFGS algorithm is very effective for high-dimensional, unconstrained optimization problems with a dense Hessian. On these problems it generally runs much faster than Newton's method, despite taking more iterations to converge. After the initial step, each iteration of the BFGS method runs in $O(n^2 + d)$ time, where d is the cost of evaluating the gradient $Df(\mathbf{x})^\mathsf{T}$. When n is large, this is a big improvement over the $O(d + h + n^3)$ cost of computing $D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^\mathsf{T}$ in Newton's method.

Remark 12.5.5. The standard modifications to the learning rate for Newton's method and gradient descent can be applied to the BFGS algorithm. That is, we can change (12.25) to

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k A_k^{-1} Df(\mathbf{x}_k)^\mathsf{T}$$

for an appropriate choice of α_k. We can either use optimal line search

$$\alpha_k = \operatorname*{argmin}_{\alpha} f(\mathbf{x}_k - \alpha A_k^{-1} Df(\mathbf{x}_k)^\mathsf{T}) \tag{12.34}$$

or use a simple backtracking algorithm to find a choice of α_k that results in descent (or compliance with the Armijo, Wolfe, or other conditions).

Remark 12.5.6. After several iterations the matrices A_k can become very small, and their inverses A_k^{-1} very large, which can make the algorithm unstable. In such cases a standard approach is to restart the method, either with $A_k = D^2 f(\mathbf{x}_k)$ or with an easily invertible, positive definite matrix like $A_k = I$.

Remark 12.5.7. Storing the approximation A_k^{-1} in memory can be costly if n is large. There is a variant of BFGS, called *limited memory-BFGS* or *L-BFGS*, for dealing with this problem.

12.5.5 *Sherman–Morrison–Woodbury

The Sherman–Morrison formula (12.27) is a special case of a more general formula called the *Sherman–Morrison–Woodbury* formula, which has many uses in applied and computational mathematics.

Proposition 12.5.8 (Sherman–Morrison–Woodbury). *Let A be a nonsingular $n \times n$ matrix, B an $n \times \ell$ matrix, C a nonsingular $\ell \times \ell$ matrix, and D an $\ell \times n$ matrix. We have*

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}. \tag{12.35}$$

Proof. The proof is Exercise 12.32. □

Vista 12.5.9. Many techniques in machine learning boil down to choosing parameters $\mathbf{w} \in \mathbb{R}^m$ for a family of functions $f(\mathbf{x}; \mathbf{w})$ in such a way that the predictions made by the resulting function closely approximate some data set of the form $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, meaning that $\widehat{y}_i = f(\mathbf{x}_i; \mathbf{w})$ should be close to y_i, on average. The distance between the data and the predictions is typically measured by some "loss" function of the form $\ell(y, \widehat{y})$, for example, $\ell(y, \widehat{y}) = (y - \widehat{y})^2$. Thus the problem to solve is

$$\operatorname*{minimize}_{\mathbf{w} \in \mathbb{R}^m} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(\mathbf{x}_i; \mathbf{w})),$$

where the dimension m of the parameter space may be large.

Low-memory BFGS and gradient descent (and a variant of it called *stochastic gradient descent*) are very important numerical methods for optimization in machine learning.

12.6 Conjugate-Gradient Methods

Throughout this chapter, we have seen an inverse relationship between the rate of convergence and the amount of computing required per iteration. For example, Newton's method and the quasi-Newton methods usually converge faster but come at a higher cost per iteration than gradient-descent methods, which usually converge more slowly but at a lower cost per iteration. In this section, we consider a "middle ground" class of algorithms called *conjugate-gradient methods*, which typically require more iterations than Newton methods but at less cost per iteration, and fewer iterations than gradient descent methods, albeit at a higher cost per iteration.

One of the primary applications of conjugate-gradient methods is solving large linear systems of the form

$$A\mathbf{x} = \mathbf{b}, \tag{12.36}$$

where $A > 0$. As we see in Exercise 12.3, solving such a system is equivalent to minimizing the quadratic objective $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} A\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x}$. There is a standard conjugate-gradient algorithm for solving such systems, and when we refer to *the conjugate-gradient method* we usually mean this standard choice. But for higher-degree (nonquadratic) optimization problems, there are many variants of conjugate-gradient methods (see Section 12.6.3 for more on the nonquadratic case).

In exact arithmetic (the absence of round-off error), the conjugate-gradient method is guaranteed to minimize a quadratic objective $f : \mathbb{R}^n \to \mathbb{R}$ in n iterations and thus can be considered a direct method (meaning that it produces an exact solution with a finite number of iterations). In the presence of round-off error, however, the conjugate-gradient method will not actually terminate, but continuing to iterate can give successively better approximations. We show in Section 12.7 that the convergence is linear with a rate (see Definition 12.2.1) that is related to the matrix condition number $\kappa(A)$, that is, a better conditioned A gives faster convergence.

Each iteration of the conjugate-gradient method requires one matrix-vector multiplication of the form $A\mathbf{d}$ for some direction $\mathbf{d}$ and the rest of the computations have a cost of $O(n)$. If the matrix A is sparse with only m entries, where $m \ll n^2$, then each matrix-vector multiplication costs only $O(m)$ FLOPs, and thus solving the entire system (12.36) costs only $O(mn+n^2)$ FLOPs. This can be much cheaper than the $O(n^3)$ cost of using the LU or QR decompositions. In other words, the conjugate-gradient method is a good sparse solver.

Fortunately, the conjugate-gradient method can usually be terminated in fewer than n iterations, because each step moves in the direction of maximal improvement, and once the most important directions have been handled, the resulting intermediate solution is often good enough. If the algorithm can reach a satisfactory approximation in $K \ll n$ iterations, then the total time complexity to solve the system (12.36) is $O(mK + nK)$.

12.6.1 Conjugate Directions

Recall that exact gradient descent can suffer from the problem of bouncing back and forth inefficiently when the Hessian A has a large condition number $\kappa(A)$; see Figure 12.4. The conjugate-gradient method seeks to remedy this by finding better directions of travel.

Geometric Motivation

As a first step to understanding the conjugate-gradient method, consider the special case where the minimizer is at the origin and $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}\Lambda\mathbf{x}$, where Λ is diagonal and positive definite. The level sets of f are ellipsoids with all their axes agreeing with the coordinate axes.

In this case, start at any point $\mathbf{x}_0 = (a_1, a_2, \ldots, a_n)$, but instead of moving in the direction of the negative gradient, move in a direction parallel to the first standard basis vector $\pm\mathbf{e}_1$. It is straightforward to check, and easy to see geometrically, that the minimizer of $\phi(\alpha) = f(\mathbf{x}_0 + \alpha\mathbf{e}_1)$ occurs at $\alpha = -a_1$, so that $\mathbf{x}_1 = (0, a_2, \ldots, a_n)$. Move now in a direction parallel to $\mathbf{e}_2$. A line search to find the optimizer gives $\mathbf{x}_2 = (0, 0, a_3, \ldots, a_n)$. Continuing in this manner is called *coordinate descent*, and in the special case that Λ is diagonal, it arrives at the minimizer (the origin) in n steps or fewer.

Unfortunately, if $f(\mathbf{x}) = \mathbf{x}^\mathsf{T}A\mathbf{x}$, where A is not diagonal, then coordinate descent can suffer from the same bouncing phenomenon that exact gradient descent suffers from. But a change of basis can solve this problem. Let $A = P^\mathsf{T}\Lambda P$ for some invertible P and a diagonal matrix $\Lambda > 0$. Since A is positive definite, it is orthonormally diagonalizable, which shows that A can always be written as $P^\mathsf{T}\Lambda P$ for some invertible P and diagonal Λ, but we do not require that P be orthonormal for the rest of this discussion.

Let $\mathbf{d}_1, \ldots, \mathbf{d}_n$ be the columns of P^{-1}. Changing basis in $\mathbb{R}^n$ from the standard basis $S = [\mathbf{e}_1, \ldots, \mathbf{e}_n]$ to the basis $T = [\mathbf{d}_1, \ldots, \mathbf{d}_n]$ means that the coordinates $[\mathbf{x}]_S$ of $\mathbf{x}$ in the standard basis can be written as P^{-1} times the coordinates in the eigenbasis: $[\mathbf{x}]_S = P^{-1}[\mathbf{x}]_T$. Thus, f can be expressed as

$$f(\mathbf{x}) = \frac{1}{2}[\mathbf{x}]_S^\mathsf{T}A[\mathbf{x}]_S = \frac{1}{2}\left(P^{-1}[\mathbf{x}]_T\right)^\mathsf{T}P^\mathsf{T}\Lambda P\left(P^{-1}[\mathbf{x}]_T\right) = \frac{1}{2}[\mathbf{x}]_T^\mathsf{T}\,\Lambda\,[\mathbf{x}]_T$$

in the new coordinates. This means that coordinate descent, performed in terms of the new basis, must reach the optimizer in n steps or fewer. Expressed in terms of the original coordinates, this means that moving first in the direction $\mathbf{d}_1$ and then the direction $\mathbf{d}_2$ and so forth, will reach the optimizer in n steps or fewer.

Conjugate Directions

Any two of the vectors $\mathbf{d}_i$ and $\mathbf{d}_j$ described above satisfy

$$\mathbf{d}_i^\mathsf{T}A\mathbf{d}_j = \mathbf{d}_i^\mathsf{T}P^\mathsf{T}\Lambda P\mathbf{d}_j = \mathbf{e}_i^\mathsf{T}\Lambda\mathbf{e}_j = 0$$

whenever $i \neq j$. This property is important enough to deserve a name and a formal definition.

Definition 12.6.1. *Assume $A \in M_n(\mathbb{R})$ is positive definite. Nonzero vectors $\mathbf{d}_1, \ldots, \mathbf{d}_k$ are called* A-conjugate *if*

$$\mathbf{d}_i^\mathsf{T}A\mathbf{d}_j = 0 \tag{12.37}$$

for all $i \neq j$.

Remark 12.6.2. Condition (12.37) is equivalent to saying the vectors $\mathbf{d}_1, \dots, \mathbf{d}_k$ are orthogonal in the weighted inner product $\langle \mathbf{x}, \mathbf{y} \rangle_A = \mathbf{x}^\mathsf{T} A \mathbf{y}$. Thus, any set of A-conjugate vectors is linearly independent by Corollary 3.2.5 of Volume 1.

Conjugate directions are useful because, as the previous discussion suggests, moving toward the minimizer in direction $\mathbf{d}_1$, followed by moving in direction $\mathbf{d}_2$, and so forth, must reach the minimizer in n steps or fewer.

Proposition 12.6.3 (Conjugate Directions). *Assume the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is quadratic*

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} A \mathbf{x} - \mathbf{x}^\mathsf{T} \mathbf{b} + c, \tag{12.38}$$

with $A > 0$, $\mathbf{b} \in \mathbb{R}^n$, and $c \in \mathbb{R}$. Given any starting point $\mathbf{x}_0$ and a sequence $\mathbf{d}_1, \dots, \mathbf{d}_n$ of A-conjugate directions, define $\mathbf{r}_k$, α_k, and $\mathbf{x}_k$ iteratively for each $k \in \{1, \dots, n\}$ as

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_{k-1}, \qquad \alpha_k = \frac{\mathbf{d}_k^\mathsf{T} \mathbf{r}_k}{\mathbf{d}_k^\mathsf{T} A \mathbf{d}_k}, \qquad \text{and} \qquad \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k. \tag{12.39}$$

The vector $\mathbf{r}_k$ is called the kth residual. *For each k the quantity α_k is the minimizer of the function $\phi_k(\alpha) = f(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k)$, and $\mathbf{x}_n$ is the unique minimizer of f, that is, $\mathbf{x}_n = \mathbf{x}^*$.*

Proof. The proof is Exercise 12.35. □

Remark 12.6.4. The algorithm only requires the use of $\mathbf{x}_{k-1}$ and $\mathbf{d}_k$ at step k. None of the points $\mathbf{x}_0, \dots, \mathbf{x}_{k-2}$ and none of the other directions $\mathbf{d}_j$ for $j \neq k$ is needed to compute $\mathbf{r}_k$, α_k, and $\mathbf{x}_k$. This means this algorithm can be used without remembering or knowing any of the other $\mathbf{d}_j$, provided we know that each $\mathbf{d}_k$ is A-conjugate to all the other $\mathbf{d}_j$.

Given the A-conjugate vectors $\mathbf{d}_1, \dots, \mathbf{d}_n$, this algorithm determines the minimizer of f in n steps or fewer. Thus we need a fast way to compute the A-conjugate vectors. A naïve way to produce these would be, as discussed above, to factor A as $P^\mathsf{T} \Lambda P$ for some diagonal Λ with invertible P, and then let the directions $\mathbf{d}_1, \dots, \mathbf{d}_n$ be the columns of P^{-1}. But doing this is at least as difficult as solving the system $A\mathbf{x} = \mathbf{b}$.

Alternatively, one could start with any basis and use the Gram–Schmidt process with the inner product $\langle \cdot, \cdot \rangle_A$, as described in Volume 1, Section 3.3. If the initial basis is the standard basis, doing this is essentially equivalent to finding the QR decomposition of A, which could be used to backsolve for $\mathbf{x}$ in the system $A\mathbf{x} = \mathbf{b}$. Therefore, finding a basis of A-conjugate directions in this way is also not easier than solving the system $A\mathbf{x} = \mathbf{b}$ via the QR decomposition.

In the following subsection we present a fast method, called the *conjugate-gradient method*, of obtaining a sequence of A-conjugate vectors.

12.6.2 Conjugate-Gradient Method

The *conjugate-gradient method* is a clever approach to constructing a set of A-conjugate directions. It does this by using, at step k, the previous direction $\mathbf{d}_{k-1}$

and the negative gradient $-Df(\mathbf{x}_{k-1})^\mathsf{T}$ to define the next A-conjugate direction. As discussed above, this means that for quadratic objectives in exact arithmetic, it is guaranteed to converge in n steps. Thus, if run to completion, it solves the linear system $A\mathbf{x} = \mathbf{b}$ exactly. But, of course, we use floating-point arithmetic, not exact arithmetic, and on very large systems we usually cannot or do not want to let it run all the way to completion because a good approximate solution can usually be found in fewer than n steps.

There are three main reasons that the conjugate-gradient method works well as an iterative algorithm:

(i) At each step there are many possible A-conjugate directions, but the algorithm chooses the one that is closest to the current negative gradient, and moving in that direction tends to reduce the value of f as much as possible.

(ii) The algorithm requires only the previous direction $\mathbf{d}_{k-1}$ in order to compute the next direction and the next approximation $\mathbf{x}_k$. This means that each iteration is cheap to compute and only one direction needs to be stored or computed at a time.

(iii) Because of the previous two steps, the algorithm can be terminated before all n steps are complete—often long before.

As an iterative algorithm, the method converges linearly with each step, and the rate of convergence is related to the condition number $\kappa(A)$, where a better conditioned A gives faster convergence; see Section 12.7.

The conjugate-gradient method merges gradient descent and the conjugate-direction method. It constructs the A-conjugate directions $\mathbf{d}_1, \dots, \mathbf{d}_n$ by applying Gram–Schmidt with the inner product $\langle \cdot, \cdot \rangle_A$ to the negative gradient sequence $-Df(\mathbf{x}_0)^\mathsf{T}, -Df(\mathbf{x}_1)^\mathsf{T}, \dots, -Df(\mathbf{x}_{n-1})^\mathsf{T}$. This differs from exact gradient descent because instead of proceeding in the steepest direction $-Df(\mathbf{x}_k)^\mathsf{T}$, it projects out, via Gram–Schmidt, the directions that have already been used.

Given an initial point $\mathbf{x}_0$, the algorithm moves in the direction $-Df(\mathbf{x}_0)^\mathsf{T}$ to the optimal point on that line (this is the same as exact gradient descent), but at each subsequent step, it A-orthogonally projects away the directions that have already been used before and then moves to the optimal point in the remaining direction.

Remark 12.6.5. If the objective $f : \mathbb{R}^n \to \mathbb{R}$ is quadratic, then the conjugate-gradient method reaches the minimizer in n steps or fewer. In two dimensions, as in Figure 12.10, this means the minimizer is reached by the second step. Compare this to Newton's method, which always converges in a single step for a quadratic objective; but in high dimensions that one step costs more than all of the conjugate-gradient steps combined.

The Gram–Schmidt process normally requires computing inner products of the new direction $-Df(\mathbf{x}_k)^\mathsf{T}$ with all the previous A-conjugate directions $\mathbf{d}_1, \dots, \mathbf{d}_{k-1}$. An important reason for the usefulness of the conjugate-gradient method is that it completely avoids computing most of these inner products, thanks to the following lemma.

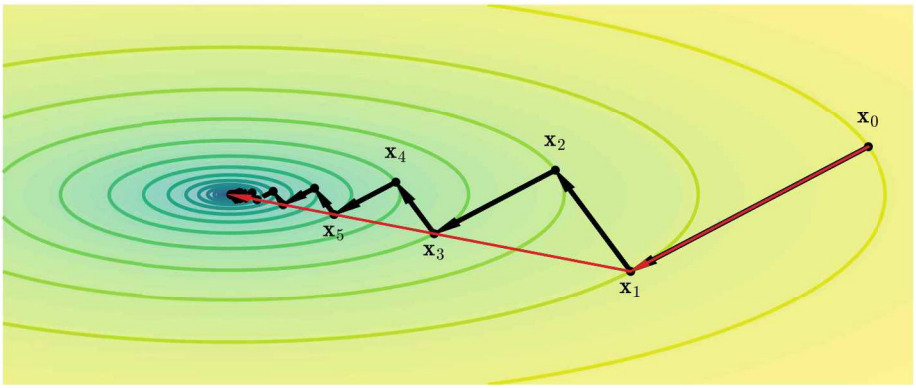


Figure 12.10. *The conjugate-gradient method (red) avoids much of the bouncing back and forth that occurs with exact gradient descent (black). The first step of the conjugate-gradient method is the same as exact gradient descent, but the second step moves in a direction that is A-conjugate to the first, the third step moves in a direction that is A-conjugate to both the first and the second, and so forth.*

Lemma 12.6.6. *Given a quadratic objective function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} A\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$ and a set of A-conjugate directions $\mathbf{d}_1, \ldots, \mathbf{d}_k$, the residual $\mathbf{r}_k$ computed by the conjugate direction method* (12.39) *satisfies $\mathbf{r}_k = -Df(\mathbf{x}_{k-1})^\mathsf{T}$ and*

$$\mathbf{d}_i^\mathsf{T}\mathbf{r}_k = 0 \tag{12.40}$$

for all $i \in \{1, \ldots, k-1\}$.

Proof. The fact that $\mathbf{r}_k = -Df(\mathbf{x}_{k-1})^\mathsf{T}$ is a straightforward computation. Next we prove that for any $k \in \{2, \ldots, n\}$ we have

$$\mathbf{d}_{k-1}^\mathsf{T}\mathbf{r}_k = 0, \tag{12.41}$$

as follows:

$$\begin{aligned}
\mathbf{d}_{k-1}^\mathsf{T}\mathbf{r}_k &= \mathbf{d}_{k-1}^\mathsf{T}(\mathbf{b} - A\mathbf{x}_{k-1}) \\
&= \mathbf{d}_{k-1}^\mathsf{T}\mathbf{b} - \mathbf{d}_{k-1}^\mathsf{T}A(\mathbf{x}_{k-2} + \alpha_{k-1}\mathbf{d}_{k-1}) \\
&= \mathbf{d}_{k-1}^\mathsf{T}(\mathbf{b} - A\mathbf{x}_{k-2}) - \alpha_{k-1}\mathbf{d}_{k-1}^\mathsf{T}A\mathbf{d}_{k-1} \\
&= \mathbf{d}_{k-1}^\mathsf{T}(\mathbf{b} - A\mathbf{x}_{k-2}) - \mathbf{d}_{k-1}^\mathsf{T}\mathbf{r}_{k-1} = 0.
\end{aligned}$$

The rest of the proof is by induction. For $k = 1$, there is nothing to prove, and the case of $k = 2$ follows from (12.41). Assume that for some $k \geq 2$ we have $\mathbf{d}_i^\mathsf{T}\mathbf{r}_k = 0$ for all $i = 1, 2, \ldots, k-1$. Since

$$\mathbf{r}_{k+1} - \mathbf{r}_k = A(\mathbf{x}_{k-1} - \mathbf{x}_k) = -\alpha_k A\mathbf{d}_k \qquad \text{and} \qquad \alpha_k = \frac{\mathbf{r}_k^\mathsf{T}\mathbf{d}_k}{\mathbf{d}_k^\mathsf{T}A\mathbf{d}_k}, \tag{12.42}$$

it follows that $\mathbf{d}_i^\mathsf{T}\mathbf{r}_{k+1} = \mathbf{d}_i^\mathsf{T}\mathbf{r}_k - \alpha_k\mathbf{d}_i^\mathsf{T}A\mathbf{d}_k = 0$ for $i = 1, \ldots, k-1$, by the induction hypothesis and A-conjugacy of $\mathbf{d}_i$ with $\mathbf{d}_k$. Finally, $\mathbf{d}_k^\mathsf{T}\mathbf{r}_{k+1} = 0$ by (12.41). Therefore, $\mathbf{d}_i^\mathsf{T}\mathbf{r}_{k+1} = 0$ for all $i = 1, 2, \ldots, k$. ∎

The conjugate-gradient method starts with $\mathbf{x}_0$ and computes

$$\mathbf{d}_1 = \mathbf{r}_1 = -Df(\mathbf{x}_0)^\mathsf{T} = \mathbf{b} - A\mathbf{x}_0,$$

which gives $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_1 \mathbf{d}_1$, where α_1 is computed by (12.42). At each subsequent step it computes $\mathbf{r}_k = -Df(\mathbf{x}_{k-1})^\mathsf{T} = \mathbf{b} - A\mathbf{x}_{k-1}$ and then performs one step of the Gram–Schmidt process, using the inner product $\langle \cdot, \cdot \rangle_A$, projecting out all the previous directions $\mathbf{d}_1, \dots, \mathbf{d}_{k-1}$ to get $\mathbf{d}_k$ as

$$\mathbf{d}_k = \mathbf{r}_k - \operatorname{proj}_{\operatorname{span}(\mathbf{d}_1,\dots,\mathbf{d}_{k-1})} \mathbf{r}_k = \mathbf{r}_k - \operatorname{proj}_{\mathbf{d}_{k-1}} \mathbf{r}_k = \mathbf{r}_k - \frac{\mathbf{d}_{k-1}^\mathsf{T} A \mathbf{r}_k}{\mathbf{d}_{k-1}^\mathsf{T} A \mathbf{d}_{k-1}} \mathbf{d}_{k-1}, \quad (12.43)$$

where $\operatorname{proj}_{\mathscr{K}} \mathbf{r}$ denotes the orthogonal projection, with respect to the inner product $\langle \cdot, \cdot \rangle_A$, of $\mathbf{r}$ onto a subspace $\mathscr{K}$, and where the second equality follows from Lemma 12.6.6. Note that the usual Gram–Schmidt process would also normalize the vector $\mathbf{d}_k$, but that is an unnecessary step, since we do not need the A-conjugate vectors to be orthonormal. After $\mathbf{d}_k$ is constructed, α_k is computed by (12.42), and $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$. The entire process is summarized in Algorithm 12.2.

If $\mathbf{r}_k = \mathbf{0}$ for any k, then $A\mathbf{x}_{k-1} = \mathbf{b}$, and the minimizer has been found: $\mathbf{x}_{k-1} = \mathbf{x}^*$. The sequence $\mathbf{d}_1, \dots, \mathbf{d}_n$ in the algorithm above is A-conjugate by construction. This fact is important enough to be its own theorem.

Theorem 12.6.7. *In the conjugate-gradient method, the set $\{\mathbf{d}_i\}_{i=1}^n$ is A-conjugate.*

Because of Theorem 12.6.7, the conjugate-gradient method (Algorithm 12.2) is a special case of the conjugate direction method (Proposition 12.6.3) and hence must converge in n or fewer steps for quadratic objective functions.

Complexity of the Conjugate-Gradient Method

The computational complexity of each step involves several inner products with complexity $O(n)$ and two matrix-vector multiplications: $A\mathbf{x}_{k+1}$ and $A\mathbf{d}_k$. Sparse matrix-vector multiplication can generally be performed in $O(m)$ FLOPs, where m is the number of nonzero entries of A. So each step has complexity $O(m+n)$, and running the algorithm to completion would take n steps for a total of $O(n^2 + nm)$ FLOPs. If A is not sparse or $m \approx n^2$, then the entire process has complexity $\sim 4n^3$, which is six times slower than solving the linear system directly by LU decomposition. But if n is large and $m \ll n^2$, this is much faster than direct methods.

In situations where n is very large, running the algorithm to completion is not usually advantageous because (i) the approximation is often good enough after fewer steps, (ii) it may be prohibitively expensive to carry out all n steps, and (iii) numerical stability issues often prevent the algorithm from giving the exact answer after n steps anyway.

12.6.3 *Conjugate-Gradient Method for Nonquadratic Problems

The conjugate-gradient method can also be applied to more general, nonquadratic objective functions. As with Newton's method, the idea is to fit a quadratic

```python
import numpy as np
from numpy.linalg import norm

def conjugate_gradient(A, b, x0, tol=1e-8, max_iters=None):
    """Solve A x = b by minimizing 1/2 x.T A x - b.T x using
    the conjugate gradient algorithm, starting at the initial
    guess of x0."""

    if not max_iters:
        max_iters = len(b) # Theoretical max number of steps.

    # Initialize
    x = x0
    d = r = (b - A @ x) # First conj dir is the residue.
    done = (norm(r) < tol)  # Stop if the reside is small.
    iters = 0

    while not done:
        dA = d @ A   # This product is used often.
        alpha = r @ d / (dA @ d)
        x = x + alpha * d  # Updated x.
        r = b - A @ x      # Updated residue.
        beta = (dA @ r)/(dA @ d)
        d = r - beta * d   # Next conjugate direction.
        iters += 1
        #Stopping criteria
        done = (norm(r) < tol or iters >= max_iters)
    return x
```

Algorithm 12.2. *Implementation in Python of the conjugate-gradient method for solving the linear system* $A\mathbf{x} = \mathbf{b}$ *by minimizing a quadratic objective function of the form* (12.38).

approximation to the objective function using a second-order Taylor polynomial. The Hessian of the function is required for the Taylor polynomial. But one of the main reasons for using the conjugate-gradient method is to avoid computing the Hessian (otherwise we'd just use Newton's method), so it is beneficial to use a variant of the conjugate-gradient method that doesn't require the Hessian.

The only places the Hessian A occurs in the basic (quadratic) conjugate-gradient method are when computing α_k and β_k. But α_k can be found by minimizing the function $\phi_k(\alpha) = f(\mathbf{x}_k + \alpha\mathbf{d}_k)$, and this can be accomplished with a line search, which can be done without A.

For β_k there are several approximation formulas that do not require A and can be used in place of the original conjugate-gradient method above. We list these formulas here but omit the proofs. For details of their derivations see the references in the notes for this chapter.

In each of these cases set $\mathbf{r}_k = -Df(\mathbf{x}_{k-1})^{\mathsf{T}}$.

Hestenes–Stiefel Formula

$$\beta_k = \frac{\mathbf{r}_{k+1}^{\mathsf{T}}(\mathbf{r}_{k+1} - \mathbf{r}_k)}{\mathbf{d}_k^{\mathsf{T}}(\mathbf{r}_{k+1} - \mathbf{r}_k)}.$$

Polak–Ribière Formula

$$\beta_k = \frac{\mathbf{r}_{k+1}^{\mathsf{T}}(\mathbf{r}_{k+1} - \mathbf{r}_k)}{\mathbf{r}_k^{\mathsf{T}}\mathbf{r}_k}.$$

Fletcher–Reeves Formula

$$\beta_k = \frac{\mathbf{r}_{k+1}^{\mathsf{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathsf{T}}\mathbf{r}_k}.$$

These modifications to the conjugate-gradient method do not converge in n steps when applied to functions that are not quadratic. Instead, they should run until meeting a suitable stopping criterion. Moreover, the A-conjugacy of the direction vectors tends to deteriorate over time. Thus, it is common to reinitialize the direction to the negative gradient after every n steps or so.

12.7 *Convergence of the Conjugate-Gradient Method

In this section we show that the conjugate-gradient method, when considered as an iterative algorithm for optimizing a quadratic function, converges linearly. We assume throughout that the conjugate-gradient method has been applied to the quadratic objective[51]

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathsf{T}}A\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x}, \qquad \mathbf{x} \in \mathbb{R}^n, \tag{12.44}$$

where $A > 0$. Following Algorithm 12.2, we start with $\mathbf{x}_0$ and produce nonzero residuals $\mathbf{r}_1, \ldots, \mathbf{r}_m$, the values $\alpha_1, \ldots, \alpha_m$, and conjugate directions $\mathbf{d}_1, \ldots, \mathbf{d}_m$. If using exact arithmetic, then for some $m \leq n$, we have that $\mathbf{r}_{m+1} = \mathbf{0}$ and $\mathbf{x}_m = \mathbf{x}^*$ is the global minimizer of f. However, when using finite-precision arithmetic, the sequence iterates, converging linearly to $\mathbf{x}^*$, and stops only once a terminal condition is met. In theory m could be any positive integer, but in practice $m \leq n$.

To begin, we need the idea of a *Krylov subspace* of $\mathbb{R}^n$; see also Volume 1, Section 13.2.

Definition 12.7.1. *For $A \in M_n(\mathbb{R})$ and $\mathbf{y} \in \mathbb{R}^n$, the kth* Krylov subspace *of A generated by $\mathbf{y}$ is*

$$\mathscr{K}_k(A, \mathbf{y}) = \operatorname{span}(\mathbf{y}, A\mathbf{y}, A^2\mathbf{y}, \ldots, A^{k-1}\mathbf{y}).$$

The residuals $\mathbf{r}_1, \ldots, \mathbf{r}_k$ and the conjugate directions $\mathbf{d}_1, \ldots, \mathbf{d}_k$ all lie in the Krylov subspace generated by $\mathbf{r}_1$, as the next lemma shows.

[51] Since the minimizer of f is the same as the minimizer of $f - c$, it suffices to assume that $c = 0$.

Lemma 12.7.2. *For all $k \in \{1, \dots, n\}$ we have*

$$\operatorname{span}(\mathbf{d}_1, \dots, \mathbf{d}_k) = \operatorname{span}(\mathbf{r}_1, \dots, \mathbf{r}_k) = \mathscr{K}_k(A, \mathbf{r}_1). \tag{12.45}$$

Proof. First, note that left-multiplying the definition of $\mathbf{x}_k$ (12.39) by A gives

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A \mathbf{d}_k. \tag{12.46}$$

We now prove (12.45) by induction. The case $k = 1$ follows because $\mathbf{d}_1 = \mathbf{r}_1$. For the induction step assume that (12.45) holds for $k = \ell - 1$. This implies that

$$A\mathbf{d}_{\ell-1} \in A\mathscr{K}_{\ell-1}(A, \mathbf{r}_1) = \operatorname{span}(A\mathbf{r}, \dots, A\mathbf{r}^{\ell}) \subset \mathscr{K}_{\ell}(A, \mathbf{r}_1),$$

which, when combined with (12.46), gives

$$\mathbf{r}_{\ell} \in \mathscr{K}_{\ell}(A, \mathbf{r}_1)$$

and

$$\operatorname{span}(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_{\ell}) \subset \mathscr{K}_{\ell}(A, \mathbf{r}_1).$$

Since $d_{\ell} = \mathbf{r}_{\ell} - \beta_{\ell}\mathbf{d}_{\ell-1}$, we also have

$$\operatorname{span}(\mathbf{d}_1, \dots, \mathbf{d}_{\ell}) \subset \operatorname{span}(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_{\ell}) \subset \mathscr{K}_{\ell}(A, \mathbf{r}_1).$$

But since the $\mathbf{d}_i$ are A-conjugate and nonzero, they are linearly independent and thus $\dim(\operatorname{span}(\mathbf{d}_1, \dots, \mathbf{d}_{\ell})) = \ell \geq \dim \mathscr{K}_{\ell}(A, \mathbf{r}_1)$, and hence

$$\operatorname{span}(\mathbf{d}_1, \dots, \mathbf{d}_{\ell}) = \operatorname{span}(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_{\ell}) = \mathscr{K}_{\ell}(A, \mathbf{r}_1). \quad \square$$

Lemma 12.7.3. *The point $\mathbf{x}_k$ is the minimizer of f among all $\mathbf{x}$ that lie in the set $\mathbf{x}_0 + \mathscr{K}_k(A, \mathbf{r}_1) = \{\mathbf{x}_0 + \mathbf{q} \mid \mathbf{q} \in \mathscr{K}_k(A, \mathbf{r}_1)\}$; that is,*

$$\mathbf{x}_k = \operatorname*{argmin}_{\mathbf{x} \in \mathbf{x}_0 + \mathscr{K}_k(A, \mathbf{r}_1)} f(\mathbf{x}). \tag{12.47}$$

Moreover, $\mathbf{x}_k$ is also the minimizer of $\varepsilon(\mathbf{x})^2 = \|\mathbf{x} - \mathbf{x}^*\|_A^2$ on that same set; that is

$$\mathbf{x}_k = \operatorname*{argmin}_{\mathbf{x} \in \mathbf{x}_0 + \mathscr{K}_k(A, \mathbf{r}_1)} \varepsilon(\mathbf{x})^2. \tag{12.48}$$

Proof. Note first that $A\mathbf{x}^* = \mathbf{b}$, which implies that

$$\begin{aligned} \varepsilon(\mathbf{x})^2 &= (\mathbf{x} - \mathbf{x}^*)^{\mathsf{T}} A (\mathbf{x} - \mathbf{x}^*) \\ &= \mathbf{x}^{\mathsf{T}} A \mathbf{x} - 2\mathbf{x}^{\mathsf{T}} A \mathbf{x}^* + (\mathbf{x}^*)^{\mathsf{T}} A \mathbf{x}^* \\ &= \mathbf{x}^{\mathsf{T}} A \mathbf{x} - 2\mathbf{x}^{\mathsf{T}} \mathbf{b} + (\mathbf{x}^*)^{\mathsf{T}} \mathbf{b} \\ &= 2f(\mathbf{x}) + \mathbf{b}^{\mathsf{T}} \mathbf{x}^*. \end{aligned}$$

Therefore a point $\mathbf{x}^*$ is the minimizer of f if and only if $\mathbf{x}^*$ is the minimizer of ε^2. Thus it suffices to prove (12.48).

Since $\mathbf{x}_k \in \mathbf{x}_0 + \mathscr{K}_k(A, \mathbf{r}_1)$ we have

$$\mathbf{x}_0 + \mathscr{K}_k(A, \mathbf{r}_1) = \mathbf{x}_k + \mathscr{K}_k(A, \mathbf{r}_1) = \mathbf{x}_k + \operatorname{span}(\mathbf{d}_1, \dots, \mathbf{d}_k).$$

Therefore, any $\mathbf{x} \in \mathbf{x}_0 + \mathscr{K}_k(A, \mathbf{r}_1)$ can be written as $\mathbf{x} = \mathbf{x}_k + \mathbf{v}$ for some $\mathbf{v} \in \mathscr{K}_k(A, \mathbf{r}_1) = \operatorname{span}(\mathbf{d}_1, \ldots, \mathbf{d}_k)$.

Since $\mathbf{x}^* = \mathbf{x}_m$ for some $m \geq k$ we have $\mathbf{x}^* = \mathbf{x}_k + \sum_{i=k+1}^{m} \alpha_i \mathbf{d}_i$. By Exercise 12.39 we have

$$\varepsilon(\mathbf{x}^*)^2 = \varepsilon(\mathbf{x}_k)^2 + \|\mathbf{v}\|_A^2, \tag{12.49}$$

which shows that ε^2 is minimized precisely when $\|\mathbf{v}\|_A^2 = 0$. Hence $\mathbf{x}_k$ is the minimizer of ε^2. □

Lemma 12.7.4. *Let $\varepsilon_k = \varepsilon(\mathbf{x}_k)$ and $\mathscr{P}_k$ be the set of all polynomials of degree at most k with constant term 1. If $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of the $n \times n$ matrix A, then*

$$\frac{\varepsilon_k}{\varepsilon_0} \leq \min_{q \in \mathscr{P}_k} \max_j |q(\lambda_j)|.$$

Proof. The Krylov subspace $\mathscr{K}_k(A, \mathbf{r}_1)$ can be written as

$$\mathscr{K}_k(A, \mathbf{r}_1) = \{p(A)\mathbf{r}_1 \mid p \in \mathbb{R}[x; k-1]\},$$

where $\mathbb{R}[x; k-1]$ is the vector space of all polynomials of degree at most $k-1$, and $p(A) = \sum_{i=0}^{k-1} a_i A^i$ for any polynomial $p(\mathbf{x}) = \sum_{i=0}^{k-1} a_i x^i \in \mathbb{R}[x; k-1]$. Therefore, another way to express Lemma 12.7.3 is to say that $\mathbf{x}_k = \mathbf{x}_0 + p_k(A)\mathbf{r}_1$, where

$$p_k = \operatorname*{argmin}_{p \in \mathbb{R}[x;k-1]} f(\mathbf{x}_0 + p(A)\mathbf{r}_1) = \operatorname*{argmin}_{p \in \mathbb{R}[x;k-1]} \varepsilon(\mathbf{x}_0 + p(A)\mathbf{r}_1). \tag{12.50}$$

Since $\mathbf{r}_1 = A\mathbf{x}_0 - \mathbf{b} = A\mathbf{x}_0 - A\mathbf{x}^* = A(\mathbf{x}_0 - \mathbf{x}^*)$, we have

$$\mathbf{x}_k - \mathbf{x}^* = \mathbf{x}_0 + p_k(A)\mathbf{r}_1 - \mathbf{x}^* = (I + Ap_k(A))(\mathbf{x}_0 - \mathbf{x}^*).$$

Therefore, $\mathbf{x}_k - \mathbf{x}^* = q_k(A)(\mathbf{x}_0 - \mathbf{x}^*)$, where $q_k(x) = 1 + xp_k(x)$. Since $\mathscr{P}_k$ is the set of all polynomials of degree at most k with constant term 1, we can write

$$\varepsilon_k = \|\mathbf{x}_k - \mathbf{x}^*\|_A = \min_{q \in \mathscr{P}_k} \|q(A)(\mathbf{x}_0 - \mathbf{x}^*)\|_A.$$

Exercise 12.40 shows that

$$\varepsilon_k^2 \leq \|q(A)(\mathbf{x}_0 - \mathbf{x}^*)\|_A^2 \leq \varepsilon_0^2 \max_j (q(\lambda_j))^2. \tag{12.51}$$

Therefore we have

$$\frac{\varepsilon_k^2}{\varepsilon_0^2} \leq \min_{q \in \mathscr{P}_k} \max_j (q(\lambda_j))^2. \quad \square$$

Lemma 12.7.5. *The error $\varepsilon_k = \|\mathbf{x}_k - \mathbf{x}^*\|_A$ satisfies*

$$\frac{\varepsilon_k}{\varepsilon_0} \leq \frac{1}{2^{k-1}\widehat{T}_k\left(\frac{\kappa+1}{\kappa-1}\right)},$$

where $\kappa = \lambda_{max}/\lambda_{min}$ is the 2-norm condition number of A (see Definition 11.2.15) and $\widehat{T}_k$ is the kth nonmonic Chebyshev polynomial (see Section 9.3.3).

Proof. By the previous lemma, it suffices to show that

$$\min_{q \in \mathscr{P}_k} \max_j |q(\lambda_j)| \leq \frac{1}{2^{k-1} \widehat{T}_k\left(\frac{\kappa+1}{\kappa-1}\right)}.$$

Since $A > 0$, all the eigenvalues λ of A satisfy $0 < \lambda_{\min} \leq \lambda \leq \lambda_{\max}$. Recall from Proposition 9.3.1 that the (nonmomic) Chebyshev polynomials have the property that $|\widehat{T}_k(x)| \leq 2^{-(k-1)}$ when $|x| \leq 1$. Moreover the map

$$x \mapsto \frac{\lambda_{\max} + \lambda_{\min} - 2x}{\lambda_{\max} - \lambda_{\min}}$$

takes the interval $[\lambda_{\min}, \lambda_{\max}]$ into the interval $[-1, 1]$. Thus, the polynomial

$$\widehat{q}_k(x) = \frac{\widehat{T}_k\left(\frac{\lambda_{\max}+\lambda_{\min}-2x}{\lambda_{\max}-\lambda_{\min}}\right)}{\widehat{T}_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}}\right)}$$

is in $\mathscr{P}_k$ since it satisfies $\widehat{q}(0) = 1$. Moreover, the numerator satisfies

$$\left|\widehat{T}_k\left(\frac{\lambda_{\max} + \lambda_{\min} - 2x}{\lambda_{\max} - \lambda_{\min}}\right)\right| < \frac{1}{2^{k-1}}$$

whenever $x \in [\lambda_{\min}, \lambda_{\max}]$. Hence,

$$\min_{q \in \mathscr{P}_k}\left(\max_j |q(\lambda_j)|\right) \leq \max_j |\widehat{q}(\lambda_j)| \leq \frac{1}{2^{k-1}\widehat{T}_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}}\right)} = \frac{1}{2^{k-1}\widehat{T}_k\left(\frac{\kappa+1}{\kappa-1}\right)}. \qquad \square$$

Theorem 12.7.6. *The error $\varepsilon_k = \|\mathbf{x}_k - \mathbf{x}^*\|_A$ satisfies*

$$\frac{\varepsilon_k}{\varepsilon_0} \leq 2\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right]^{-1}, \tag{12.52}$$

where κ is the 2-norm condition number of A.

Proof. It suffices to show that

$$\widehat{T}_k\left(\frac{\kappa+1}{\kappa-1}\right) = \frac{1}{2^k}\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right]. \tag{12.53}$$

This follows from Exercises 12.41 and 12.42. $\square$

Corollary 12.7.7. *The error $\varepsilon_k = \|\mathbf{x}_k - \mathbf{x}^*\|_A$ in the conjugate-gradient method converges linearly. More precisely, we have*

$$\frac{\varepsilon_k}{\varepsilon_0} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k. \tag{12.54}$$

Remark 12.7.8. Corollary 12.7.7 shows that if κ is close to 1, then convergence is very fast, but as $\kappa \to \infty$, the corollary isn't very helpful and we must resort to using the theorem. Asymptotically, we have

$$\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \sim 1 + \frac{2}{\sqrt{\kappa}} \qquad \text{and} \qquad \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \sim 1 - \frac{2}{\sqrt{\kappa}}.$$

With a little algebra (see Exercise 12.43), we can show that

$$\frac{\varepsilon_k}{\varepsilon_0} \le \left(1 + \binom{k}{2}\left(\frac{2}{\sqrt{\kappa}}\right)^2 + \binom{k}{4}\left(\frac{2}{\sqrt{\kappa}}\right)^4 + \cdots\right)^{-1}. \tag{12.55}$$

This is not much less than 1 unless k is large, relative to $\sqrt{\kappa}$.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

12.1. Find all the critical points of the function

$$f(x) = \frac{1}{5}x^5 - \frac{5}{2}x^4 + \frac{35}{3}x^3 - 25x^2 + 24x$$

on $\mathbb{R}$ and identify which of those is a local minimizer or maximizer.

12.2. Find all the critical points of the function

$$f(x,y) = 3x^2y + 4xy^2 + xy$$

on $\mathbb{R}^2$ and identify which of those is a local minimizer or maximizer.

12.3. ⚠ Prove all the claims of Example 12.1.17 about minimization of a quadratic function; that is, prove the following:

(i) For any square matrix A the matrix $Q = A^T + A$ is symmetric, and $\mathbf{x}^T Q \mathbf{x} = \mathbf{x}^T A^T \mathbf{x} + \mathbf{x}^T A \mathbf{x} = 2\mathbf{x}^T A \mathbf{x}$, so (12.2) is equal to (12.3). Thus we may always assume that quadratic functions are of the form (12.3) with Q symmetric.

(ii) Any minimizer $\mathbf{x}^*$ of a quadratic function f of the form (12.2) or (12.3) is a solution of the linear equation (12.4).

(iii) If $Q > 0$, then the quadratic minimization problem (12.2) has a minimizer.

(iv) If the quadratic minimization problem (12.2) has a minimizer, then $Q \geq 0$ and the minimizer is the solution of the linear system (12.4).

Explain why this shows that solving the system (12.4) with positive definite Q is equivalent to solving the quadratic optimization problem (12.3).

12.4. Let $A \in M_{m\times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$. Consider the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2. \tag{12.56}$$

Prove the following:

(i) The Hessian satisfies $D^2 f(\mathbf{x}) = A^\mathsf{T} A \geq 0$.

(ii) A necessary condition for $\mathbf{x}^*$ to be a minimizer of f is that it satisfies the linear system

$$A^\mathsf{T} A\mathbf{x}^* = A^\mathsf{T}\mathbf{b}. \tag{12.57}$$

This is called the *normal equation* for (12.56).

12.5. Consider the quadratic function f given in (12.56). Prove the following:

(i) If A has full column rank, then $D^2 f(\mathbf{x}) > 0$, and thus any solution of the normal equation is a local minimizer of f.

(ii) If A has full column rank, the point $\mathbf{x}^* = (A^\mathsf{T} A)^{-1} A^\mathsf{T}\mathbf{b}$ is the unique local minimizer of f.

(iii) Even if $A^\mathsf{T} A$ is not invertible, a solution $\mathbf{x}^*$ to (12.57) always exists. Hint: Think about the orthogonal projection of $\mathbf{b}$ to the subspace $\mathscr{R}(A)$, as described in Volume 1, Section 3.9.1, and consider the fundamental subspaces of A (Volume 1, Theorem 3.8.9),

(iv) Even if $A^\mathsf{T} A$ is not invertible, any solution of (12.57) can be written as $\mathbf{x}^* = \mathbf{x}^\dagger + \mathbf{v}$, with $\mathbf{v} \in \mathscr{N}(A)$ and $\mathbf{x}^\dagger = A^\dagger\mathbf{b}$, where $A^\dagger = V_1 \Sigma_1^{-1} U_1^\mathsf{T}$ is the Moore–Penrose pseudoinverse of A, and $A = U_1 \Sigma_1 V_1^\mathsf{T}$ is the compact form of the singular value decomposition (see Volume 1, Theorem 4.5.10 and Remark 4.5.14).

12.6.* Prove the claim in Remark 12.1.22 that a critical point (x_0, y_0) of twice-differentiable function f is a strict local minimizer if and only if (12.6) holds.

12.7. Fix $m > 1$. For each of the following convergent sequences, decide whether the convergence is sublinear, linear, superlinear but not quadratic, quadratic, or better than quadratic. If it is linear, determine the rate of convergence μ.

(i) $x_k = 1 + m^{-k}$.

(ii) $x_k = m^{-(k+3)}$.

(iii) $x_k = 1 + m^{-2k}$.

(iv) $x_k = 2 + m^{-2^k}$.

12.8. Prove carefully that the sequence $x_k = 1 + k^{-2}$ converges to 1 sublinearly.

12.9. Consider the problem of minimizing $f(x) = x^{4/3}$. Note that 0 is the global minimizer of f.

(i) Write down the algorithm for Newton's method applied to this problem.

(ii) Show that as long as the starting point is not 0, this algorithm does not converge to 0, no matter how close to 0 it starts. Why doesn't this contradict Theorem 12.2.6?

12.10. Code up an implementation of Newton's method for finding a critical point of a function of one variable. Your code should accept a twice-differentiable function f, an initial guess x_0, a desired level of accuracy ε, and a maximum number of iterations M. It should return an approximation to a critical point of f, provided the algorithm reaches the desired accuracy in fewer than M iterations. For the stopping criterion, use $|x_{k+1} - x_k| < \varepsilon$. Be sure your code has methods for identifying and handling cases where the algorithm fails or the sequence does not converge.

12.11. Just as Newton's method can be used to find roots of a function, the secant method has an obvious analogue for rootfinding.

(i) Describe the rootfinding version of the secant method and use it to devise a method to approximate $\log_b(x)$ for any $b \in (1, \infty)$ and any $x > 0$ *using only basic arithmetic operations and exponentiation.*

(ii) Code up your algorithm without importing any libraries or modules. Your code should accept two floats $x > 0$ and $b > 1$ and return a close approximation to $\log_b(x)$.

12.12. Prove that an unconstrained linear objective function $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + c$, with $\mathbf{a} \in \mathbb{R}^n$ and $c \in \mathbb{R}$, either is constant or has no minimum.

12.13. Let $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} Q \mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x}$, where $Q \in M_n(\mathbb{R})$ is positive definite (denoted $Q > 0$) and $\mathbf{b} \in \mathbb{R}^n$. Show that exact gradient descent (that is, gradient descent with optimal line search) converges in one step (that is, $\mathbf{x}_1 = Q^{-1}\mathbf{b}$) if and only if $\mathbf{x}_0$ is chosen such that $Df(\mathbf{x}_0)^\mathsf{T} = Q\mathbf{x}_0 - \mathbf{b}$ is an eigenvector of Q and α_0 satisfies (12.15).

12.14. Assume that $f \in C^1(\mathbb{R}^n; \mathbb{R})$. Let $\{\mathbf{x}_k\}_{k=0}^{\infty}$ be defined by exact gradient descent. Show that $\mathbf{x}_{k+1} - \mathbf{x}_k$ is orthogonal to $\mathbf{x}_{k+2} - \mathbf{x}_{k+1}$ for each k.

12.15. Write a simple implementation of the exact gradient descent method for quadratic functions. Your code should accept a small number ε, arrays $\mathbf{x}_0, \mathbf{b}$ of length n, and an $n \times n$ matrix $Q > 0$, and your code should return a close approximation to a local minimizer $\mathbf{x}^*$ of $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} Q \mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$. For the stopping criterion, use the condition $\|Df(\mathbf{x}_k)^\mathsf{T}\| < \varepsilon$.

12.16. Construct an implementation of exact gradient descent for arbitrary functions, using Newton's method (Exercise 12.10) for the line search (you must choose some initial value of α for Newton's method—justify your choice). Your method should accept a callable function f, a starting value $\mathbf{x}_0$ (an array of length n), and a small number ε. It should return a close approximation to a local minimizer $\mathbf{x}^*$ of f. For the stopping criterion, use the condition $\|Df(\mathbf{x}_k)\| < \varepsilon$.

12.17. Apply your code from the previous problem to the Rosenbrock function

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

with an initial guess of $(x_0, y_0) = (-2, 2)$. Does it converge? If not, explain why not. If it does, how many iterations does it take to get within 10^{-5} of the true minimizer?

12.18. Prove Proposition 12.4.6 as follows:

(i) Let $\mathbf{d}_k = -D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^\mathsf{T}$ and set $\phi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{d}_k)$. Show that $\phi'(0) = D_{\mathbf{d}_k} f(\mathbf{x}_k) = Df(\mathbf{x}_k)\mathbf{d}_k$.

(ii) Show that $Df(\mathbf{x}_k)\mathbf{d}_k < 0$. Hint: Recall that $D^2 f(\mathbf{x}_k) > 0$.

(iii) Show that there exists an $\bar{\alpha} > 0$ such that $\phi'(\alpha) < 0$ for all $\alpha \in (0, \bar{\alpha})$.

(iv) Show that $f(\mathbf{x}_k + \alpha \mathbf{d}_k) < f(\mathbf{x}_k)$ for all $\alpha \in (0, \bar{\alpha})$.

12.19. Give an example of a smooth (infinitely differentiable) function $f : \mathbb{R}^2 \to \mathbb{R}$ and an initial point $\mathbf{x}_0$ such that $D^2 f(\mathbf{x}_0) > 0$ but $f(\mathbf{x}_0 - D^2 f(\mathbf{x}_0)^{-1} Df(\mathbf{x}_0)^\mathsf{T}) > f(\mathbf{x}_0)$.

12.20. Write an implementation of Newton's method for finding a local minimizer of a function $f : \mathbb{R}^n \to \mathbb{R}$. Your code should accept a twice-differentiable function f, an initial guess $\mathbf{x}_0$, a desired level of accuracy ε, and a maximum number of iterations M. At each step it should calculate $\mathbf{x}_{k+1} = \mathbf{x}_k - D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^\mathsf{T}$ and then repeat until it reaches a good approximation to a critical point of f or exceeds M iterations. For the stopping criterion, use $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \varepsilon$. Be sure your code has methods for identifying and handling cases where the algorithm fails or the sequence does not converge.

12.21. Apply your Newton code from Exercise 12.20 to the Rosenbrock function with an initial guess of $\mathbf{x}_0 = (x_0, y_0) = (-2, 2)$. Does it converge? If not, explain why not. If it does, how many iterations does it take to get within 10^{-5} of the true minimizer?

12.22. Prove that if $A \in M_n(\mathbb{F})$ has eigenvalues $\lambda_1, \ldots, \lambda_n$ and $B = A + \mu I$, then the eigenvectors of A and B are the same, and the eigenvalues of B are $\mu + \lambda_1, \mu + \lambda_2, \ldots, \mu + \lambda_n$.

12.23. Code up an implementation of the Gauss–Newton algorithm for solving NLS problems. Your code should accept a differentiable function $\mathbf{r} : \mathbb{R}^n \to \mathbb{R}^m$, defining an objective function $f(\mathbf{x}) = \mathbf{r}(\mathbf{x})^\mathsf{T} \mathbf{r}(\mathbf{x})$, an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, a desired level of accuracy ε, and a maximum number of iterations M. At each step it should calculate $J(\mathbf{x}_k) = D\mathbf{r}(\mathbf{x}_k)^\mathsf{T}$ and compute $\mathbf{x}_{k+1}$ via (12.23), and then repeat until it reaches a good approximation to a critical point of f, or exceeds M iterations. For the stopping criterion, use $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \varepsilon$.
Apply your code to the range finder problem (Example 12.4.10 and Figure 12.9) with the data given in the example. Verify that when starting at $\mathbf{x}_0 = (2, 0)$ your code converges in eight steps to the minimizer (2.9546367, 2.88618843).

12.24.* Adapt your code from Exercise 12.20 to check at each iteration whether the Newton step descends, and then use backtracking to guarantee that each step descends.

Apply your adapted Newton code to the Rosenbrock function with the initial guess of $\mathbf{x}_0 = (x_0, y_0) = (-2, 2)$. Compare the results to those of Exercise 12.21.

12.25. Prove Proposition 12.5.1 (the Sherman–Morrison formula). Hint: Recall that to prove $X = Y^{-1}$ it suffices to show $XY = YX = I$.

12.26. Prove that for the BFGS method, as given in (12.29), the matrix A_k is symmetric whenever A_{k-1} is symmetric.

12.27. Prove that for the BFGS method, as given in (12.29), we have $A_k \mathbf{s}_k = \mathbf{y}_k$, and A_k satisfies the constraint (12.28).

12.28. Apply (12.27) twice to derive (12.33).

12.29. Consider the problem of minimizing the function

$$f(x, y) = x - y + 2x^2 + 2xy + y^2.$$

Apply the BFGS algorithm by hand to this problem for two steps (compute $\mathbf{x}_1$ and $\mathbf{x}_2$), starting at $\mathbf{x}_0 = (0, 0)$ and taking $A_0 = I$.
Note: Since f is quadratic, if we had taken $A_0 = D^2 f(\mathbf{x}_0)$, then the method would converge to the minimizer in a single step, because BFGS would have been identical to Newton's method at that initial step. But since $A_0 = I \neq D^2 f(\mathbf{x}_0)$, there is no reason to expect convergence in a single step.

12.30. Write an implementation of the BFGS algorithm or use the code in Algorithm 12.1.

(i) Apply your code to the function in Example 12.5.3 with an initial guess of $\mathbf{x}_0 = (x_0, y_0) = (4, 4)$ and $A_0 = D^2 f(\mathbf{x}_0) = \left[\begin{smallmatrix} 18 & 0 \\ 0 & 2 \end{smallmatrix}\right]$. How many iterations does it take to get within 10^{-5} of the true minimizer?

(ii) Repeat the previous step with the initial point $\mathbf{x}_0 = (4, 4)$ and $A_0 = I$.

(iii) Repeat the previous step with the initial point $\mathbf{x}_0 = (10, 10)$ and $A_0 = D^2 f(\mathbf{x}_0) = \left[\begin{smallmatrix} 54 & 0 \\ 0 & 2 \end{smallmatrix}\right]$.

(iv) Repeat the previous step with the initial point $\mathbf{x}_0 = (10, 10)$ and $A_0 = I$.

(v) What happens when the algorithm begins at the initial point $\mathbf{x}_0 = (0, 0)$? Explain.

12.31.* The *Davidon–Fletcher–Powell (DFP) method* is a quasi-Newton method that is similar to the BFGS method. Like BFGS, the DFP method makes a rank-two update to maintain a positive definite Hessian approximation. Suppose that $f \in C^1(\mathbb{R}^n; \mathbb{R})$. The DFP algorithm for minimizing f is as follows:

(i) Initialize a starting matrix B_0 and starting position $\mathbf{x}_0$ and let $k = 0$.

(ii) $\mathbf{d}_k = -B_k Df(\mathbf{x}_k)^\mathsf{T}$ (search direction).

(iii) $\alpha_k = \operatorname{argmin}_{a>0} f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$ (line search).

(iv) $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.

(v) $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$.

(vi) $\mathbf{q}_k = Df(\mathbf{x}_k)^\mathsf{T} - Df(\mathbf{x}_{k+1})^\mathsf{T}$.

(vii) If $||Df(\mathbf{x}_{k+1})|| < \varepsilon$ for some predetermined $\varepsilon > 0$, then stop.

(viii) $B_{k+1} = B_k + \frac{\mathbf{p}_k \mathbf{p}_k^\mathsf{T}}{\mathbf{p}_k^\mathsf{T} \mathbf{q}_k} - \frac{B_k \mathbf{q}_k \mathbf{q}_k^\mathsf{T} B_k}{\mathbf{q}_k^\mathsf{T} B_k \mathbf{q}_k}$ (DFP inverse-Hessian approximation).

(ix) Set $k = k + 1$ and go to step (ii).

Prove the following:

(i) Show that if B_0 is symmetric, then B_k is symmetric.

(ii) Use the DFP method to minimize $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1 x_2 + x_2^2$ from the starting point $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^\mathsf{T}$ with $B_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\varepsilon = 0.01$.

(iii) Show that if the function is quadratic with Hessian $Q > 0$ (i.e., $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} Q \mathbf{x} - \mathbf{x}^\mathsf{T} \mathbf{b} + c$), then $B_{k+1} Q \mathbf{p}_i = \mathbf{p}_i$ for $0 \le i \le k$.

(iv) Show that if the function is quadratic with Hessian $Q > 0$, then $\mathbf{p}_i^\mathsf{T} Q \mathbf{p}_j = 0$ for $0 \le i < j \le k$.

(v) If f is quadratic with Hessian $Q > 0$, how many steps will it take, at most, for DFP to converge?

12.32.* Prove the Sherman–Morrison–Woodbury formula (12.35).

12.33. Prove that in the special case of where the minimizer is at the origin in $\mathbb{R}^n$ and $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} \Lambda \mathbf{x}$, where Λ is diagonal and positive definite, then coordinate descent (as described on page 557) converges in no more than n steps, regardless of the starting point $\mathbf{x}_0$.

12.34. Let $A \in M_n(\mathbb{R})$ satisfy $A > 0$, and let f be the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T} A \mathbf{x} - \mathbf{b}^\mathsf{T} \mathbf{x} + c. \tag{12.58}$$

Given a starting point $\mathbf{x}_0$ and A-conjugate directions $\mathbf{d}_1, \ldots, \mathbf{d}_n$ in $\mathbb{R}^n$, show that the optimal line search solution for $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$, that is, the α which minimizes $\phi_k(\alpha) = f(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k)$, is given by $\alpha_k = \frac{\mathbf{r}_k^\mathsf{T} \mathbf{d}_k}{\mathbf{d}_k^\mathsf{T} A \mathbf{d}_k}$, where $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_{k-1}$.

12.35. Let $A \in M_n(\mathbb{R})$ satisfy $A > 0$, and let f be the quadratic function (12.58). Assume a starting point $\mathbf{x}_0$ and A-conjugate directions $\mathbf{d}_1, \ldots, \mathbf{d}_n$ in $\mathbb{R}^n$. For each $k > 0$ let $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$, with α_k as in the previous problem.

(i) Let the kth error vector be $\boldsymbol{\varepsilon}_k = \mathbf{x}_k - \mathbf{x}^*$, where $\mathbf{x}^*$ is the minimizer of f. Writing $\boldsymbol{\varepsilon}_0$ in terms of the basis $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ as $\boldsymbol{\varepsilon}_0 = \sum_{j=0}^{n-1} \delta_j \mathbf{d}_j$, show that $\mathbf{d}_k^\mathsf{T} A \boldsymbol{\varepsilon}_0 = \delta_k \mathbf{d}_k^\mathsf{T} A \mathbf{d}_k$ and hence

$$\delta_k = \frac{\mathbf{d}_k^\mathsf{T} A \boldsymbol{\varepsilon}_0}{\mathbf{d}_k^\mathsf{T} A \mathbf{d}_k}.$$

(ii) Show that $\boldsymbol{\varepsilon}_k = \boldsymbol{\varepsilon}_0 + \sum_{j=0}^{k-1} \alpha_j \mathbf{d}_j$ and use this to show that

$$\frac{\mathbf{d}_k^\mathsf{T} A \boldsymbol{\varepsilon}_k}{\mathbf{d}_k^\mathsf{T} A \mathbf{d}_k} = \frac{\mathbf{d}_k^\mathsf{T} A \boldsymbol{\varepsilon}_0}{\mathbf{d}_k^\mathsf{T} A \mathbf{d}_k} = \delta_k.$$

(iii) Use the results of the previous problem to show that $\alpha_k = -\delta_k$.

(iv) Show that $\boldsymbol{\varepsilon}_n = 0$, hence for the quadratic function f this method converges to the minimizer in no more than n steps.

12.36. Prove that all the equalities in (12.43) hold.

12.37. For the conjugate-gradient method prove that the vectors $\mathbf{r}_i$ and $\mathbf{d}_i$ satisfy the following relations for all $k \in \{2, \ldots, n\}$:

(i) $\mathbf{r}_i^\mathsf{T} \mathbf{r}_k = 0$ for all $i < k$.

(ii) Generalize (12.42) to prove that for all $\ell < k$ we have

$$\mathbf{r}_k - \mathbf{r}_\ell = -\sum_{j=\ell}^{k-1} \alpha_j A \mathbf{d}_j.$$

(iii) For each $\ell < k$ prove that

$$\mathbf{r}_k^\mathsf{T} \mathbf{r}_k = -\sum_{i=\ell}^{k-1} \alpha_i \mathbf{d}_i^\mathsf{T} A \mathbf{r}_k = -\sum_{i=\ell}^{k-1} \alpha_i \langle \mathbf{d}_i, \mathbf{r}_k \rangle_A .$$

12.38. ⚠ For each of the multivariable optimization methods discussed in this chapter, list the following:

(i) The basic idea of the method, including how it differs from the other methods in the list. Include any geometric description you can give of the method.

(ii) What types of optimization problems it can solve and cannot solve.

(iii) Relative strengths of the method.

(iv) Relative weaknesses of the method.

12.39.* Prove that (12.49) holds as follows:

(i) Show that for all $i > k$ the vector $\mathbf{d}_i^\mathsf{T} A \mathbf{v} = 0$ for all $\mathbf{v} \in \mathscr{K}_k(A, \mathbf{r}_1)$.

(ii) Expand out $\|\mathbf{x} - \mathbf{x}^*\|_A^2$, using the relations $\mathbf{x} = \mathbf{x}_k + \mathbf{v}$ for some $\mathbf{v} \in \mathscr{K}_k(A, \mathbf{r}_1) = \operatorname{span}(\mathbf{d}_1, \ldots, \mathbf{d}_k)$ and $\mathbf{x}^* = \mathbf{x}_k + \sum_{i=k+1}^m \alpha_i \mathbf{d}_i$.

(iii) Combine the first two parts to get $\varepsilon(\mathbf{x}^*)^2 = \varepsilon(\mathbf{x}_k)^2 + \|\mathbf{v}\|_A^2$.

12.40.* Prove (12.51) as follows:

(i) Show that $A = U^\mathsf{T} \Lambda U$ for an orthonormal matrix U and a diagonal matrix $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$.

(ii) Set $\mathbf{y} = U(\mathbf{x}_0 - \mathbf{x}^*)$ and show that $\|q(A)(\mathbf{x}_0 - \mathbf{x}^*)\|_A^2 = \sum_{i=1}^n y_i^2 \lambda_i (q(\lambda_i))^2$.

(iii) Show that $\sum_{i=1}^n y_i^2 \lambda_i (q(\lambda_i))^2 \le \varepsilon(\mathbf{x}_0)^2 \max_j (q(\lambda_j))^2$.

12.41.* For $k \in \mathbb{Z}^+$, let $\widehat{T}_k$ be the kth monic Chebyshev polynomial. Prove that

$$\widehat{T}_k\left(\frac{z + z^{-1}}{2}\right) = \frac{z^k + z^{-k}}{2^k}.$$

Hint: Use induction via (9.20).

12.42.* Prove Theorem 12.7.6 by showing the following:

(i) If $z = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}$, then $\frac{z+z^{-1}}{2} = \frac{\kappa+1}{\kappa-1}$.

(ii) Use the previous exercise to prove (12.53).

12.43.* Derive the expansion in Remark 12.7.8.

Notes

Useful references for the optimization techniques in this chapter include [CZ01] and [NW99]. For details on the limited-memory BFGS algorithm, see [NW99, Section 7.2]. For a detailed discussion of the use of Levenberg–Marquardt methods in NLS problems, see [TMS10]. The NLS range finder example was inspired by [Van18]. It is closely related to the *geodetic* problem of updating geographic survey data (measuring distances, angles, and altitudes), which was the problem for which Gauss developed the method of least squares. For more on this, see [Dem97, Example 3.3]. For more on conjugate-gradient methods see [GS92, Dan67, Dan70, She94]. For more about the conjugate-gradient method for nonquadratic objectives, see [CZ01].

13 Linear Optimization

The simplex method is so easy I could even teach it to MBA students.
—Emily Evans

If the objective function in an unconstrained optimization problem is linear and nontrivial, then, as shown in Exercise 12.12 it has no minimum nor maximum. But many interesting and important problems correspond to minimizing a linear function with some additional constraints. Problems where the objective and the constraints are all linear are called *linear optimization* problems. Linear optimization problems arise in many important applications, including resource allocation, production planning, labor scheduling, transportation, portfolio management, marketing, and military logistics, to name just a few.

We begin this chapter with a discussion of *convex* and *affine* sets, which play an important role in linear optimization problems. We then state and prove the fundamental theorem of linear optimization, which guarantees that if there is an optimizer for a linear problem, then one of the vertices of the feasible set is an optimizer. Thus, one strategy to finding an optimizer is to search among the vertices of the feasible set. This is the key idea for the most famous and widely used method for solving a linear optimization problem, namely, the *simplex method*, which was developed by George Dantzig in 1947. It remains one of the most notable algorithms of our time.

13.1 Convex and Affine Sets

Before beginning the subject of linear optimization, we need a little background about *convex sets* and *affine sets*. Throughout this section assume that V is a given vector space.

13.1.1 Convex Sets

Definition 13.1.1. *A nonempty set $C \subset V$ is* convex *if for each $\mathbf{x}, \mathbf{y} \in C$, we have*

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C$$

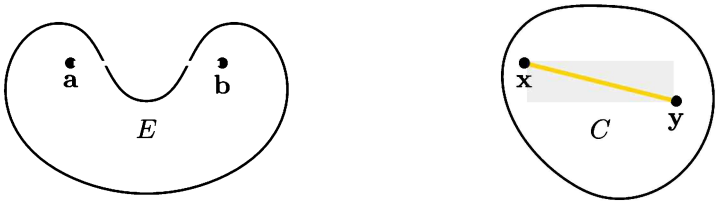


Figure 13.1. *The set E (left) is not convex because the line segment (red) between $\mathbf{a}$ and $\mathbf{b}$ is not contained in E. But the set C (right) is convex because the line segment between any two points $\mathbf{x}, \mathbf{y} \in C$ is always contained in C.*

for all λ with $0 \leq \lambda \leq 1$. Said differently, C is convex if for each $\mathbf{x}, \mathbf{y} \in C$ the line segment $\ell(\mathbf{x}, \mathbf{y}) = \{\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \mid \lambda \in [0,1]\}$ between $\mathbf{x}$ and $\mathbf{y}$ is contained in C for every $\mathbf{x}, \mathbf{y} \in C$; see Figure 13.1.

Example 13.1.2. If $(V, \|\cdot\|)$ is a normed linear space, then the open ball $B(\mathbf{v}, r)$ centered at $\mathbf{v} \in V$ is convex. To see this, consider $\mathbf{x}, \mathbf{y} \in B(\mathbf{v}, r)$, and let $0 \leq \lambda \leq 1$. We have $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in B(\mathbf{v}, r)$, since

$$\begin{aligned} \|\lambda\mathbf{x} + (1-\lambda)\mathbf{y} - \mathbf{v}\| &= \|\lambda(\mathbf{x} - \mathbf{v}) + (1-\lambda)(\mathbf{y} - \mathbf{v})\| \\ &\leq \lambda\|\mathbf{x} - \mathbf{v}\| + (1-\lambda)\|\mathbf{y} - \mathbf{v}\| \\ &< \lambda r + (1-\lambda)r = r. \end{aligned}$$

Proposition 13.1.3. *The intersection of a collection $\{C_\alpha\}_{\alpha \in J} \subset V$ of convex sets is convex if it is not empty.*

Proof. Let $C = \bigcap_{\alpha \in J} C_\alpha$. If $\mathbf{x}, \mathbf{y} \in C$ and $0 \leq \lambda \leq 1$, then $\mathbf{x}, \mathbf{y} \in C_\alpha$ for each $\alpha \in J$, which implies $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in C_\alpha$ for each $\alpha \in J$. Thus, $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in C$. □

13.1.2 Convex Combinations and Convex Hulls

Definition 13.1.4. *Let S be a nonempty subset of V. The* convex hull *of S, denoted* $\operatorname{conv}(S)$*, is the set of all convex combinations of elements of S, that is, the set of all finite sums of the form*

$$\lambda_1\mathbf{x}_1 + \cdots + \lambda_k\mathbf{x}_k \quad \text{for } \mathbf{x}_i \in S \text{ and } k \in \mathbb{Z}^+,$$

where each $\lambda_i \geq 0$ and $\lambda_1 + \cdots + \lambda_k = 1$. This set is also sometimes called the convex span *of S.*

Proposition 13.1.5. *If S is a nonempty subset of V, then $\operatorname{conv}(S)$ is convex.*

Proof. See Exercise 13.2. □

Proposition 13.1.6. *If C is a convex subset of V, then $\operatorname{conv}(C) = C$.*

Proof. It is immediate from the definition that $C \subset \text{conv}(C)$, so it suffices to show $\text{conv}(C) \subset C$; this follows by induction on the number of terms in the convex combination. The cases $n = 1$ and $n = 2$ follow by the definition of a convex set.

Suppose that all convex combinations of C of length n are in C, and consider a convex combination of length $n+1$, say, $\mathbf{x} = \lambda_1\mathbf{x}_1 + \lambda_2\mathbf{x}_2 + \cdots + \lambda_n\mathbf{x}_n + \lambda_{n+1}\mathbf{x}_{n+1}$, where, after possibly reindexing, we may assume $0 < \lambda_{n+1} < 1$. Rewrite $\mathbf{x}$ as

$$\mathbf{x} = (1-\lambda_{n+1}) \underbrace{\left(\frac{\lambda_1\mathbf{x}_1}{1-\lambda_{n+1}} + \cdots + \frac{\lambda_n\mathbf{x}_n}{1-\lambda_{n+1}} \right)}_{\mathbf{w}} + \lambda_{n+1}\mathbf{x}_{n+1}.$$

Thus, if $\mathbf{w} = \frac{\lambda_1\mathbf{x}_1}{1-\lambda_{n+1}} + \cdots + \frac{\lambda_n\mathbf{x}_n}{1-\lambda_{n+1}}$, then by the inductive hypothesis, we have $\mathbf{w} \in C$. This implies $\mathbf{x} = (1-\lambda_{n+1})\mathbf{w} + \lambda_{n+1}\mathbf{x}_{n+1} \in C$, by the definition of a convex set. Hence, all convex combinations of $n+1$ elements are also in C. Therefore, by induction, all finite convex combinations of elements of C are in C, and $\text{conv}(C) \subset C$. □

Theorem 13.1.7. *Assume $S \subset V$ is not empty. The convex hull of S is the smallest convex set that contains S, meaning that if D is any convex set containing S, then $\text{conv}(S) \subset D$. Moreover, $\text{conv}(S)$ is equal to the intersection of all convex sets containing S.*

Proof. If D is any convex set with $S \subset D$, then $\text{conv}(S) \subset \text{conv}(D) = D$, by Proposition 13.1.6. This also implies that $\text{conv}(S)$ is a subset of the intersection of all convex sets containing S. Conversely, since $\text{conv}(S)$ is itself a convex set containing S, it must contain the intersection of all such sets. □

13.1.3 Affine Sets and Functions

Definition 13.1.8. *A set $A \subset V$ is called* affine *if there exist some linear subspace $W \subset V$ and a point $\mathbf{v} \in V$ such that $A = W + \mathbf{v} = \{\mathbf{x} \in V \mid \exists\, \mathbf{w} \in W, \mathbf{x} = \mathbf{v} + \mathbf{w}\}$. In other words, A is affine if A is a translate (a coset) of a linear subspace W of V; see also Volume 1, Section 1.5.1. The* dimension *of an affine space $A = W + \mathbf{v}$ is the dimension of the linear subspace W.*

Example 13.1.9.

(i) Any linear subspace is an affine set (with translation $\mathbf{0}$).

(ii) A single point $\{\mathbf{c}\}$ is an affine set of dimension 0 because it is a translate of the zero subspace: $\{\mathbf{c}\} = \{0\} + \mathbf{c}$.

(iii) For any matrix $A \in M_{m\times n}$ and any $\mathbf{b} \in \mathbb{R}^m$ the set $C = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b}\}$ is an affine set if it is not empty. To see this, let $\mathbf{x}_0$ be any element of C and observe that $C = \mathscr{N}(A) + \mathbf{x}_0 = \{\mathbf{w} + \mathbf{x}_0 \in \mathbb{R}^n \mid A\mathbf{w} = \mathbf{0}\}$, and the kernel $\mathscr{N}(A)$ of A is a subspace of $\mathbb{R}^n$.

Proposition 13.1.10. *If an affine set A can be written in two ways as $A = W + \mathbf{v} = W' + \mathbf{v}'$, with W and W' both linear subspaces of V, then $W = W'$ and $\mathbf{v} - \mathbf{v}' \in W$. Thus the dimension of A is well defined.*

Proof. The proof is Exercise 13.9. □

Proposition 13.1.11. *An affine set is convex.*

Proof. The proof is Exercise 13.1(ii) □

Affine functions are those that differ only by a constant from linear transformations. We (and everyone else) often call these *linear functions*. Beware that this name is easily confused with linear transformations.

Definition 13.1.12. *Let V and W be vector spaces. A function $f : V \to W$ is* affine *if there is a linear transformation $L : V \to W$ and a constant $\mathbf{c} \in W$ such that $f(\mathbf{x}) = L(\mathbf{x}) + \mathbf{c}$ for all $\mathbf{x} \in V$.*

Example 13.1.13.

(i) Any linear transformation is an affine function.

(ii) Any constant function is affine.

(iii) The function $f : \mathbb{R}^2 \to \mathbb{R}$ given by $f(x, y) = 3x + 4y + 6$ is not a linear transformation because it does not map the origin to 0, but it is affine, because $f(x, y) - 6$ is linear.

Proposition 13.1.14. *If $C \subset V$ is a convex set and $f : V \to W$ is an affine function, then $f(C)$ is convex. Also, if $D \subset W$ is convex, then $f^{-1}(D) = \{\mathbf{v} \in V \mid f(\mathbf{v}) \in D\}$ is convex.*

Proof. The proof is Exercise 13.3. □

Proposition 13.1.15. *If $A \subset V$ is an affine subset of V and $f : V \to W$ is an affine function, then $f(A)$ is an affine subset of W. Similarly, if $B \subset W$ is an affine subset of W, then $f^{-1}(B)$ is affine subset of V.*

Proof. The proof is Exercise 13.10. □

13.1.4 Hyperplanes and Half Spaces

Throughout the rest of this chapter, unless otherwise indicated, let $(V, \langle \cdot, \cdot \rangle)$ be a given inner product space over $\mathbb{R}$, and let $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$ be the corresponding norm.

Definition 13.1.16. *A* hyperplane *in V is a set of the form $\{\mathbf{x} \in V \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$, where $\mathbf{a} \in V$ is not zero, and $b \in \mathbb{R}$.*

Remark 13.1.17. Algebraically, the hyperplane $H = \{\mathbf{x} \in V \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ is the solution set of the linear system $\langle \mathbf{a}, \mathbf{x} \rangle = b$, which is

$$a_1 x_1 + \cdots + a_n x_n = b,$$

where $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{x} = (x_1, \ldots, x_n)$. This is the same as the *zero locus* $\{\mathbf{x} \in V \mid h(\mathbf{x}) = 0\}$ of the affine function $h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle - b$. Sometimes people call a function of the form $h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle - b$ a hyperplane, but this is a little sloppy.

Remark 13.1.18. Geometrically, H is the set of all vectors that have a constant inner product with $\mathbf{a}$. If $\mathbf{x}_0 \in H$ is given, then we may write the hyperplane as $H = \{\mathbf{x} \in V \mid \langle \mathbf{a}, \mathbf{x} - \mathbf{x}_0 \rangle = 0\}$. This implies that H is a translate of the orthogonal complement $W = \operatorname{span}(\mathbf{a})^\perp$ of $\mathbf{a}$, which is the linear subspace of V consisting of vectors orthogonal to $\mathbf{a}$; see Figure 13.2. If V is finite dimensional, then every such W has dimension $\dim(V) - 1$, and so the dimension of a hyperplane in V is always $\dim(V) - 1$.

Example 13.1.19. Given two points $\mathbf{c}, \mathbf{d} \in V$, the *perpendicular bisector* is the unique hyperplane that goes through the midpoint $\mathbf{m} = \frac{1}{2}(\mathbf{c} + \mathbf{d})$ of $\mathbf{c}$ and $\mathbf{d}$ and is perpendicular to the line segment $\ell(\mathbf{c}, \mathbf{d})$ joining the two points. The perpendicular bisector is given by the hyperplane $\langle \mathbf{d} - \mathbf{c}, \mathbf{x} - \mathbf{m} \rangle = 0$, or, alternatively, $\langle \mathbf{a}, \mathbf{x} \rangle = b$, where $\mathbf{a} = \mathbf{d} - \mathbf{c}$ and

$$b = \left\langle \mathbf{d} - \mathbf{c}, \frac{1}{2}(\mathbf{c} + \mathbf{d}) \right\rangle = \frac{1}{2}(\|\mathbf{d}\|^2 - \|\mathbf{c}\|^2);$$

see Figure 13.3.

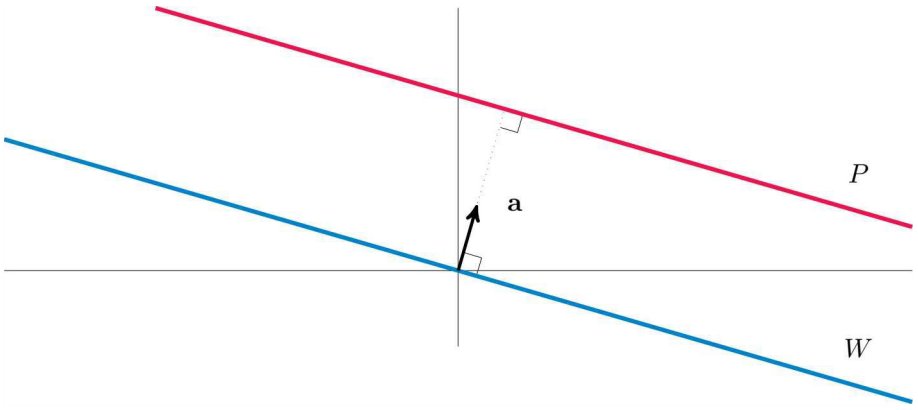


Figure 13.2. *A set of the form $H = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ is called a* hyperplane *(see Definition 13.1.16). Any hyperplane is an affine set because it is a translate (coset) of the linear subspace $W = \operatorname{span}(\mathbf{a})^\perp$ of all vectors orthogonal to $\mathbf{a}$; see Remark 13.1.17.*

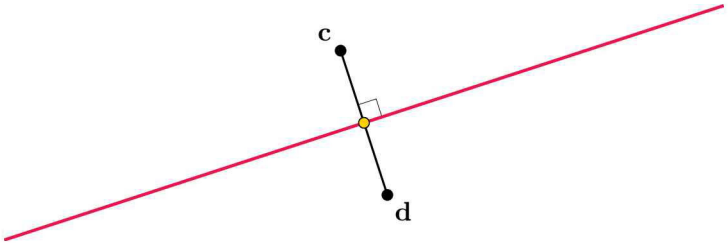


Figure 13.3. *The perpendicular bisector of* $\mathbf{c}$ *and* $\mathbf{d}$ *is the hyperplane (red) through the midpoint* $\frac{1}{2}(\mathbf{c}+\mathbf{d})$ *(yellow) that is perpendicular to* $\mathbf{d}-\mathbf{c}$*, as described in Example 13.1.19. This can also be described as the set* $\{\mathbf{x} \mid \langle \mathbf{d}-\mathbf{c}, \mathbf{x}-\mathbf{m}\rangle = 0\}$.

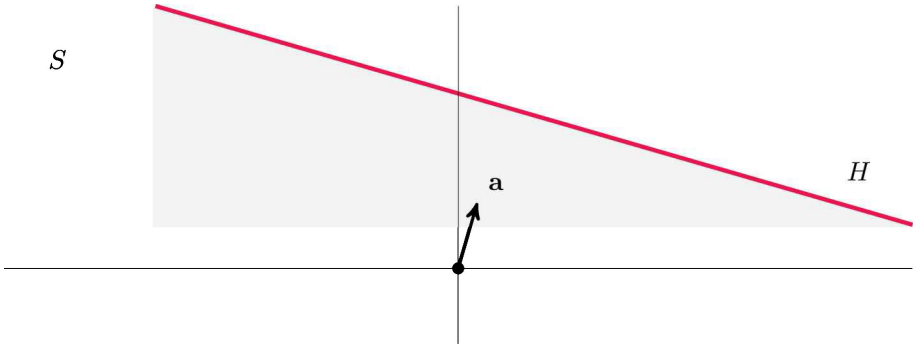


Figure 13.4. *The half space* $S = \{\mathbf{x} \in V \mid \langle \mathbf{a}, \mathbf{x}\rangle \leq b\}$ *(gray) is "half" of the space determined by the supporting hyperplane* $H = \partial S = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}, \mathbf{x}\rangle = b\}$ *(red). If* b *is positive, then the half space lies on the side of the hyperplane that contains the origin.*

Definition 13.1.20. *A* half space *in* V *is a set of the form* $S = \{\mathbf{x} \in V \mid \langle \mathbf{a}, \mathbf{x}\rangle \leq b\}$*, where* $\mathbf{a} \in V$ *is not zero, and* $b \in \mathbb{R}$*. The hyperplane* $H = \partial S = \{\mathbf{x} \in V \mid \langle \mathbf{a}, \mathbf{x}\rangle = b\}$ *is called the* supporting hyperplane *of* S*. See Figure 13.4 for an illustration.*

Remark 13.1.21. The set $\{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x}\rangle \geq b\}$ is also a half space, since it can be written as $\{\mathbf{x} \mid \langle -\mathbf{a}, \mathbf{x}\rangle \leq -b\}$.

13.2 Projection, Support, and Separation

Throughout this section we assume that $(V, \langle \cdot, \cdot \rangle)$ is a finite-dimensional inner product space. Many of the results of this section also hold in the infinite-dimensional case, but their proofs would take us beyond the scope of this text. Details on the infinite-dimensional case can be found in any good book on functional analysis.

13.2.1 Projection to a Convex Set

The notion of a projection in linear algebra extends in a very natural way to convex sets. This is a very powerful concept.

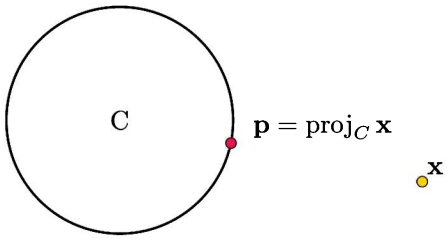


Figure 13.5. *The projection* $\text{proj}_C \mathbf{x}$ *of the point* $\mathbf{x}$ *onto the convex set* C *is the point* $\mathbf{p} \in C$ *nearest to* $\mathbf{x}$.

Definition 13.2.1. *Let* $C \subset V$ *be nonempty, closed, and convex, and let* $\mathbf{x} \in V$. *The* projection of $\mathbf{x}$ onto C, *denoted* $\text{proj}_C \mathbf{x}$, *is the point* $\mathbf{p} \in C$ *satisfying* $\|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{y}\|$ *for all* $\mathbf{y} \in C$. *In other words,* $\text{proj}_C \mathbf{x}$ *is the closest point in* C *to* $\mathbf{x}$. *See Figure 13.5.*

The problem of finding the projection of a point $\mathbf{x}$ onto a convex set C is a constrained optimization problem of the form

$$\begin{aligned} &\text{minimize} && \|\mathbf{x} - \mathbf{p}\| \\ &\text{subject to} && \mathbf{p} \in C. \end{aligned} \tag{13.1}$$

Remark 13.2.2. Since the squaring function is strictly increasing, (13.1) is equivalent to the problem of minimizing $\|\mathbf{x} - \mathbf{p}\|^2$ subject to $\mathbf{p} \in C$, which is generally easier to compute.

Theorem 13.2.3. *If* $C \subset V$ *is a nonempty, closed, and convex set, then every* $\mathbf{x} \in V$ *has a unique projection* $\mathbf{p} \in C$.

Proof. Let $\mathbf{x} \in V$ be given. Since C is not empty, there exists at least one $\mathbf{z} \in C$. Consider the closed and bounded subset

$$C' = \{\mathbf{y} \in C : \|\mathbf{y} - \mathbf{x}\| \leq \|\mathbf{z} - \mathbf{x}\|\}.$$

Since V is finite dimensional, the Heine–Borel theorem (Theorem 5.5.4 of Volume 1) guarantees that C' is compact. Let $f : C \to \mathbb{R}$ be the function $f(\mathbf{y}) = \|\mathbf{y} - \mathbf{x}\|$. Notice that $\inf_{\mathbf{y} \in C} f(\mathbf{y}) = \inf_{\mathbf{y} \in C'} f(\mathbf{y})$. Since f is continuous and C' is compact, the extreme value theorem guarantees that some $\mathbf{p} \in C'$ is a minimizer of f on C', and thus also a minimizer of f on all of C.

To prove uniqueness, suppose there exists another minimizer $\mathbf{q} \in C$ of f with $\mathbf{q} \neq \mathbf{p}$. Since $\mathbf{p}$ and $\mathbf{q}$ are minimizers of f on C, it follows that the midpoint $\mathbf{m} = \frac{1}{2}(\mathbf{p} + \mathbf{q}) \in C$ is also a minimizer, since $\mathbf{x} - \mathbf{m} = \frac{1}{2}(\mathbf{x} - \mathbf{p}) + \frac{1}{2}(\mathbf{x} - \mathbf{q})$, and thus

$$\|\mathbf{x} - \mathbf{m}\| \leq \frac{\|\mathbf{x} - \mathbf{p}\|}{2} + \frac{\|\mathbf{x} - \mathbf{q}\|}{2} = \|\mathbf{x} - \mathbf{p}\|.$$

However, $\mathbf{x}-\mathbf{m}$ is orthogonal to $\mathbf{m}-\mathbf{q}$, since

$$\begin{aligned}\langle \mathbf{x}-\mathbf{m}, \mathbf{q}-\mathbf{m}\rangle &= \left\langle \mathbf{x}-\frac{\mathbf{p}+\mathbf{q}}{2}, \mathbf{q}-\frac{\mathbf{p}+\mathbf{q}}{2}\right\rangle \\ &= \left\langle \frac{\mathbf{x}-\mathbf{p}}{2}+\frac{\mathbf{x}-\mathbf{q}}{2}, \frac{\mathbf{x}-\mathbf{p}}{2}-\frac{\mathbf{x}-\mathbf{q}}{2}\right\rangle \\ &= \frac{\|\mathbf{x}-\mathbf{p}\|^2}{4}-\frac{\|\mathbf{x}-\mathbf{q}\|^2}{4}=0.\end{aligned}$$

It follows from the Pythagorean law (Theorem 3.1.20 of Volume 1) that

$$\|\mathbf{x}-\mathbf{q}\|^2=\|\mathbf{x}-\mathbf{m}+\mathbf{m}-\mathbf{q}\|^2=\|\mathbf{x}-\mathbf{m}\|^2+\|\mathbf{m}-\mathbf{q}\|^2>\|\mathbf{x}-\mathbf{m}\|^2,$$

which is a contradiction, since $\mathbf{q}$ and $\mathbf{m}$ are equidistant from $\mathbf{x}$. □

Example 13.2.4. Since every subspace of a vector space is convex, an orthogonal projection of a vector to a subspace is the same as the convex projection of the vector to the subspace.

The following theorem gives a very useful condition for deciding when a point is a projection.

Theorem 13.2.5 (Convex Projection Formula). *Assume $C \subset V$ is nonempty, closed, and convex. A point $\mathbf{p} \in C$ is the projection of $\mathbf{x}$ onto C if and only if*

$$\langle \mathbf{x}-\mathbf{p}, \mathbf{p}-\mathbf{y}\rangle \geq 0 \quad \text{for all } \mathbf{y} \in C. \tag{13.2}$$

Proof. The proof is Exercise 13.13. □

13.2.2 Support and Separation

An important concept in convex analysis is that of a *supporting* hyperplane.

Definition 13.2.6. *Let C be a set in V. The hyperplane $H=\{\mathbf{v}\in V \mid \langle \mathbf{a},\mathbf{v}\rangle = b\}$* supports *$C$ if*

(i) *$C \cap H \neq \emptyset$, and*

(ii) *C lies entirely in only one of the two half spaces $\{\mathbf{v}\in V \mid \langle \mathbf{a},\mathbf{v}\rangle \leq b\}$ or $\{\mathbf{v}\in V \mid \langle \mathbf{a},\mathbf{v}\rangle \geq b\}$ defined by H. In this case, the half space containing C is also said to* support *C.*

If $D \subset V$ is another set, then C and D are separated *by H if $\langle \mathbf{a},\mathbf{x}\rangle \leq b$ for all $\mathbf{x}\in C$ and $\langle \mathbf{a},\mathbf{x}\rangle \geq b$ for all $\mathbf{x}\in D$, or vice versa. The sets are* strictly separated *if $\langle \mathbf{a},\mathbf{x}\rangle < b$ for all $\mathbf{x}\in C$ and $\langle \mathbf{a},\mathbf{x}\rangle > b$ for all $\mathbf{x}\in D$, or vice versa.*

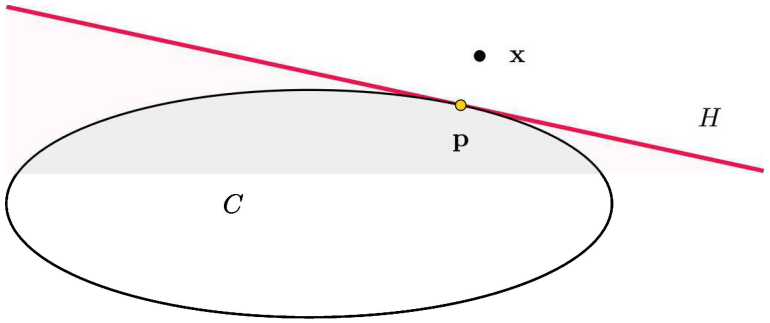


Figure 13.6. *The separation lemma (Lemma 13.2.7) guarantees that for any nonempty, closed, and convex set C, and any $\mathbf{x} \notin C$, there exists a hyperplane H (red) separating C from $\{\mathbf{x}\}$ and supporting C, meaning that one of the half spaces (pink) defined by H contains all of C, that H itself contains at least one point (here that point is $\mathbf{p}$), and that $\mathbf{x}$ is not in the half space containing C.*

Lemma 13.2.7 (Separation Lemma). *Assume $C \subset V$ is nonempty, closed, and convex. If $\mathbf{x} \notin C$ and $\mathbf{p} = \operatorname{proj}_C \mathbf{x}$, then the hyperplane $H = \{\mathbf{z} \mid \langle \mathbf{x} - \mathbf{p}, \mathbf{z} - \mathbf{p} \rangle = 0\}$ supports C and the half space $S = \{\mathbf{z} \mid \langle \mathbf{x} - \mathbf{p}, \mathbf{z} - \mathbf{p} \rangle \leq 0\}$ contains C but not $\mathbf{x}$, as illustrated in Figure 13.6.*

Proof. The proof is Exercise 13.12. □

Theorem 13.2.8 (Supporting Hyperplane Theorem). *Assume $C \subset V$ is closed, convex, and not empty. If $\mathbf{p} \in C$ and there exists a sequence $(\mathbf{x}_k)_{k=0}^{\infty} \subset C^c$ outside of C that converges to $\mathbf{p}$, then there exists a supporting hyperplane of C that contains $\mathbf{p}$.*

Proof. For each $k \in \mathbb{N}$ let $\mathbf{p}_k = \operatorname{proj}_C \mathbf{x}_k$, let $\mathbf{a}_k = (\mathbf{x}_k - \mathbf{p}_k)/\|\mathbf{x}_k - \mathbf{p}_k\|$, and let $b_k = \langle \mathbf{a}_k, \mathbf{p}_k \rangle$. Since $\|\mathbf{a}_k\| = 1$ the vector $\mathbf{a}_k$ lies in the closed unit sphere, which is compact. Therefore, the sequence $(\mathbf{a}_k)_{k=0}^{\infty}$ must have a subsequence that converges to some $\mathbf{a} \in V$ with $\|\mathbf{a}\| = 1$. Passing to that subsequence, we may assume that $\mathbf{a}_k \to \mathbf{a}$. Let $\mathbf{x} = \mathbf{a} + \mathbf{p}$. Since $\mathbf{p}_k$ is the projection of $\mathbf{x}_k$ to C for every $k \in \mathbb{N}$, Lemma 13.2.7 shows that

$$\langle \mathbf{x}_k - \mathbf{p}_k, \mathbf{z} - \mathbf{p}_k \rangle \leq 0$$

and every $\mathbf{z} \in C$. This shows that

$$\langle \mathbf{a}_k, \mathbf{z} \rangle \leq \langle \mathbf{a}_k, \mathbf{p}_k \rangle \tag{13.3}$$

for all $k \in \mathbb{N}$ and $\mathbf{z} \in \mathbb{C}$. Taking the limit of (13.3) gives $\langle \mathbf{a}, \mathbf{z} - \mathbf{p} \rangle \leq 0$, which can be rewritten as $\langle \mathbf{x} - \mathbf{p}, \mathbf{p} - \mathbf{z} \rangle \geq 0$. This implies $\mathbf{p}$ is the projection of $\mathbf{x}$ onto C by Theorem 13.2.5. Moreover, the hyperplane $H = \{\mathbf{z} \in V \mid \langle \mathbf{a}, \mathbf{z} \rangle = b\}$, with $b = \langle \mathbf{a}, \mathbf{p} \rangle$, is the desired hyperplane containing $\mathbf{p}$. □

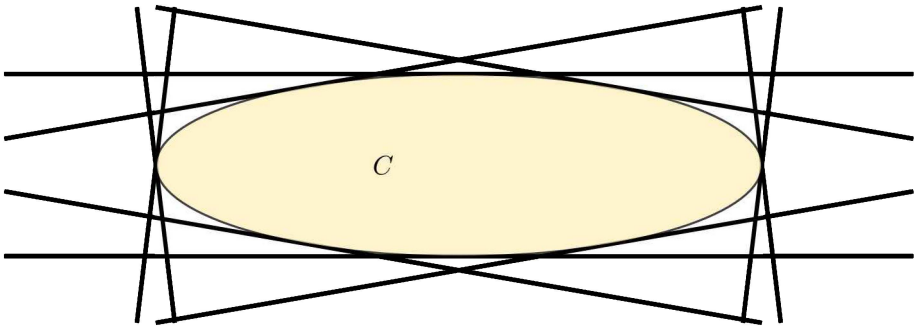


Figure 13.7. *Minkowski's theorem (Corollary 13.2.9) guarantees that any closed convex set C can be described as the intersection of supporting half spaces. Here several of the supporting hyperplanes are drawn.*

Corollary 13.2.9 (Minkowski). *Any nonempty, closed, convex set in $C \subset V$ is equal to the intersection of all half spaces that support it. Figure 13.7 gives an illustration of this.*

Proof. Let $\mathscr{H}$ be the set of all half spaces that support C, and let K be the intersection of all half spaces in $\mathscr{H}$. Clearly, $C \subset K$, so it suffices to show that $K \subset C$. Suppose there exists $\mathbf{x} \in K \setminus C$. By the separation lemma (Lemma 13.2.7), there exists a supporting half space containing C but not $\mathbf{x}$. It follows that $\mathbf{x} \notin K$, a contradiction. Therefore, $C = K$. □

13.2.3 Separating Hyperplanes

Given any convex set C and point $\mathbf{x} \notin C$, the separation lemma (Lemma 13.2.7) guarantees that there is a hyperplane H supporting C that separates C and $\mathbf{x}$. Since H supports C, the separation is not strict. However, a much stronger separation result holds, namely, any two disjoint convex sets, where one is compact and the other closed, can always be strictly separated by a hyperplane. This is an important result, called the *separating hyperplane theorem* or the *Hahn–Banach separation theorem.* This is illustrated in Figure 13.8.

Theorem 13.2.10 (Separating Hyperplane Theorem). *For any nonempty, disjoint, convex subsets C and D with C compact and D closed, there exists a strictly separating hyperplane.*

Proof. Since C and D are disjoint subsets, with C compact and D closed, Exercise 5.33 of Volume 1 shows that

$$\operatorname{dist}(C, D) = \inf\{\|\mathbf{u} - \mathbf{v}\| \mid \mathbf{u} \in C, \mathbf{v} \in D\} > 0.$$

Moreover, there exist points $\mathbf{c} \in C$ and $\mathbf{d} \in D$ that achieve the minimum distance, that is, $\|\mathbf{c} - \mathbf{d}\| = \operatorname{dist}(C, D)$. Consider the hyperplane $\langle \mathbf{a}, \mathbf{x} \rangle = b$ defined by the perpendicular bisector of the points $\mathbf{c}$ and $\mathbf{d}$; see Example 13.1.19 and Figure 13.8,

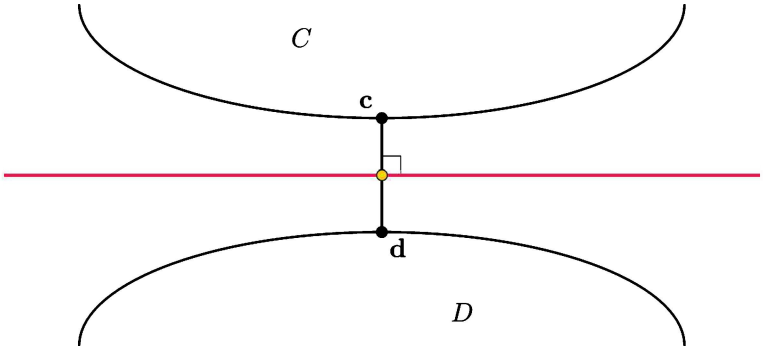


Figure 13.8. *Construct a strictly separating hyperplane for the convex sets C and D by taking the two distance-minimizing points $\mathbf{c} \in C$ and $\mathbf{d} \in D$ and choosing the perpendicular bisector (red) through the midpoint $\frac{\mathbf{c}+\mathbf{d}}{2}$ (yellow).*

that is, let

$$\mathbf{a} = \mathbf{d} - \mathbf{c} \qquad \text{and} \qquad b = \frac{\|\mathbf{d}\|^2 - \|\mathbf{c}\|^2}{2}.$$

Set $h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle - b$. It suffices to show that h is negative on C and positive on D. Suppose there exists $\mathbf{u} \in D$ such that $h(\mathbf{u}) \leq 0$. We have

$$\begin{aligned} h(\mathbf{u}) &= \langle \mathbf{d} - \mathbf{c}, \mathbf{u} \rangle - \frac{\|\mathbf{d}\|^2 - \|\mathbf{c}\|^2}{2} = \left\langle \mathbf{d} - \mathbf{c}, \mathbf{u} - \frac{\mathbf{d}+\mathbf{c}}{2} \right\rangle \\ &= \left\langle \mathbf{d} - \mathbf{c}, \mathbf{u} - \mathbf{d} + \frac{\mathbf{d}-\mathbf{c}}{2} \right\rangle = \langle \mathbf{d} - \mathbf{c}, \mathbf{u} - \mathbf{d} \rangle + \frac{\|\mathbf{d}-\mathbf{c}\|^2}{2}. \end{aligned}$$

Hence $\langle \mathbf{d} - \mathbf{c}, \mathbf{u} - \mathbf{d} \rangle < 0$. However, since

$$\frac{d}{dt} \|(1-t)\mathbf{d} + t\mathbf{u} - \mathbf{c}\|^2 \Big|_{t=0} = 2 \langle \mathbf{d} - \mathbf{c}, \mathbf{u} - \mathbf{d} \rangle < 0,$$

there exists some $0 < t < 1$ such that $\|(1-t)\mathbf{d} + t\mathbf{u} - \mathbf{c}\|^2 < \|\mathbf{d} - \mathbf{c}\|^2$. In other words, $(1-t)\mathbf{d} + t\mathbf{u}$ lies in D and is closer to $\mathbf{c}$ than $\mathbf{d}$ is. This is a contradiction since $\mathbf{d}$ is the point in D that is closest to $\mathbf{c}$. Thus $h(\mathbf{u}) > 0$ for all $\mathbf{u} \in D$. The proof follows similarly for C. □

Corollary 13.2.11. *If $C \subset V$ is closed, convex, and not empty, then for any $\mathbf{x} \notin C$ there exists a hyperplane that strictly separates $\{\mathbf{x}\}$ and C.*

Remark 13.2.12. It is important that one of the sets be compact for the strict separation in the theorem to hold. In the noncompact case it can still be shown that there exists a separating hyperplane, but the separation might not be strict. In Exercise 13.14 you are asked to give an example of two nonempty closed convex disjoint sets that cannot be strictly separated.

13.3 Fundamentals of Linear Optimization

A linear optimization problem is one where both the objective and the constraints are all linear (affine) functions.

13.3.1 Standard Form for Linear Optimization

Many seemingly different optimization problems are actually equivalent. In order to avoid having to talk about all the different equivalent forms of a linear optimization problem every time we want to prove a result, we usually assume that our problem has been put into the following standard form.

Definition 13.3.1. *An optimization problem is* linear *if the objective function and the constraint functions are linear functions. A linear optimization problem is in* standard form *if it can be written as*

$$\begin{aligned} &\text{minimize} && \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ &\textit{subject to} && A\mathbf{x} \preceq \mathbf{b}, \\ &&& \mathbf{x} \succeq \mathbf{0}, \end{aligned} \tag{13.4}$$

where $A \in M_{m\times n}(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. The symbols $\preceq$ and $\succeq$ denote entrywise inequality. Written out in coordinates, this takes the form

$$\begin{aligned} &\text{minimize} && c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ &\textit{subject to} && a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1, \\ &&& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2, \\ &&& \qquad\vdots \\ &&& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m, \\ &&& x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0. \end{aligned} \tag{13.5}$$

The inputs $x_1, \ldots, x_n$ are called the decision variables. *A point $\mathbf{x}$ is said to be* feasible *if it satisfies the constraints. The set $\mathscr{F}$ of feasible points is called the* feasible set. *If $\mathscr{F} = \emptyset$, the problem is* infeasible*; otherwise it is* feasible.

Nota Bene 13.3.2. Linear optimization is often called *linear programming*. The word *programming* here is an old-fashioned way of referring to the tables that were traditionally made to solve linear optimization problems. We avoid this usage of the term *programming* in this book to prevent confusion with computer programming.

Example 13.3.3. The linear optimization problem

$$\begin{aligned} &\text{maximize} && 5x_1 - x_2 + 6x_3 + 7 \\ &\text{subject to} && 4x_1 + x_2 + x_3 \leq -2, \\ &&& x_2 - x_1 + 8 \geq 1, \\ &&& x_1 \geq 0, x_2 \leq 0, x_3 \geq 0 \end{aligned}$$

can be rewritten in standard form as follows. First, the constant 7 in the objective does not change the optimizer, so it may be dropped. Second, the problem may be converted to a minimization problem by changing the sign of

the objective. Third, the constant 8 in the constraint may be combined with the bound of 1 and that constraint multiplied by -1 to change the constraint from $\geq$ to $\leq$, giving

$$\begin{aligned} \text{minimize} \quad & -5x_1 + x_2 - 6x_3 \\ \text{subject to} \quad & 4x_1 + x_2 + x_3 \leq -2, \\ & x_1 - x_2 \leq 7, \\ & x_1 \geq 0, x_2 \leq 0, x_3 \geq 0. \end{aligned}$$

Finally, replacing x_2 by $-x_2$ gives

$$\begin{aligned} \text{minimize} \quad & -5x_1 - x_2 - 6x_3 \\ \text{subject to} \quad & 4x_1 - x_2 + x_3 \leq -2, \\ & x_1 + x_2 \leq 7, \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0. \end{aligned}$$

This has the matrix form (13.4), where

$$A = \begin{bmatrix} 4 & -1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ 7 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} -5 \\ -1 \\ -6 \end{bmatrix}.$$

Remark 13.3.4. In the standard form the two constraints $A\mathbf{x} \preceq \mathbf{b}$ and $\mathbf{x} \succeq \mathbf{0}$ can be combined into one inequality with $\left[\begin{smallmatrix} -I \\ A \end{smallmatrix}\right] \mathbf{x} \preceq \left[\begin{smallmatrix} \mathbf{0} \\ \mathbf{b} \end{smallmatrix}\right]$. Thus, we sometimes write a linear optimization problem as

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{subject to} \quad & \widetilde{A}\mathbf{x} \preceq \widetilde{\mathbf{b}}, \end{aligned} \tag{13.6}$$

where $\widetilde{A} = \left[\begin{smallmatrix} -I \\ A \end{smallmatrix}\right]$ and $\widetilde{\mathbf{b}} = \left[\begin{smallmatrix} \mathbf{0} \\ \mathbf{b} \end{smallmatrix}\right]$.

13.3.2 The Fundamental Theorem

The feasible set of a linear optimization problem is an intersection of finitely many half spaces. Such an intersection is called a *polyhedron*. The fundamental theorem of linear optimization (Theorem 13.3.13) guarantees that every linear optimization problem with a nonempty, bounded feasible set always has a minimizer that is a vertex of the feasible polyhedron. In other words, the search for a minimizer boils down to only searching among the vertices, and there are only finitely many of these.

Definition 13.3.5. *A* polyhedron P *in* $\mathbb{R}^n$ *is the intersection of finitely many half spaces. In other words, a polyhedron* P *is a set that can be written in the form*

$$P = \bigcap_{i=1}^{m} \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}_i^{\mathsf{T}}\mathbf{x} \leq b_i\} = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \preceq \mathbf{b}\},$$

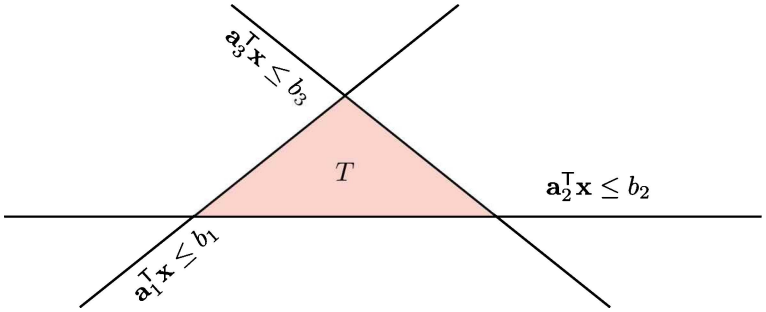


Figure 13.9. *The intersection of half spaces forms a polyhedron. The intersection of the half spaces* $\mathbf{a}_1^\mathsf{T}\mathbf{x} \leq b_1$ *with* $\mathbf{a}_2^\mathsf{T}\mathbf{x} \leq b_2$ *and* $\mathbf{a}_3^\mathsf{T}\mathbf{x} \leq b_3$ *is the triangle* T*.*

where A *is the* $m \times n$ *matrix whose* i*th row is* $\mathbf{a}_i^\mathsf{T}$ *and* $\mathbf{b}$ *is the column vector whose* i*th entry is* b_i*; see Figure 13.9 for a graphical depiction. The* dimension *of* P*, denoted* $\dim P$*, is the dimension of the smallest affine subset of* $\mathbb{R}^n$ *containing* P*.*

Remark 13.3.6. A polyhedron is a closed, convex set, since it is the intersection of closed (convex) half spaces.

Definition 13.3.7. *Let* $C \subset \mathbb{R}^n$ *be a convex set. An* extreme point*, or* vertex*, of* C *is a point* $\mathbf{v} \in C$ *that cannot be written as the midpoint of any two distinct points in* C*. In other words, a point* $\mathbf{v} \in C$ *is an extreme point if the equality* $\mathbf{v} = \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2)$ *for* $\mathbf{x}_1, \mathbf{x}_2 \in P$ *implies that* $\mathbf{v} = \mathbf{x}_1 = \mathbf{x}_2$*. This is depicted in Figure 13.10. We denote the set of vertices of* C *by* $\operatorname{vert}(C)$*.*

A vertex of a polyhedron can be identified by the space of constraints that are *active* at that vertex.

Definition 13.3.8. *Let* $P = \bigcap_{i=1}^m \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}_i^\mathsf{T}\mathbf{x} \leq b_i\}$ *be a polyhedron defined by each of the half spaces* $\mathbf{a}_i^\mathsf{T}\mathbf{x} \leq b_i$*. For any* $\mathbf{p} \in P$*, the* i*th constraint* $\mathbf{a}_i^\mathsf{T}\mathbf{x} \leq b_i$ *is* active *(or* binding*) at* $\mathbf{p}$ *if it satisfies the equality*

$$\mathbf{a}_i^\mathsf{T}\mathbf{p} = b_i;$$

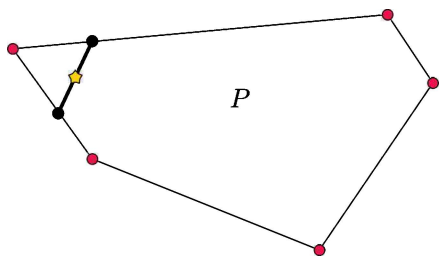


Figure 13.10. *A point (yellow) that is the midpoint of two other distinct points in a polyhedron* P *(gray) is not a vertex of* P*. The vertices (red) of* P *are the points that cannot be written as the midpoint of two other distinct points in* P*.*

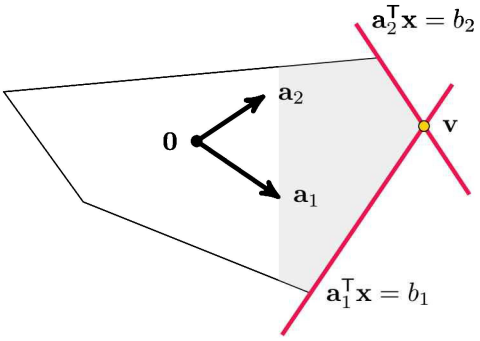


Figure 13.11. *The active constraints at a vertex* $\mathbf{v}$ *of a polyhedron are those which are equalities at* $\mathbf{v}$*. The active constraints are* $\mathbf{a}_1^\mathsf{T}\mathbf{x} \leq b_1$ *and* $\mathbf{a}_2^\mathsf{T}\mathbf{x} \leq b_2$*, corresponding to the two red hyperplanes. The active set is* $A_\mathbf{v} = \{\mathbf{a}_1, \mathbf{a}_2\}$*, which spans* $\mathbb{R}^2$*, as required by Theorem 13.3.9.*

otherwise it is inactive*, or* nonbinding*; see Figure 13.11. The* active set *of* $\mathbf{p} \in P$ *is the set* $A_\mathbf{p} = \{\mathbf{a}_i \mid \mathbf{a}_i^\mathsf{T}\mathbf{p} = b_i \text{ for some } i = 1, \ldots, m\}$.

Theorem 13.3.9. *Let* $P \subset \mathbb{R}^n$ *be a polyhedron. A point* $\mathbf{v} \in P$ *is a vertex of* P *if and only if the active set* $A_\mathbf{v}$ *spans all of* $\mathbb{R}^n$*, that is,* $\operatorname{span} A_\mathbf{v} = \mathbb{R}^n$.

Proof. Suppose $A_\mathbf{v}$ spans $\mathbb{R}^n$ but $\mathbf{v}$ is not a vertex. This implies $\mathbf{v} = \frac{1}{2}(\mathbf{v}_1 + \mathbf{v}_2)$ for distinct points $\mathbf{v}_1, \mathbf{v}_2 \in P$. Let $\mathbf{a}_i \in A_\mathbf{v}$. Since $\mathbf{v}_1, \mathbf{v}_2 \in P$, we have that $\mathbf{a}_i^\mathsf{T}\mathbf{v}_1 \leq b_i$ and $\mathbf{a}_i^\mathsf{T}\mathbf{v}_2 \leq b_i$. However, if either of the inequalities is strict, then $\mathbf{a}_i^\mathsf{T}\mathbf{v} = \frac{1}{2}(\mathbf{a}_i^\mathsf{T}\mathbf{v}_1 + \mathbf{a}_i^\mathsf{T}\mathbf{v}_2) < b_i$, which is a contradiction, and thus we have that both $\mathbf{a}_i^\mathsf{T}\mathbf{v}_1 = b_i$ and $\mathbf{a}_i^\mathsf{T}\mathbf{v}_2 = b_i$. Let M be the matrix whose rows are $\mathbf{a}_i^\mathsf{T}$ for each $\mathbf{a}_i \in A_\mathbf{v}$. Since $\mathbf{a}_i^\mathsf{T}\mathbf{v}_1 = \mathbf{a}_i^\mathsf{T}\mathbf{v}_2$ for all $\mathbf{a}_i \in A_\mathbf{v}$, it follows that $M\mathbf{v}_1 = M\mathbf{v}_2$, which implies that the null space of M is not trivial, and thus M is not of full rank. This is a contradiction, since the row space of M was assumed to span $\mathbb{R}^n$.

Now suppose that $A_\mathbf{v}$ does not span $\mathbb{R}^n$, and so there exists a nonzero $\mathbf{u} \in (\operatorname{span} A_\mathbf{v})^\perp$. This implies that $\mathbf{a}_i^\mathsf{T}(\mathbf{v} + \alpha\mathbf{u}) = b_i$ for each $\mathbf{a}_i \in A_\mathbf{v}$. For any $\mathbf{a}_i \notin A_\mathbf{v}$, we have $\mathbf{a}_i^\mathsf{T}\mathbf{v} < b_i$, and thus there exists some $\varepsilon > 0$ so that $\mathbf{v} + \varepsilon\mathbf{u}, \mathbf{v} - \varepsilon\mathbf{u} \in P$. Note that $\mathbf{v} = \frac{1}{2}((\mathbf{v} + \varepsilon\mathbf{u}) + (\mathbf{v} - \varepsilon\mathbf{u}))$, and thus $\mathbf{v}$ is not a vertex of P. ∎

Corollary 13.3.10. *If* $\mathbf{v} \in P \subset \mathbb{R}^n$ *is a vertex, then* $|A_\mathbf{v}| \geq n$*, that is,* $\mathbf{v}$ *must lie on at least* n *hyperplanes defining* P.

Corollary 13.3.11. *The number of vertices of a polyhedron* $P \subset \mathbb{R}^n$ *defined by the intersection of* k *half spaces is at most* $\binom{k}{n}$.

Proof. Every vertex must lie on at least n of the k hyperplanes defining P, and there are $\binom{k}{n}$ ways to choose a collection of n things from among the k hyperplanes defining P. ∎

The following theorem (Minkowski–Steinitz) is key to proving the fundamental theorem of linear optimization. Although we prove it only for $\mathbb{R}^n$, the Minkowski–

Steinitz theorem also holds in the infinite-dimensional case, where it is called the *Krein–Milman* theorem.

Theorem 13.3.12 (Minkowski–Steinitz). *Every nonempty compact convex set $C \subset \mathbb{R}^n$ is the convex hull of its vertices, that is, $C = \operatorname{conv}(\operatorname{vert}(C))$.*

The proof of the Minkowski–Steinitz theorem is given in Section 13.3.4.

Theorem 13.3.13 (Fundamental Theorem of Linear Optimization). *If the feasible set $\mathscr{F}$ of a linear optimization problem is bounded and not empty, then there exists a vertex $\mathbf{v} \in \mathscr{F}$ which is optimal. See Figure 13.12 for an illustration.*

Proof. The intersection of closed half spaces is closed, so $\mathscr{F}$ is both closed and bounded and thus compact. By the extreme value theorem (Volume 1, Corollary 5.5.7) there exists a minimizer $\mathbf{x}^* \in \mathscr{F}$, that is, $\mathbf{c}^\mathsf{T}\mathbf{x} \geq \mathbf{c}^\mathsf{T}\mathbf{x}^*$ for all $\mathbf{x} \in \mathscr{F}$.

Suppose that no vertex of $\mathscr{F}$ is a minimizer. In this case, $\mathbf{c}^\mathsf{T}\mathbf{v} > \mathbf{c}^\mathsf{T}\mathbf{x}^*$ for each vertex $\mathbf{v}$ in the finite set $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ of all vertices $\mathscr{F}$. Since $\mathscr{F}$ is a polyhedron, it can be written as the convex hull of its vertices (by Minkowski–Steinitz), hence the minimizer $\mathbf{x}^*$ can be written as

$$\mathbf{x}^* = \lambda_1 \mathbf{v}_1 + \cdots + \lambda_k \mathbf{v}_k,$$

where $\lambda_i \geq 0$ and $\sum_{i=1}^k \lambda_i = 1$. This gives

$$\mathbf{c}^\mathsf{T}\mathbf{x}^* = \lambda_1 \mathbf{c}^\mathsf{T}\mathbf{v}_1 + \cdots + \lambda_k \mathbf{c}^\mathsf{T}\mathbf{v}_k > (\lambda_1 + \cdots + \lambda_k)\mathbf{c}^\mathsf{T}\mathbf{x}^* = \mathbf{c}^\mathsf{T}\mathbf{x}^*,$$

which is a contradiction. Thus, $\mathbf{c}^\mathsf{T}\mathbf{x}^* = \mathbf{c}^\mathsf{T}\mathbf{v}$ for some $\mathbf{v} \in V$. □

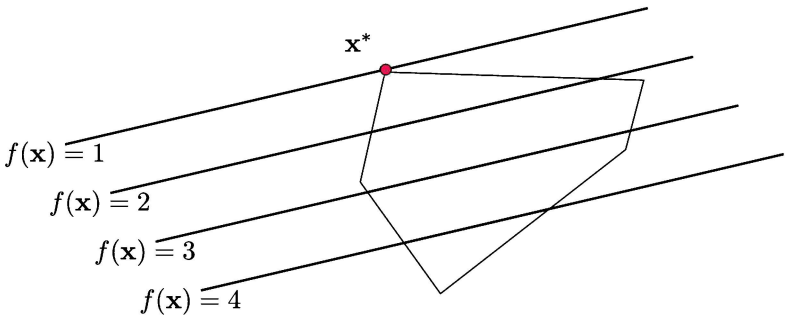


Figure 13.12. *The fundamental theorem of linear optimization says that a nonempty, bounded feasible set has a vertex that is a minimizer. The level sets of the objective function for a linear optimization problem are hyperplanes. Of all level sets that intersect the feasible set, the one with the smallest value must intersect at either a vertex or along a face of the feasible set. In either case, there is an optimizer at a vertex.*

Remark 13.3.14. This theorem is fundamental to solving linear optimization problems because it guarantees that once feasibility and boundedness are established, one of at most $\binom{m}{n}$ vertices must be a minimizer. One of the main methods for finding a minimizer is the *simplex method*, which moves along adjacent vertices with decreasing values of the objective function until a minimizing vertex is found. This is described in detail in the next section.

13.3.3 Applications

Linear optimization problems arise in many real-world settings. We consider two examples here.

Application: Production Schedules

A company produces n different products using m different raw materials. Let r_j denote the revenue associated with the production of one unit of product j, and let x_j denote the number of units of product j produced. Thus, the total revenue is $\sum_{j=1}^{n} r_j x_j$.

Let ξ_i denote the cost of purchasing one unit of the ith raw material and assume that there are at most b_i units of the ith raw material available for purchase. Assume that producing one unit of product j requires a_{ij} units of raw material i, so that the cost of producing one unit of product j is given by $c_j = \sum_{i=1}^{m} \xi_i a_{ij}$.

The net profit p_j for each unit of product j is $p_j = r_j - \sum_{i=1}^{m} \xi_i a_{ij}$ and the total profit is $\sum_{j=1}^{n} p_j x_j$. Therefore, the production planner should solve

$$\begin{aligned} \text{minimize} \quad & -\sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, m, \\ & x_j \geq 0, \qquad\qquad\quad j = 1, 2, \ldots, n. \end{aligned} \tag{13.7}$$

Application: Network Flow

Consider a directed graph $G = (V, E)$, where each node represents a producer or consumer of a given product and edges are transportation channels for the product. To make things concrete, we assume the product is cheese, and edges correspond to trucking routes for transporting cheese.

Let b_i denote the amount of cheese produced or consumed at the ith node, where a negative value means demand. Assume that $\sum_{i \in V} b_i = 0$ so that the system does not grow or shrink over time (economists would say the *market clears*). For each edge $(i, j) \in E$, let c_{ij} denote the cost of transporting one unit cheese from node i to node j, and let x_{ij} denote the number of units moved from node i to node j. The total transportation cost is therefore

$$\text{cost} = \sum_{(i,j) \in E} c_{ij} x_{ij}.$$

We want to move all the cheese from the producers to the consumers as cheaply as possible. We also want to do this in such a way that there is no surplus cheese left

at any node, that is, so that the flow is *balanced* at each node, which means that

$$\sum_{(v,j)\in E} x_{vj} - \sum_{(i,v)\in E} x_{iv} = b_v$$

for each $v \in V$. Moreover, assume that each $x_{ij} \geq 0$ so that the flow goes with the direction of the edge. This is important because flowing backward on edge (i,j) would be the same as flowing forward on edge (j,i), which should incur a positive cost at the rate of c_{ji} per unit of cheese, instead of a negative cost (namely $x_{ij}c_{ij}$, if $x_{ij} < 0$). There may also be some upper bound $x_{ij} \leq a_{ij}$ on the flow along each edge—for example, if the number of trucks available on each route is limited.

This gives the linear optimization problem of finding $\mathbf{x} \in \mathbb{R}^{|E|}$ to

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{(v,j)\in E} x_{vj} - \sum_{(i,v)\in E} x_{iv} = b_v \quad \forall v \in V, \\ & \mathbf{0} \preceq \mathbf{x} \preceq \mathbf{a}, \end{aligned}$$

where $\mathbf{x}$ is the vector of all the x_{ij} and $\mathbf{a} \in \mathbb{R}^{|E|}$ is the vector of all the a_{ij}.

13.3.4 *Proof of Minkowski–Steinitz

We now prove the Minkowski–Steinitz theorem (Theorem 13.3.12); however, we first need the following lemma.

Lemma 13.3.15. *Let $C \subset \mathbb{R}^n$ be nonempty, compact, and convex. If H is a supporting hyperplane of C, then* $\operatorname{vert}(C \cap H) = \operatorname{vert}(C) \cap H$.

Proof. Let $\mathbf{v} \in \operatorname{vert}(C) \cap H$. If $\mathbf{x}, \mathbf{y} \in C \cap H$ satisfy $\mathbf{v} = \frac{1}{2}(\mathbf{x}+\mathbf{y})$, then $\mathbf{v} = \mathbf{x} = \mathbf{y}$, which implies $\mathbf{v} \in \operatorname{vert}(C \cap H)$. Thus, $\operatorname{vert}(C) \cap H \subset \operatorname{vert}(C \cap H)$. Conversely, suppose $\mathbf{v} \in C \cap H$ is not an extreme point of C. Thus we can find $\mathbf{x}, \mathbf{y} \in C$, with $\mathbf{x} \neq \mathbf{y}$, such that $\mathbf{v} = \frac{1}{2}(\mathbf{x} + \mathbf{y})$. Assume without loss of generality that $H = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ and $C \subset \{\mathbf{z} \in \mathbb{R}^n \mid \langle \mathbf{a}, \mathbf{z} \rangle \leq b\}$. If $\mathbf{x} \notin H$ or $\mathbf{y} \notin H$, then $\langle \mathbf{a}, \mathbf{x} \rangle < b$ or $\langle \mathbf{a}, \mathbf{y} \rangle < b$, which implies $b = \langle \mathbf{a}, \mathbf{v} \rangle = \frac{1}{2}(\langle \mathbf{a}, \mathbf{x} \rangle + \langle \mathbf{a}, \mathbf{y} \rangle) < b$, a contradiction. Hence, $\mathbf{x}, \mathbf{y} \in H$, which implies that $\mathbf{v}$ is not an extreme point of $C \cap H$. It follows that $\operatorname{vert}(C \cap H) \subset \operatorname{vert}(C) \cap H$. ∎

The proof of the Minkowski–Steinitz theorem now proceeds by induction on the dimension n. In the case of $n = 1$, a nonempty compact convex set C in $\mathbb{R}$ is an interval of the form $C = [a, b]$. It is clear that $\operatorname{vert}(C) = \{a, b\}$ and $C = \operatorname{conv}(\{a, b\})$.

For $n > 1$, assume by induction that the theorem holds for $\mathbb{R}^{n-1}$ and consider a nonempty closed convex set $C \subset \mathbb{R}^n$. Let $\mathbf{x} \in C$ be a point that is not a vertex. If $\mathbf{x}$ lies in a supporting hyperplane H of C, then $C \cap H$ is convex. Translating this set by $-\mathbf{x}$ gives a convex set $(C \cap H) - \mathbf{x}$ in the $(n-1)$-dimensional linear subspace $H - \mathbf{x} \subset \mathbb{R}^n$. By the induction hypothesis the conclusion of the theorem holds on the set $(C \cap H) - \mathbf{x}$. It is straightforward to check that it also holds on $C \cap H$.

Combining this with the lemma gives $\mathbf{x} \in \operatorname{conv}(\operatorname{vert}(C \cap H)) = \operatorname{conv}(\operatorname{vert}(C) \cap H) \subset \operatorname{conv}(\operatorname{vert}(C))$.

If $\mathbf{x}$ does not lie in the boundary of any supporting half space, then since $\mathbf{x}$ is also not a vertex, there must be distinct $\mathbf{r}, \mathbf{s} \in C$ such that $\mathbf{x} = \frac{1}{2}(\mathbf{r} + \mathbf{s})$. Let L be the one-dimensional affine set $L = \{t\mathbf{r} + (1-t)\mathbf{s} \mid t \in \mathbb{R}\} = \mathbf{s} + \operatorname{span}(\mathbf{r} - \mathbf{s})$. The set $(L \cap C) - \mathbf{s}$ is a compact and convex subset of a one-dimensional vector space, hence the conclusion of the theorem applies to $(L \cap C) - \mathbf{s}$, and also to $L \cap C$. Let $\mathbf{r}'$ and $\mathbf{s}'$ denote the vertices of $L \cap C$. Thus $\mathbf{x}$ lies in the convex span of $\mathbf{r}'$ and $\mathbf{s}'$. For each of the points $\mathbf{r}'$ and $\mathbf{s}'$ there is a sequence of points outside of C that converges to the point, so by the supporting half space theorem (Theorem 13.2.8), $\mathbf{r}'$ and $\mathbf{s}'$ lie in the boundary H of a supporting half space and thus lie in the convex span $\operatorname{conv}(\operatorname{vert}(C \cap H)) = \operatorname{conv}(\operatorname{vert}(C)) \cap H \subset \operatorname{conv}(\operatorname{vert}(C))$. Therefore $\mathbf{x}$ lies in $\operatorname{conv}(\{\mathbf{r}', \mathbf{s}'\}) \subset \operatorname{conv}(\operatorname{vert}(C))$, as required. □

13.4 The Simplex Method I

The fundamental theorem of linear optimization (Theorem 13.3.13) says that an optimizer can be found among the vertices of the feasible set. The simplex method is an algorithm for searching through the set of all vertices to find the minimizer. In most cases it performs very well, but it is possible to design pathological cases where it visits every single vertex before finding the optimizer.

The geometric idea behind the simplex method is to start at any feasible vertex $\mathbf{v}$ and examine the vertices neighboring $\mathbf{v}$ to see if one improves the objective f; that is, find a neighbor $\mathbf{w}$ of $\mathbf{v}$ such that $f(\mathbf{w}) \leq f(\mathbf{v})$. If such a neighbor can be found, then move to the neighbor (set $\mathbf{v} = \mathbf{w}$) and repeat. If $f(\mathbf{w}) > f(\mathbf{v})$ for every neighbor $\mathbf{w}$, then stop and return $\mathbf{v}$ as a minimizer. See Figure 13.13 for an illustration.

This is the main idea of the simplex method, but there are a few technical details to work out, including how to find an initial feasible vertex, and what to do when $f(\mathbf{w}) = f(\mathbf{v})$ in order to ensure that the algorithm does not visit the same vertex more than once. The rest of this section and the next give the details of how to execute the algorithm.

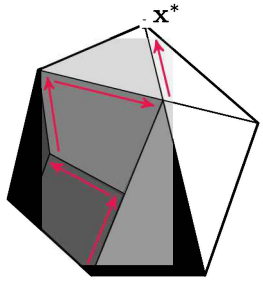


Figure 13.13. *The feasible set for a linear optimization problem is a polyhedron. If an optimal point exists, then an optimal point occurs at a vertex of the polyhedron. The simplex method searches for optimal points by moving among adjacent vertices in a direction that decreases the value of the objective function until it finds a minimizing vertex* $\mathbf{x}^*$.

13.4.1 Slack Variables and Vertices

Corollary 13.3.10 gives a method for identifying vertices, namely, look for points that lie on at least n of the hyperplanes defining the feasible set. While these points may not always be vertices (in the unlikely case that the hyperplanes are linearly dependent), every vertex is of this form.

Feasible points lie on a bounding hyperplane when the inequality in the corresponding constraint is an equality. To track these more easily, we introduce nonnegative variables $w_1, \ldots, w_m$, called *slack variables*, and rewrite the linear optimization problem as a system with equalities:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} + \mathbf{w} = \mathbf{b}, \\ & \mathbf{x} \succeq \mathbf{0}, \\ & \mathbf{w} \succeq \mathbf{0}. \end{aligned} \tag{13.8}$$

This system is equivalent to the original problem, since $A\mathbf{x} \preceq \mathbf{b}$ if and only if $A\mathbf{x} + \mathbf{w} = \mathbf{b}$ and $\mathbf{w} \succeq \mathbf{0}$. We summarize this idea in the following proposition.

Proposition 13.4.1. *For a linear optimization problem with n variables and m constraints in standard form* (13.4)*, a point $\mathbf{x} \in \mathbb{R}^n$ is feasible if and only if there exists $\mathbf{w} \in \mathbb{R}^m$ such that $\mathbf{x} \succeq \mathbf{0}$, $\mathbf{w} \succeq \mathbf{0}$, and $A\mathbf{x} + \mathbf{w} = \mathbf{b}$.*

We call any assignment of $\mathbf{x}$ and $\mathbf{w}$ a *configuration*. Note that we can always solve for $\mathbf{w}$ in terms of $\mathbf{x}$ as $\mathbf{w} = \mathbf{b} - A\mathbf{x}$.

Example 13.4.2. Adding slack variables to the problem

$$\begin{aligned} \text{minimize} \quad & x_1 - x_2 \\ \text{subject to} \quad & 2x_1 + x_2 \leq 4, \\ & x_1 + 2x_2 \leq 3, \\ & x_1, x_2 \geq 0 \end{aligned}$$

gives an equivalent problem

$$\begin{aligned} \text{minimize} \quad & x_1 - x_2 \\ \text{subject to} \quad & w_1 + 2x_1 + x_2 = 4, \\ & w_2 + x_1 + 2x_2 = 3, \\ & x_1, x_2, w_1, w_2 \geq 0. \end{aligned}$$

This polyhedron is depicted in Figure 13.14.

The advantage of this new formulation is that it gives a simpler description of the bounding hyperplanes. In the original version (13.4), the hyperplanes bounding the feasible set are given by the equations $x_1 = 0$, $x_2 = 0$, $\ldots$, $x_n = 0$ and by

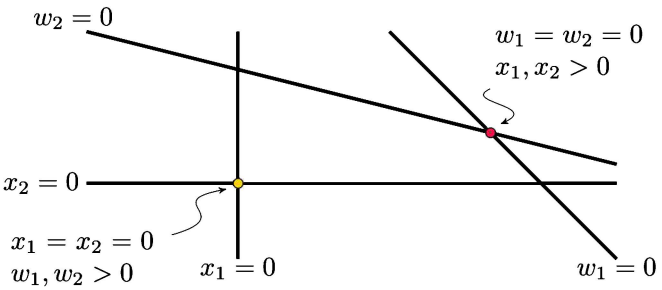


Figure 13.14. *The polyhedron of Example 13.4.2. The locus where $w_1 = 0$ is the hyperplane $2x_1 + x_2 = 4$, and the locus $w_2 = 0$ is the hyperplane $x_1 + 2x_2 = 3$. The red point is the intersection $w_1 = w_2$, and at that point both x_1 and x_2 are strictly positive. Similarly, the yellow point is the intersection $x_1 = x_2 = 0$ and at that point w_1 and w_2 are both strictly positive.*

each of the rows of the equation $A\mathbf{x} = \mathbf{b}$. But in the new description, the bounding hyperplanes are given by each of the equations $x_1 = 0$, $x_2 = 0$, ..., $x_n = 0$ and $w_1 = 0$, $w_2 = 0$, ..., $w_m = 0$. So, in the previous example the equation $w_1 = 0$ corresponds to the hyperplane $2x_1 + x_2 = 4$ in $\mathbb{R}^2$.

Nota Bene 13.4.3. Although a configuration of $\mathbf{x}$ and $\mathbf{w}$ lies in $\mathbb{R}^{n+m}$, when we discuss *the hyperplane given by* $w_i = 0$, we mean the hyperplane in $\mathbb{R}^n$ defined by the ith row of $A\mathbf{x} = \mathbf{b}$ and not a hyperplane in the larger space $\mathbb{R}^{n+m}$.

13.4.2 The Simplex Method

As outlined in the introduction to this section, the simplex method starts at a feasible vertex $\mathbf{v}$, examines the vertices neighboring $\mathbf{v}$ to see if one improves the objective f, and if such a neighbor can be found, it moves to the neighbor and repeats.

We begin with an example of how this method is actually executed, solving the linear optimization problem in Example 13.4.2. Using the formulation of the problem with slack variables, we wish to choose x_1, x_2, w_1, w_2 in order to

$$\begin{aligned} \text{minimize} \quad & x_1 - x_2 \\ \text{subject to} \quad & w_1 + 2x_1 + x_2 = 4, \\ & w_2 + x_1 + 2x_2 = 3, \\ & x_1, x_2, w_1, w_2 \geq 0. \end{aligned}$$

Let f be the objective function. Note that setting $x_1 = 0$ and $x_2 = 0$ gives $w_1 = 4$ and $w_2 = 3$, which is a feasible configuration, so $(x_1, x_2) = (0, 0)$ is a vertex of the feasible polyhedron $\mathscr{F}$; see Figure 13.14. We can solve for $\mathbf{w}$ in terms of $\mathbf{x}$ and

write the following system of equations, called a *dictionary*:

$$\begin{array}{rcrcrcr} f & = & 0 & + & x_1 & - & x_2 \\ \hline w_1 & = & 4 & - & 2x_1 & - & x_2 \\ w_2 & = & 3 & - & x_1 & - & 2x_2 \\ \text{dependent} & & & & \text{independent} & & \end{array} \tag{13.9}$$

Since all the variables must be nonnegative, we don't bother including that information in the dictionary. The objective function and the variables w_1 and w_2 are written on the left and they are expressed in terms of the variables x_1 and x_2.

Choosing a vertex of $\mathscr{F}$ corresponds to choosing two variables from the set $\{x_1, x_2, w_1, w_2\}$ to set to 0; we call these the *independent variables* and solve for the other, *dependent*, variables in terms of the independent variables.[52] In the preceding dictionary, the independent variables are x_1 and x_2, while the dependent variables are w_1 and w_2.

Each step of the simplex method starts at a given vertex, corresponding to a choice of n independent variables, and moves to a new vertex by trading one of the independent variables with a dependent variable. That means one of the formerly independent variables may become nonzero (hence positive) and one of the formerly dependent variables must be set to 0. In the current example, that means we may choose one of x_1 or x_2 to allow to become positive (and be dependent) and choose one of w_1 or w_2 to set to 0 (and be independent).

Since the coefficient of x_1 in f is positive, allowing x_1 to become positive (hence, dependent) would increase the value of the objective, which is undesirable, whereas the coefficient of x_2 is negative, so allowing x_2 to increase would decrease the objective, as desired. Thus x_2 must become dependent, and either w_1 or w_2 must become independent. Keeping $x_1 = 0$, the requirement $w_1 \geq 0$ gives $x_2 \leq 4$, and the requirement $w_2 \geq 0$ gives $x_2 \leq \frac{3}{2}$. Therefore, the largest x_2 can be is $\frac{3}{2}$, and $w_2 = 0$ gives the *binding constraint* on x_2. Thus w_2 must be the new independent variable.

Solving for x_2 in terms of x_1 and w_2 gives $x_2 = \frac{3}{2} - \frac{1}{2}x_1 - \frac{1}{2}w_2$. Substituting this into the equations of (13.9) for w_1 and f gives the following new dictionary:

$$\begin{array}{rcrcrcr} f & = & -\frac{3}{2} & + & \frac{3}{2}x_1 & + & \frac{1}{2}w_2 \\ \hline w_1 & = & \frac{5}{2} & - & \frac{3}{2}x_1 & + & \frac{1}{2}w_2 \\ x_2 & = & \frac{3}{2} & - & \frac{1}{2}x_1 & - & \frac{1}{2}w_2. \\ \text{dependent} & & & & \text{independent} & & \end{array}$$

Setting the independent variables, x_1 and w_2, to 0 corresponds to the vertex on the upper left of Figure 13.14, and the objective has the value $-\frac{3}{2}$ at this point. Since the objective f is now a positive linear combination of the independent variables x_1 and w_2, and those two variables must always remain nonnegative, the vertex corresponding to $x_1 = w_2 = 0$ must be a minimizer of f. So, $\mathbf{x}^* = (0, \frac{3}{2})$.

[52] The dependent variables are often called *basic* variables, and the independent variables are often called *nonbasic* variables. But we find these names confusing, since the nonbasic variables are the ones defining the basis of hyperplanes active at the current vertex.

Remark 13.4.4. Both the slack variables $w_1, \dots, w_m$ and the original variables $x_1, \dots, x_n$ are all treated the same way throughout this process, so it makes notation simpler if we relabel the slack variables to be $x_{n+1}, \dots, x_{n+m}$ by writing $x_{n+i} = w_i$. We do this from now on. This means that each of the hyperplanes bounding $\mathscr{F}$ is of the form $\{x_i = 0\}$ for some $i \in \{1, \dots, n+m\}$, and the corresponding equation is given by the ith row of $\widetilde{A}\mathbf{x} = \widetilde{\mathbf{b}}$ in (13.6).

In Algorithm 13.1 we give a more detailed description of how the simplex method is executed in general.

Begin at a feasible vertex, defined by hyperplanes

$$x_{i_1} = \cdots = x_{i_n} = 0.$$

Let $I = \{i_1, \dots, i_n\}$ be the set of indices of these independent variables and $D = \{d_1, \dots, d_m\}$ be the indices of the dependent variables, so that $I \cup D = \{1, \dots, n+m\}$.

(i) Solve for the objective f and all the x_j with $j \in D$ in terms of the independent variables (all x_i with $i \in I$), and write the corresponding dictionary, whose top row is the expression for f in terms of the independent variables, and each remaining row is the expression for some x_j with $j \in D$ in terms of the independent variables.

(ii) Identify an independent variable x_i for which the coefficient of x_i in f is strictly negative, so that increasing x_i would decrease f. If none exists, then stop: the current vertex, defined by I, is a minimizer.

(iii) For each $j \in D$, the inequality $x_j \geq 0$ combined with the conditions $x_{i'} = 0$ for all $i' \in I \setminus \{i\}$ gives a bound on x_i that must be satisfied.

  (a) If none of these is an upper bound for x_i, then x_i may increase arbitrarily, the feasible set $\mathscr{F}$ for this problem is unbounded, and f has no minimizer because it approaches $-\infty$. Stop.

  (b) Otherwise, the smallest of these upper bounds is called the *binding* (or *active*) constraint. Let $j \in D$ be an index corresponding to the binding constraint (there may be more than one dependent variable that yields the same constraint).

(iv) Trade the positions of i and j; that is, remove i from I and put it into D, and remove j from D and put it into I. The variable x_i is called the *entering* variable, as it enters D, and the variable x_j is the *leaving* variable.

(v) Repeat from step (i).

Algorithm 13.1. *Outline of the basic simplex method for solving linear optimization problems, beginning at a feasible vertex. The problem should be of the form* (13.8) *but where the slack variables* $w_1, \dots, w_m$ *are renamed to* $x_{n+1}, \dots, x_{n+m}$. *Coding up this algorithm is an exercise in the computing labs for this volume.*

Nota Bene 13.4.5. In the basic simplex method there are two choices to be made with each iteration. First, one independent variable must be chosen as the entering variable. Any independent variable x_i for which the coefficient of x_i in f is strictly negative in the objective f will do, but there may be some choices that improve the objective more than others. Second, once the entering variable is chosen, the leaving variable must be chosen. Again, there may be more than one candidate, and usually any dependent variable that imposes a binding constraint may be used (but see the next section for a discussion of *cycling*, where not just any choice will work).

There are many rules for choosing which variables to take as the entering and leaving variables. In the next section we discuss one of these rules, called *Bland's rule*. But in many cases the simplex method works well with just choosing the first variables you find that can be used; that is, choose as the entering variable the first variable you find that has a negative coefficient in the objective, and choose as the leaving variable the first variable you find that imposes a binding constraint on the entering variable.

The basic version of the simplex method converges in most cases, as described in the following proposition.

Proposition 13.4.6. *Assume that the feasible set of a linear optimization problem is bounded and not empty. If no vertex is visited more than once by the simplex method (for example, if the objective function strictly decreases at every step of the algorithm), then the simplex method terminates at a minimizer of the objective.*

Proof. Since the feasible set is bounded, it has a finite number of vertices. Since no vertex is visited more than once, the algorithm must terminate. The terminating vertex corresponds to a choice of n independent variables $x_{i_1}, \ldots, x_{i_n}$. Because the algorithm terminates at this vertex, the coefficient c_{i_ℓ} of each independent variable x_{i_ℓ} in the objective function $f = \zeta + \sum_{k=1}^{n} c_{i_k} x_{i_k}$ (where ζ is the constant term) is nonnegative. But every feasible point must satisfy $x_{i_\ell} \geq 0$, and hence f is minimized at the vertex $x_{i_1} = \cdots = x_{i_n} = 0$. □

Remark 13.4.7. Usually, every step of the simplex method strictly decreases the objective, but there are settings where that is not the case. Theorem 13.5.4 in Section 13.5.3 gives a method to ensure that no vertex is visited more than once, even in the case the objective is not strictly decreased.

Remark 13.4.8. Although we have worked out the steps of the simplex method in the examples of this section by hand, these are straightforward to program using standard methods of numerical linear algebra (see the accompanying labs for this volume). Except for the purpose of learning the method, executing these steps by hand is as silly as solving a large linear system of equations by hand.

13.4.3 Another Example

Consider the problem of choosing x_1, x_2, x_3 to

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2x_2 + 3x_3 \\ \text{subject to} \quad & 2x_1 + x_2 + x_3 \leq 4, \\ & -x_1 - 2x_2 + 4x_3 \leq 2, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Adding slack variables x_4 and x_5 gives an equivalent linear optimization problem

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2x_2 + 3x_3 \\ \text{subject to} \quad & 2x_1 + x_2 + x_3 + x_4 = 4, \\ & -x_1 - 2x_2 + 4x_3 + x_5 = 2, \\ & x_1, x_2, x_3, x_4, x_5 \geq 0. \end{aligned}$$

The point $x_1 = x_2 = x_3 = 0$ is feasible because plugging these values into the equations above gives $x_4 = 4$ and $x_5 = 2$ (both positive). Starting at this configuration, and solving for the objective f, as well as x_4 and x_5 in terms of the independent variables x_1, x_2, x_3, gives the initial dictionary

$$\begin{array}{rcrcrcrcr} f & = & 0 & - & x_1 & - & 2x_2 & + & 3x_3 \\ \hline x_4 & = & 4 & - & 2x_1 & - & x_2 & - & x_3 \\ x_5 & = & 2 & + & x_1 & + & 2x_2 & - & 4x_3. \end{array}$$

Since the coefficients of x_1 and x_2 in f are negative, increasing either x_1 or x_2 will decrease f. It doesn't matter which we choose, so take x_1. The constraint on x_1 given by $x_4 \geq 0$ (and all other independent variables set to 0) is $x_1 \leq 2$. The constraint of $x_5 \geq 0$ implies that $x_1 \geq -2$, which always holds, since x_1 is nonnegative. Thus the only binding constraint is given by $x_4 \geq 0$, so swap x_1 with x_4, making x_4 independent and x_1 dependent.

Solving the first equation for x_1 in terms of x_4 gives

$$x_1 = 2 - \frac{1}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4. \tag{13.10}$$

Replace the first constraint with (13.10), and substitute this expression for x_1 everywhere else x_1 occurs in the first dictionary. The resulting dictionary is

$$\begin{array}{rcrcrcrcr} f & = & -2 & - & \frac{3}{2}x_2 & + & \frac{7}{2}x_3 & + & \frac{1}{2}x_4 \\ \hline x_1 & = & 2 & - & \frac{1}{2}x_2 & - & \frac{1}{2}x_3 & - & \frac{1}{2}x_4 \\ x_5 & = & 4 & + & \frac{3}{2}x_2 & - & \frac{9}{2}x_3 & - & \frac{1}{2}x_4. \end{array}$$

Only x_2 has a negative coefficient in the objective function, so x_2 is the new entering variable. The first constraint $x_1 \geq 0$ gives $x_2 \leq 4$, and the second constraint $x_5 \geq 0$ gives $x_2 \geq -\frac{8}{3}$, which is not binding. Therefore, x_1 is the leaving variable.

Solve the first equation for x_2 to get $x_2 = 4 - 2x_1 - x_3 - x_4$, and substitute this into the previous dictionary to get

$$\begin{array}{rcrcrcrcr} f & = & -8 & + & 3x_1 & + & 5x_3 & + & 2x_4 \\ \hline x_2 & = & 4 & - & 2x_1 & - & x_3 & - & x_4 \\ x_5 & = & 10 & - & 3x_1 & - & 6x_3 & - & 2x_4. \end{array}$$

The independent variables x_1, x_3, and x_4 correspond to the vertex $x_1 = x_3 = x_4 = 0$, which gives $x_2 = 4$ and $x_5 = 10$.

Since all coefficients in the objective function are positive, the objective function cannot be further decreased by swapping a variable. Thus the solution to the original problem is $\mathbf{x}^* = (0, 4, 0)$.

The final dictionary corresponds to the minimization problem

$$\begin{aligned} \text{minimize} \quad & -8 + 3x_1 + 5x_3 + 2x_4 \\ \text{subject to} \quad & 2x_1 + x_3 + x_4 \leq 4, \\ & 3x_1 + 6x_3 + 2x_4 \leq 10, \\ & x_1, x_3, x_4 \geq 0, \end{aligned}$$

which is equivalent to the initial optimization problem. But now it is easy to see that $x_1 = x_3 = x_4 = 0$ is optimal, because increasing any of these variables increases the objective.

13.4.4 *An Unbounded Example

Consider the following linear optimization problem:

$$\begin{aligned} \text{minimize} \quad & -x_1 + 2x_2 + x_3 \\ \text{subject to} \quad & -x_1 + x_2 \leq 5, \\ & 2x_2 - x_3 \leq 2, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Adding slack variables x_4 and x_5 gives the following dictionary representation:

$$\begin{array}{rcrcrcrcr} f & = & 0 & - & x_1 & + & 2x_2 & + & x_3 \\ \hline x_4 & = & 5 & + & x_1 & - & x_2 & & \\ x_5 & = & 2 & & & - & 2x_2 & + & x_3. \end{array}$$

To decrease the objective function, x_1 must be the entering variable, since it is the only independent variable with a negative coefficient in f. The constraint $x_5 \geq 0$ does not bound x_1, and so the only constraint on x_1 is given by $x_4 \geq 0$, which implies that $x_1 \geq -5$. Since there is no upper bound on x_1, it may be arbitrarily large, which means that the feasible set is unbounded. The objective has no minimizer because setting $x_1 = z > 0$ gives the valid configuration $x_1 = z, x_2 = 0, x_3 = 0, x_4 = 5 + z, x_5 = 2$ with $f(\mathbf{x}) = -z$.

13.5 The Simplex Method II

The simplex method works by proceeding from one known feasible vertex to another. So the simplex method needs a feasible vertex from which to start. In this section we discuss how to find such a vertex. We also discuss how to ensure that no vertex is visited more than once, thereby ensuring that the algorithm terminates. Finally, we discuss some aspects of the temporal complexity of the simplex method.

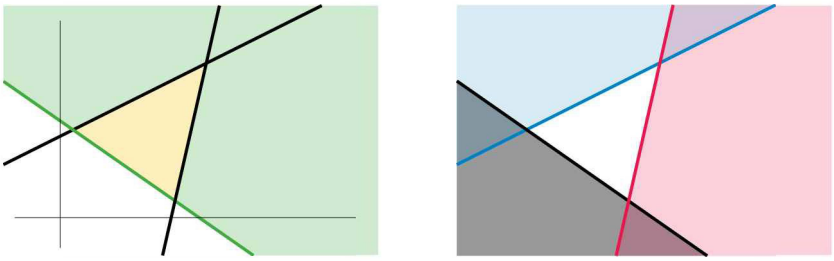


Figure 13.15. *The left panel illustrates a well-posed linear optimization problem (bounded and feasible) where the feasible set (yellow triangle) does not contain the origin. This occurs whenever an entry in* $\mathbf{b}$ *is negative. In this case the negative value of* b_i *corresponds to the green half space. The first step to using the simplex method to solve a problem like this is to use an auxiliary problem to find a feasible point. The right panel illustrates a linear optimization problem that is infeasible because the half spaces have no mutual intersection. There can be no optimizer if there is no feasible point at all.*

13.5.1 Determining Feasibility

The feasible set may actually be empty, and even if it is not empty, the simplex method needs a feasible starting vertex. So far we have always initialized the simplex method by starting at the origin: $x_1 = x_2 = \cdots = x_n = 0$. If this point is feasible, then the feasible set must be nonempty.

Determining whether the origin is feasible is easy. Recall that the constraints take the form $A\mathbf{x} \preceq \mathbf{b}$. When $\mathbf{x} = \mathbf{0}$, the left-hand side is $\mathbf{0}$. Therefore, the origin is feasible whenever $\mathbf{b} \succeq \mathbf{0}$, that is, whenever all the entries of $\mathbf{b}$ are nonnegative. However, if any entry of $\mathbf{b}$ is negative, the origin is not feasible. See Figure 13.15 for an illustration.

When the origin is not feasible, we need to find a feasible starting point or determine that no such point exists. To accomplish this we use an *auxiliary problem* constructed by adding a new variable (increasing the dimension of the original problem), as we now describe.

Definition 13.5.1. *Given a linear optimization problem in standard form* (13.4) *the* auxiliary problem *is the problem*

$$\begin{aligned} &\text{minimize} && x_0 \\ &\textit{subject to} && A\mathbf{x} - x_0\mathbb{1} \preceq \mathbf{b}, \\ &&& x_0 \geq 0, \\ &&& \mathbf{x} \succeq 0. \end{aligned} \tag{13.11}$$

Here $\mathbb{1} = (1, 1, \ldots, 1)$ *denotes the all-ones vector, so if* $\mathbf{a}_i^\mathsf{T}$ *denotes the* i*th row of* A*, then each of the original constraints* $\mathbf{a}_i^\mathsf{T}\mathbf{x} \leq b_i$ *is replaced by* $\mathbf{a}_i^\mathsf{T}\mathbf{x} - x_0 \leq b_i$ *in the auxiliary problem.*

Three important properties of the auxiliary problem can be seen immediately:

(i) Feasible points of the form $(x_0, x_1, \ldots, x_n)$ with $x_0 = 0$ correspond to points $(x_1, \ldots, x_n)$ of the original feasible set, and

(ii) The point $(b, 0, 0, \ldots, 0)$ is feasible for the auxiliary problem, where $b = -\min_i b_i$ (note that this is positive, since at least one of the b_i is negative).

(iii) The auxiliary objective x_0 is bounded below by 0.

The first fact means that the original problem has a feasible point if and only if the auxiliary problem has a feasible point with $x_0 = 0$. The second fact gives an explicit feasible starting vertex for the auxiliary problem. The last fact implies that the problem is not unbounded and so the simplex method applied to the auxiliary problem will terminate (assuming vertices are visited at most once). We summarize this in the following proposition

Proposition 13.5.2. *A linear optimization problem in standard form is feasible if and only if either*

(i) *all the entries of* $\mathbf{b}$ *are nonnegative, in which case the origin is feasible for the original problem; or*

(ii) *some entries are negative, but the associated (bounded) auxiliary problem has an optimal value of* 0 *with optimizer* $(0, x_1^+, \ldots, x_n^+)$*, in which case the point* $(x_1^+, \ldots, x_n^+)$ *is a feasible vertex for the original problem.*

Remark 13.5.3. Geometrically, the set $H_i = \{(x_0, x_1, \ldots, x_n) \mid \mathbf{a}_i^\mathsf{T}\mathbf{x} - x_0 \leq b_i\}$ in $\mathbb{R}^{n+1}$ is a half space containing the point $(b_i, 0, 0, \ldots, 0)$. The set $\{(0, \mathbf{x}) \mid \mathbf{a}_i^\mathsf{T}\mathbf{x} \leq b_i\}$ also lies in H, where we have written $(0, x_1, \ldots, x_n)$ as $(0, \mathbf{x})$ for convenience. The feasible set $\widetilde{\mathscr{F}}$ of the auxiliary problem is the intersection of half spaces $H_1 \cap \cdots \cap H_m$ with the half spaces $x_0 \geq 0, \ldots, x_n \geq 0$, which is a polyhedron in $\mathbb{R}^{n+1}$ having the property that a point $(x_0, \mathbf{x}) \in \mathbb{R}^{n+1}$ lies in $\widetilde{\mathscr{F}}$ if and only if the point $\mathbf{x} \in \mathbb{R}^n$ lies in the feasible set $\mathscr{F}$ for the original problem.

13.5.2 Example of an Auxiliary Problem

Consider the problem of finding x_1, x_2 to

$$\begin{aligned} \text{minimize} \quad & -2x_1 - x_2 \\ \text{subject to} \quad & -x_1 + x_2 \leq -1, \\ & -x_1 - 2x_2 \leq -2, \\ & x_2 \leq 1, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Since there are negative values on the right-hand side of the constraints, the origin is not feasible for this problem. The auxiliary problem is

$$\begin{aligned} \text{minimize} \quad & x_0 \\ \text{subject to} \quad & -x_1 + x_2 - x_0 \leq -1, \\ & -x_1 - 2x_2 - x_0 \leq -2, \\ & x_2 - x_0 \leq 1, \\ & x_0, x_1, x_2 \geq 0. \end{aligned}$$

A feasible starting vertex for the auxiliary problem is $x_0 = 2$, with $x_1 = x_2 = 0$.

Introduce slack variables x_3, x_4, x_5 for the constraints to get the relations

$$\begin{aligned} -x_1 + x_2 - x_0 + x_3 &= -1, \\ -x_1 - 2x_2 - x_0 + x_4 &= -2, \\ x_2 - x_0 + x_5 &= 1 \end{aligned}$$

and substitute the values at the vertex into these equations to get $x_3 = 1$, $x_4 = 0$, and $x_5 = 3$. Thus this vertex corresponds to choosing x_1, x_2, x_4 as the independent variables and gives the dictionary

$$\begin{array}{rcrcrcrcr} f & = & 2 & - & x_1 & - & 2x_2 & + & x_4 \\ \hline x_3 & = & 1 & & & - & 3x_2 & + & x_4 \\ x_0 & = & 2 & - & x_1 & - & 2x_2 & + & x_4 \\ x_5 & = & 3 & - & x_1 & - & 3x_2 & + & x_4. \end{array}$$

The coefficients of x_1 and x_2 are both negative, so we may choose x_1 as the entering variable. The binding variable for x_1 is x_0, so we choose that as the leaving variable, which yields the dictionary

$$\begin{array}{rcrcrcrcr} f & = & 0 & & & & & & x_0 \\ \hline x_3 & = & 1 & - & 3x_2 & + & x_4 & & \\ x_1 & = & 2 & - & 2x_2 & + & x_4 & - & x_0 \\ x_5 & = & 1 & - & x_2 & & & + & x_0. \end{array}$$

None of the coefficients of the objective function is negative, so this point is optimal and the optimal value is 0, as desired. This gives $x_1 = 2$, $x_2 = 0$ as a feasible point for the original problem.

Removing x_0 from the dictionary above gives a valid dictionary for the original problem, except that the objective function for the auxiliary problem must be replaced with the objective function $-2x_1 - x_2$ for the original problem. But the objective function must be rewritten in terms of the independent variables, x_2, x_4. This can be done using the second constraint, which gives an expression for x_1 in terms of the independent variables. This gives the dictionary

$$\begin{array}{rcrcrcrcr} f & = & -4 & + & 3x_2 & - & 2x_4 & & \\ \hline x_3 & = & 1 & - & 3x_2 & + & x_4 & & \\ x_1 & = & 2 & - & 2x_2 & + & x_4 & & \\ x_5 & = & 1 & - & x_2. & & & & \end{array}$$

This dictionary is feasible for the original problem and has an objective function value -4. From this point the simplex method continues in the normal fashion until it finds an optimal point or determines that the problem is unbounded.

13.5.3 Cycling and Degeneracy

As proved above, the simplex method terminates and returns a minimizing vertex provided it never visits the same vertex twice. In most cases the objective function

(i) If every entry in $\mathbf{b}$ is nonnegative, then the origin is feasible. Use the basic simplex method (Algorithm 13.1) instead of this one. Otherwise (if at least one entry of $\mathbf{b}$ is negative) the origin is infeasible.

(ii) **Begin phase one:** Set up the auxiliary problem by introducing a variable x_0

  (a) Set the objective function equal to x_0.

  (b) Subtract x_0 from each of the inequality constraints.

  (c) Add a new constraint that $x_0 \geq 0$.

(iii) Introduce slack variables, and set up the initial dictionary.

(iv) To make the dictionary feasible, pivot on the row with the minimal (most negative) entry of $\mathbf{b}$; that is, make $\mathbf{x}_0$ the entering variable and take as leaving variable the slack variable corresponding to the most negative entry of $\mathbf{b}$.

(v) Proceed as usual for the simplex method until either:

  (a) The objective function is zero and the dictionary is feasible. This indicates a feasible dictionary has been found. Go to step (vi).

  (b) An optimal value has been reached with (no negative coefficients in the objective function) but the value of the objective function is positive. This indicates an infeasible dictionary. Stop. There is no solution.

(vi) **Begin phase two:** Delete the column with the variable x_0 from the dictionary. Replace the objective function with the original objective function, replacing all dependent variables with independent variables

(vii) Continue with the basic simplex method (Algorithm 13.1) until an optimizer is found.

Algorithm 13.2. *A summary of the two-phase simplex method. The first phase uses the simplex method to solve an auxiliary problem that gives a feasible point for the original problem. The second phase uses the simplex method, starting at the feasible point found in phase one, to solve the original problem. Coding up this algorithm is an exercise in the computing labs for this volume.*

strictly decreases with each step, so it does not cycle back to a previously visited vertex, but unfortunately, it is possible for the value of the objective function to remain unchanged after a step of the simplex method.

The following dictionary gives an example:

$$\begin{array}{rcrcrcrcr} f & = & 0 & + & \frac{1}{2}x_1 & - & 2x_2 & + & \frac{3}{2}x_3 \\ \hline x_4 & = & 1 & - & \frac{1}{2}x_1 & & & - & \frac{1}{2}x_3 \\ x_5 & = & 0 & + & x_1 & - & x_2 & + & x_3. \end{array}$$

The only negative coefficient is that of x_2, so that must be the entering variable. The constraint induced by $x_5 \geq 0$ is binding and implies $x_2 \leq 0$, so x_2 cannot increase at all without making x_5 negative. Nevertheless, we can still pivot (make a new choice of entering and leaving variables) to get

$$\begin{array}{rcrcrcrcr} f & = & 0 & - & \frac{3}{2}x_1 & - & \frac{1}{2}x_3 & + & 2x_5 \\ \hline x_4 & = & 1 & - & \frac{1}{2}x_1 & - & \frac{1}{2}x_3 & & \\ x_2 & = & 0 & + & x_1 & + & x_3 & - & x_5. \end{array}$$

While the dependent and independent variables have changed, the value of the objective function has not. Moreover, the actual configuration of $x_1 = x_2 = x_3 = 0$, $x_4 = 1$, $x_5 = 0$ is the same for both this and the previous dictionaries, so we have not moved to a different vertex.

This situation is called *degeneracy*. A *degenerate* dictionary is one in which the constant term of one of the constraints is 0. In this case, some dependent variables are equal to zero, in addition to the n independent variables. This corresponds to being on the intersection of more than n hyperplanes. See Figure 13.16 for an illustration.

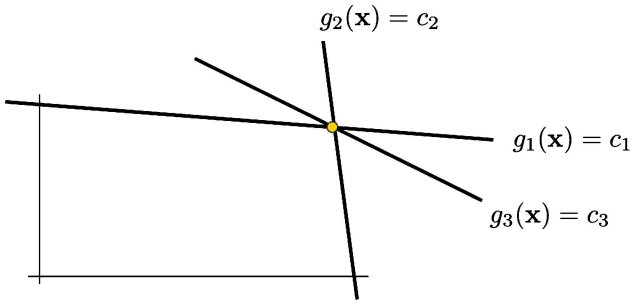


Figure 13.16. *Degeneracy occurs when more than n hyperplanes intersect at a single vertex. Moving from n of these hyperplanes to n others does not result in moving to a different feasible point or decreasing the value of the objective function; it only changes which variables are considered dependent and which are considered independent.*

Degeneracy seems problematic, but making the right choices of entering and leaving variables (pivots) eventually moves to a new vertex. In the previous example, choosing x_1 as the entering variable makes x_4 binding and moves to a new vertex with an improved objective.

One way to escape a degenerate vertex is to apply *Bland's rule.*

Theorem 13.5.4 (Bland's Rule). *If the entering variable is always chosen to be the independent variable with smallest index having a negative coefficient in the objective function and the leaving variable is chosen to be the dependent variable with the smallest index among all dependent variables* that impose a binding constraint on the entering variable*, then the simplex method never cycles.*

The proof of Bland's rule is long and somewhat unenlightening, so we do not include it here, but the interested reader can find a proof in [Van14].

Nota Bene 13.5.5. Bland's rule is just a tie-breaking rule for the usual simplex method. The entering variable must still be one with a negative coefficient in the objective function; see Nota Bene 13.4.5. There is no need to invoke Bland's rule unless there is more than one such variable. Similarly, once the entering variable is chosen, the leaving variable must be chosen from among those dependent variables that impose a binding constraint on that chosen entering variable. Often there is only one of these, and again, Bland's rule needs to be invoked only if there is more than one dependent variable that is binding.

Since the total number of configurations is finite, we have the following corollary.

Corollary 13.5.6. *If the simplex method is implemented using Bland's rule, then it is guaranteed to terminate after a finite number of iterations and produce the correct result for a linear optimization problem in standard form.*

An alternative approach to preventing cycling is the *lexicographic perturbation rule*. This approach chooses a small ε_1, a much smaller ε_2, a still smaller ε_3, and so on, and then shifts b_i to $b_i + \varepsilon_i$. This removes all degeneracy and, if ε_1 is sufficiently small, it does not change the optimal choice of dependent and independent variables.

Neither Bland's rule nor lexicographic perturbation is necessarily the fastest route to the optimizer. Indeed, there are often other choices that result in a much greater decrease in the objective function. Note also that Bland's rule could easily result in very small pivot values (the coefficient of the entering variable), which reduces the stability of the algorithm.

Finally note that cycling is actually a rare phenomenon. It often makes more sense to use another selection rule and worry about cycling only if it actually occurs.

13.5.4 Complexity of Linear Optimization

The worst-case run time of the simplex method is exponential because the number of potential vertices to check is $\binom{m+n}{n}$. In 1972 Klee and Minty constructed a pathological example in n variables and $m = n$ constraints where the feasible region is a slightly tilted n-dimensional hypercube with one vertex at the origin [KM72]. Choosing the starting vertex to be the origin and following standard pivoting rules, the simplex method on this example visits every other vertex of the hypercube before reaching the optimizer. There are 2^n vertices, so the simplex method makes $2^n - 1$ pivots before finding the optimizer. Soon after the Klee–Minty paper was published, several other authors produced similar examples for many other pivoting rules, including Bland's rule. So in the worst case, the simplex method can perform very poorly.

However, these worst cases are very unusual. It can be shown that on average very few of the vertices need be to examined, and the average-case performance is

generally very good, typically taking $2m$ to $3m$ iterations (where m is the number of constraints) and converging in expected polynomial time for certain distributions of random inputs.

Remark 13.5.7. The dual problem, described in the next section, interchanges the number of constraints and the number of variables, so the dual problem is typically solved in $2n$ to $3n$ iterations. If the number m of constraints is much larger than the dimension n, then switching to the dual problem can significantly reduce the complexity of solving the problem with the simplex method.

Remark 13.5.8. In Section 15.6 we describe a very different class of methods for solving linear and other optimization problems. These solve linear optimization problems in polynomial time, namely $O(\sqrt{n}L \log L)$, where L is the number of bits of input to the problem. It can be shown that $L \in O(mn + \log_2(|P|))$, where P is the product of all nonzero coefficients of A, $\mathbf{b}$, and $\mathbf{c}$. Despite the fact that these methods are polynomial and the simplex method is exponential in the worst case, the simplex method in practice usually takes only $O(m)$ or $O(n)$ iterations to find the optimizer.

13.6 Duality

In this section we discuss *duality* for linear optimization problems. The weak duality theorem guarantees that for every linear minimization problem there is another, *dual* linear maximization problem with the property that every feasible value of the dual objective is bounded above by every feasible value of the original (*primal*) objective. The strong duality theorem guarantees that the minimal value of the primal problem is equal to the maximal value of the dual problem. Moreover, the optimizers for the primal and dual problems are closely related. Indeed knowledge about one of these usually gives useful knowledge about the other. One of the most important connections between the primal and dual problems is the property of *complementary slackness*, which is discussed in Section 13.6.3.

In some cases the dual problem is easier to solve than its corresponding primal problem and in other cases the primal problem is easier to solve. In either case, once one of the optimizers is known, the other can be computed quickly. As shown in Chapter 15, some of the best methods for solving large linear optimization problems rely heavily on the interplay between the primal and dual problems.

13.6.1 Weak Duality

Every linear minimization problem has a natural *dual* problem that is formulated as a linear maximization problem.

Definition 13.6.1. *Let $A \in M_{m\times n}(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. The linear optimization problem*

$$\begin{aligned} &\text{minimize} && \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ &\textit{subject to} && A\mathbf{x} \preceq \mathbf{b}, \\ & && \mathbf{x} \succeq \mathbf{0} \end{aligned} \tag{13.12}$$

has the following dual problem*:*

$$\begin{aligned} \text{maximize} \quad & -\mathbf{b}^\mathsf{T}\mathbf{y} \\ \textit{subject to} \quad & -A^\mathsf{T}\mathbf{y} \preceq \mathbf{c}, \\ & \mathbf{y} \succeq \mathbf{0}. \end{aligned} \tag{13.13}$$

We call the original problem (13.12) *the* primal problem *to distinguish it from the dual problem* (13.13)*.*

Remark 13.6.2. Following Example 13.3.3, it is sometimes helpful to write the dual as a minimization problem in standard form

$$\begin{aligned} \text{minimize} \quad & \mathbf{b}^\mathsf{T}\mathbf{y} \\ \text{subject to} \quad & -A^\mathsf{T}\mathbf{y} \preceq \mathbf{c}, \\ & \mathbf{y} \succeq \mathbf{0}. \end{aligned} \tag{13.14}$$

The next proposition shows that, as one might expect from the name *dual*, the dual of the dual is the primal.

Proposition 13.6.3. *For a linear optimization problem in standard form, the dual of the dual optimization problem is again the primal problem. Stated more precisely, if the dual problem is recast as the minimization problem* (13.14)*, then its dual is a maximization problem that, when recast as a minimization problem, is the original problem* (13.12)*.*

Proof. The proof is Exercise 13.33. □

Nota Bene 13.6.4. Many textbooks develop linear optimization theory with maximization problems being their "standard form." In these cases, the primal problem is the maximization problem and the corresponding dual problem is a minimization problem.

The weak duality theorem guarantees that every feasible value of the primal objective is an upper bound for every feasible value of the dual objective.

Theorem 13.6.5 (Weak Duality Theorem). *If* $\mathbf{x} \in \mathbb{R}^n$ *is feasible for the primal problem* (13.12) *and* $\mathbf{y} \in \mathbb{R}^m$ *is feasible for the dual problem* (13.13)*, then the objective for the dual problem is bounded above by the objective for the primal problem, that is, the following inequality holds:*

$$-\mathbf{b}^\mathsf{T}\mathbf{y} \leq \mathbf{c}^\mathsf{T}\mathbf{x}. \tag{13.15}$$

Proof. It is straightforward to see that

$$-\mathbf{b}^\mathsf{T}\mathbf{y} = -\mathbf{y}^\mathsf{T}\mathbf{b} \preceq -\mathbf{y}^\mathsf{T}A\mathbf{x} = -\mathbf{x}^\mathsf{T}A^\mathsf{T}\mathbf{y} \preceq \mathbf{x}^\mathsf{T}\mathbf{c} = \mathbf{c}^\mathsf{T}\mathbf{x}. \quad \square$$

Corollary 13.6.6. *If the dual problem is unbounded, then the primal problem is infeasible. Similarly, if the primal problem is unbounded, then the dual problem is infeasible.*

Proof. If the primal problem is feasible and the dual problem is unbounded, then weak duality fails, since $\mathbf{c}^\mathsf{T}\mathbf{x}$ is an upper bound for all feasible values of $-\mathbf{b}^\mathsf{T}\mathbf{y}$. Similarly, if the dual problem is feasible and the primal problem is unbounded, then weak duality fails, since $-\mathbf{b}^\mathsf{T}\mathbf{y}$ is a lower bound for all values of $\mathbf{c}^\mathsf{T}\mathbf{x}$. □

The converse of the corollary is not true. When the primal is infeasible, we do not know that the dual is necessarily unbounded. Both problems could be infeasible.

Example 13.6.7. The linear optimization problem

$$\begin{aligned} \text{minimize} \quad & 2x_2 - x_1 \\ \text{subject to} \quad & -x_1 + x_2 \leq 2, \\ & x_2 \leq -1, \\ & x_1, x_2 \geq 0 \end{aligned}$$

is infeasible because the constraints require that $x_2 \geq 0$ and $x_2 \leq -1$. The dual problem

$$\begin{aligned} \text{maximize} \quad & -2y_1 + y_2 \\ \text{subject to} \quad & y_1 \leq -1, \\ & -y_1 - y_2 \leq 2, \\ & y_1, y_2 \geq 0 \end{aligned}$$

is also infeasible, since the constraints require that $y_1 \geq 0$ and $y_1 \leq -1$.

13.6.2 Strong Duality

The weak duality theorem only guarantees that feasible points of the dual problem provide lower bounds for the primal problem. It does not indicate how tight that bound is. The strong duality theorem guarantees that the bound is sharp and that the minimal value of the primal is equal to the maximal value of the dual.

Theorem 13.6.8 (Strong Duality). *If the primal problem* (13.12) *has a minimizer* $\mathbf{x}^* \in \mathbb{R}^n$ *and the dual problem* (13.13) *has a maximizer* $\mathbf{y}^* \in \mathbb{R}^m$*, then the following equality holds:*

$$-\mathbf{b}^\mathsf{T}\mathbf{y}^* = \mathbf{c}^\mathsf{T}\mathbf{x}^*. \tag{13.16}$$

Remark 13.6.9. The proof of strong duality is given in Section 13.6.5. Strong duality gives us a *certificate of optimality*, meaning that if a primal feasible solution $\mathbf{x}^*$ and a dual feasible solution $\mathbf{y}^*$ satisfy (13.16), then they are guaranteed to be, respectively, a primal optimizer and a dual optimalizer.

13.6.3 Complementary Slackness

Strong duality also implies a very useful relation called *complementary slackness* between the optimizer $\mathbf{x}^*$ of the primal problem and the optimizer $\mathbf{y}^*$ of the dual problem.

Theorem 13.6.10 (Complementary Slackness). *Suppose that* $\mathbf{x} = (x_1, \ldots, x_n)$ *is a feasible point of the primal problem* (13.12) *and* $\mathbf{y} = (y_1, \ldots, y_m)$ *is a feasible point of the dual problem* (13.13). *Let* $\mathbf{w} = (w_1, \ldots, w_m) \succeq \mathbf{0}$ *be the slack variable for the primal problem so that* $A\mathbf{x} + \mathbf{w} = \mathbf{b}$, *and let* $\mathbf{z} = (z_1, \ldots, z_n) \succeq \mathbf{0}$ *be the slack variable for the dual problem so that* $-A^\mathsf{T}\mathbf{y} + \mathbf{z} = \mathbf{c}$. *The points* $\mathbf{x}$ *and* $\mathbf{y}$ *are both optimal for their respective problems if and only if*

$$x_i z_i = 0 \quad \forall i \in \{1, \ldots, n\} \qquad \textit{and} \qquad y_j w_j = 0 \quad \forall j \in \{1, \ldots, m\}. \tag{13.17}$$

Proof. Assume $\mathbf{x}$ and $\mathbf{y}$ are, respectively, optimizers for the primal and dual problems. As in the proof of Theorem 13.6.5, strong duality implies

$$-\mathbf{y}^\mathsf{T}\mathbf{b} = -\mathbf{y}^\mathsf{T}A\mathbf{x} = \mathbf{c}^\mathsf{T}\mathbf{x}. \tag{13.18}$$

This gives

$$\mathbf{0} = \mathbf{y}^\mathsf{T}(\mathbf{b} - A\mathbf{x}) = \mathbf{y}^\mathsf{T}\mathbf{w} \qquad \text{and} \qquad \mathbf{0} = (\mathbf{c} + A^\mathsf{T}\mathbf{y})^\mathsf{T}\mathbf{x} = \mathbf{z}^\mathsf{T}\mathbf{x}. \tag{13.19}$$

The right-hand sides of (13.19) are sums of nonnegative products, so each term $y_i w_i$ and $z_j x_j$ must be zero.

Conversely, if (13.17) holds, then so does (13.19), which implies (13.18). Since equality of the primal and dual objective functions can occur only at optimal points, $\mathbf{x}$ and $\mathbf{y}$ must be optimal. □

If the optimizer of either the primal or the dual problem is known (but not both), then complementary slackness gives some linear relationships that help solve the other problem.

Example 13.6.11. Suppose it is known that $\mathbf{x}^* = (11, 1/2)$ is an optimizer for the primal problem

$$\begin{aligned} \text{minimize} \quad & 24x_1 + 60x_2 \\ \text{subject to} \quad & -\tfrac{1}{2}x_1 - x_2 \leq -6, \\ & -2x_1 - 2x_2 \leq -14, \\ & -x_1 - 4x_2 \leq -13, \\ & x_1, x_2 \geq 0. \end{aligned}$$

The corresponding dual problem is

$$\begin{aligned} \text{maximize} \quad & 6y_1 + 14y_2 + 13y_3 \\ \text{subject to} \quad & \tfrac{1}{2}y_1 + 2y_2 + y_3 \leq 24, \\ & y_1 + 2y_2 + 4y_3 \leq 60, \\ & y_1, y_2, y_3 \geq 0. \end{aligned}$$

Computing $A\mathbf{x} + \mathbf{w} = \mathbf{b}$, it is easy to see that $w_1 = w_3 = 0$ and $w_2 = 9$. Complementary slackness implies that $y_1 \geq 0$, $y_2 = 0$, and $y_3 \geq 0$ and that $z_1 = z_2 = 0$. Computing $-A^\mathsf{T}\mathbf{y} + \mathbf{z} = \mathbf{c}$ we have

$$\begin{bmatrix} \frac{1}{2} & 2 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} y_1 \\ 0 \\ y_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 24 \\ 60 \end{bmatrix},$$

which simplifies to

$$\begin{bmatrix} \frac{1}{2} & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 60 \end{bmatrix}.$$

This can be solved to give $\mathbf{y}^* = (36, 6)$.

To check that this is an optimizer, we verify strong duality (13.16) and see that $-\mathbf{b}^\mathsf{T}\mathbf{y}^* = \mathbf{c}^\mathsf{T}\mathbf{x}^* = 294$.

13.6.4 An Economic Interpretation of Duality

Recall the production-schedule problem described in Section 13.3.3. In this situation, the factory scheduler wants to maximize total profit $\sum_{j=1}^{n} p_j x_j$, subject to the constraints that it requires a_{ij} units of material i to make one unit of product j, and there are at most b_i units of material i available. If we set $\mathbf{c} = -\mathbf{p}$, then the problem (13.7) can be expressed in standard primal form (13.12).

For the dual problem, assume an investor is interested in buying all the resources that the factory has available. If she pays y_i dollars for resource i, then the total amount she will be paying is $\sum_{i=1}^{m} b_i y_i = \mathbf{b}^\mathsf{T}\mathbf{y}$ dollars. She would obviously prefer to minimize this amount, but if she offers too little, then the factory won't be willing to sell, because it can make more money by using the resources to make products. Specifically, for every $j \in \{1, \ldots, n\}$ the value $\sum_{i=1}^{m} a_{ij} y_i$ that the investor offers to pay for the materials to make one product of type j must be at least as much as the value p_j of that widget; that is, the factory will not accept the deal unless $A^\mathsf{T}\mathbf{y} \succeq -\mathbf{c}$. Also, $\mathbf{y} \succeq \mathbf{0}$ because the factory won't pay the investor to take the resources. So the investor's problem is to choose nonnegative prices y_i to minimize $\mathbf{b}^\mathsf{T}\mathbf{y}$ (or maximize $-\mathbf{b}^\mathsf{T}\mathbf{y}$) subject to $A^\mathsf{T}\mathbf{y} \succeq -\mathbf{c}$ (or $-A^\mathsf{T}\mathbf{y} \preceq \mathbf{c}$). This is precisely the dual problem (13.13).

Weak duality $-\mathbf{b}^\mathsf{T}\mathbf{y} \leq \mathbf{c}^\mathsf{T}\mathbf{x}$ is equivalent to saying $\mathbf{p}^\mathsf{T}\mathbf{x} \leq \mathbf{b}^\mathsf{T}\mathbf{y}$ for all feasible $\mathbf{x}$ and $\mathbf{y}$; that is, the total value of any feasible offer the investor makes will be no less than the factory's profit making products, no matter what combination of products it makes. Strong duality guarantees that the investor's optimal (cost-minimizing) offer will cost her exactly the same amount of money that the factory could earn by making products in the optimal (profit-maximizing) way.

The investor's optimal prices $\mathbf{y}^*$ are often called *shadow prices*. At the shadow price, the factory should be willing to buy or sell (a small number of) its resources, because selling would bring in the same amount of profit as making more products would. Note, however, that selling or buying a large number of resources could change the optimal values of $\mathbf{x}^*$ and $\mathbf{y}^*$ and thus change the shadow prices.

Complementary slackness also has an interpretation in this economic setting. If an optimal primal slack variable w_i^* is positive, then not all of resource i is used in the optimizer. Thus, the factory has a surplus of that resource that it cannot use, and this extra resource is of no value to the factory. Therefore, the shadow price of resource i should be 0. This shows that $w_i^* y_i^* = 0$ for all $i \in \{1, \ldots, m\}$.

Similarly, if the optimal dual slack variable z_j^* is positive, that means the investor is paying more for the resources needed to make one widget of type j than the profit the factory can make by producing a widget of type j. Thus, the factory would make more money by making fewer widgets of type j. But since x_j^* is optimal, this means x_j^* must be zero. This shows that $z_j^* x_j^* = 0$ for all $j \in \{1, \ldots, n\}$.

13.6.5 *Proof of Strong Duality

We conclude the section by proving Theorem 13.6.8, which guarantees that whenever the primal problem has an optimizer $\mathbf{x}^* \in \mathbb{R}^n$, then the dual also has an optimizer $\mathbf{y}^* \in \mathbb{R}^m$ such that $-\mathbf{b}^\mathsf{T}\mathbf{y}^* = \mathbf{c}^\mathsf{T}\mathbf{x}^*$. The first step in the proof is Farkas' lemma.

Lemma 13.6.12 (Farkas' Lemma). *Given $A \in M_{m\times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$, denote $P = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \succeq \mathbf{0}, A\mathbf{x} \preceq \mathbf{b}\}$ and $D = \{\mathbf{y} \in \mathbb{R}^m \mid \mathbf{y} \succeq \mathbf{0}, A^\mathsf{T}\mathbf{y} \succeq \mathbf{0}, \mathbf{b}^\mathsf{T}\mathbf{y} < 0\}$. The set P is not empty if and only if the set D is empty.*

Proof. If $P \neq \emptyset$, then there exists $\mathbf{x} \succeq \mathbf{0}$ with $A\mathbf{x} \preceq \mathbf{b}$. For any $\mathbf{y} \succeq \mathbf{0}$ satisfying $A^\mathsf{T}\mathbf{y} \succeq \mathbf{0}$, we have $0 \leq \mathbf{x}^\mathsf{T}A^\mathsf{T}\mathbf{y} = \mathbf{y}^\mathsf{T}A\mathbf{x} \leq \mathbf{y}^\mathsf{T}\mathbf{b}$. Thus, $D = \emptyset$.

If $P = \emptyset$, let $\widetilde{A} = \begin{bmatrix} A & I_m \end{bmatrix}$ be the matrix constructed by adjoining A and the $m \times m$ identity matrix. The set

$$K = \{\widetilde{A}\mathbf{w} \mid \mathbf{w} \in \mathbb{R}^{n+m} \text{ with } \mathbf{w} \succeq \mathbf{0}\}$$

is convex and closed. Suppose $\mathbf{b} \in K$, meaning $\mathbf{b} = \widetilde{A}\mathbf{w}$ for some $\mathbf{w} \succeq \mathbf{0}$. Write

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} \quad \text{with } \mathbf{w}_1 \in \mathbb{R}^n \text{ and } \mathbf{w}_2 \in \mathbb{R}^m.$$

This implies $A\mathbf{w}_1 \preceq A\mathbf{w}_1 + \mathbf{w}_2 = \mathbf{b}$, so $\mathbf{w}_1 \in P$, which is a contradiction. Therefore $\mathbf{b} \notin K$.

Since K and $\{\mathbf{b}\}$ are disjoint, then by the separation lemma (Lemma 13.2.7), there is a hyperplane that separates K from $\mathbf{b}$; that is, there is a vector $\mathbf{y}$ and a real number d such that $\mathbf{y}^\mathsf{T}\mathbf{x} > d$ for all $\mathbf{x} \in K$ and $\mathbf{y}^\mathsf{T}\mathbf{b} < d$. In particular, since $\mathbf{0} \in K$, we have $0 = \mathbf{y}^\mathsf{T}\mathbf{0} > d$.

Finally, suppose that $\mathbf{y}^\mathsf{T}\widetilde{A}$ has a negative value in the ith coordinate. Hence, $\alpha\mathbf{y}^\mathsf{T}\widetilde{A}\mathbf{e}_i < d$ for some $\alpha > 0$. This implies that $\mathbf{y}^\mathsf{T}\widetilde{A}(\alpha\mathbf{e}_i) < d$, but this is a contradiction since $\widetilde{A}(\alpha\mathbf{e}_i) \in K$. Thus, $\mathbf{y}^\mathsf{T}\widetilde{A}$ has only nonnegative entries. Taking the transpose gives $A^\mathsf{T}\mathbf{y} \succeq \mathbf{0}$ and $\mathbf{y} \succeq \mathbf{0}$, which implies that $\mathbf{y} \in D$. □

We can now prove the strong duality theorem.

Proof. Assume that $\mathbf{x}^*$ is a minimizer for the primal problem (13.12) so that $\mathbf{c}^\mathsf{T}\mathbf{x}^*$ is the minimal value. Therefore, for any $\varepsilon > 0$ the system of inequalities $A\mathbf{x} \preceq \mathbf{b}$,

$\mathbf{c}^\mathsf{T}\mathbf{x} \leq \mathbf{c}^\mathsf{T}\mathbf{x}^* - \varepsilon$ has no solution with $\mathbf{x} \succeq \mathbf{0}$. Writing this in matrix notation gives the system

$$\begin{bmatrix} A \\ \mathbf{c}^\mathsf{T} \end{bmatrix} \mathbf{x} \preceq \begin{bmatrix} \mathbf{b} \\ \mathbf{c}^\mathsf{T}\mathbf{x}^* - \varepsilon \end{bmatrix} \quad \text{with} \quad \mathbf{x} \succeq \mathbf{0},$$

which has no solution. By Farkas' lemma, there must exist $\mathbf{y} \in \mathbb{R}^m$ with $\mathbf{y} \succeq \mathbf{0}$ and $\xi \geq 0$ satisfying

$$\begin{bmatrix} A^\mathsf{T} & \mathbf{c} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \xi \end{bmatrix} \succeq \mathbf{0} \qquad \text{and} \qquad \begin{bmatrix} \mathbf{b}^\mathsf{T} & \mathbf{c}^\mathsf{T}\mathbf{x}^* - \varepsilon \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \xi \end{bmatrix} < 0.$$

Thus we have

$$A^\mathsf{T}\mathbf{y} + \xi\mathbf{c} \succeq \mathbf{0} \qquad \text{and} \qquad \mathbf{b}^\mathsf{T}\mathbf{y} + \xi(\mathbf{c}^\mathsf{T}\mathbf{x}^* - \varepsilon) < 0.$$

If $\xi = 0$, then we have $A^\mathsf{T}\mathbf{y} \succeq \mathbf{0}$ and $\mathbf{b}^\mathsf{T}\mathbf{y} < 0$, so by Farkas' lemma, there is no point $\mathbf{x} \succeq \mathbf{0}$ satisfying $A\mathbf{x} \preceq \mathbf{b}$; but $\mathbf{x}^*$ is such a point, so $\xi > 0$.

Rescaling $\mathbf{y}$ by $1/\xi$, we may assume that $\xi = 1$. Simplifying gives

$$-A^\mathsf{T}\mathbf{y} \preceq \mathbf{c} \qquad \text{and} \qquad -\mathbf{b}^\mathsf{T}\mathbf{y} > \mathbf{c}^\mathsf{T}\mathbf{x}^* - \varepsilon.$$

This implies that $\mathbf{y}$ is a feasible point for the dual problem (13.13). Since there is a feasible point to the dual problem and all values of the objective function are bounded above, there is a maximizer $\mathbf{y}^*$ satisfying $\mathbf{c}^\mathsf{T}\mathbf{x}^* - \varepsilon < -\mathbf{b}^\mathsf{T}\mathbf{y}^* \leq \mathbf{c}^\mathsf{T}\mathbf{x}^*$. Since this holds for all $\varepsilon > 0$, it follows that (13.16) holds. □

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

13.1. Prove that each of the following sets is convex:

(i) A half space.

(ii) An affine set.

(iii) The positive orthant $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \succeq \mathbf{0}\}$.

(iv) The set $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}^\mathsf{T} A\mathbf{x} \leq c\}$ for any $c \geq 0$, where $A \in M_n(\mathbb{R})$ and $A > 0$. Hint: Since $A > 0$, the function $\mathbf{x} \mapsto \sqrt{\mathbf{x}^\mathsf{T} A\mathbf{x}}$ defines a norm on $\mathbb{R}^n$. Use the triangle inequality applied to $\lambda\mathbf{x} + (1-\lambda)\mathbf{y}$.

(v) The set $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}^\mathsf{T} A\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c \leq 0\}$ where $A \in M_n(\mathbb{R})$, $A > 0$, $\mathbf{b} \in \mathbb{R}^n$, and $c \in \mathbb{R}$.

13.2. Prove Proposition 13.1.5.

13.3. Prove Proposition 13.1.14.

13.4. Provide an example showing that the union of two convex sets need not be convex. Given a nondecreasing sequence $K_0 \subset K_1 \subset \cdots$ of convex sets, prove that the union $\bigcup_i K_i$ is convex.

13.5. Define the *convex cone* $\operatorname{cone}(S)$ of a set S to be the set of all nonnegative linear combinations $\sum_{i=1}^k a_i \mathbf{s}_i$ of elements of S:

$$\operatorname{cone}(S) = \left\{ \sum_{i=1}^k a_i \mathbf{s}_i \mid a_i \geq 0, \mathbf{s}_i \in S \right\}.$$

Prove that the convex cone of any set S is convex.

13.6. Let $\mathrm{PSD}_n(\mathbb{R}) \subset M_n(\mathbb{R})$ be the set of all positive semidefinite (and hence symmetric) $n \times n$ matrices over $\mathbb{R}$. Prove that $\mathrm{PSD}_n(\mathbb{R})$ is a convex cone, meaning that $\operatorname{cone}(\mathrm{PSD}_n(\mathbb{R})) = \mathrm{PSD}_n(\mathbb{R})$, and hence $\mathrm{PSD}_n(\mathbb{R})$ is convex.

13.7. Two hyperplanes in $\mathbb{R}^n$ are *parallel* if they never intersect.

(i) What is a necessary and sufficient algebraic condition for two hyperplanes to be parallel? (Warning: The inner product representation of the hyperplane is not necessarily unique.)

(ii) Find a formula for the distance (assume the Euclidean metric $\|\cdot\|_2$) between two parallel hyperplanes.

(iii) Describe necessary and sufficient algebraic conditions for one half space to contain another.

13.8.* Let $S_1, S_2 \subset V$ be subsets of a vector space V. The *Minkowski sum* $S_1 + S_2$ of S_1 and S_2 is the set

$$S_1 + S_2 = \{\mathbf{s}_1 + \mathbf{s}_2 \mid \mathbf{s}_i \in S_i\}.$$

Prove the following:

(i) The Minkowski sum of convex sets is convex.

(ii) The Minkowski sum of affine sets is affine.

(iii) $\operatorname{conv}(S_1) + \operatorname{conv}(S_2) = \operatorname{conv}(S_1 + S_2)$.

(iv) $\operatorname{cone}(S_1) + \operatorname{cone}(S_2) = \operatorname{cone}(S_1 + S_2)$.

13.9.* Prove Proposition 13.1.10.

13.10.* Prove Proposition 13.1.15.

13.11.* Prove that if S is compact, then $\operatorname{conv}(S)$ is compact.

13.12. Prove the separation lemma (Lemma 13.2.7).

13.13. Prove the convex projection theorem (Theorem 13.2.5). To do this, first prove the statements below, and then write a complete proof of the theorem.

(i) $\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}-\mathbf{p}\|^2 + \|\mathbf{p}-\mathbf{y}\|^2 + 2\langle \mathbf{x}-\mathbf{p}, \mathbf{p}-\mathbf{y}\rangle$.

(ii) If (13.2) holds, then $\|\mathbf{x}-\mathbf{y}\| \geq \|\mathbf{x}-\mathbf{p}\|$ for all $\mathbf{y} \in C$, $\mathbf{y} \neq \mathbf{p}$. Hint: Use the identity in (i).

(iii) If $\mathbf{z} = \lambda\mathbf{y} + (1-\lambda)\mathbf{p}$, where $0 \leq \lambda \leq 1$, then

$$\|\mathbf{x}-\mathbf{z}\|^2 = \|\mathbf{x}-\mathbf{p}\|^2 + 2\lambda\langle \mathbf{x}-\mathbf{p}, \mathbf{p}-\mathbf{y}\rangle + \lambda^2\|\mathbf{y}-\mathbf{p}\|^2. \tag{13.20}$$

(iv) If $\mathbf{p}$ is a projection of $\mathbf{x}$ onto the convex set C, then $\langle \mathbf{x}-\mathbf{p}, \mathbf{p}-\mathbf{y}\rangle \geq 0$ for all $\mathbf{y} \in C$. Hint: Use (13.20) to show that

$$0 \leq 2\langle \mathbf{x}-\mathbf{p}, \mathbf{p}-\mathbf{y}\rangle + \lambda\|\mathbf{y}-\mathbf{p}\|^2 \quad \forall \mathbf{y} \in C, \lambda \in [0,1].$$

13.14. Give an example of convex sets C and D that are disjoint and both closed (but not compact) that have no strictly separating hyperplane.

13.15. Let $C, D \subset \mathbb{R}^n$ be disjoint closed convex sets (not necessarily compact). Prove that if the set $E = C - D = \{\mathbf{c}-\mathbf{d} \mid \mathbf{c} \in C, \mathbf{d} \in D\}$ is closed, then there exists a strictly separating hyperplane, that is, there exist $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $\langle \mathbf{a}, \mathbf{c}\rangle \leq b$ for every $\mathbf{c} \in C$ and $\langle \mathbf{a}, \mathbf{d}\rangle > b$ for every $\mathbf{d} \in D$.

(i) Prove that $C - D$ is convex and $\mathbf{0} \notin E$.

(ii) Assume that E is closed. Prove that there exists a point $\mathbf{z} \in E$ such that if $\mathbf{a} = -\mathbf{z}$ and $b' = -\mathbf{a}^\mathsf{T}\mathbf{a}$, then the half space $H' = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x}\rangle \leq b'\}$ supports E and does not contain $\mathbf{0}$.

(iii) Prove that $\sup_{\mathbf{c}\in C} \mathbf{a}^\mathsf{T}\mathbf{c} + \|\mathbf{a}\|^2 \leq \inf_{\mathbf{d}\in D} \langle \mathbf{a}, \mathbf{d}\rangle$.

(iv) Let $b = \sup_{\mathbf{c}\in C} \mathbf{a}^\mathsf{T}\mathbf{c} + \|\mathbf{a}\|^2$, and let $H = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x}\rangle \leq b\}$. Prove that $C \subset H$ and $D \cap H = \emptyset$.

13.16. Prove that if $C, D \subset \mathbb{R}^n$ are disjoint closed convex sets (but $C - D$ is not necessarily closed), then there exists a separating hyperplane, but it need not be strict. Hint: Take an increasing sequence C_n of compact subsets of C producing sequences $(\mathbf{a}_n)_{n\in\mathbb{N}}$. Normalize the vectors so that $\mathbf{v}_n = \mathbf{a}_n/\|\mathbf{a}_n\|$. Show that $\mathbf{v}_n$ converges and use the limit to define the desired half space.

13.17. Consider the linear optimization problem

$$\begin{aligned} \text{minimize} \quad & -5x + 4y \\ \text{subject to} \quad & 2x - 3y \leq 4, \\ & x - 6y \leq 1, \\ & x + y \leq 6, \\ & x, y \geq 0. \end{aligned}$$

Draw (or plot) the feasible set. Graph the objective function over the feasible set. Find an optimizer for this problem.

13.18. Draw the feasible polygon of the following linear optimization problem. Identify all the vertices, and use the fundamental theorem to solve the linear optimization problem (that is, check all the vertices). Give both the optimizer and the optimal value of the objective function.

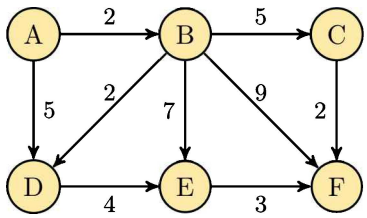
$$\begin{aligned} \text{minimize} \quad & -3x_1 - x_2 \\ \text{subject to} \quad & x_1 + 3x_2 \leq 15, \\ & 2x_1 + 3x_2 \leq 18, \\ & x_1 - x_2 \leq 4, \\ & x_1, x_2 \geq 0. \end{aligned}$$

13.19. Draw the feasible polygon of the following linear optimization problem. Identify all the vertices, and use the fundamental theorem to solve the linear optimization problem (that is, check all the vertices). Give both the optimizer and the optimal value of the objective function.

$$\begin{aligned} \text{minimize} \quad & -4x - 6y \\ \text{subject to} \quad & -x + y \leq 11, \\ & x + y \leq 27, \\ & 2x + 5y \leq 90, \\ & x, y \geq 0. \end{aligned}$$

13.20. Kenny's Toy Co. manufactures two types of toys: a GI Barb soldier and a Joey doll. A GI Barb soldier sells for \$12 and uses \$5 worth of raw materials. Each solider that is manufactured increases Kenny's general overhead costs by \$3. A Joey doll sells for \$10 and uses \$3 worth of raw materials. Each Joey doll built increases Kenny's overhead costs by \$4. The manufacture of soldiers and dolls requires two types of labor: molding and finishing. A soldier requires 15 minutes of finishing labor and 2 minutes of molding labor. A Joey doll requires 10 minutes of finishing and 2 minutes of molding labor. Each week, Kenny can obtain all the needed raw material but only 30 finishing hours and 5 molding hours of labor. Demand for GI Barbs is unlimited but at most 200 Joey dolls are bought each week. Formulate a linear optimization problem in standard form whose solution would maximize Kenny's profit on these two toys.

13.21. Consider the following network, where the weights of each edge represent the carrying cost per unit of that edge.



Assume that the supply (or demand, depending on the sign) at the nodes is $b_A = 10, b_B = 1, b_C = -2, b_D = -3, b_E = 4, b_F = -10$ and that the capacity of each edge is bounded by 6. Write a linear optimization problem in standard form whose solution gives the optimal (cheapest) flow in this network with these constraints.

13.22. For the linear optimization problem in Exercise 13.17, solve the linear problem using the simplex method. Show the dictionary after each pivot. Give both the optimizer and the optimal value of the objective function. Verify that your answers agree with those of Exercise 13.17.

13.23. For the linear optimization problem in Exercise 13.18, solve the linear problem using the simplex method. Show the dictionary after each pivot. Give both the optimizer and the optimal value of the objective function. Verify that your answers agree with those of Exercise 13.18.

13.24. For the linear optimization problem in Exercise 13.19, solve the linear problem using the simplex method. Show the dictionary after each pivot. Give both the optimizer and the optimal value of the objective function. Verify that your answers agree with those of Exercise 13.19.

13.25. Solve the Kenny's Toys linear problem of Exercise 13.20 using the simplex method. Show the dictionary after each pivot. Give both the optimal choice of how much of each toy to manufacture and the maximal profit.

13.26. Give an example of a three-dimensional linear minimization problem where the feasible set is closed and unbounded (hence nonempty) and

(i) the objective function has no minimizer;

(ii) the objective function has a unique feasible minimizer.

13.27. Solve the following linear problems using the simplex method. Show the dictionary after each pivot. If there is a solution, give a minimizer and the minimal value of the objective function. If there isn't a solution, tell whether the problem is unbounded or infeasible.

(i)

$$\begin{aligned} \text{minimize} \quad & x_1 + 3x_2 + x_3 \\ \text{subject to} \quad & -x_1 - x_2 - x_3 \leq -2, \\ & 2x_1 - x_2 + x_3 \leq 1, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

(ii)

$$\begin{aligned} \text{minimize} \quad & -5x_1 - 2x_2 \\ \text{subject to} \quad & 5x_1 + 3x_2 \leq 15, \\ & 3x_1 + 5x_2 \leq 15, \\ & 4x_1 - 3x_2 \leq -12, \\ & x_1, x_2 \geq 0. \end{aligned}$$

(iii)

$$\begin{aligned} \text{minimize} \quad & 3x_1 - x_2 \\ \text{subject to} \quad & x_2 \leq 4, \\ & -2x_1 + 3x_2 \leq 6, \\ & x_1, x_2 \geq 0. \end{aligned}$$

13.28. Give an example of a three-dimensional linear optimization problem where

(i) the feasible set is empty;

(ii) the feasible set is closed, bounded, and not empty, but $\mathbf{0}$ is not feasible. Write an auxiliary problem whose solution gives a feasible vertex for starting the original problem.

13.29. Solve the following linear problem using Bland's rule to resolve degeneracy. Show the dictionary after each pivot. Give the minimizer and the minimum value of the objective function.

$$\begin{aligned} \text{minimize} \quad & -10x_1 + 57x_2 + 9x_3 + 24x_4 \\ \text{subject to} \quad & 0.5x_1 - 1.5x_2 - 0.5x_3 + x_4 \leq 0, \\ & 0.5x_1 - 5.5x_2 - 2.5x_3 + 9x_4 \leq 0, \\ & x_1 \leq 1, \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

13.30. Solve the previous problem using lexicographic perturbation to remove all degeneracy. Note that the minimizing vertex for the perturbed problem does not lie in exactly the same position as the minimizer for the original, unperturbed problem, but it is identified by the same set of independent variables in the final, optimal dictionary, and hence those independent variables identify the minimizing vertex for the original problem.

13.31. Consider the linear problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{0}, \\ & \mathbf{x} \succeq \mathbf{0}, \end{aligned}$$

with the right-hand side of each constraint equal to zero. Show that either $\mathbf{x} = \mathbf{0}$ is a minimizer or the problem is unbounded.

13.32.* Suppose that the initial dictionary of a given linear optimization problem is not degenerate and, when solved by the simplex method, there is never a tie for the choice of binding constraint.

(i) Can such a problem have degenerate dictionaries? Explain.

(ii) Can such a problem cycle? Explain.

13.33. Prove Proposition 13.6.3 that the dual of the dual is the primal, as follows:

(i) Beginning with a linear minimization problem in the form (13.12), compute its dual and recast that as a minimization problem in the form (13.14).

(ii) Compute the dual of the new minimization problem and then recast it as a minimization problem. Verify that this agrees with the original primal problem.

13.34. Give the dual of the linear problem

$$\begin{aligned} \text{minimize} \quad & -x_1 - x_2 \\ \text{subject to} \quad & 2x_1 + x_2 \leq 3, \\ & x_1 + 3x_2 \leq 5, \\ & 2x_1 + 3x_2 \leq 4, \\ & x_1, x_2 \geq 0. \end{aligned}$$

By graphing the constraints, solve both the primal and dual problems and verify that the optimal values are equal.

13.35. Show, using the weak duality theorem, but without finding the optimizer or using the simplex method, that the linear problem

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2x_2 - x_3 \\ \text{subject to} \quad & x_1 + x_2 + 2x_3 \leq 4, \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

has an optimizer, and give upper and lower bounds on the optimal value.

13.36. Give an example of a three-dimensional linear optimization problem ($n = m = 3$) where both the primal and its dual are infeasible.

13.37. For each of the linear optimization problems Exercises 13.17–13.19 provide the dual formulations (13.13). Use the optimal solutions for those primal problems (which you found earlier) and complementary slackness to find optimizers for the dual problems.

13.38. Write the dual of the linear problem Exercise 13.27(i) and solve the dual problem using the simplex method. Use your solution to the dual problem to construct a solution to the primal problem.

Notes

This chapter is inspired by [Van14], especially in the use of dictionaries instead of the more commonly used, but more confusing, tableaux. Other useful sources include [NW99, Bie15, Ped04] and [BV04]. The proof that Bland's rule does not cycle as well as Exercises 13.29, 13.31, and 13.32 are from [Van14]. Klee and Minty's famous example first appeared in [KM72]. For more on these sorts of pathological examples see [PSZ09]. For more on the complexity of the simplex method see [Gol94, FS15, DS19].

For a proof of the infinite-dimensional versions of some of the theorems in this chapter from convex analysis, including the Hahn–Banach theorem (Theorem 13.2.10) and the Krein–Millman theorem (Theorem 13.3.12), see [Con90, Pro08, Rud91].

14 Nonlinear Constrained Optimization

The more constraints one imposes, the more one frees oneself.
—Igor Stravinsky

In the previous chapter, we considered optimization problems (13.4) with linear constraints, which resulted in feasible sets characterized by convex polytopes. In this chapter, we consider optimization problems with nonlinear constraints, which produce much more complex geometries. For a nonempty open set $\Omega \subset \mathbb{R}^n$ we consider two kinds of nonlinear constraints. One is called an *equality constraint* and is defined as the set of points satisfying $H(\mathbf{x}) = \mathbf{0}$ for some vector-valued function $H : \Omega \to \mathbb{R}^\ell$. The other is called an *inequality constraint* and is defined as the set of points satisfying $G(\mathbf{x}) \preceq \mathbf{0}$ for some vector-valued function $G : \Omega \to \mathbb{R}^m$.

Given objective function $f \in C^1(\Omega; \mathbb{R})$, we define the *standard form* of a nonlinear constrained minimization problem to be

$$\begin{array}{ll} \underset{\mathbf{x}\in\Omega}{\text{minimize}} & f(\mathbf{x}) \\ \text{subject to} & G(\mathbf{x}) \preceq \mathbf{0}, \\ & H(\mathbf{x}) = \mathbf{0}. \end{array} \tag{14.1}$$

Any point $\mathbf{x} \in \Omega$ satisfying both $G(\mathbf{x}) \preceq \mathbf{0}$ and $H(\mathbf{x}) = 0$ is called a *feasible point* of the problem (14.1). The set $\mathscr{F}$

$$\mathscr{F} = \{\mathbf{x} \in \Omega \mid G(\mathbf{x}) \preceq \mathbf{0} \text{ and } H(\mathbf{x}) = \mathbf{0}\} \subset \Omega \tag{14.2}$$

of feasible points is called the *feasible set* of the problem (14.1).

Constrained optimization problems are generally more difficult to solve than unconstrained problems. This is because the minimizer $\mathbf{x}^* \in \mathscr{F}$ need not lie in the interior of $\mathscr{F}$, and thus it need not satisfy $Df(\mathbf{x}^*) = \mathbf{0}$. Another complication is that the feasible set can be nonconvex, which makes it more difficult to work with.

We begin the chapter with the problem of equality constraints and later treat the more complicated case where there are both equality constraints and inequality constraints.

14.1 Equality-Constrained Optimization

We assume $\Omega \subset \mathbb{R}^n$ is open and an objective function $f \in C^1(\Omega;\mathbb{R})$ is given. The *standard form* of an equality-constrained optimization problem is given by

$$\begin{aligned} \underset{\mathbf{x}\in\Omega}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & H(\mathbf{x}) = \mathbf{0}, \end{aligned} \tag{14.3}$$

where $H : \Omega \to \mathbb{R}^\ell$ is the equality constraint and no inequality constraint G is present. The feasible set for this problem is the set $\mathscr{F} = \{\mathbf{x} \in \Omega \mid H(\mathbf{x}) = \mathbf{0}\}$.

Definition 14.1.1. *A point $\mathbf{x}^* \in \mathscr{F}$ is a* local minimizer *for the problem* (14.3) *if there exists an open neighborhood $U \subset \Omega$ of $\mathbf{x}^*$ such that $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for every $\mathbf{x} \in U \cap \mathscr{F}$. The point $\mathbf{x}^* \in \mathscr{F}$ is a* global minimizer *for the problem* (14.3) *if $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for every $\mathbf{x} \in \mathscr{F}$.*

14.1.1 Regular and Singular Points

A feasible set can have some special points that are not smooth. These could correspond to a sharp cusp, a place where two or more branches intersect, or be badly behaved in some other way. All of these "bad" points are where the derivative of H fails to have full rank.

Definition 14.1.2. *Let $H \in C^1(\Omega;\mathbb{R}^\ell)$. A point $\mathbf{x} \in \mathscr{F}$ is called a* regular point *of $\mathscr{F}$ if $\operatorname{rank}(DH(\mathbf{x})) = \ell$, that is, if $DH(\mathbf{x})$ has full row rank. A point that is not regular is called* singular.

Example 14.1.3. If $\ell = 1$ and $n = 2$, then the feasible set $\mathscr{F} = \{(x,y) \mid H(x,y) = 0\} \subset \mathbb{R}^2$ is a one-dimensional subset of $\mathbb{R}^2$. The derivative $DH(x,y)$ is a single row, so it has maximal row rank unless it vanishes. Here are several examples. See Figure 14.1 for plots of these curves.

(i) If $H(x,y) = x^2 - y^2$, then $\mathscr{F} = \{(x,y) \in \mathbb{R}^2 \mid x^2 = y^2\}$ is the union of the two lines $\{x = y\}$ and $\{x = -y\}$. The derivative $DH(x,y) = \begin{bmatrix} 2x & -2y \end{bmatrix}$ only vanishes at the origin, and the origin is contained in $\mathscr{F}$; so the origin is a singular point of $\mathscr{F}$, while every other point of $\mathscr{F}$ is regular.

(ii) If $H(x,y) = x^5 - y^4$, then the derivative only vanishes at the origin, and the origin lies on the curve $\mathscr{F}$; so the origin is a singular point of the curve $\mathscr{F}$, while every other point of $\mathscr{F}$ is regular.

(iii) If $H(x,y) = x^2 + y^2$, then $\mathscr{F}$ consists of a single point—the origin, which is also singular.

(iv) If $H(x,y) = x^5 - y^3$, then $\mathscr{F}$ has a singular point at the origin, but $\mathscr{F}$ looks "smooth."

(v) If $H(x,y) = x^5 - x - y^3$, then the derivative $DH(x,y) = (5x^4 - 1, 3y^2)$ vanishes only at the points $(x,y) = (\pm 5^{-1/4}, 0)$, but these points do not lie on $\mathscr{F}$, so there are no singular points of $\mathscr{F}$. Every point of $\mathscr{F}$ is regular.

(vi) If $H(x,y) = x^5 + x - y^3$, then the derivative $DH(x,y) = (5x^4 + 1, 3y^2)$ never vanishes, so there are no singular points of $\mathscr{F}$. Every point of $\mathscr{F}$ is regular.

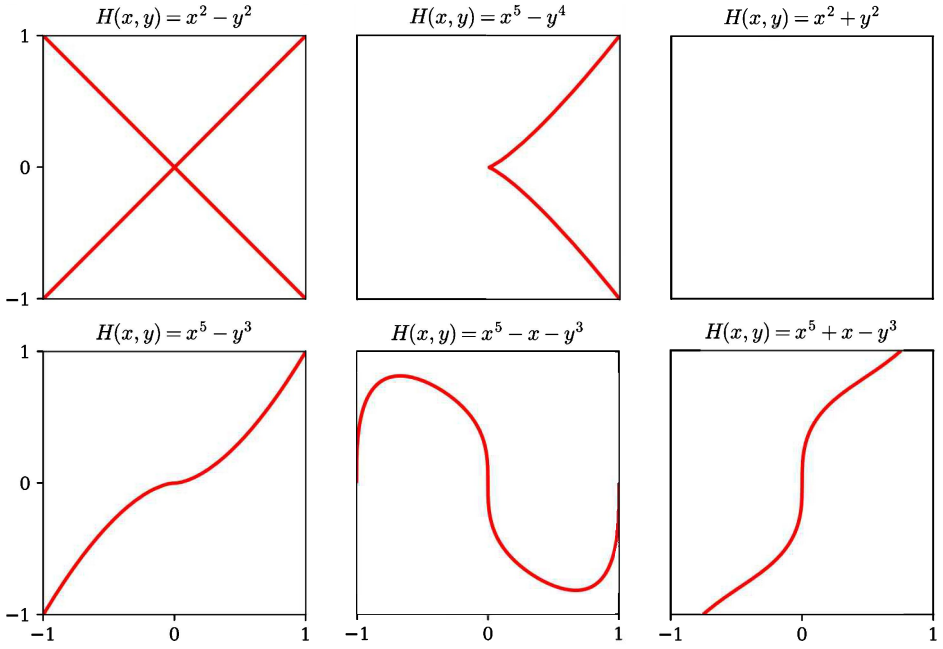


Figure 14.1. *Six plane curves, as described in Example 14.1.3. The first four examples have singular points at the origin, while the last two curves have no singular points. In the top row the singularities are visibly obvious. But in the bottom left case (example* (iv)*) the singularity is not easily visible.*

Remark 14.1.4. We are primarily interested in the regular points of the feasible set of the minimization problem (14.3). Since the rank of $DH(\mathbf{x})$ is never more than the number of columns, there can be no regular points if $\ell > n$, that is, if the number of constraints is greater than the dimension of the space $\mathbb{R}^n$. Further, if a regular point $\mathbf{x} \in \mathscr{F}$ satisfies $\ell = n$, then it is an isolated point (see Exercise 14.5), which doesn't make for a very interesting optimization problem. Therefore, we focus on the case where $\ell < n$. If there are n or more constraints, then it is often the case that some of them are redundant, and eliminating the redundant constraints may give $\ell < n$.

14.1.2 The Geometry of Feasible Sets

Consider the minimization problem (14.3), where the equality constraint function $H \in C^1(\Omega; \mathbb{R}^\ell)$ satisfies $\ell < n$. We have ℓ equations, given by $h_1(\mathbf{x}) = 0$, $h_2(\mathbf{x}) = 0$, $\ldots$, $h_\ell(\mathbf{x}) = 0$, and n unknowns. In the case of affine constraints, if the rows of $DH(\mathbf{x})$ are linearly independent,[53] then the feasible set is an $(n - \ell)$-dimensional affine space.

In the case of nonlinear constraints, if the rows of the derivative $DH(\mathbf{x})$ at a point $\mathbf{x} \in \mathscr{F}$ are linearly independent (that is, $\mathbf{x}$ is a regular point), then we can express the constraints locally (meaning in a neighborhood of $\mathbf{x}$) as an $n - \ell$ dimensional parametric surface (or manifold; see Volume 1, Section 10.3). In other words, there is an open set $U \subset \mathbb{R}^{n-\ell}$, a neighborhood V of $\mathbf{x}$, and an injective function $\boldsymbol{\alpha} \in C^1(U; V)$ such that $\boldsymbol{\alpha}(U) = V \cap \mathscr{F}$.

This parameterization allows us to locally reduce the minimization problem (14.3) to an unconstrained minimization problem. We begin with the following lemma.

Lemma 14.1.5. *Let $H \in (\Omega; \mathbb{R}^\ell)$ satisfy $\ell < n$. At any regular point $\mathbf{x}_0$ of the set*

$$\mathscr{F} = \{\mathbf{x} \in \Omega \mid H(\mathbf{x}) = \mathbf{0}\}, \tag{14.4}$$

there is a neighborhood of $\mathbf{x}_0$ which is an $(n-\ell)$-dimensional parametric surface (or parametrized manifold; see Volume 1, Definition 10.3.1). In other words, there is an open set $U \subset \mathbb{R}^{n-\ell}$, a point $\mathbf{z}_0 \in U$, and an injective C^1 function $\boldsymbol{\alpha} : U \to \Omega \subset \mathbb{R}^n$ with $\operatorname{Im}(\boldsymbol{\alpha}) = V \cap \mathscr{F}$, $D\boldsymbol{\alpha}$ injective, and $\boldsymbol{\alpha}(\mathbf{z}_0) = \mathbf{x}_0$.

Proof. Near any regular point $\mathbf{x}_0$ of $\mathscr{F}$, since

$$DH(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_\ell}{\partial x_1} & \frac{\partial h_\ell}{\partial x_2} & \cdots & \frac{\partial h_\ell}{\partial x_n} \end{bmatrix}$$

has rank ℓ, there are ℓ coordinates $i_1, \ldots, i_\ell$ such that the square submatrix

$$\begin{bmatrix} \frac{\partial h_1}{\partial x_{i_1}} & \frac{\partial h_1}{\partial x_{i_2}} & \cdots & \frac{\partial h_1}{\partial x_{i_\ell}} \\ \frac{\partial h_2}{\partial x_{i_1}} & \frac{\partial h_2}{\partial x_{i_2}} & \cdots & \frac{\partial h_2}{\partial x_{i_\ell}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_\ell}{\partial x_{i_1}} & \frac{\partial h_\ell}{\partial x_{i_2}} & \cdots & \frac{\partial h_\ell}{\partial x_{i_\ell}} \end{bmatrix}$$

is nonsingular. Let $\mathbf{y} = (y_1, \ldots, y_\ell) = (x_{i_1}, \ldots, x_{i_\ell})$ and let $\mathbf{z} = (z_1, \ldots, z_{n-\ell})$ be the remaining $\mathbf{x}$ variables. Rearranging the order of the variables as necessary, we may write $\mathbf{x} = (\mathbf{z}, \mathbf{y})$ and $H : \mathbb{R}^{n-\ell} \times \mathbb{R}^\ell \to \mathbb{R}^\ell$ with $D_{\mathbf{y}} H(\mathbf{z}, \mathbf{y})$ invertible. By the implicit function theorem (Theorem 10.3.2), there are open sets $U \subset \mathbb{R}^{n-\ell}$ and

[53] In the affine case $DH(\mathbf{x})$ is independent of $\mathbf{x}$, so all of the points in the feasible set are regular points.

$V \subset \mathbb{R}^n$, a point $\mathbf{z}_0$ of U, and an injective C^1 function $\boldsymbol{\beta} : U \to \mathbb{R}^\ell$ such that $\mathbf{x}_0 = (\mathbf{z}_0, \boldsymbol{\beta}(\mathbf{z}_0))$ with $H(\mathbf{z}, \boldsymbol{\beta}(\mathbf{z})) = \mathbf{0}$ for all $\mathbf{z} \in U$ and $\boldsymbol{\alpha}(U) = V \cap \mathscr{F}$. Setting $\boldsymbol{\alpha}(\mathbf{z}) = (\mathbf{z}, \boldsymbol{\beta}(\mathbf{z}))$ gives the desired result. □

Remark 14.1.6. The lemma guarantees that in a neighborhood V of a regular point there is always a parametrization $\boldsymbol{\alpha} : U \to \mathbb{R}^n$ whose image is contained in the feasible set. This allows the local optimization of an equality-constrained problem to be reformulated into an unconstrained local optimization problem. In other words, if the regular point $\mathbf{x}^*$ is a local minimum of f constrained to $V \cap \mathscr{F}$, then it is also a local minimum of $f \circ \boldsymbol{\alpha}$ on U. This allows us to generalize the FONC to equality-constrained minimization problems, which we do in Section 14.2.

Remark 14.1.7. Although the implicit function theorem guarantees $\boldsymbol{\alpha}$ exists, it does not provide an explicit expression for $\boldsymbol{\alpha}$. Luckily, it does provide an explicit expression for the first derivative $D\boldsymbol{\alpha}$ via (10.16), which is what we need to derive a constrained FONC; see Section 14.2 for details.

14.1.3 Tangent Spaces and Normal Spaces of Parametrizations

Throughout the remainder of this section, we assume that $H \in C^1(\Omega; \mathbb{R}^\ell)$ with $\ell < n$, that the feasible set is (14.4), and that $\mathbf{x}_0 \in \mathscr{F}$ is a regular point. From Lemma 14.1.5, there exists an open neighborhood $U \subset \mathbb{R}^{n-\ell}$ of $\mathbf{x}_0$ and a parametrized surface (or manifold) $\boldsymbol{\alpha} : U \to \mathbb{R}^n$ with $\operatorname{Im}(\boldsymbol{\alpha}) = V \cap \mathscr{F}$, $D\boldsymbol{\alpha}$ injective, and $\boldsymbol{\alpha}(\mathbf{z}_0) = \mathbf{x}_0$.

For any $\mathbf{a} \in \mathbb{R}^{n-\ell}$, we can define a line in U of the form $\sigma(t) = \mathbf{a}t + \mathbf{z}_0$. This defines a curve $\gamma(t) = \boldsymbol{\alpha}(\sigma(t))$ in $\mathscr{F}$, whose derivative defines a tangent vector of $\mathscr{F}$ at $\mathbf{x}_0$

$$\mathbf{v} = \gamma'(t) = \frac{d}{dt}\boldsymbol{\alpha}(\sigma(t))\bigg|_{t=0} = D\boldsymbol{\alpha}(\mathbf{z}_0)\sigma'(0) = D\boldsymbol{\alpha}(\mathbf{z}_0)\mathbf{a}.$$

The space of all such tangent vectors is equal to $\mathscr{R}(D\boldsymbol{\alpha}(\mathbf{x}_0))$; see Volume 1, Section 10.3.2, for more details. This motivates the following definition.

Definition 14.1.8. *The* tangent space $T_{\mathbf{x}_0}\mathscr{F}$ *of the parametrization* $\boldsymbol{\alpha} : U \to \mathbb{R}^n$ *at* $\mathbf{x}_0 = \boldsymbol{\alpha}(\mathbf{z}_0)$ *is the range of* $D\alpha(\mathbf{z}_0)$ *in* $\mathbb{R}^n$.

As an immediate consequence of the previous discussion, we have the following proposition.

Proposition 14.1.9. *If* $\mathbf{x}$ *is a regular point of* $\mathscr{F}$*, and if* $\mathbf{v} \in T_{\mathbf{x}}\mathscr{F}$*, then there is an interval* $(-b, b) \subset \mathbb{R}$ *and a curve* $\gamma : (-b, b) \to \mathscr{F}$ *in* $\mathscr{F}$ *that is differentiable at* 0 *and such that* $\gamma(0) = \mathbf{x}$ *and* $\gamma'(0) = \mathbf{v}$.

Proof. Let $\mathbf{z}_0 \in U$ and let $\boldsymbol{\alpha} : \mathbb{R}^k \to \mathbb{R}^n$ be a parametrization of $\mathscr{F}$ near $\mathbf{x}$ with $\boldsymbol{\alpha}(\mathbf{z}_0) = \mathbf{x}_0$. By definition, any $\mathbf{v} \in T_{\mathbf{x}_0}M$ is in the range of $D\boldsymbol{\alpha}(\mathbf{z}_0)$, so there is a vector $\mathbf{w} \in \mathbb{R}^k$ such that $D\boldsymbol{\alpha}(\mathbf{z}_0)\mathbf{w} = \mathbf{v}$. Letting $\gamma(t) = \boldsymbol{\alpha}(\mathbf{z}_0 + t\mathbf{w})$ gives $\gamma(0) = \boldsymbol{\alpha}(\mathbf{z}_0) = \mathbf{x}_0$ and $\gamma'(0) = D\boldsymbol{\alpha}(\mathbf{z}_0)\mathbf{w} = \mathbf{v}$. □

The next lemma gives a characterization of the tangent space in terms of the function H instead of the parametrization $\boldsymbol{\alpha}$.

Lemma 14.1.10. $T_{\mathbf{x}_0}\mathscr{F} = \mathscr{N}\left(DH(\mathbf{x}_0)\right)$.

Proof. Since $H(\boldsymbol{\alpha}(\mathbf{z})) = \mathbf{0}$ for all $\mathbf{z} \in U$, the chain rule gives

$$D(H(\boldsymbol{\alpha}(\mathbf{z}_0))) = DH(\mathbf{x}_0)D\boldsymbol{\alpha}(\mathbf{z}_0) = \mathbf{0}.$$

Since every tangent vector $\mathbf{v}$ is of the form $\mathbf{v} = D\boldsymbol{\alpha}(\mathbf{z}_0)\mathbf{a}$ for some $\mathbf{a} \in \mathbb{R}^{n-\ell}$, we have $DH(\mathbf{x}_0)\mathbf{v} = \mathbf{0}\mathbf{a} = \mathbf{0}$. Hence, $T_{\mathbf{x}_0}\mathscr{F} \subset \mathscr{N}\left(DH(\mathbf{x}_0)\right)$.

For equality, it suffices to show that $\dim T_{\mathbf{x}_0}\mathscr{F} = \dim \mathscr{R}\left(D\boldsymbol{\alpha}(\mathbf{x}_0)\right) = n - \ell$. Let $\boldsymbol{\alpha}(\mathbf{z}) = (\mathbf{z}, \boldsymbol{\beta}(\mathbf{z}))$ be the parametrization given in the proof of Lemma 14.1.5. A direct computation gives $D\boldsymbol{\alpha}(\mathbf{z}_0) = \begin{bmatrix} I_{n-\ell} \\ D\boldsymbol{\beta}(\mathbf{z}_0) \end{bmatrix}$, and this clearly has rank $n - \ell$. ∎

Definition 14.1.11. *The* normal space $N_{\mathbf{x}_0}\mathscr{F}$ *of the regular point* $\mathbf{x}_0 \in \mathscr{F}$ *is the orthogonal complement of the tangent space* $T_{\mathbf{x}_0}\mathscr{F} \subset \mathbb{R}^n$*; that is,* $N_{\mathbf{x}_0}\mathscr{F} = T_{\mathbf{x}_0}\mathscr{F}^{\perp}$.

Corollary 14.1.12. $N_{\mathbf{x}_0}\mathscr{F} = T_{\mathbf{x}_0}\mathscr{F}^{\perp} = \mathscr{N}\left(DH(\mathbf{x}_0)\right)^{\perp} = \mathscr{R}\left(DH(\mathbf{x}_0)^{\mathsf{T}}\right)$.

Remark 14.1.13. The observation that the tangent space and normal space are orthogonal is a generalization of the fact that the gradient of a function is orthogonal to level sets. Here the gradient generalizes to $DH(\mathbf{x}_0)^{\mathsf{T}}$ and the tangent vectors to the level sets generalize to the tangent space at $\mathbf{x}_0$.

Nota Bene 14.1.14. Do not confuse the two complementary ways of thinking about the feasible set $\mathscr{F}$ and its tangent space.

One way to think about it is parametrically, that is, as a parametrized manifold $\boldsymbol{\alpha} : U \to \mathbb{R}^n$. This is the approach taken in Volume 1, Chapter 10. In the parametric formulation, the tangent space at a regular point $\mathbf{x}_0 = \boldsymbol{\alpha}(\mathbf{z}_0)$ is

$$T_{\mathbf{x}_0}\mathscr{F} = \mathscr{R}\left(D\boldsymbol{\alpha}(\mathbf{z}_0)\right),$$

and the columns of $D\boldsymbol{\alpha}(\mathbf{z}_0)$ are all tangent to $\mathscr{F}$ at $\mathbf{x}_0$.

Another way to think about $\mathscr{F}$ is implicitly, that is, as the zero set of the function $H : \Omega \to \mathbb{R}^{\ell}$. This is the most natural way to think about a manifold defined by equality constraints. In the implicit formulation the tangent space at a regular point $\mathbf{x}_0$ is given by

$$T_{\mathbf{x}_0}\mathscr{F} = \mathscr{N}\left(DH(\mathbf{x}_0)\right),$$

and the rows of $DH(\mathbf{x}_0)$ are normal to the tangent space of $\mathscr{F}$ at $\mathbf{x}_0$.

Example 14.1.15. If $H : \mathbb{R}^3 \to \mathbb{R}$ is given by $H(x, y, z) = 9xz - 7y^2$, then the surface $\mathscr{F} = \{(x, y, z) \in \mathbb{R}^3 \mid H(x, y, z) = 0\}$ can be parametrized by $\boldsymbol{\alpha}(t, u) = (t^2, 3ut, 7u^2)$. The point $\mathbf{x}_0 = \boldsymbol{\alpha}(1, 1) = (1, 3, 7) \in \mathscr{F}$ is a regular point of $\mathscr{F}$ because $DH(\mathbf{x}_0)$ has rank one.

In the parametric formulation we have

$$T_{\mathbf{x}_0}\mathscr{F} = \mathscr{R}\left(D\boldsymbol{\alpha}(1,1)\right) = \mathscr{R}\left(\begin{bmatrix} 2 & 0 \\ 3 & 3 \\ 0 & 14 \end{bmatrix}\right) = \operatorname{span}\left(\left\{\begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3 \\ 14 \end{bmatrix}\right\}\right),$$

whereas in the implicit formulation we have

$$T_{\mathbf{x}_0}\mathscr{F} = \mathscr{N}\left(DH(1,3,7)\right) = \mathscr{N}\left(\begin{bmatrix} 63 & -42 & 9 \end{bmatrix}\right).$$

The columns $D\boldsymbol{\alpha}((1,1))$ are orthogonal to $DH(\mathbf{x}_0)^\mathsf{T}$ and span $\mathscr{N}\left(DH(\mathbf{x}_0)\right)$, while the vector $DH((\mathbf{x}_0))^\mathsf{T}$ is normal to $\mathscr{F}$ at $\mathbf{x}_0$.

Example 14.1.16. If $H : \mathbb{R}^3 \to \mathbb{R}^2$ is given by $H(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}))$, then

$$DH(\mathbf{x}_0) = \begin{bmatrix} D_1h_1(\mathbf{x}_0) & D_2h_1(\mathbf{x}_0) & D_3h_1(\mathbf{x}_0) \\ D_1h_2(\mathbf{x}_0) & D_2h_2(\mathbf{x}_0) & D_3h_1(\mathbf{x}_0) \end{bmatrix}$$

has maximal row rank (two) if and only if the vectors $Dh_1(\mathbf{x}_0)^\mathsf{T}$ and $Dh_2(\mathbf{x}_0)^\mathsf{T}$ are linearly independent. Lemma 14.1.10 shows that each $Dh_i(\mathbf{x}_0)$ is normal to the surface $S_i = \{\mathbf{x} \mid h_i(\mathbf{x}) = 0\}$ at the point $\mathbf{x}_0$, so a point of $S_1 \cap S_2$ is singular if and only if the normal to S_1 at $\mathbf{x}_0$ is a scalar multiple of the normal to S_2 at $\mathbf{x}_0$. An example of this is depicted in Figure 14.2.

14.2 Lagrange's First-Order Condition

Chapter 12 discussed the first-order necessary condition (FONC) for an optimizer in an unconstrained optimization problem; see Theorem 12.1.6. In this section, we generalize that result to the case of equality-constrained optimization problems (14.3). The main result is Lagrange's first-order condition, which gives a necessary condition similar to that for unconstrained problems.

An important feature of Lagrange's first-order condition is that the necessary condition does not require an explicit parametrization of the feasible set. In other words, it doesn't require reducing the constrained optimization problem into an unconstrained optimization problem.

14.2.1 Lagrange's First-Order Condition

Theorem 14.2.1 (Lagrange's First-Order Necessary Condition). *Let* $\mathbf{x}^*$ *be a local minimizer of the equality-constrained optimization problem* (14.3). *If* $\mathbf{x}^*$ *is*

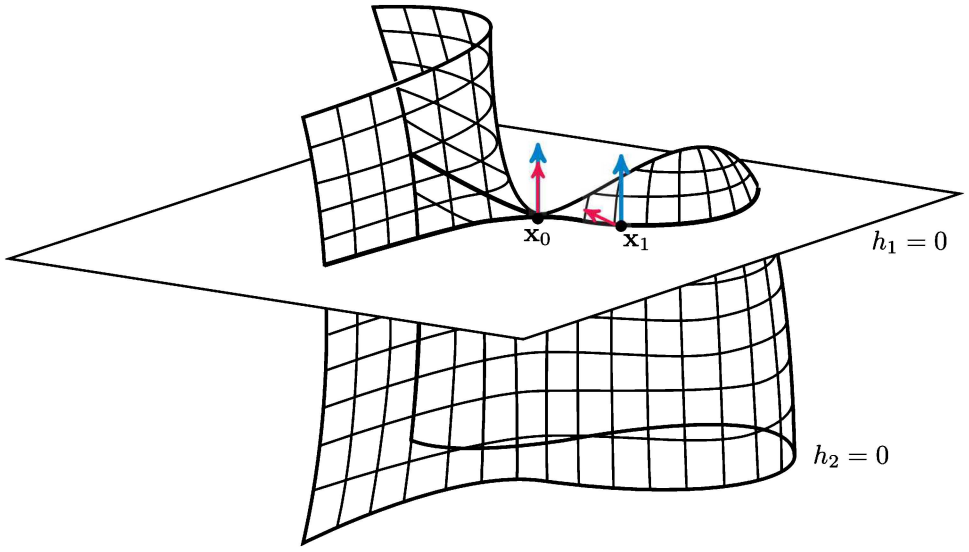


Figure 14.2. *Intersection of two surfaces, defined by $h_1 = 0$ (a plane) and $h_2 = 0$ (the saddle shape), respectively. The intersection is a curve in $\mathbb{R}^3$. At every point $\mathbf{x}$ in the plane $h_1 = 0$ the vector $Dh_1(\mathbf{x})^\mathsf{T}$ (blue) points directly upward (normal) from the plane. Similarly, at every point $\mathbf{x}$ of the surface $h_2 = 0$ the vector $Dh_2(\mathbf{x})^\mathsf{T}$ (red) points in a direction normal to the surface. At the point $\mathbf{x}_1$, and at most other points of the curve of intersection, the vector $Dh_2(\mathbf{x}_1)^\mathsf{T}$ is linearly independent of $Dh_1(\mathbf{x}_1)$, so the matrix $DH(\mathbf{x}_1)$ has maximal rank (rank two) at $\mathbf{x}_1$, and the point is regular. But at $\mathbf{x}_0$ the vector $Dh_2(\mathbf{x}_0)^\mathsf{T}$ (red) and the vector $Dh_1(\mathbf{x}_0)$ are both vertical, and so $Dh_2(\mathbf{x}_0)^\mathsf{T}$ is a scalar multiple of $Dh_1(\mathbf{x}_0)^\mathsf{T}$. This means $DH(\mathbf{x}_0)$ only has rank one at $\mathbf{x}_0$, and therefore $\mathbf{x}_0$ is a singular point of the curve.*

a regular point of the feasible set $\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid H(\mathbf{x}) = \mathbf{0}\} \subset \Omega$, then there exists $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ such that

$$Df(\mathbf{x}^*) + \boldsymbol{\lambda}^{*\mathsf{T}} DH(\mathbf{x}^*) = \mathbf{0}. \tag{14.5}$$

Proof. Let $\boldsymbol{\alpha} : U \to \mathbb{R}^n$ be a parametrization of the feasible manifold near $\mathbf{x}^*$, with U open, and with $\boldsymbol{\alpha}(\mathbf{z}^*) = \mathbf{x}^*$ as given by Lemma 14.1.5. The FONC (Theorem 12.1.6) for $f \circ \boldsymbol{\alpha} : U \to \mathbb{R}$ guarantees that $D(f \circ \boldsymbol{\alpha})(\mathbf{z}^*) = \mathbf{0}$, and the chain rule gives

$$\mathbf{0} = D(f \circ \boldsymbol{\alpha})(\mathbf{z}^*) = Df(\mathbf{x}^*) D\boldsymbol{\alpha}(\mathbf{z}^*).$$

Since $\mathscr{R}(D\boldsymbol{\alpha}(\mathbf{z}^*)) = T_{\mathbf{x}^*}\mathscr{F}$, we have $Df(\mathbf{x}^*)^\mathsf{T} \in N_{\mathbf{x}^*}\mathscr{F} = \mathscr{R}\left(DH(\mathbf{x}^*)^\mathsf{T}\right)$ by Lemma 14.1.10. Therefore $Df(\mathbf{x}^*)^\mathsf{T} = DH(\mathbf{x}^*)^\mathsf{T}\mathbf{v}$ for some $\mathbf{v} \in \mathbb{R}^n$. Setting $\boldsymbol{\lambda}^* = -\mathbf{v}$ gives the result. ∎

The preceding theorem provides a necessary condition for a regular point $\mathbf{x}$ to be a local minimizer or maximizer of f on the feasible set $\mathscr{F}$, namely that the condition in (14.5) is satisfied. Such a point $\mathbf{x}$ is called a *critical point* of the equality-constrained optimization problem (14.3). In the next section we develop

both necessary and sufficient second-derivative tests, which enable us, in many situations, to determine whether a given critical point is a maximizer, a minimizer, or neither.

Remark 14.2.2. We can give a nice geometric interpretation of the Lagrange first-order condition in the special case of single equality constraint $h(\mathbf{x}) = 0$; for an illustration, see Figure 14.3. At each feasible point $\mathbf{x}$, the gradient $Dh(\mathbf{x})^\mathsf{T}$ is normal to the locus $\mathscr{F}$. Similarly, the gradient $Df(\mathbf{x})^\mathsf{T}$ of the objective function is normal to the contour lines of the objective and points in the direction of greatest increase. For a local minimizer $\mathbf{x}^*$, the Lagrange first-order condition guarantees that the two gradients are parallel. They could point in either the same direction or opposite directions, but they must be parallel. If the two gradients are not parallel at a point $\mathbf{x}$, then $-Df(\mathbf{x})^\mathsf{T}$ (which points in the direction of greatest decrease) can be orthogonally projected onto the tangent space of $\mathscr{F}$, and moving in that direction along $\mathscr{F}$ will decrease the value of the objective function.

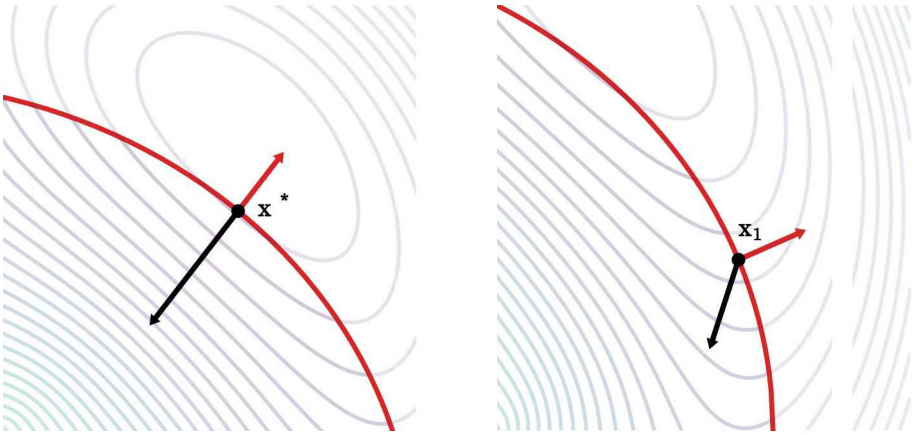


Figure 14.3. *If a problem has a single equality constraint h, then for each point $\mathbf{x}$ the gradient $Dh(\mathbf{x})^\mathsf{T}$ (red arrows) is normal to the feasible set $\mathscr{F}$ (red curves). The gradient $Df(\mathbf{x})^\mathsf{T}$ (black arrows) of the objective is normal to the contour lines of the objective (light purple and green). In the left panel the point $\mathbf{x}^*$ is a local minimizer, and the Lagrange first-order condition guarantees that the two gradients are parallel. In the right panel, the point $\mathbf{x}_1$ is not a local minimizer, and the gradient $Df(\mathbf{x}_1)^\mathsf{T}$ is not parallel to $Dh(\mathbf{x}_1)^\mathsf{T}$. For more see Remark 14.2.2.*

Remark 14.2.3. Any maximization problem of the form

$$\begin{aligned} &\text{maximize} && f(\mathbf{x}) \\ &\text{subject to} && H(\mathbf{x}) = \mathbf{0} \end{aligned} \tag{14.6}$$

can be easily adapted to standard form (14.3) by changing the sign of the objective f. If $\mathbf{x}^*$ is a local maximizer of (14.6), then $\mathbf{x}^*$ must be a local minimizer of $-f$ with the same equality constraint. Therefore, the Lagrange condition (14.5) holds for equality-constrained maximizers.

Example 14.2.4. To maximize the volume $V(x, y, z) = xyz$ of a rectangular box given a fixed surface area S, we solve the minimization problem

$$\begin{aligned} &\text{minimize} && f(x, y, z) = -xyz \\ &\text{subject to} && H(x, y, z) = 2(xy + yz + xz) - S = 0. \end{aligned}$$

By Lagrange's condition (14.5), a minimizer $\mathbf{x}^* = (x, y, z)$ must satisfy

$$\mathbf{0} = Df(\mathbf{x}^*) + \boldsymbol{\lambda}^{*\mathsf{T}} DH(\mathbf{x}^*) = -(yz, xz, yx)^\mathsf{T} + \lambda^* 2(y + z, x + z, y + x)^\mathsf{T}$$

or, equivalently,

$$yz = 2\lambda^*(y + z), \qquad xz = 2\lambda^*(x + z), \qquad \text{and} \qquad xy = 2\lambda^*(x + y).$$

Multiplying the first equation by x, the second by y, and the third by z gives three equalities for xyz. Setting these equal and simplifying gives the relations

$$0 = x(y - z) = y(x - z) = z(x - y).$$

If any coordinate is equal to 0, then the volume is 0, which is not a maximizer. Thus, $x = y = z = \sqrt{S/6}$ is the maximizer—a perfect cube.

Example 14.2.5. We find the points on the unit circle $\{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\|^2 = 1\}$ that are closest to, and farthest from, the point $\mathbf{p} = (3, 4)$. The unit circle corresponds to the equality constraint $H(\mathbf{x}) = \|\mathbf{x}\|^2 - 1 = 0$. The objective function is the distance function $\|\mathbf{x} - \mathbf{p}\|_2$, but the optimizer of the distance function is the same as that of its square, so we can assume that $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{p}\|_2^2$.

We have $DH(\mathbf{x}) = 2\mathbf{x}^\mathsf{T}$ and $Df(\mathbf{x}) = 2(\mathbf{x} - \mathbf{p})^\mathsf{T}$, so the Lagrange condition is equivalent to $\lambda\mathbf{x} = \mathbf{p} - \mathbf{x}$. Thus for $\mathbf{x} = (x, y)$, we have

$$\frac{3 - x}{x} = \lambda = \frac{4 - y}{y},$$

which reduces to

$$\frac{3}{x} = \frac{4}{y}.$$

Solving for y in terms of x and substituting into the constraint $H(\mathbf{x}) = 0$ gives

$$x = \pm\frac{3}{5} \qquad \text{and} \qquad y = \frac{4}{3}x.$$

It is straightforward to check that $(\frac{3}{5}, \frac{4}{5})$ is the minimizer and $(-\frac{3}{5}, -\frac{4}{5})$ is the maximizer.

14.2.2 The Lagrangian

Definition 14.2.6. *The* Lagrangian *corresponding to the equality-constrained optimization problem* (14.3) *is the function* $\mathscr{L} : \Omega \times \mathbb{R}^\ell \to \mathbb{R}$ *given by*

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^\mathsf{T} H(\mathbf{x}). \tag{14.7}$$

The Lagrangian is a convenient tool for lifting the constrained optimization problem (14.3) to a higher-dimensional space, where it becomes an unconstrained problem. In the proof of the Lagrange condition, we used an explicit parametrization of the feasible manifold $\mathscr{F}$ to convert (14.3) into an unconstrained problem, but the Lagrangian allows us to do this without reference to an explicit parametrization of the feasible manifold. Critical points of the Lagrangian correspond to critical points of (14.3). More precisely, if $D\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}$ for some pair $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$, then

$$D\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \begin{bmatrix} Df(\mathbf{x}^*) + \boldsymbol{\lambda}^{*\mathsf{T}} DH(\mathbf{x}^*) & H(\mathbf{x}^*)^\mathsf{T} \end{bmatrix} = \mathbf{0},$$

which is a restatement of (14.5) and the constraint $H(\mathbf{x}^*) = \mathbf{0}$. Thus, for a regular point $\mathbf{x}^*$ to be a local minimizer (or maximizer), it is necessary that there exist $\boldsymbol{\lambda}^*$ such that $D\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}$.

Nota Bene 14.2.7. Although the optimal points of f occur at critical points of the Lagrangian, these critical points are not necessarily optimizers for the Lagrangian. In fact, they are usually saddle points of the Lagrangian.

Example 14.2.8. Recall that the multinomial distribution (see Section 5.7.5) comes from a sequence of n repeated trials of a categorical distribution with parameters $\mathbf{p} = (p_1, p_2, \ldots, p_k) \succeq \mathbf{0}$, such that $\sum_{i=1}^k p_i = 1$. Its support consists of k-tuples of nonnegative integers $\mathbf{x} = (x_1, x_2, \ldots, x_k)$, satisfying $\sum_{j=1}^k x_j = n$. Here x_j represents the number of experiments that had result j. The p.m.f. is

$$\mathbf{x} \mapsto \binom{n}{x_1, x_2, \ldots, x_k} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k}.$$

Given a draw $\mathbf{x} = (x_1, \ldots, x_k)$ from a multinomial distribution with unknown parameters $\mathbf{p}$, we wish to compute the MLE for the parameters $\mathbf{p}$. To do this we must maximize the log-likelihood of the multinomial

$$\ell(\mathbf{p}) = \log \binom{n}{x_1, x_2, \ldots, x_k} + \sum_{j=1}^k x_j \log p_j$$

subject to the equality constraint $\sum_{i=1}^k p_i = 1$ and the inequality constraint $\mathbf{p} \succeq \mathbf{0}$. Since the first term of the log-likelihood has no $\mathbf{p}$ dependence, it has no effect on the choice of maximizer, and so we can disregard it. Thus the

MLE of $\mathbf{p}$ is the solution of the optimization problem

$$\begin{array}{ll} \text{minimize} & -\sum_{j=1}^{k} x_j \log p_j \\ \text{subject to} & \sum_{j=1}^{k} p_j = 1, \\ & p_j \geq 0 \quad \forall j. \end{array} \tag{14.8}$$

No p_j can be zero, since that would make the objective go to infinity, and so we can always assume each $p_j > 0$. Thus, we assume $\Omega = \{\mathbf{p} \in \mathbb{R}^k \mid \mathbf{p} \succ \mathbf{0}\}$. On Ω we have the single equality constraint $\sum_{j=1}^{k} p_j = 1$. The Lagrangian is

$$\mathscr{L}(\mathbf{p}, \lambda) = -\sum_{j=1}^{k} x_j \log p_j + \lambda \left(\left(\sum_{j=1}^{k} p_j \right) - 1 \right). \tag{14.9}$$

The first-order condition requires

$$\mathbf{0} = D_{\mathbf{p}} \mathscr{L}(\mathbf{p}, \lambda) = -\begin{bmatrix} \frac{x_1}{p_1} & \frac{x_2}{p_2} & \cdots & \frac{x_k}{p_k} \end{bmatrix} + \lambda \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}.$$

Thus we have $x_j = \lambda p_j$ for each j. Taking the sum gives

$$n = \sum_{j=1}^{k} x_j = \lambda \sum_{j=1}^{k} p_j = \lambda.$$

It follows that any minimizer $\widehat{\mathbf{p}}$ must satisfy $\widehat{p}_j = \frac{x_j}{n}$ for each $j \in \{1, 2, \ldots, k\}$.

Example 14.2.9. We can generalize Example 14.2.5 to the problem of finding the points on the zero set $\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid H(\mathbf{x}) = \mathbf{0}\}$ which are closest to and farthest from a given point $\mathbf{p} \notin \mathscr{F}$. In the special case that $\mathscr{F}$ is a convex set C or the boundary of a convex set C, then the minimizer $\mathbf{x}^*$ is exactly the projection of $\mathbf{p}$ to C.

Since the objective function is $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{p}\|_2^2$ and the constraint is $H(\mathbf{x}) = \mathbf{0}$, the Lagrangian is

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{x}^\mathsf{T}\mathbf{x} - 2\mathbf{p}^\mathsf{T}\mathbf{x} + \mathbf{p}^\mathsf{T}\mathbf{p} + \boldsymbol{\lambda}^\mathsf{T} H(\mathbf{x}).$$

Taking the derivative yields

$$D\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}) = \begin{bmatrix} 2\mathbf{x}^\mathsf{T} - 2\mathbf{p}^\mathsf{T} + \boldsymbol{\lambda}^\mathsf{T} DH(\mathbf{x}) & H(\mathbf{x}) \end{bmatrix}.$$

Thus, a necessary condition for a minimizer is that $DH(\mathbf{x})^\mathsf{T}\boldsymbol{\lambda} = 2(\mathbf{p} - \mathbf{x})$, which implies that $\mathbf{p} - \mathbf{x} \in \mathscr{R}\left(DH(\mathbf{x})^\mathsf{T}\right) = \mathscr{N}\left(DH(\mathbf{x})\right)^\perp$. In other words, the line segment $\mathbf{p} - \mathbf{x}$ must be orthogonal to the tangent space $T_{\mathbf{x}}\mathscr{F}$.

14.3 Lagrange's Second-Order Conditions

Just as in the unconstrained case, there are second-order conditions for equality-constrained optimization. In this section we state these conditions as theorems, provide some examples, and then rigorously prove the theorems.

14.3.1 Statement of Results and Examples

Recall that the second-order necessary and sufficient conditions relied on the Hessian of the objective function. In the equality-constrained case, they rely on the Hessian of the Lagrangian.

Theorem 14.3.1 (Lagrange Second-Order Necessary Condition). *Let $\mathbf{x}^*$ be a local minimizer of the equality-constrained optimization problem (14.3), where f and H are both C^2 on a neighborhood of $\mathbf{x}^*$. If $\mathbf{x}^*$ is a regular point of $\mathscr{F}$ with corresponding Lagrange multiplier $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ satisfying (14.5), then $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} \geq 0$ whenever $\mathbf{v} \in T_{\mathbf{x}^*}\mathscr{F}$.*

Theorem 14.3.2 (Lagrange Second-Order Sufficient Condition). *Consider the equality-constrained optimization problem (14.3). Assume there exists $\mathbf{x}^* \in \mathbb{R}^n$ and $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ such that (14.5) holds. If f and H are both C^2 in a neighborhood of $\mathbf{x}^*$ and $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} > 0$ for all nonzero $\mathbf{v} \in T_{\mathbf{x}^*}\mathscr{F}$, then $\mathbf{x}^*$ is a strict local minimizer.*

Remark 14.3.3. As in Remark 14.2.3 with the first-order condition, we can provide second-order necessary and sufficient conditions for maximization problems of the form (14.6). In particular, by changing the direction of the inequalities, we have $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} \leq 0$ for the SONC, and $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} < 0$ for the SOSC. For more details, see Exercise 14.13.

Example 14.3.4. Consider again the problem of finding the MLE for the multinomial distribution (see Example 14.2.8). Recall that the Lagrangian $\mathscr{L}(\mathbf{p}, \boldsymbol{\lambda})$ is given by (14.9) and has the derivative

$$D_{\mathbf{p}} \mathscr{L}(\mathbf{p}, \lambda) = -\begin{bmatrix} \frac{x_1}{p_1} & \frac{x_2}{p_2} & \dots & \frac{x_k}{p_k} \end{bmatrix} + \lambda \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}.$$

Thus for all $\mathbf{p} > 0$, we have

$$D^2_{\mathbf{p}} \mathscr{L}(\mathbf{p}, \boldsymbol{\lambda}) = \begin{bmatrix} x_1/p_1^2 & 0 & 0 & \dots & 0 \\ 0 & x_2/p_2^2 & 0 & \dots & 0 \\ 0 & 0 & x_3/p_3^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & x_n/p_n^2 \end{bmatrix} > 0.$$

Therefore, this unique critical point is a minimizer.

Remark 14.3.5. The SOSC does not require the Hessian $D^2_{\mathbf{p}}\mathscr{L}(\mathbf{x})$ of the Lagrangian at the critical point $\mathbf{x}$ to be positive definite but rather that $\mathbf{v}^{\mathsf{T}} D^2_{\mathbf{x}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v}$ is positive for each $\mathbf{v} \in T_{\mathbf{x}}\mathscr{F}$. However, if the Hessian is positive definite, then $\mathbf{v}^{\mathsf{T}} D^2_{\mathbf{x}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} > 0$ holds for all $\mathbf{v} \neq \mathbf{0}$. Thus, we can think of a positive definite Hessian as a "strong" sufficient condition.

Example 14.3.6. Consider the problem

$$\begin{aligned} \text{minimize} \quad & -x_1x_2 - x_2x_3 - x_1x_3 \\ \text{subject to} \quad & h = x_1 + x_2 + x_3 - 3 = 0. \end{aligned}$$

The Lagrangian is

$$\mathscr{L}(x_1, x_2, x_3, \lambda) = -x_1x_2 - x_2x_3 - x_1x_3 + \lambda(x_1 + x_2 + x_3 - 3)$$

with

$$\begin{aligned} &D\mathscr{L}(x_1, x_2, x_3, \lambda) \\ &\quad = \begin{bmatrix} \lambda - x_2 - x_3 & \lambda - x_1 - x_3 & \lambda - x_1 - x_2 & x_1 + x_2 + x_3 - 3 \end{bmatrix}. \end{aligned}$$

A little algebra shows that the Lagrange first-order condition is satisfied only at $(1, 1, 1)$. The Hessian (in $\mathbf{x}$) is

$$D^2_{\mathbf{x}}\mathscr{L} = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix},$$

which is not positive definite, because its eigenvalues are -2, 1, and 1. However, the feasible set $\mathscr{F}$ is a plane in $\mathbb{R}^3$ with normal vector $\mathbf{n} = D_{\mathbf{x}}h = (1, 1, 1)$, so a vector $\mathbf{v} \in \mathbb{R}^3$ is a tangent vector if and only if $\mathbf{n}^{\mathsf{T}}\mathbf{v} = 0$ or, equivalently, $v_1 + v_2 + v_3 = 0$. For any tangent vector $\mathbf{v}$, we have $\mathbf{v}^{\mathsf{T}} D^2_{\mathbf{x}}\mathscr{L}\mathbf{v} = v_1(-v_2 - v_3) + v_2(-v_1 - v_3) + v_3(-v_1 - v_2) = v_1^2 + v_2^2 + v_3^2 \geq 0$ with equality if and only if $\mathbf{v} = \mathbf{0}$. Therefore the Lagrange SOSC guarantees that $(1, 1, 1)$ is a local minimizer.

Example 14.3.7. Assume that the benefit (utility) enjoyed by consuming the amounts x and y, respectively, of two goods is given by the function $U(x, y) = a\ln(x) + b\ln(y)$. The available budget for these goods is fixed (say, A dollars) and the unit cost of the goods is p_x and p_y, respectively, imposing the constraint $xp_x + yp_y = A$.

To maximize utility let $\Omega = \{(x, y) \in \mathbb{R}^2 \mid (x, y) \succ \mathbf{0}\}$ (note that zero consumption in either category produces $-\infty$ utility) and solve

$$\begin{aligned} \underset{(x,y)\in\Omega}{\text{minimize}} \quad & -U(x, y) = -a\ln(x) - b\ln(y) \\ \text{subject to} \quad & xp_x + yp_y - A = 0. \end{aligned} \tag{14.10}$$

It is easy to solve for y in terms of x, which would turn this into an unconstrained optimization problem. But we leave this as a constrained problem to illustrate how to apply the Lagrange second-order criterion.

The Lagrangian is

$$\mathscr{L}(x, y, \lambda) = -a\ln(x) - b\ln(y) + \lambda(xp_x + yp_y - A).$$

Its derivative is $D\mathscr{L}(x, y, \lambda) = \begin{bmatrix} -\frac{a}{x} + \lambda p_x & -\frac{b}{y} + \lambda p_y & xp_x + yp_y - A \end{bmatrix}$. A critical point must satisfy

$$\lambda = \frac{a}{(xp_x)} = \frac{b}{(yp_y)} \qquad \text{and} \qquad yp_y = A - xp_x.$$

Solving for x, y, and λ gives

$$x = \frac{aA}{p_x(a+b)}, \qquad y = \frac{bA}{p_y(a+b)}, \qquad \text{and} \qquad \lambda = \frac{a+b}{A}$$

as the only critical point.

The second derivative with respect to $\mathbf{x} = (x, y)$ is

$$D^2_{\mathbf{x}}\mathscr{L} = \begin{bmatrix} a/x^2 & 0 \\ 0 & b/y^2 \end{bmatrix} > 0 \qquad \text{for all nonzero } x \text{ and } y,$$

so the Lagrange SOSC guarantees that this critical point is a local minimizer of (14.10).

Example 14.3.8. Recall the problem of Example 14.2.9, of finding the points on the zero set $\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid H(\mathbf{x}) = 0\}$ which are closest to and farthest from a point $\mathbf{p} \notin \mathscr{F}$. In this problem, the second derivative is

$$D^2_{\mathbf{x}}\mathscr{L} = 2I + \lambda D^2 H(\mathbf{x}).$$

Consider the case where $H : \mathbb{R}^2 \to \mathbb{R}$ is $H(\mathbf{x}) = x^2 - y^2 - 1$, and $\mathbf{p} = \begin{bmatrix} 8 & 2\sqrt{3} \end{bmatrix}^\mathsf{T}$. In this case we have $DH(\mathbf{x}) = \begin{bmatrix} 2x & -2y \end{bmatrix}$ and $D^2H(\mathbf{x}) = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$. The Lagrange FONC is

$$DH(\mathbf{x})^\mathsf{T}\lambda = 2(\mathbf{p} - \mathbf{x}).$$

One can easily check that the point $\mathbf{x} = (2, -\sqrt{3})$ satisfies the FONC with $\lambda = 3$. But for the second order condition, we have $D^2_{\mathbf{x}}\mathscr{L} = \begin{bmatrix} 8 & 0 \\ 0 & -4 \end{bmatrix}$, which is indefinite.

The tangent space at $\mathbf{x}$ is the kernel

$$T_{\mathbf{x}}\mathscr{F} = \mathscr{N}\left(DH(\mathbf{x})\right) = \mathscr{N}\left(\begin{bmatrix} 4 & -2\sqrt{3} \end{bmatrix}\right) = \operatorname{span}\left(\begin{bmatrix} \sqrt{3} \\ 2 \end{bmatrix}\right).$$

Thus, every vector $\mathbf{v} \in T_{\mathbf{x}}\mathscr{F}$ is of the form $\mathbf{v} = \alpha(\sqrt{3}, 2)$. Checking this in the Lagrange second-order condition gives

$$\mathbf{v}^{\mathsf{T}} D^2_{\mathbf{x}} \mathscr{L} \mathbf{v} = \alpha^2(24 - 16) > 0,$$

and so $\mathbf{x}$ is a local minimizer by Lagrange's SOSC.

Example 14.3.9. Consider again the problem in Example 14.2.9, but where $H : \mathbb{R}^n \to \mathbb{R}$ is $H(\mathbf{x}) = \mathbf{x}^{\mathsf{T}}\mathbf{x} - 1$, so that $\mathscr{F} = \{\mathbf{x} \mid H(\mathbf{x}) = 0\}$ is the unit sphere. We have $DH(\mathbf{x}) = 2\mathbf{x}^{\mathsf{T}}$ and $D^2H(\mathbf{x}) = 2I$. The necessary condition for an optimizer is that $DH(\mathbf{x})^{\mathsf{T}}\lambda = 2(\mathbf{p} - \mathbf{x})$, so we have $\mathbf{x} = \mathbf{p}/(1 + \lambda)$ for some λ. Combining this with $H(\mathbf{x}) = 0$ gives $1 = \mathbf{p}^{\mathsf{T}}\mathbf{p}/(1 + \lambda)^2$ or

$$\lambda = \pm\sqrt{\mathbf{p}^{\mathsf{T}}\mathbf{p}} - 1 \qquad \text{and} \qquad \mathbf{x} = \pm\frac{\mathbf{p}}{\|\mathbf{p}\|}.$$

The Hessian is $D^2_{\mathbf{x}}\mathscr{L} = (2+2\lambda)I$, which is positive definite if $\lambda > -1$, which occurs if we take $\lambda = \|\mathbf{p}\| - 1$, which gives $\mathbf{x} = \frac{\mathbf{p}}{\|\mathbf{p}\|}$. Therefore Lagrange's SOSC guarantees that $\mathbf{x} = \frac{\mathbf{p}}{\|\mathbf{p}\|}$ is a minimizer. Similarly, Lagrange's SOSC guarantees that $\mathbf{x} = -\frac{\mathbf{p}}{\|\mathbf{p}\|}$ is a maximizer.

14.3.2 Proof of the Lagrange Second-Order Necessary Condition

We prove the Lagrange SONC, Theorem 14.3.1.

Proof. Let $\mathbf{v} \in T_{\mathbf{x}^*}\mathscr{F}$, where $\mathbf{x}^* \in \mathscr{F}$ is a local minimizer. By Proposition 14.1.9, there exists a curve $\gamma(t)$ in $\mathscr{F}$ with $\gamma(0) = \mathbf{x}^*$ and $\gamma'(0) = \mathbf{v}$. Moreover, $t = 0$ is a local minimizer of $\phi(t) = f(\gamma(t))$, and thus $\phi''(0) \geq 0$. Since

$$\frac{d^2\phi}{dt^2}(0) = \gamma'(0)^{\mathsf{T}} D^2 f(\gamma(0))\gamma'(0) + Df(\gamma(0))\gamma''(0) = \mathbf{v}^{\mathsf{T}} D^2 f(\mathbf{x}^*)\mathbf{v} + Df(\mathbf{x}^*)\gamma''(0),$$

we have

$$\mathbf{v}^{\mathsf{T}} D^2 f(\mathbf{x}^*)\mathbf{v} + Df(\mathbf{x}^*)\gamma''(0) \geq 0. \tag{14.11}$$

Moreover, since $H(\gamma(t)) = \mathbf{0}$ (and writing $H(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) & h_2(\mathbf{x}) & \cdots & h_\ell(\mathbf{x}) \end{bmatrix}^{\mathsf{T}}$), we have

$$\begin{aligned} 0 = \frac{d^2}{dt^2}\boldsymbol{\lambda}^{*\mathsf{T}} H(\gamma(t))\Big|_{t=0} &= \sum_{j=1}^{\ell} \lambda_j \frac{d}{dt}\left(Dh_j(\gamma(t))\gamma'(t)\right)\Big|_{t=0} \\ &= \sum_{j=1}^{\ell} \lambda_j \left(\gamma'(0)^{\mathsf{T}} D^2 h_j(\gamma(0))\gamma'(0) + Dh_j(\gamma(0))\gamma''(0)\right) \\ &= \left(\sum_{j=1}^{\ell} \lambda_j \mathbf{v}^{\mathsf{T}} D^2 h_j(\mathbf{x}^*)\mathbf{v}\right) + \boldsymbol{\lambda}^{*\mathsf{T}} DH(\mathbf{x}^*)\gamma''(0), \end{aligned} \tag{14.12}$$

where $\boldsymbol{\lambda}^* = \begin{bmatrix}\lambda_1 & \lambda_2 & \cdots & \lambda_\ell\end{bmatrix}^\mathsf{T}$. Adding (14.11) and (14.12) yields

$$\mathbf{v}^\mathsf{T}\left(D^2 f(\mathbf{x}^*) + \sum_{j=1}^{\ell} \lambda_j D^2 h_j(\mathbf{x}^*)\right)\mathbf{v} + \Big(Df(\mathbf{x}^*) + \boldsymbol{\lambda}^{*\mathsf{T}} DH(\mathbf{x}^*)\Big)\gamma''(0) \geq 0.$$

Lagrange's first-order condition (14.5) gives $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} \geq 0$, since

$$D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = D^2 f(\mathbf{x}^*) + \sum_{j=1}^{\ell} \lambda_j D^2 h_j(\mathbf{x}^*). \quad \square$$

14.3.3 Proof of the Lagrange Second-Order Sufficient Condition

We prove the Lagrange SOSC, Theorem 14.3.1.

Proof. Suppose that $\mathbf{x}^* \in \mathscr{F}$ satisfies the hypothesis yet is not a strict local minimizer. This implies there exists a sequence $\{\mathbf{x}_k\}_{k=1}^{\infty} \subset \mathscr{F} \setminus \{\mathbf{x}^*\}$ that converges to $\mathbf{x}^*$ such that $f(\mathbf{x}_k) \leq f(\mathbf{x}^*)$. Let

$$\boldsymbol{\varepsilon}_k = \mathbf{x}_k - \mathbf{x}^* \qquad \text{and} \qquad \mathbf{s}_k = \frac{\boldsymbol{\varepsilon}_k}{\|\boldsymbol{\varepsilon}_k\|}.$$

Note that $\{\mathbf{s}_k\}_{k=1}^{\infty}$ is bounded and thus has a convergent subsequence $\mathbf{s}_{k_i} \to \mathbf{s}^*$. By reindexing the subsequence we may assume that $\mathbf{x}_k \to \mathbf{x}^*$ and $\mathbf{s}_k \to \mathbf{s}^*$. Note that $H(\mathbf{x}_k) - H(\mathbf{x}^*) = \mathbf{0}$ for each $k \in \mathbb{N}$. Dividing by $\|\boldsymbol{\varepsilon}_k\|$ and taking the limit gives $DH(\mathbf{x}^*)\mathbf{s}^* = \mathbf{0}$, which implies that $\mathbf{s}^* \in T_{\mathbf{x}^*}\mathscr{F}$.

Taylor's theorem (Theorem 10.3.8) gives

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) = Df(\mathbf{x}^*)\boldsymbol{\varepsilon}_k + \int_0^1 (1-t)\boldsymbol{\varepsilon}_k^\mathsf{T} D^2 f(\mathbf{x}^* + t\boldsymbol{\varepsilon}_k)\boldsymbol{\varepsilon}_k \, dt,$$

and if $H = (h_1, \ldots, h_\ell)$, then for every $j \in \{1, \ldots, m\}$ we also have

$$0 = Dh_j(\mathbf{x}^*)\boldsymbol{\varepsilon}_k + \int_0^1 (1-t)\boldsymbol{\varepsilon}_k^\mathsf{T} D^2 h_j(\mathbf{x}^* + t\boldsymbol{\varepsilon}_k)\boldsymbol{\varepsilon}_k \, dt.$$

The second equality holds since $H(\mathbf{x}_k) = H(\mathbf{x}^*) = \mathbf{0}$. This implies

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) = \|\mathbf{x}_k - \mathbf{x}^*\| Df(\mathbf{x}^*)\mathbf{s}_k + \|\mathbf{x}_k - \mathbf{x}^*\|^2 \int_0^1 (1-t)\mathbf{s}_k^\mathsf{T} D^2 f(\mathbf{x}^* + t\boldsymbol{\varepsilon}_k)\mathbf{s}_k \, dt$$

and

$$0 = \|\mathbf{x}_k - \mathbf{x}^*\| Dh_j(\mathbf{x}^*)\mathbf{s}_k + \|\mathbf{x}_k - \mathbf{x}^*\|^2 \int_0^1 (1-t)\mathbf{s}_k^\mathsf{T} D^2 h_j(\mathbf{x}^* + t\boldsymbol{\varepsilon}_k)\mathbf{s}_k \, dt.$$

Letting $\boldsymbol{\lambda}^* = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)$ and multiplying the second equality by λ_j, summing over j, and then adding to the first equality gives

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) = \|\mathbf{x}_k - \mathbf{x}^*\|^2 \int_0^1 (1-t)\mathbf{s}_k^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^* + t\boldsymbol{\varepsilon}_k, \boldsymbol{\lambda}^*)\mathbf{s}_k \, dt. \tag{14.13}$$

By hypothesis, we have $(\mathbf{s}^*)^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{s}^* > 0$. Since $\mathscr{L}$ is C^2 in a neighborhood of $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$, it follows that $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}^*)\mathbf{v} > 0$ holds for $\mathbf{x}$ in a neighborhood of $\mathbf{x}^*$ and $\mathbf{v}$ sufficiently close to $\mathbf{s}^*$. Thus, if k sufficiently large, then the integral is positive. Hence, the right-hand side of (14.13) is positive, whereas the left-hand side is nonpositive, which is a contradiction. Thus, $\mathbf{x}^* \in \mathscr{F}$ is a strict local minimizer. □

14.4 Karush–Kuhn–Tucker First-Order Conditions

We now turn to the case where the feasible set is defined by both equality constraints and inequality constraints. Throughout this section and the next we consider constrained optimization problems in standard form (14.1), where f is continuously differentiable on the open set $\Omega \subset \mathbb{R}^n$. We also assume that $G : \Omega \to \mathbb{R}^m$ and $H : \Omega \to \mathbb{R}^\ell$ are continuously differentiable and have the form

$$G(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_m(\mathbf{x}) \end{bmatrix} \quad \text{and} \quad H(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ \vdots \\ h_\ell(\mathbf{x}) \end{bmatrix}.$$

An example of an inequality-constrained problem is depicted in Figure 14.4.

Definition 14.4.1. *A point $\mathbf{x} \in \Omega$ is a* feasible point *of the optimization problem* (14.1) *if it satisfies all the constraints, that is, if $G(\mathbf{x}) \preceq \mathbf{0}$ and $H(\mathbf{x}) = \mathbf{0}$. The set $\mathscr{F}$ of all feasible points is called the* feasible set *and satisfies* (14.2). *A point $\mathbf{x}^* \in \mathscr{F}$ is a* local minimizer *for the problem* (14.1) *if there exists an open set $U \subset \Omega$ such that $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for every $\mathbf{x} \in U \cap \mathscr{F}$. The point $\mathbf{x}^*$ is a* global minimizer *for the problem* (14.1) *if $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for every $\mathbf{x} \in \mathscr{F}$.*

The Karush–Kuhn–Tucker (KKT) conditions, given below in Theorem 14.4.5, provide conditions for inequality-constrained optimizers analogous to the Lagrange conditions for equality-constrained optimizers.

14.4.1 The Locus of Binding Constraints

Before we can describe the KKT conditions, we need some definitions about points that satisfy the binding constraints.

Definition 14.4.2. *Given a point $\mathbf{x} \in \mathscr{F}$, an inequality constraint $g_j(\mathbf{x}) \leq 0$ is* binding *(or* active*) at $\mathbf{x}$ if $g_j(\mathbf{x}) = 0$. If instead $g_j(\mathbf{x}) < 0$, then the constraint is* nonbinding *(or* inactive*) at $\mathbf{x}$. Let $J(\mathbf{x})$ denote the index set of binding constraints at $\mathbf{x}$, that is,*

$$J(\mathbf{x}) = \{j \mid g_j(\mathbf{x}) = 0\}.$$

The locus of binding constraints at $\mathbf{x}$ *is the set*

$$\widetilde{\mathscr{F}}(\mathbf{x}) = \{\mathbf{y} \in \Omega \mid H(\mathbf{y}) = \mathbf{0} \text{ and } g_j(\mathbf{y}) = 0 \ \forall j \in J(\mathbf{x})\} \subset \mathscr{F}.$$

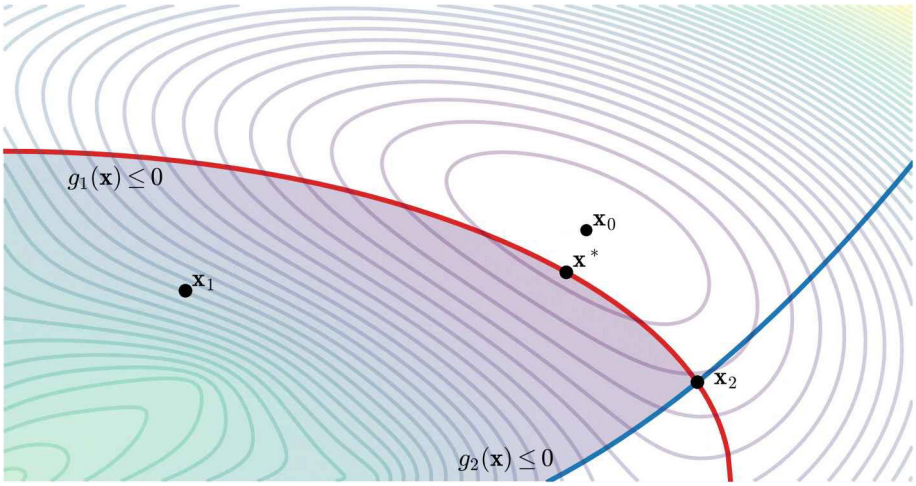


Figure 14.4. *An inequality-constrained optimization problem in* $\mathbb{R}^2$ *with constraints* $g_1(\mathbf{x}) \le 0$ *and* $g_2(\mathbf{x}) \le 0$. *The minimizer is* $\mathbf{x}^*$. *For more on this figure see Example 14.4.4.*

A point $\mathbf{x}$ *is a* regular point *for the optimization problem* (14.1) *if it is regular for* $\widetilde{\mathscr{F}}(\mathbf{x})$, *that is, if the set*

$$\{Dh_i(\mathbf{x})\}_{i=1}^{\ell} \cup \{Dg_j(\mathbf{x})\}_{j \in J(\mathbf{x})}$$

is linearly independent. If $\mathbf{x}$ *is regular, let* $\widetilde{T}(\mathbf{x}) = T_{\mathbf{x}}\widetilde{\mathscr{F}}(\mathbf{x})$ *be the tangent space to* $\widetilde{\mathscr{F}}(\mathbf{x})$ *at* $\mathbf{x}$.

Remark 14.4.3. If the inequality constraints are all affine, then the binding (active) constraints at a point $\mathbf{x}$ are the same as the active constraints defined in Definition 13.3.8.

Example 14.4.4. Figure 14.4 illustrates an inequality-constrained optimization problem in $\mathbb{R}^2$ with constraints $g_1(\mathbf{x}) \le 0$ and $g_2(\mathbf{x}) \le 0$. The feasible set $\mathscr{F} = \{\mathbf{x} \mid g(\mathbf{x}) \le 0\}$ is shaded in color (lower left). The objective function f is shown as a contour plot with smaller values colored darker (more purple) and larger values colored lighter (more yellow). The point $\mathbf{x}^*$ is the minimizer of this problem. The point $\mathbf{x}_0$ is not the minimizer for this problem because it is not feasible, but it would be the minimizer if the problem were unconstrained.

The only binding constraint at the point $\mathbf{x}^*$ is g_1, and so $J(\mathbf{x}^*) = \{1\}$ and the locus $\widetilde{\mathscr{F}}$ of binding constraints at $\mathbf{x}^*$ is the red curve $\{\mathbf{x} \mid g_1(\mathbf{x}) = 0\}$. There are no active constraints at $\mathbf{x}_1$, so $J(\mathbf{x}_1) = \emptyset$. The binding constraints at $\mathbf{x}_2$ are g_1 and g_2, so $J(\mathbf{x}_2) = \{1, 2\}$, and the locus $\widetilde{\mathscr{F}}$ of binding constraints at $\mathbf{x}_2$ is the singleton set $\{\mathbf{x}_2\}$ (the intersection $\{\mathbf{x} \mid g_1(x) = 0\} \cap \{\mathbf{x} \mid g_2(\mathbf{x}) = 0\}$ of the red and blue curves).

14.4.2 The KKT First-Order Conditions

We can now give the KKT first-order conditions, which give a powerful set of necessary conditions for a local minimizer.

Theorem 14.4.5 (KKT First-Order Conditions). *Assume that* $\mathbf{x}^* \in \mathscr{F}$ *is a local minimizer of the constrained optimization problem* (14.1). *If* $\mathbf{x}^* \in \mathscr{F}$ *is a regular point, then there exists* $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ *and* $\boldsymbol{\mu}^* \in \mathbb{R}^m$ *such that*

(i) $Df(\mathbf{x}^*) + (\boldsymbol{\lambda}^*)^\mathsf{T} DH(\mathbf{x}^*) + (\boldsymbol{\mu}^*)^\mathsf{T} DG(\mathbf{x}^*) = \mathbf{0}$,

(ii) $\boldsymbol{\mu}^* \succeq \mathbf{0}$, *and*

(iii) $\mu_i^* g_i(\mathbf{x}^*) = 0$ *for all* $i \in \{1, \ldots, m\}$.

We refer to (i)–(iii) *as the* KKT conditions *of* (14.1).

We give the proof of the KKT first-order conditions in Section 14.4.5.

Example 14.4.6. An illustration of the KKT conditions in $\mathbb{R}^2$ for the case of no equality constraints ($\ell = 0$) and three inequality constraints $G : \mathbb{R}^2 \to \mathbb{R}^3$ is given in Figure 14.5. The local optimizer is $\mathbf{x}^*$ in that figure. The binding constraints at the point $\mathbf{x}^*$ are g_1 and g_2, and the locus $\widetilde{\mathscr{F}}$ of binding constraints at $\mathbf{x}^*$ is the singleton set $\{\mathbf{x}^*\}$. Because g_3 is not binding at $\mathbf{x}^*$, the multiplier μ_3^* is 0 (by condition (iii)), which means that condition (i) does not involve the gradient of g_3. The KKT conditions guarantee that

$$Df(\mathbf{x}^*) + \mu_1^* Dg_1(\mathbf{x}^*) + \mu_2^* Dg_2(\mathbf{x}^*) = 0$$

with $\mu_1^*, \mu_2^* \geq 0$. This is equivalent to saying that $Df(\mathbf{x}^*)^\mathsf{T} = -\mu_1^* Dg_1(\mathbf{x}^*)^\mathsf{T} - \mu_2^* Dg_2(\mathbf{x}^*)^\mathsf{T}$, which means that the gradient $Df(\mathbf{x}^*)^\mathsf{T}$ (black arrow) is a negative linear combination of the two constraint gradients $Dg_1(\mathbf{x}^*)^\mathsf{T}$ (red arrow) and $Dg_2(\mathbf{x}^*)^\mathsf{T}$ (blue arrow).

Contrast this with the situation in Figure 14.6. The point $\mathbf{x}_1$ cannot be a minimizer because the gradient $Df(\mathbf{x}^*)^\mathsf{T}$ is not a negative linear combination of the two binding-constraint gradients $Dg_1(\mathbf{x}^*)^\mathsf{T}$ and $Dg_2(\mathbf{x}^*)^\mathsf{T}$. But projecting $-Df(\mathbf{x}^*)^\mathsf{T}$ orthogonally onto the tangent space of one of the binding-constraint loci (the red curve) gives a vector (orange) that points in a direction that decreases f.

Remark 14.4.7. The feasibility conditions $H(\mathbf{x}^*) = \mathbf{0}$ and $G(\mathbf{x}^*) \preceq \mathbf{0}$ are sometimes called the *primal feasibility* conditions, to distinguish them from the two KKT conditions $D_\mathbf{x}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$ and $\boldsymbol{\mu}^* \succeq \mathbf{0}$, which are sometimes called the *dual feasibility* conditions. The third KKT condition that $\mu_i^* g_i(\mathbf{x}^*) = 0$ for all i is called *complementary slackness* and is considered neither primal nor dual. Exercise 14.22 shows that these conditions correspond exactly to their counterparts of the same name for linear optimization. We discuss duality in more depth in the next chapter.

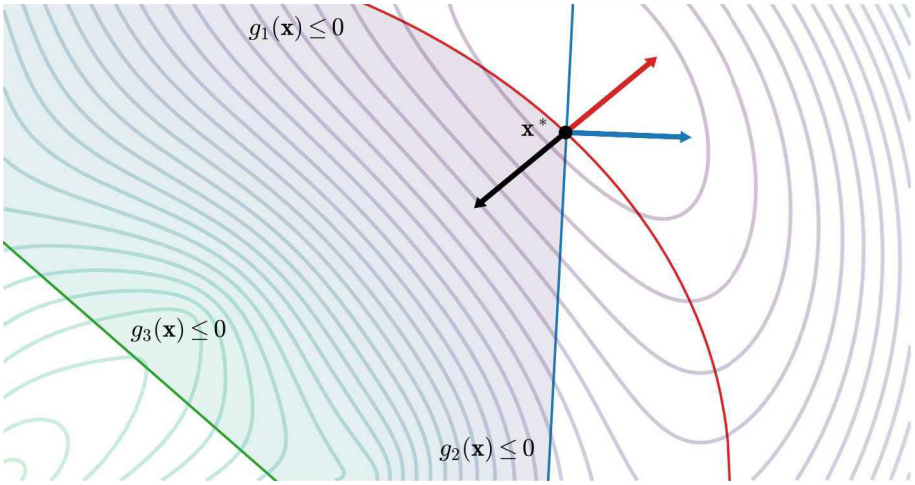


Figure 14.5. *A local optimizer $\mathbf{x}^*$ of an inequality-constrained optimization problem in $\mathbb{R}^2$ with two binding constraints. The first-order KKT constraints guarantee that the gradient $Df(\mathbf{x}^*)^\mathsf{T}$ (black arrow) is a negative linear combination of the two binding-constraint gradients $Dg_1(\mathbf{x}^*)^\mathsf{T}$ (red arrow) and $Dg_2(\mathbf{x}^*)^\mathsf{T}$ (blue arrow). For more details, see Example 14.4.6.*

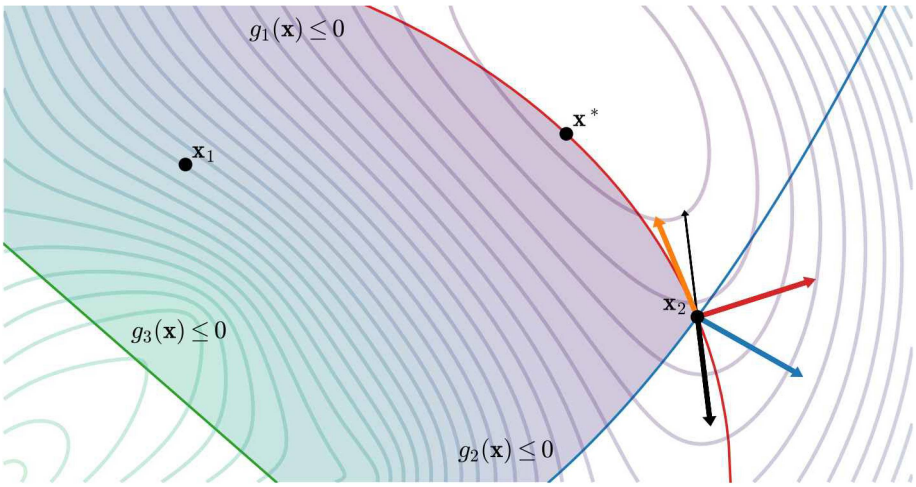


Figure 14.6. *The point $\mathbf{x}_1$ in this inequality-constrained optimization problem in $\mathbb{R}^2$ cannot be a minimizer because the gradient $Df(\mathbf{x}_1)^\mathsf{T}$ (black arrow) is not a negative linear combination of the two binding-constraint gradients $Dg_1(\mathbf{x}_1)^\mathsf{T}$ (red arrow) and $Dg_2(\mathbf{x}_1)^\mathsf{T}$ (blue arrow). For more details, see Example 14.4.6.*

14.4.3 Examples

Example 14.4.8. Consider the problem

$$\begin{aligned} \text{minimize} \quad & f(x,y) = (x-2)^2 + 2(y-1)^2 \\ \text{subject to} \quad & x + 4y - 3 \leq 0, \\ & y - x \leq 0. \end{aligned}$$

The KKT conditions require that there exist $\boldsymbol{\mu} = (\mu_1, \mu_2) \succeq \mathbf{0}$ such that

$$\mu_1(x + 4y - 3) = 0, \tag{14.14a}$$

$$\mu_2(y - x) = 0, \tag{14.14b}$$

$$\begin{bmatrix} 2(x-2) & 4(y-1) \end{bmatrix} + \boldsymbol{\mu}^\mathsf{T} \begin{bmatrix} 1 & 4 \\ -1 & 1 \end{bmatrix} = \mathbf{0}^\mathsf{T}. \tag{14.14c}$$

To solve this, consider the possible cases:

(i) If $\mu_1 = \mu_2 = 0$, then (14.14c) implies that $x = 2$ and $y = 1$. But this is not feasible.

(ii) If $\mu_1 = 0$ and $\mu_2 \neq 0$, then (14.14b) implies that $x = y$. Substituting into (14.14c) gives $2x - 4 = \mu_2 = -4x + 4$. Hence $x = y = 4/3$ and $\mu_2 = -4/3$, which does not satisfy the nonnegativity condition for $\boldsymbol{\mu}$.

(iii) If $\mu_2 = 0$ and $\mu_1 \neq 0$, then (14.14a) implies that $x = 3 - 4y$, and substituting this into (14.14c) gives $2(3 - 4y - 2) = -\mu_1 = y - 1$. Hence $y = \frac{1}{3}$, $x = \frac{5}{3}$, and $\mu_1 = \frac{2}{3}$, which is feasible.

(iv) If μ_1 and μ_2 are both nonzero, then $x = y$ and $x = 3 - 4y$, so $x = y = \frac{3}{5}$ and $\boldsymbol{\mu} = (\frac{22}{25}, -\frac{48}{25})$. But this does not satisfy the nonnegativity condition for $\boldsymbol{\mu}$.

Therefore, the only case with a feasible point satisfying the KKT conditions is (iii), and the only candidate for a minimizer is $(\frac{5}{3}, \frac{1}{3})$.

Example 14.4.9. Consider the problem

$$\begin{aligned} \text{minimize} \quad & f(x,y) = x^2 + y^2 + z^2 \\ \text{subject to} \quad & h(\mathbf{x}) = x + y + z - 1 = 0, \\ & z \leq 0. \end{aligned}$$

The KKT conditions require that there exist $\mu > 0$ and $\lambda \in \mathbb{R}$ such that

$$\mu z = 0, \tag{14.15a}$$

$$\begin{bmatrix} 2x + \lambda, 2y + \lambda, 2z + \lambda + \mu \end{bmatrix}^\mathsf{T} = \mathbf{0}. \tag{14.15b}$$

Equation (14.15b) and a little algebra give $z + \frac{1}{2}\mu = x = y$. If $z \neq 0$, then $\mu = 0$, which gives $x = y = z$. Combining this with $h(\mathbf{x}) = 0$ gives $x = y = z = \frac{1}{3}$, which is not feasible (because $z > 0$).

If $\mu > 0$, then we have $z = 0$ and $2x = 2y = 1$, so $x = y = \frac{1}{2}$. This is the only feasible solution to the KKT constraints.

Example 14.4.10.* Consider the problem

$$\begin{aligned} \text{minimize} \quad & f(x, y) = x^2 + y^2 + xy - 3x \\ \text{subject to} \quad & x \geq 0, \\ & y \geq 0. \end{aligned}$$

The KKT conditions require that there exist $\boldsymbol{\mu} = (\mu_1, \mu_2)^\mathsf{T} \succeq \mathbf{0}$ such that

$$-\mu_1 x = 0, \tag{14.16a}$$

$$-\mu_2 y = 0, \tag{14.16b}$$

$$\begin{bmatrix} 2x + y - 3 & 2y + x \end{bmatrix} + \boldsymbol{\mu}^\mathsf{T} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = \mathbf{0}^\mathsf{T}. \tag{14.16c}$$

Again consider the different possible cases:

(i) If $\mu_1 = \mu_2 = 0$, then (14.16c) implies that $x = -2y$ and $3 = 2x + y$, so $y = -1$ and $x = 2$. But this is not feasible.

(ii) If $\mu_1 = 0$ and $\mu_2 \neq 0$, then (14.16b) implies that $y = 0$. Substituting into (14.16c) gives $2x - 3 = 0$ and $\mu_2 = x$. Hence $x = \mu_2 = \frac{3}{2}$ and $y = \mu_1 = 0$. This is feasible.

(iii) If $\mu_2 = 0$ and $\mu_1 \neq 0$, then (14.16a) implies that $x = 0$, and substituting this into (14.16c) gives $y = 0$ and $\mu_1 = y - 3 = -3$, which does not satisfy the nonnegativity condition.

(iv) Finally, if $\mu_1 \neq 0$ and $\mu_2 \neq 0$, then $x = y = 0$, and $2y + x - \mu_2 \neq 0$, so $\boldsymbol{\mu}$ does not satisfy the nonnegativity condition.

Therefore, the only case with a feasible point satisfying the KKT conditions is (ii), and the only candidate for a minimizer is $(\frac{3}{2}, 0)$.

Definition 14.4.11. *The* Lagrangian *of the constrained optimization problem* (14.1) *is the function* $\mathscr{L} : \Omega \times \mathbb{R}^\ell \times \mathbb{R}^m \to \mathbb{R}$ *given by*

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^\mathsf{T} H(\mathbf{x}) + \boldsymbol{\mu}^\mathsf{T} G(\mathbf{x}). \tag{14.17}$$

The Lagrangian derivative condition $D_{\mathbf{x}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$ gives the first KKT condition (Theorem 14.4.5(i)). In contrast, the Lagrangian derivative condition

$D_{\boldsymbol{\lambda}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$ merely satisfies the constraint $H(\mathbf{x}^*) = \mathbf{0}$. The Lagrangian derivative condition $D_{\boldsymbol{\mu}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \preceq \mathbf{0}$ satisfies the constraint $G(\mathbf{x}^*) \preceq \mathbf{0}$.

14.4.4 Lagrange as a Special Case of KKT

If an optimization problem has no inequality constraints or the optimizer does not lie on the boundary of the inequality constraints, then the KKT first-order conditions reduce to the Lagrange first-order conditions.

Consider first the case of no inequality constraints. Here the KKT conditions are clearly the same as the Lagrange condition, and there are no μ_i, so the first-order KKT conditions are the same as those for equality constraints.

In the case that we know that no optimizer lies on the boundary of any equality constraint, we may work in the open subset of Ω where these strict inequalities hold. Inside that open subset, we have only equality constraints, and the KKT conditions reduce to the Lagrange conditions.

Alternatively, we can work with the full KKT Lagrangian (14.17), but the fact that all inequalities are strict for all optimizers implies that every $g_i(\mathbf{x}) \neq 0$ for every optimizer $\mathbf{x}^*$ and every i. Therefore, complementary slackness guarantees that $\boldsymbol{\mu}^* = \mathbf{0}$. The only remaining KKT first-order condition is

$$\mathbf{0} = D_{\mathbf{x}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{0}) = D_{\mathbf{x}}f + \boldsymbol{\lambda}^* D_{\mathbf{x}}H + \mathbf{0}^{\mathsf{T}} D_{\mathbf{x}}G,$$

which is the same as the equality-constrained first-order Lagrange condition.

14.4.5 Proof of the KKT First-Order Conditions

We prove the KKT first-order conditions, Theorem 14.4.5.

Proof. Let $\widetilde{\mathscr{F}} = \widetilde{\mathscr{F}}(\mathbf{x}^*)$. Since $\widetilde{\mathscr{F}} \subset \mathscr{F}$, the point $\mathbf{x}^* \in \mathscr{F}$ is also a local minimizer of f on $\widetilde{\mathscr{F}}$, which involves only equality constraints. Thus, by Lagrange's first-order condition (Theorem 14.2.1), there exist $\boldsymbol{\lambda}^* = (\lambda_1^*, \ldots, \lambda_\ell^*) \in \mathbb{R}^\ell$ and constants $\{\mu_j^*\}_{j \in J(\mathbf{x}^*)}$ so that

$$Df(\mathbf{x}^*) + \sum_{i=1}^{\ell} \lambda_i^* Dh_i(\mathbf{x}^*) + \sum_{j \in J(\mathbf{x}^*)} \mu_j^* Dg_j(\mathbf{x}^*) = \mathbf{0}.$$

Setting $\mu_j^* = 0$ for all $j \notin J(\mathbf{x}^*)$ gives $\boldsymbol{\mu}^* = (\mu_1^*, \ldots, \mu_m^*) \in \mathbb{R}^m$ so that

$$Df(\mathbf{x}^*) + (\boldsymbol{\lambda}^*)^{\mathsf{T}} DH(\mathbf{x}^*) + (\boldsymbol{\mu}^*)^{\mathsf{T}} DG(\mathbf{x}^*) = \mathbf{0}.$$

This immediately gives $\mu_i^* g_i(\mathbf{x}) = 0$ for every $i \in \{1, \ldots, m\}$.

Now, to show that $\boldsymbol{\mu}^* \succeq \mathbf{0}$, suppose by way of contradiction that $\mu_k^* < 0$ for some $k \in J(\mathbf{x}^*)$. Let $\widetilde{\mathscr{F}}_k$ be the enlargement of the feasible set obtained by removing the constraint $g_k(\mathbf{x}^*) \leq 0$ from the definition of $\widetilde{\mathscr{F}}$, that is,

$$\widetilde{\mathscr{F}}_k = \{\mathbf{x} \in \Omega \mid H(\mathbf{x}) = \mathbf{0} \text{ and } g_j(\mathbf{x}) = 0 \; \forall j \in J(\mathbf{x}^*), j \neq k\},$$

and let $T_{\mathbf{x}^*}\widetilde{\mathscr{F}}_k$ be the tangent space of $\widetilde{\mathscr{F}}_k$ at $\mathbf{x}^*$, that is,

$$T_{\mathbf{x}^*}\widetilde{\mathscr{F}}_k = \{\mathbf{v} \in \mathbb{R}^n \mid DH(\mathbf{x}^*)\mathbf{v} = \mathbf{0} \text{ and } Dg_j(\mathbf{x})\mathbf{v} = 0 \; \forall j \in J(\mathbf{x}^*), j \neq k\}.$$

We claim that there exists a $\mathbf{v} \in T_{\mathbf{x}^*}\widetilde{\mathscr{F}_k}$ such that $Dg_k(\mathbf{x}^*)\mathbf{v} \neq 0$. If not, then $Dg_k(\mathbf{x}^*)\mathbf{v} = 0$ for all $\mathbf{v} \in T_{\mathbf{x}^*}\widetilde{\mathscr{F}_k}$. Thus $Dg_k(\mathbf{x}^*)^\mathsf{T} \in T_{\mathbf{x}^*}\widetilde{\mathscr{F}_k}^\perp$, that is, it is in the normal space to $\widetilde{\mathscr{F}_k}$ at $\mathbf{x}^*$. Hence,

$$Dg_k(\mathbf{x}^*) \in \operatorname{span}\left(\{Dh_i(\mathbf{x}^*)\}_{i=1}^{\ell} \cup \{Dg_j(\mathbf{x}^*)\}_{j \in J(\mathbf{x}^*) \setminus \{k\}}\right),$$

contradicting the assumption that $\mathbf{x}^*$ is a regular point. Therefore $Dg_k(\mathbf{x}^*)\mathbf{v} \neq 0$ for some $\mathbf{v} \in T_{\mathbf{x}^*}\widetilde{\mathscr{F}_k}$.

Changing the sign of $\mathbf{v}$, if necessary, we may assume $Dg_k(\mathbf{x}^*)\mathbf{v} < 0$. Now write the Lagrange condition as

$$Df(\mathbf{x}^*) = -\sum_{i=1}^{\ell} \lambda_i^* Dh_i(\mathbf{x}^*) - \sum_{j \neq k} \mu_j^* Dg_j(\mathbf{x}^*) - \mu_k^* Dg_k(\mathbf{x}^*).$$

Applying this to $\mathbf{v}$, and using the fact that $Dh_i(\mathbf{x}^*)\mathbf{v} = 0$ for all i and $Dg_j(\mathbf{x}^*)\mathbf{v} = 0$ for all $j \in J(\mathbf{x}^*) \setminus \{k\}$ gives

$$Df(\mathbf{x}^*)\mathbf{v} = -\mu_k^* Dg_k(\mathbf{x}^*)\mathbf{v} < 0.$$

Now, since $\mathbf{v} \in T_{\mathbf{x}^*}\widetilde{\mathscr{F}_k}$, Proposition 14.1.9 guarantees there is a differentiable curve $\gamma : (-b, b) \to \widetilde{\mathscr{F}_k}$ with $\gamma(0) = \mathbf{x}^*$ and $\gamma'(0) = \mathbf{v}$. We then have

$$\frac{d}{dt} f(\gamma(t))\bigg|_{t=0} = Df(\mathbf{x}^*)\mathbf{v} < 0,$$

which means that the function $f(\gamma(t))$ is strictly decreasing at $t = 0$. Moreover,

$$\frac{d}{dt} g_k(\gamma(t))\bigg|_{t=0} = Dg_k(\mathbf{x}^*)\mathbf{v} < 0,$$

and so the function $g_k(\gamma(t))$ is also strictly decreasing at $t = 0$.

Thus, there exists $\delta > 0$ such that $f(\gamma(t)) < f(\gamma(0)) = f(\mathbf{x}^*)$ and $g_k(\gamma(t)) < 0$ whenever $t \in (0, \delta)$. Thus, we have that $\gamma(t) \in \mathscr{F}$ and $f(\gamma(t)) < f(\mathbf{x}^*)$ for all such $t \in (0, \delta)$. This contradicts the statement that $\mathbf{x}^*$ is a local minimizer for f. ∎

14.5 *Second-Order KKT

Just as for the unconstrained and equality-constrained cases, there are both KKT SONCs and KKT SOSCs.

14.5.1 Second-Order Necessary Condition

Theorem 14.5.1 (KKT Second-Order Necessary Condition). *Assume that $\mathbf{x}^* \in \mathscr{F}$ is a regular point and a local minimizer of the optimization problem* (14.1). *Let $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ and $\boldsymbol{\mu}^* \in \mathbb{R}^m$ be the vectors satisfying the first-order KKT conditions, and let*

$$\widetilde{T}(\mathbf{x}^*) = \{\mathbf{v} \in \mathbb{R}^n \mid DH(\mathbf{x}^*)\mathbf{v} = \mathbf{0}, Dg_j(\mathbf{x}^*)\mathbf{v} = 0 \ \forall j \in J(\mathbf{x}^*)\}.$$

If f, H, G are all C^2 in a neighborhood of $\mathbf{x}^*$, then for every $\mathbf{v} \in \widetilde{T}(\mathbf{x}^*)$ we have

$$\mathbf{v}^{\mathsf{T}} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)\mathbf{v} \geq 0.$$

Proof. Since $\mathbf{x}^*$ is a local minimizer in the feasible set $\mathscr{F}$ in (14.2), it is also a local minimizer in $\widetilde{\mathscr{F}} = \{\mathbf{x} \in \mathscr{F} \mid H(\mathbf{x}) = \mathbf{0}, g_j(\mathbf{x}) = 0 \, \forall j \in J(\mathbf{x}^*)\}$. Since $\mathbf{x}^*$ is a regular point of $\widetilde{\mathscr{F}}$, and since $\widetilde{T}(\mathbf{x}^*) = T_{\mathbf{x}^*}\widetilde{\mathscr{F}}$, the second-order necessary Lagrange condition immediately gives the result. □

Theorem 14.5.2 (KKT Second-Order Sufficient Condition). *Consider the optimization problem (14.1), where f, H, and G are all C^2. Assume that $\mathbf{x}^* \in \mathscr{F}$ is a regular point and $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ and $\boldsymbol{\mu}^* \in \mathbb{R}^m$ are vectors satisfying the first-order KKT conditions. Let*

$$\widehat{J}(\mathbf{x}^*, \boldsymbol{\mu}^*) = \{i \in J(\mathbf{x}^*) \mid \mu_i^* > 0\},$$

and let

$$\widehat{T}(\mathbf{x}^*) = \{\mathbf{v} \in \mathbb{R}^n \mid DH(\mathbf{x}^*)\mathbf{v} = \mathbf{0}, Dg_j(\mathbf{x}^*)\mathbf{v} = 0 \, \forall j \in \widehat{J}(\mathbf{x}^*, \boldsymbol{\mu}^*)\}.$$

If for every nonzero $\mathbf{v} \in \widehat{T}(\mathbf{x}^*)$ we have

$$\mathbf{v}^{\mathsf{T}} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)\mathbf{v} > 0,$$

then $\mathbf{x}^*$ is a strict local minimizer for this problem.

Proof. As in the proof of the Lagrange SOSCs, if $\mathbf{x}^*$ is not a strict local minimizer, we can take a sequence $\mathbf{x}_k \to \mathbf{x}^*$ with $f(\mathbf{x}_k) \leq f(\mathbf{x}^*)$ and construct $\boldsymbol{\varepsilon}_k = \mathbf{x}_k - \mathbf{x}^*$ and $\mathbf{s}_k = \boldsymbol{\varepsilon}_k / \|\boldsymbol{\varepsilon}_k\|$ with $\mathbf{s}_k \to \mathbf{s}^*$ for some $\mathbf{s}^*$.

As before, we have $DH(\mathbf{x}^*)\mathbf{s}^* = 0$. We now show that for every $j \in \widehat{J}(\mathbf{x}^*)$ we have $Dg_j(\mathbf{x}^*)\mathbf{s}^* = 0$. To see this, note that $g_j(\mathbf{x}^*) = 0$ and for every $k \in \mathbb{Z}^+$ we have $g_j(\mathbf{x}^* + \boldsymbol{\varepsilon}_k) = g_j(\mathbf{x}_k) \leq 0$, since $\mathbf{x}_k$ is feasible. This gives

$$Dg_j(\mathbf{x}^*)\mathbf{s}^* = \frac{g_j(\mathbf{x}^* + \boldsymbol{\varepsilon}_k) - g_j(\mathbf{x}^*)}{\|\boldsymbol{\varepsilon}_k\|} = \lim_{k \to \infty} \frac{g_j(\mathbf{x}^* + \boldsymbol{\varepsilon}_k)}{\|\boldsymbol{\varepsilon}_k\|} \leq 0.$$

On the other hand, since the first-order KKT conditions hold, we have

$$Df(\mathbf{x}^*)\mathbf{s}^* = -\boldsymbol{\lambda}^{\mathsf{T}} DH(\mathbf{x}^*)\mathbf{s}^* - (\boldsymbol{\mu}^*)^{\mathsf{T}} DG(\mathbf{x}^*)\mathbf{s}^* = -(\boldsymbol{\mu}^*)^{\mathsf{T}} DG(\mathbf{x}^*)\mathbf{s}^*.$$

But each nonzero μ_j^* is strictly positive, and $Dg_j(\mathbf{x}^*)\mathbf{s}^* \leq 0$, so $Df(\mathbf{x}^*)\mathbf{s}^* \geq 0$. However,

$$Df(\mathbf{x}^*)\mathbf{s}^* = \lim_{k = \to \infty} \frac{f(\mathbf{x}^* + \boldsymbol{\varepsilon}_k) - f(\mathbf{x}^*)}{\|\boldsymbol{\varepsilon}_k\|} \leq 0.$$

Therefore, $Df(\mathbf{x}^*)\mathbf{s}^* = 0$ and $Dg_j(\mathbf{x}^*)\mathbf{s}^* = 0$ for every $j \in \widehat{J}$.

The rest of the proof is essentially the same as for Lagrange, but with $\widehat{T}(\mathbf{x}^*)$ and $\widehat{J}(\mathbf{x}^*, \boldsymbol{\mu}^*)$ substituted for $\widetilde{T}(\mathbf{x}^*)$ and $J(\mathbf{x}^*)$. The details are Exercise 14.28. □

14.5.2 Examples

Example 14.5.3. In Example 14.4.8 the only active constraint at the point $\mathbf{x}^* = \begin{bmatrix}\frac{5}{3} & \frac{1}{3}\end{bmatrix}^\mathsf{T}$ and $\mu^* = \begin{bmatrix}\frac{2}{3} & 0\end{bmatrix}^\mathsf{T}$ is $g_1(x, y) = x + 4y - 3$, with $\mu_1 > 0$, so

$$\widehat{T}(\mathbf{x}^*, \mu^*) = \mathscr{N}\left(Dg_1(\mathbf{x}^*)\right) = \{\mathbf{v} \mid \begin{bmatrix}1 & 4\end{bmatrix}\mathbf{v} = 0\} = \operatorname{span}(\begin{bmatrix}-4 & 1\end{bmatrix}^\mathsf{T}).$$

It suffices to check that $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \mu^*)\mathbf{v} > 0$ for any such $\mathbf{v} \neq \mathbf{0}$. But

$$D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}, \mu) = \begin{bmatrix}2 & 0\\ 0 & 4\end{bmatrix} > 0,$$

so the condition holds. This guarantees that the one feasible candidate is indeed a minimizer.

Example 14.5.4. In Example 14.4.10 we have

$$D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \mu^*) = \begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix} > 0,$$

so the KKT SOSC guarantees that all feasible first-order candidates are indeed minimizers.

Example 14.5.5. Consider the problem of finding $\mathbf{x} \in \mathbb{R}^2$ to

$$\begin{aligned} &\text{minimize} && \|\mathbf{x}\|_2^2\\ &\text{subject to} && 2 - x_1 x_2 + 3x_1 \leq 0. \end{aligned}$$

The Lagrangian for this problem is

$$\mathscr{L}(x_1, x_2, \mu) = x_1^2 + x_2^2 + \mu(2 - x_1 x_2 + 3x_1)$$

with $D_{\mathbf{x}} \mathscr{L}(x_1, x_2, \mu) = \begin{bmatrix}2x_1 - \mu(x_2 - 3) & 2x_2 - \mu x_1\end{bmatrix}$. It is straightforward to check that $\mathbf{x} = (2, 4)$ where $\mu = 4$ is a solution of the first-order KKT conditions, and the constraint $h(\mathbf{x}) = 2 - x_1 x_2 + 3x_1$ is active there with $\mu > 0$. We have

$$\widehat{T}(\mathbf{x}) = \mathscr{N}\left(Dh(\mathbf{x})\right) = \operatorname{span}\left(\begin{bmatrix}2\\ -1\end{bmatrix}\right) \quad \text{and} \quad D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}) = \begin{bmatrix}2 & -4\\ -4 & 2\end{bmatrix}.$$

Hence, every nonzero $\mathbf{v} \in \widehat{T}(\mathbf{x}) = \operatorname{span}((2, -1))$, which implies that $\mathbf{v} = \alpha(2, -1)$ and

$$\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}\mathbf{v} = \alpha^2 \begin{bmatrix}2 & -1\end{bmatrix} \begin{bmatrix}2 & -4\\ -4 & 2\end{bmatrix} \begin{bmatrix}2\\ -1\end{bmatrix} = 26\alpha^2 > 0.$$

So the KKT SOSC implies that this point is a local minimizer.

14.5.3 Second-Order Lagrange as a Special Case of KKT

The second-order Lagrange conditions are a special case of the second-order KKT conditions, just as the first-order Lagrange conditions are a special case of the first-order KKT conditions. To see this, note that in the case that none of the inequality constraints are active for any optimizer, then

$$\widetilde{T}(\mathbf{x}^*) = \widehat{T}(\mathbf{x}^*) = T_{\mathbf{x}^*}\mathscr{F}.$$

Therefore, the KKT second-order conditions are identical to their equality-constrained Lagrange counterparts.

14.6 Removing Affine Constraints

Perhaps the most straightforward way to deal with constraints is to rewrite the problem as an unconstrained problem. This is always possible when all of the constraints are affine equalities, because we can use the equality constraints to solve for some of the variables in terms of the others, and then substitute those relations into the objective, reducing the dimension of the problem and making the constraints unnecessary. This method is easy to apply on problems like that of Example 14.3.7. Here we give several additional examples and then discuss some general considerations for how to do this most effectively.

Unexample 14.6.1. While affine constraints can always be removed, nonaffine constraints are trickier. In some situations removing a nonaffine constraint carelessly can be disastrous. Consider the problem

$$\begin{aligned} &\text{minimize} && x^2 + y^2 \\ &\text{subject to} && (x-1)^3 - y^2 = 0. \end{aligned}$$

It is straightforward to check that the minimizer is $(1, 0)$ with minimal value 1. Making the naïve substitution $y^2 = (x-1)^3$ turns this into the unconstrained problem of minimizing $x^2 + (x-1)^3$, which is unbounded as $x \to -\infty$. The error here is that the constraint $(x-1)^3 - y^2 = 0$ also implicitly imposes the additional constraint $(x-1)^3 \geq 0$, which was not accounted for in the naïve unconstrained version of the problem. This is not to say that one can never remove nonaffine constraints, but doing so requires extra care.

14.6.1 Example: Line Fitting with Ordinary Least Squares

Given a set of data points $\{(x_i, y_i)\}_{i=1}^d$, we wish to find the line that best fits the data. Assuming that the data is only susceptible to noise (error) in the y direction (see Figure 14.7(a)) corresponds to the model

$$y_i + \varepsilon_i = mx_i + b, \qquad i = 1, \ldots, d, \tag{14.18}$$

or

$$\mathbf{y} + \boldsymbol{\varepsilon} = m\mathbf{x} + b\mathbb{1},$$

where $\boldsymbol{\varepsilon}$ is the error and $\mathbb{1} = \sum_{i=1}^{d} \mathbf{e}_i$ is the vector of all ones. To minimize the 2-norm of the error $\boldsymbol{\varepsilon}$, we can formulate this as the constrained optimization problem

$$\begin{array}{ll} \underset{\boldsymbol{\varepsilon}, m, b}{\text{minimize}} & \|\boldsymbol{\varepsilon}\|_2^2 \\ \text{subject to} & \mathbf{y} + \boldsymbol{\varepsilon} = m\mathbf{x} + b\mathbb{1}. \end{array}$$

Solving for $\boldsymbol{\varepsilon}$ in terms of m, b, $\mathbf{x}$, and $\mathbf{y}$ gives the unconstrained optimization problem of choosing m and b to

$$\underset{m,b}{\text{minimize}} \quad \|m\mathbf{x} + b\mathbb{1} - \mathbf{y}\|_2^2 = \sum_{i=1}^{N} |mx_i + b - y_i|^2.$$

We have removed the equality constraints and reduced the dimension of the problem from $N + 2$ to 2, thereby simplifying it substantially.

This particular problem can be written in a nicer form by taking

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \qquad \text{and} \qquad \mathbf{z} = \begin{bmatrix} m \\ b \end{bmatrix}$$

so that the problem becomes that of finding $\mathbf{z}$ to

$$\underset{\mathbf{z}}{\text{minimize}} \quad \|A\mathbf{z} - \mathbf{y}\|_2^2.$$

14.6.2 Total Least Squares

Ordinary linear regression measures error in the y variable only. It could also happen that the data $\{(x_i, y_i)\}_{i=1}^{d}$ are susceptible to noise in both the x and y values, corresponding to the model

$$y_i + \varepsilon_i = m(x_i + \delta_i) + b, \qquad i = 1, \ldots, d, \tag{14.19}$$

or

$$\mathbf{y} + \boldsymbol{\varepsilon} = m(\mathbf{x} + \boldsymbol{\delta}) + b\mathbb{1}.$$

For an illustration of this, see Figure 14.7(*b*).

In this setting, the goal is to solve the following equality-constrained optimization problem[54] of choosing m, b, $\boldsymbol{\delta}$, and $\boldsymbol{\varepsilon}$ to

$$\begin{array}{ll} \underset{m,b,\boldsymbol{\delta},\boldsymbol{\varepsilon}}{\text{minimize}} & \sum_{i=1}^{N} (\delta_i^2 + \varepsilon_i^2) \\ \text{subject to} & m\delta_i + mx_i + b - \varepsilon_i = y_i, \quad i = 1, \ldots, d, \end{array}$$

or, in vector notation,

$$\begin{array}{ll} \underset{m,b,\boldsymbol{\delta},\boldsymbol{\varepsilon}}{\text{minimize}} & \|\boldsymbol{\delta}\|_2^2 + \|\boldsymbol{\varepsilon}\|_2^2 \\ \text{subject to} & m\boldsymbol{\delta} - m\mathbf{x} - b\mathbb{1} + \boldsymbol{\varepsilon} = \mathbf{y}. \end{array}$$

[54] Note that we are not actually minimizing the sum of the distances to the line, but rather the sum of the *squares* of the distances. So while we are minimizing $\|\boldsymbol{\delta}\|_2^2 + \|\boldsymbol{\varepsilon}\|_2^2$, a related, but not identical, problem would be to minimize $\|\boldsymbol{\delta}\|_2 + \|\boldsymbol{\varepsilon}\|_2$. This related problem is messier to solve and the answer is usually not very different from summing the squares of the distances.

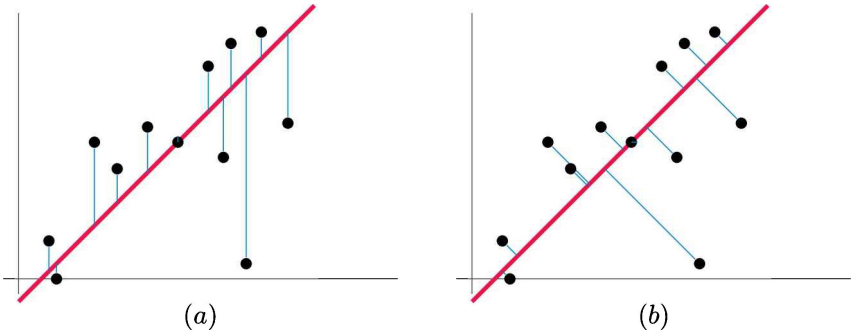


Figure 14.7. *The residuals for* (a) *ordinary least squares include only error in the y-axis. The residuals for* (b) *total least squares are the (squares of the) distances measured perpendicular to the line.*

The constraint is affine and we can rewrite this by solving for $\boldsymbol{\varepsilon}$ in terms of m, b, $\mathbf{x}$, $\mathbf{y}$, and $\boldsymbol{\delta}$ to get the unconstrained quadratic optimization problem of finding m, b, and $\boldsymbol{\delta}$ to

$$\underset{m,b,\boldsymbol{\delta}}{\text{minimize}} \quad \|\boldsymbol{\delta}\|_2^2 + \|\mathbf{y} - m\boldsymbol{\delta} + m\mathbf{x} + b\mathbb{1}\|_2^2.$$

This has turned the problem from a constrained problem to an unconstrained problem and has also reduced the dimension from $2N + 2$ down to $N + 2$, a substantial simplification.

14.6.3 Generalization: Removing Affine Constraints

The previous examples are special cases of the generic equality-constrained problem

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & C\mathbf{x} = \mathbf{d}, \end{aligned} \tag{14.20}$$

where $C \in M_{\ell\times n}(\mathbb{R})$ and $\mathbf{d} \in \mathbb{R}^\ell$. If C has rank $r \leq \ell$, then the feasible set $\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid C\mathbf{x} = \mathbf{d}\}$ has dimension $n - r$ as an affine space. Here we show how to remove the constraints by finding a bijective map $\phi : \mathbb{R}^{n-r} \to \mathscr{F}$ and reducing the constrained problem into the unconstrained problem of choosing $\mathbf{y} \in \mathbb{R}^{n-r}$ to minimize $f(\phi(\mathbf{y}))$.

We can construct a bijection ϕ using the reduced QR decomposition with pivoting to write $C = QRP^\mathsf{T}$, where Q is an $\ell \times r$ orthonormal matrix, P is an $n \times n$ permutation matrix (it is orthonormal with every entry equal to either 1 or 0), and R is an $r \times n$ upper triangular matrix of the form $R = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$, where R_1 is an $r \times r$ upper triangular, invertible matrix and R_2 is an $r \times (n - r)$ matrix. Given such a decomposition, define an affine map $\phi : \mathbb{R}^{n-r} \to \mathbb{R}^n$ by

$$\phi(\mathbf{z}) = P\begin{bmatrix} R_1^{-1}(Q^\mathsf{T}\mathbf{d} - R_2\mathbf{z}) & \mathbf{z} \end{bmatrix}^\mathsf{T}. \tag{14.21}$$

Remark 14.6.2. The formula (14.21) may feel like it comes out of nowhere, but it should feel more natural after reading the proof of Proposition 14.6.4. Pay particular attention to the nice form for $\psi = \phi^{-1}$ in that proof.

Nota Bene 14.6.3. Given a matrix $C \in M_{\ell\times n}(\mathbb{R})$ with rank $r \le d$, naïvely performing QR decomposition on C does not necessarily yield an R in the desired block form $R = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$ with R_1 upper triangular and invertible of rank r. But with pivoting, this can always be done. That is, one can always decompose C as $C = QRP$, where R has the desired form, Q has orthonormal columns, and P is a permutation matrix.

Proposition 14.6.4. *Assuming $\mathbf{d} \in \mathscr{R}(C)$, the affine map $\phi : \mathbb{R}^{n-r} \to \mathbb{R}^n$ in (14.21) is a bijection from $\mathbb{R}^{n-r}$ to the feasible set $\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid C\mathbf{x} = \mathbf{d}\}$.*

Proof. To show that ϕ is well defined, let $\mathbf{z} \in \mathbb{R}^{n-r}$. Thus,

$$\begin{aligned} C\phi(\mathbf{z}) &= Q\begin{bmatrix} R_1 & R_2 \end{bmatrix} P^{\mathsf{T}}\phi(\mathbf{z}) \\ &= Q\begin{bmatrix} R_1 & R_2 \end{bmatrix} P^{\mathsf{T}} P \begin{bmatrix} R_1^{-1}(Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z}) & \mathbf{z} \end{bmatrix}^{\mathsf{T}} \\ &= Q\begin{bmatrix} R_1 & R_2 \end{bmatrix} \begin{bmatrix} R_1^{-1}(Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z}) & \mathbf{z} \end{bmatrix}^{\mathsf{T}} \\ &= Q((Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z}) + R_2\mathbf{z}) = QQ^{-1}\mathbf{d}. \end{aligned}$$

Since $\mathbf{d} \in \mathscr{R}(C)$, it follows that $QQ^{\mathsf{T}}\mathbf{d} = \mathbf{d}$.[55] Thus, $\phi(\mathbf{z}) \in \mathscr{F}$. Define a map $\psi : \mathscr{F} \to \mathbb{R}^{n-r}$ as follows: for any $\mathbf{x} \in \mathbb{R}^n$ with $C\mathbf{x} = \mathbf{d}$, write $P^{\mathsf{T}}\mathbf{x} = \begin{bmatrix} \mathbf{y} & \mathbf{z} \end{bmatrix}^{\mathsf{T}}$ with $\mathbf{y} \in \mathbb{R}^r$ and $\mathbf{z} \in \mathbb{R}^{n-r}$, and let $\psi(\mathbf{x}) = \mathbf{z}$.

To prove the proposition, it suffices to prove that $\psi = \phi^{-1}$, that is, $\phi(\psi(\mathbf{x})) = \mathbf{x}$ for all $\mathbf{x} \in \mathscr{F}$ and $\psi(\phi(\mathbf{z})) = \mathbf{z}$ for all $\mathbf{z} \in \mathbb{R}^{n-r}$. For any $\mathbf{z} \in \mathbb{R}^{n-r}$ we have $\phi(\mathbf{z}) = P\begin{bmatrix} R_1^{-1}(Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z}) & \mathbf{z} \end{bmatrix}^{\mathsf{T}}$, and thus $\psi(\phi(\mathbf{z})) = \mathbf{z}$. Conversely, if $\mathbf{x} \in \mathscr{F}$ write $P^{\mathsf{T}}\mathbf{x} = \begin{bmatrix} \mathbf{y} & \mathbf{z} \end{bmatrix}^{\mathsf{T}}$ so that

$$\mathbf{d} = C\mathbf{x} = QRP^{\mathsf{T}}\mathbf{x} = Q\begin{bmatrix} R_1 & R_2 \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = Q(R_1\mathbf{y} + R_2\mathbf{z}).$$

Thus $Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z} = R_1\mathbf{y}$ and $\mathbf{y} = R_1^{-1}(Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z})$, giving $\mathbf{x} = \phi(\mathbf{z}) = \phi(\psi(\mathbf{x}))$. ∎

Remark 14.6.5. To solve the original problem, we need only solve the unconstrained problem

$$\underset{\mathbf{z}\in\mathbb{R}^{n-r}}{\text{minimize}}\, f(\phi(\mathbf{z})) = f(P\begin{bmatrix} R_1^{-1}(Q^{\mathsf{T}}\mathbf{d} - R_2\mathbf{z}) & \mathbf{z} \end{bmatrix}^{\mathsf{T}}),$$

for the minimizer $\mathbf{z}^*$, and let $\mathbf{x}^* = \phi(\mathbf{z}^*)$.

Remark 14.6.6. Geometrically speaking, the affine space $\mathscr{F}$ is a translate of the $(n-r)$-dimensional vector subspace $W = \{\mathbf{x} \in \mathbb{R}^n \mid C\mathbf{x} = \mathbf{0}\}$, and every subspace of dimension k is isomorphic to $\mathbb{R}^k$. The map ϕ is the composition of a translation $\mathscr{F} \to W$ with an isomorphism $W \cong \mathbb{R}^{n-r}$. See Figure 14.8 for an illustration.

[55] Beware that although $Q^{\mathsf{T}}Q = I$, the matrix Q is not square, and $QQ^{\mathsf{T}} \neq I$.

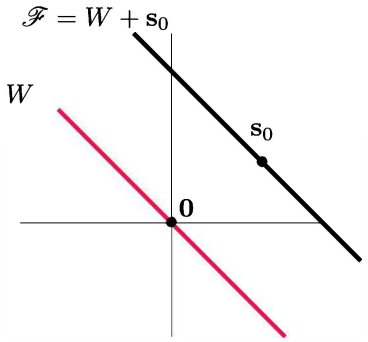


Figure 14.8. *Any affine set $\mathscr{F}$ is the translate of a vector subspace W, so we can write $\mathscr{F} = W + \mathbf{s}_0$ for any point $\mathbf{s}_0 \in \mathscr{F}$.*

Example 14.6.7. Given $A \in M_{m\times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$, the *least squares with equality constraints* (LSE) problem is given as

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \|A\mathbf{x} - \mathbf{b}\|_2^2 \\ \text{subject to} & C\mathbf{x} = \mathbf{d}, \end{array} \tag{14.22}$$

where $C \in M_{\ell\times n}(\mathbb{R})$ and $\mathbf{d} \in \mathbb{R}^\ell$.

Using the notation $\mathbf{x} = P\begin{bmatrix}\mathbf{y} & \mathbf{z}\end{bmatrix}^\mathsf{T}$ of the previous section, we rewrite the objective function as

$$A\mathbf{x} - \mathbf{b} = AP\begin{bmatrix}\mathbf{y}\\ \mathbf{z}\end{bmatrix} - \mathbf{b}.$$

By writing $AP = \begin{bmatrix}A_1 & A_2\end{bmatrix}$, we have

$$\begin{aligned} A\mathbf{x} - \mathbf{b} = AP\begin{bmatrix}\mathbf{y}\\ \mathbf{z}\end{bmatrix} - \mathbf{b} &= A_1R_1^{-1}(Q^\mathsf{T}\mathbf{d} - R_2\mathbf{z}) + A_2\mathbf{z} - \mathbf{b} \\ &= (A_2 - A_1R_1^{-1}R_2)\mathbf{z} + A_1R_1^{-1}Q^\mathsf{T}\mathbf{d} - \mathbf{b}. \end{aligned}$$

So the LSE problem (14.22) is equivalent to the following (unconstrained) ordinary least squares problem:

$$\underset{\mathbf{z}\in\mathbb{R}^{n-r}}{\text{minimize}} \|\widehat{A}\mathbf{z} - \widehat{\mathbf{b}}\|_2^2,$$

where

$$\begin{aligned} \widehat{A} &= A_2 - A_1R_1^{-1}R_2 \in M_{\ell\times(n-r)}(\mathbb{R}), \\ \widehat{\mathbf{b}} &= \mathbf{b} - A_1R_1^{-1}Q^\mathsf{T}\mathbf{d} \in \mathbb{R}^\ell. \end{aligned}$$

Remark 14.6.8. Although making an affine transformation to remove equality constraints can both reduce the dimension and allow us to use the numerical methods of Chapter 12, there are situations where it might be better not to make this

transformation. For example, in the LSE problem (14.22) if the matrix A is large but sparse, a transformation to remove equality constraints might destroy the sparse structure of A. This could result in an increased computational complexity that would completely negate any benefit that might have been gained by removing the equality constraints.

14.6.4 Application: Portfolio Optimization

Consider the problem of forming an investment portfolio of several assets. Suppose that there are n possible assets to choose from, and assume their rates of return are random variables $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ with expected values $\bar{r}_i = \mathbb{E}[r_i]$, $i = 1, 2, \ldots, n$. Assume further that the covariance matrix $\Sigma = \mathbb{E}[(\mathbf{r} - \bar{\mathbf{r}})(\mathbf{r} - \bar{\mathbf{r}})^T]$ is positive definite with components $\Sigma_{ij} = \mathbb{E}[(r_i - \bar{r}_i)(r_j - \bar{r}_j)]$.

The portfolio consists of a mix of these n assets. We characterize the mix by choosing n corresponding weights $w_1, w_2, \ldots, w_n$ which sum to one and denote the proportion of the portfolio holdings for each of the assets. The rate of return for the entire portfolio is given by

$$\rho = w_1 r_1 + w_2 r_2 + \cdots + w_n r_n.$$

The expected rate of return is given by

$$\bar{\rho} = \mathbb{E}[r] = w_1 \bar{r}_1 + w_2 \bar{r}_2 + \cdots + w_n \bar{r}_n.$$

The investor wants to maximize the rate of return, but that's not the same as maximizing the expected rate of return. Investors are usually leery of risk, which corresponds essentially to variance in the rate of return. Two different portfolios may have the same expected rate of return but wildly different variances. For a given expected rate of return $\bar{\rho}$, we wish to minimize the variance of the portfolio.

The variance is given by

$$\begin{aligned}
\operatorname{Var} \rho &= \mathbb{E}[(\rho - \bar{\rho})^2] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^n w_i r_i - \sum_{i=1}^n w_i \bar{r}_i\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^n w_i (r_i - \bar{r}_i)\right)^2\right] \\
&= \sum_{i=1}^n \sum_{j=1}^n w_i w_j \mathbb{E}[(r_i - \bar{r}_i)(r_j - \bar{r}_j)] \\
&= \sum_{i=1}^n \sum_{j=1}^n w_i w_j \Sigma_{ij}.
\end{aligned}$$

So given $\bar{r}_1, \ldots, \bar{r}_n$ and all the Σ_{ij} we have the optimization problem of choosing $w_1, \ldots, w_n$ so as to

$$\begin{array}{ll}
\text{minimize} & \sum_{i=1}^n \sum_{j=1}^n w_i w_j \Sigma_{ij} \\
\text{subject to} & w_1 + w_2 + \cdots + w_n = 1, \\
& w_1 \bar{r}_1 + \cdots + w_n \bar{r}_n = \bar{r}.
\end{array}$$

If we let $\mathbf{w} = (w_1, \ldots, w_n)$, $\mathbb{1}$ be the vector of all ones, and $\bar{\mathbf{r}} = (\bar{r}_1, \ldots, \bar{r}_n)$, then this problem could also be rewritten as the problem of choosing $\mathbf{w}$ to

$$\begin{aligned} \text{minimize} \quad & \mathbf{w}^\mathsf{T} \Sigma \mathbf{w} \\ \text{subject to} \quad & \mathbf{w}^\mathsf{T} \mathbb{1} = 1, \\ & \mathbf{w}^\mathsf{T} \bar{\mathbf{r}} = \bar{\rho}. \end{aligned} \tag{14.23}$$

Each choice of expected value $\bar{\rho}$ has a corresponding minimum possible variance $\mathrm{Var}(\rho)$. Plotting these optimal pairs on a graph with risk $\mathrm{Var}(\rho)$ on the x-axis and expected return $\bar{\rho}$ on the y-axis gives a curve called the *efficient frontier*; see Figure 14.9.

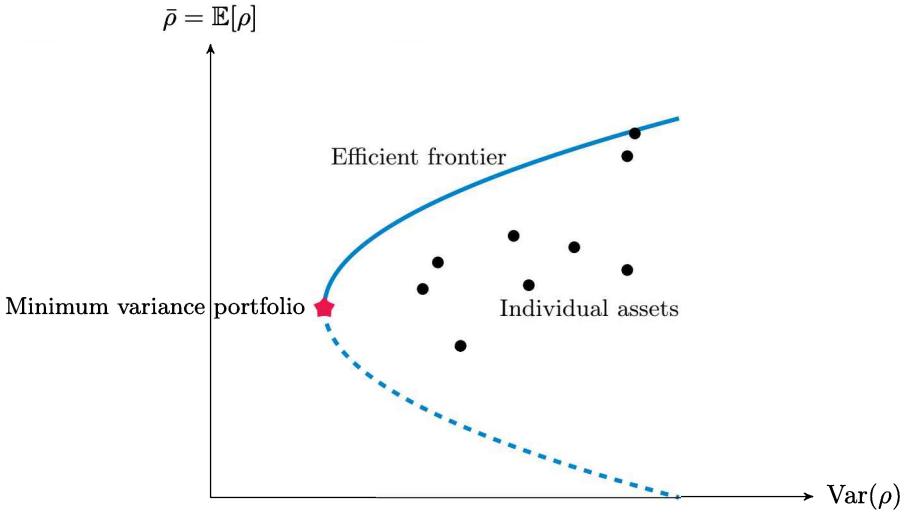


Figure 14.9. *Graphical representation of the Markowitz model. The solid blue line represents the efficient frontier, consisting of portfolios that minimize risk (variance) for a given expected return $\bar{\rho}$. The red star represents the portfolio with the overall minimum variance, and the black dots represent individual assets.*

This concept was developed by Harry Markowitz in the 1950s in his modern portfolio theory, which eventually resulted in the capital asset pricing model (CAPM). The CAPM is widely used by portfolio managers across the world. Markowitz was awarded the Nobel Prize in economics in 1990 for his work on portfolio theory.

Using the previous techniques, we can rewrite this problem (see Exercise 14.32) as the unconstrained quadratic optimization problem of choosing $\mathbf{y} \in \mathbb{R}^{n-2}$ to

$$\text{minimize} \quad \mathbf{y}^\mathsf{T} A \mathbf{y} - \mathbf{v}^\mathsf{T} \mathbf{y} + p \tag{14.24}$$

for some choice of $p \in \mathbb{R}$, and $\mathbf{v} \in \mathbb{R}^{n-2}$, and $A > 0$. In this form, we can use the techniques of Chapter 12 to find solutions to this problem.

The discussion above assumes that you can short sell an asset, which means that you essentially borrow the asset from someone and sell it to someone else. At some later date you buy it back again and return it to the person you borrowed it from.

You would do this when you think the price of an asset is going to go down. Thus, it is possible to have negative values of w_i, because you borrowed the asset, sold it, and then invested that money in the other assets in the portfolio (thus keeping the total sum of weights to one).

Prohibiting short selling corresponds to an additional requirement that $w_i \geq 0$ for each i. In this case we cannot rewrite the problem to remove all the constraints, but it is still solvable using the techniques in this text, especially those in the following chapter.

14.7 Numerical Methods for Constrained Optimization

In this section we describe a few common methods for computing minimizers of a differentiable function f on a closed, convex feasible set $\mathscr{F}$. That is, we are interested in computing numerical solutions to problems of the form

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \mathbf{x} \in \mathscr{F}, \end{aligned} \tag{14.25}$$

where $\mathscr{F}$ is closed and convex and f is continuously differentiable on an open set containing $\mathscr{F}$.

14.7.1 Conditional Gradient

The *conditional gradient method* is an iterative method for numerically solving problems of the form (14.25). Given an initial feasible point $\mathbf{x}_0 \in \mathscr{F}$ it proceeds by choosing

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k(\bar{\mathbf{x}}_k - \mathbf{x}_k), \tag{14.26}$$

where

$$\bar{\mathbf{x}}_k = \operatorname*{argmin}_{\mathbf{x} \in \mathscr{F}} Df(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k). \tag{14.27}$$

Of course, since $DF(\mathbf{x}_k)$ is a linear transformation, this is the same as minimizing $Df(\mathbf{x}_k)\mathbf{x}$ over $\mathscr{F}$.

Since $\mathscr{F}$ is convex, the point $\mathbf{x}_{k+1}$ must also lie in $\mathscr{F}$ provided $\alpha_k \in [0, 1]$. Therefore, this algorithm requires the constraint of $0 \leq \alpha_k \leq 1$ on the learning rate α_k. Aside from this constraint, the learning rate α_k can be chosen in any of the standard ways outlined in Section 12.3, including by solving the optimization problem $\alpha_k = \operatorname{argmin}_{\alpha \in [0,1]} f(\mathbf{x}_k + \alpha(\bar{\mathbf{x}}_k - \mathbf{x}_k))$, using a constant value, or backtracking.

Geometrically, this method amounts to choosing $\bar{\mathbf{x}}_k$ by minimizing the linear Taylor approximation $\ell(\mathbf{x}) = f(\mathbf{x}_k) + Df(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$ of f at $\mathbf{x}_k$. That is, the method moves in the direction that would be optimal if f were linear (and equal to $\ell(\mathbf{x})$), but the distance it travels in that direction is determined by whatever method is used to choose α_k.

The conditional gradient method replaces the problem of minimizing f over $\mathscr{F}$ with the problem of successively minimizing the linear functions $Df(\mathbf{x}_k)(\mathbf{x}-\mathbf{x}_k)$ over $\mathscr{F}$. These are often much easier to solve than the original problem. For example, if $\mathscr{F}$ is defined by purely affine constraints, then each of the new problems (14.27) is a linear optimization problem that can be solved using the simplex method.

Example 14.7.1. Given $Q > 0$, consider the problem

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{r}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \preceq \mathbf{b}, \\ & \mathbf{x} \succeq \mathbf{0}. \end{array}$$

We show how to take one step of the conditional gradient method to solve this problem, given a feasible point $\mathbf{x}_k$. Let $\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \succeq \mathbf{0}, A\mathbf{x} \preceq \mathbf{b}\}$. For each k we have $Df(\mathbf{x}_k) = \mathbf{x}_k^T Q + \mathbf{r}^T$ and so

$$\bar{\mathbf{x}}_k = \operatorname*{argmin}_{\mathbf{x}\in\mathscr{F}} (\mathbf{x}_k^T Q + \mathbf{r}^T)(\mathbf{x} - \mathbf{x}_k) = \operatorname*{argmin}_{\mathbf{x}\in\mathscr{F}} \mathbf{x}_k^T Q\mathbf{x} + \mathbf{r}^T\mathbf{x}.$$

Setting $\mathbf{c}_k = \mathbf{x}_k^T Q + \mathbf{r}^T$ shows that the problem of finding $\bar{\mathbf{x}}$ has been transformed into the linear optimization problem

$$\begin{array}{ll} \text{minimize} & \mathbf{c}_k^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \preceq \mathbf{b}, \\ & \mathbf{x} \succeq \mathbf{0}, \end{array}$$

which can be solved with the simplex method. Once $\bar{\mathbf{x}}_k$ has been found, the learning rate α_k may be found by solving

$$\alpha_k = \operatorname*{argmin}_{\alpha\in[0,1]} f(\mathbf{x}_k + \alpha(\bar{\mathbf{x}}_k - \mathbf{x}_k)).$$

One way to do this is to find the global minimizer α^* of $\varphi(\alpha) = f(\mathbf{x}_k + \alpha(\bar{\mathbf{x}}_k - \mathbf{x}_k))$, using the one-dimensional optimization methods of Section 12.2 or by solving for it exactly, by hand, since φ is a quadratic function of α. In either case, the solution of the constrained optimization problem $\alpha_k = \operatorname{argmin}_{\alpha\in[0,1]} \varphi(\alpha)$ is 0, 1, or α^*, if $\alpha^* \in [0,1]$. The computed values of α_k and $\bar{\mathbf{x}}_k$ are now plugged into (14.26) to get the next iterate.

14.7.2 Gradient Projection

The *gradient projection method* is another iterative method for solving problems of the form (14.25) with a convex feasible set $\mathscr{F}$. This method chooses

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k(\bar{\mathbf{x}}_k - \mathbf{x}_k)$$

with

$$\bar{\mathbf{x}}_k = \operatorname{proj}_{\mathscr{F}}(\mathbf{x}_k - s_k Df(\mathbf{x}_k)^T) \tag{14.28}$$

for some choice of $\alpha_k \in [0,1]$ and $s_k > 0$. Again, since the point $\bar{\mathbf{x}}_k$ lies in $\mathscr{F}$, which is convex, the point $\mathbf{x}_{k+1}$ must also lie in $\mathscr{F}$, provided $\alpha_k \in [0,1]$.

Computing a projection $\operatorname{proj}_{\mathscr{F}} \mathbf{v}$ is an optimization problem with a quadratic objective:

$$\operatorname*{minimize}_{\mathbf{x}\in\mathscr{F}} \|\mathbf{x} - \mathbf{v}\|_2^2.$$

Thus, the gradient projection method replaces the problem of minimizing f over $\mathscr{F}$ with the problem of successively minimizing quadratic functions of the form $\|\mathbf{x} - (\mathbf{x}_k - s_k Df(\mathbf{x}_k)^T)\|_2^2$ over $\mathscr{F}$.

In some situations the projection is easy to compute. Here are some important cases:

(i) If the constraints defining $\mathscr{F}$ are all of the form $\mathbf{a} \preceq \mathbf{x} \preceq \mathbf{b}$, then the projection $\operatorname{proj}_{\mathscr{F}} \mathbf{v}$ has its ith coordinate $(\operatorname{proj}_{\mathscr{F}} \mathbf{v})_i$ given by

$$(\operatorname{proj}_{\mathscr{F}} \mathbf{v})_i = \begin{cases} a_i & \text{if } v_i < a_i, \\ v_i & \text{if } a_i \le v_i \le b_i, \\ b_i & \text{if } b_i < v_i. \end{cases}$$

(ii) If the feasible set is a hyperplane $\mathscr{F} = \{\mathbf{x} \mid \mathbf{a}^\mathsf{T}\mathbf{x} = b\}$, then the projection of $\mathbf{v}$ to $\mathscr{F}$ is

$$\operatorname{proj}_{\mathscr{F}} \mathbf{v} = \mathbf{v} + \frac{b - \mathbf{a}^\mathsf{T}\mathbf{v}}{\|\mathbf{a}\|_2^2}\mathbf{a}. \tag{14.29}$$

(iii) If the feasible set is a half space $\mathscr{F} = \{\mathbf{x} \mid \mathbf{a}^\mathsf{T}\mathbf{x} \le b\}$, then the projection of $\mathbf{v}$ to $\mathscr{F}$ is just $\mathbf{v}$ if $\mathbf{a}^\mathsf{T}\mathbf{v} \le b$ and otherwise it is the projection onto the supporting hyperplane (14.29).

There are two main approaches to using the gradient projection algorithm. One approach is to make $s_k = s$ constant. Then, once each $\bar{\mathbf{x}}_k$ is chosen, choose α_k either as $\alpha_k = \operatorname{argmin}_{\alpha \in [0,1]} f(\mathbf{x}_k + \alpha(\bar{\mathbf{x}}_k - \mathbf{x}_k))$ or just choose α_k by backtracking from $\alpha = 1$.

The other approach is to fix the learning rate $\alpha_k = 1$, but at each step choose s_k judiciously. This can be done by setting $\mathbf{x}(s) = \operatorname{proj}_{\mathscr{F}}(\mathbf{x}_k - sDf(\mathbf{x}_k)^\mathsf{T})$ and then backtracking from $s = 1$ until the following analogue of the Armijo condition (12.16) holds:

$$f(\mathbf{x}_k) - f(\mathbf{x}(s)) \ge \sigma Df(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}(s)) \tag{14.30}$$

for some fixed choice of $\sigma \in (0,1)$. This effectively amounts to backtracking along the *projection arc* $\{\mathbf{x}(s) \mid s \in [0,1]\}$ until 14.30 is satisfied.

Example 14.7.2. Consider a constrained optimization problem of finding $\mathbf{x} \in \mathbb{R}^n$ to

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\mathsf{T}Q\mathbf{x} + \mathbf{r}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & \mathbf{a}^\mathsf{T}\mathbf{x} \le b \end{aligned}$$

for some fixed choices of $b \in \mathbb{R}$, $\mathbf{a}, \mathbf{r} \in \mathbb{R}^n$, and $Q \in M_n(\mathbb{R})$ with $Q > 0$. Given a feasible point $\mathbf{x}_k$, the gradient projection method with fixed $s_k = s$ first computes $\mathbf{v} = \mathbf{x}_k - sDf(\mathbf{x}_k)^\mathsf{T} = \mathbf{x}_k - sQ\mathbf{x}_k - s\mathbf{r}$ and then finds $\bar{\mathbf{x}}_k$ as $\operatorname{proj}_{\mathscr{F}} \mathbf{v}$. This is given by

$$\bar{\mathbf{x}}_k = \begin{cases} \mathbf{v} & \text{if } \mathbf{a}^\mathsf{T}\mathbf{v} \le b, \\ \mathbf{v} + \frac{b - \mathbf{a}^\mathsf{T}\mathbf{v}}{\|\mathbf{a}\|_2^2}\mathbf{a} & \text{if } \mathbf{a}^\mathsf{T}\mathbf{v} > b. \end{cases}$$

The learning rate α_k can be computed in the same manner as in Example 14.7.1 and $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k(\bar{\mathbf{x}}_k - \mathbf{x})$.

Example 14.7.3. We can also use backtracking along the projection arc for the problem in Example 14.7.2 with fixed learning rate $\alpha_k = 1$. To do this, let $\mathbf{v}(s) = \mathbf{x}_k - sDf(\mathbf{x}_k)^\mathsf{T}$, as before, and define

$$\bar{\mathbf{x}}(s) = \operatorname{proj}_{\mathscr{F}} \mathbf{v}(s) = \begin{cases} \mathbf{v}(s) & \text{if } \mathbf{a}^\mathsf{T}\mathbf{v}(s) \leq b, \\ \mathbf{v}(s) + \frac{b - \mathbf{a}^\mathsf{T}\mathbf{v}(s)}{\|\mathbf{a}\|_2^2}\mathbf{a} & \text{if } \mathbf{a}^\mathsf{T}\mathbf{v}(s) > b \end{cases}$$

for all $s \in [0,1]$. Fix $\sigma \in (0,1)$. Starting at $s = 1$, check the Armijo condition:

$$f(\mathbf{x}_k) - f(\mathbf{x}(s)) > \sigma Df(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}(s)) = \frac{1}{2}(\mathbf{x}_k^\mathsf{T}Q - \mathbf{r}^\mathsf{T})(\mathbf{x}_k - \mathbf{x}(s)). \quad (14.31)$$

If that fails, try again with $s = 1/2$ and then $s = 1/4$, and so on, until (14.31) holds, at which point, set $\mathbf{x}_{k+1} = \mathbf{x}(s)$.

Remark 14.7.4. The gradient projection method can suffer from the same sorts of inefficiencies and slow convergence as unconstrained gradient descent. There are many variations on this method to try to improve this situation, analogous to things like conjugate gradient for improving the unconstrained case of gradient descent.

14.7.3 Newton's Method with Constraints

Recall that the unconstrained version of Newton's method can be thought of as making a quadratic approximation

$$f(\mathbf{x}) \approx q(\mathbf{x}) = f(\mathbf{x}_k) + Df(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\mathsf{T} D^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$$

of $f(\mathbf{x})$ near $\mathbf{x}_k$ and then finding the minimizer $\mathbf{x}_{k+1} = \operatorname{argmin}_{\mathbf{x}} q(\mathbf{x})$. This is easily adapted to the constrained case as

$$\mathbf{x}_{k+1} = \operatorname*{argmin}_{\mathbf{x} \in \mathscr{F}} q(\mathbf{x}). \quad (14.32)$$

In the unconstrained case the minimizer of q is given by the usual Newton formula $\mathbf{x}_k - D^2 f(\mathbf{x}_k)^{-1} Df(\mathbf{x}_k)^\mathsf{T}$, but in the constrained case, computing $\mathbf{x}_{k+1}$ requires minimizing a quadratic objective over $\mathscr{F}$. As with the other methods, the difficulty of solving this problem depends heavily on the nature of $\mathscr{F}$. As in the unconstrained case, a natural variant of this method is to set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \bar{\mathbf{x}}_k$, where $\bar{\mathbf{x}}_k = (\operatorname{argmin}_{\mathbf{x} \in \mathscr{F}} q(\mathbf{x})) - \mathbf{x}_k$ and the learning rate $\alpha_k \in [0,1]$ is chosen in one of the standard ways.

14.7.4 Initialization

Since all of the methods in this section are iterative, they require an initial point $\mathbf{x}_0$. If the feasible set $\mathscr{F}$ is defined by affine constraints, then an initial value can

be found with the help of an auxiliary linear problem, exactly as in the linear case; see Section 13.5.1.

Alternatively, we can set up a different problem with the same minimizer, but with a relaxed set of constraints so that it is easier to find an initial feasible point. The key is to add a penalty to the objective for any failure to satisfy the original constraints.

Example 14.7.5. Consider a problem of the form

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && G(\mathbf{x}) \preceq \mathbf{0}. \end{aligned} \tag{14.33}$$

If no $\mathbf{x}$ is known that satisfies $G(\mathbf{x}) \preceq \mathbf{0}$, then we can consider an alternative problem of the form

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) + ct \\ &\text{subject to} && G(\mathbf{x}) \preceq t\mathbb{1}, \\ & && t \geq 0, \end{aligned} \tag{14.34}$$

where $\mathbb{1} = (1, 1, \ldots, 1)$ is the all-ones vector, and $c > 0$ is fixed. If $\mathbf{x}^*$ is a minimizer of the original problem, then $(\mathbf{x}^*, 0)$ is a minimizer for the new problem. And conversely, if $(\mathbf{x}^*, t^*)$ is a minimizer for the new problem, then $t^* = 0$ and $\mathbf{x}^*$ must be a minimizer of the original problem. Moreover, it is easy to find a feasible starting point for the new problem; namely, for any $\mathbf{x}_0$ in the domain of f, if $t_0 = \max(g_1(\mathbf{x}_0), \ldots, g_m(\mathbf{x}_0))$, then $(\mathbf{x}_0, t_0)$ is feasible for (14.34). Any of the previous iterative methods for solving constrained optimization problems may now be used to solve (14.34).

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

14.1. The unit circle $S^1 \subset \mathbb{R}^2$ is defined by the equation $h(x,y) = 0$, where

$$h(x,y) = x^2 + y^2 - 1.$$

(i) Show that every point on the circle is a regular point.

(ii) Near each point, find a one-dimensional parametrization for the circle. Hint: Remember that a parametrization is an injective map from an open interval to the circle, so you cannot find one parametrization that works for all the points of the circle at once. At each point, describe a (nonzero) normal vector as a function of your parametrization near that point.

14.2. The equation $h(x,y,z) = 0$, where

$$h(x,y,z) = \left(c - \sqrt{x^2+y^2}\right)^2 + z^2 - a^2, \quad c > a > 0,$$

defines a torus $T^2 \subset \mathbb{R}^3$.

(i) Show that every point on this torus is a regular point.

(ii) Find a two-dimensional parametrization of the torus near each point and describe a (nonzero) normal vector at each point. Hint: Recall that $Dh(\mathbf{x})^\mathsf{T}$ is always orthogonal to $T_\mathbf{x}S$.

14.3. For each of the following functions $h : \mathbb{R}^2 \to \mathbb{R}$, find all the points of the set $S = \{(x,y) \in \mathbb{R}^2 \mid h(\mathbf{x}) = 0\}$ that are singular (not regular). Plot the set S near each singular point. For each singular point (x,y), identify the vector space $\{\mathbf{v} \in \mathbb{R}^2 \mid Dh(x,y)\mathbf{v} = 0\}$ and its orthogonal complement.

(i) $h(x,y) = x^4 - 2x^3 + x^2 - y^2$. Hint: The implicit curve $S = \{\mathbf{x} \mid h(\mathbf{x}) = 0\}$ corresponds to the 0-contour curve.

(ii) $h(x,y) = x^3 - x^2 - y^2$. Hint: To see what is happening near the singularity, try plotting the level curves $\{\mathbf{x} \mid h(\mathbf{x}) = c\}$ for very small positive and negative values of c.

(iii) $h(x,y) = x^3 - y^2 + 2y - 1$.

14.4. For each of the following functions $h : \mathbb{R}^3 \to \mathbb{R}$, find all the points of the set $S = \{(x,y,z) \in \mathbb{R}^3 \mid h(\mathbf{x}) = 0\}$ that are singular (not regular). Plot the set S near each singular point (if there are an infinite number of singular points, plot the set near a large number of them).

(i) $h(x,y,z) = z^2 - x^2 + y^2$. Hint: If you do not have tools to easily plot a three-dimensional implicit surface, you can often just solve for one of the variables in terms of the others (like $z = \pm\sqrt{x^2 - y^2}$) and then plot the corresponding three-dimensional graphs (in this case both the positive and negative parts).

(ii) $h(x,y,z) = x^2 y - z^2$.

14.5. Prove that if $\ell = n$, then a regular point $\mathbf{x} \in \mathscr{F}$ is an isolated point, that is, there is a neighborhood of $\mathbf{x}$ in which there are no other points of $\mathscr{F}$.

14.6. Consider the problem

$$\begin{array}{ll} \underset{(x,y)\in\mathbb{R}^2}{\text{maximize}} & xy \\ \text{subject to} & x^2+4y^2=1. \end{array}$$

Find all the points that satisfy the Lagrange first-order condition.

14.7. Consider the problem

$$\begin{array}{ll} \underset{(x,y,z)\in\mathbb{R}^3}{\text{maximize}} & x^2+2xy+3y^2+4x+5y+6z \\ \text{subject to} & x+2y=3, \\ & 4x+5z=6. \end{array}$$

Find all the points that satisfy the Lagrange first-order condition.

14.8. Find the dimensions of the box of maximal volume that can be inscribed in the ellipsoid

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1.$$

14.9. Let $A \in M_{m\times n}(\mathbb{R})$ be of maximal rank. Show that the minimizer of the problem

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \|A\mathbf{x}\|_2^2 \\ \text{subject to} & \|\mathbf{x}\|_2^2=1 \end{array}$$

is the unit-length right singular vector corresponding to the smallest singular value of A, that is, $\mathbf{x}^*$ satisfies $A^\mathsf{T}A\mathbf{x}^*=\sigma_r^2\mathbf{x}^*$.

14.10. Consider the problem

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^2}{\text{maximize}} & \|\mathbf{x}-\mathbf{x}_0\|_2^2 \\ \text{subject to} & \|\mathbf{x}\|_2^2=9, \end{array}$$

where $\mathbf{x}_0=(1,\sqrt{3})$. Find all the points that satisfy the Lagrange first-order condition.

14.11. For Exercise 14.6 use the second-order conditions to determine which, if any, of the points are local maximizers.

14.12. For Exercise 14.7 use the second-order conditions to determine which, if any, of the points are local maximizers.

14.13. For any equality-constrained maximization problem of the form (14.6), define the Lagrangian of the maximization problem to be

$$\mathscr{L}(\mathbf{x},\boldsymbol{\lambda})=f(\mathbf{x})+\boldsymbol{\lambda}^\mathsf{T}H(\mathbf{x}).$$

Use the Lagrange FONC for maximization problems (see Remark 14.2.3) and the Lagrange second-order conditions for minimization to prove the following corresponding second-order conditions in the case that f and H are both C^2 in a neighborhood of $\mathbf{x}^*$:

(i) If $\mathbf{x}^*$ is a local maximizer (14.6) and a regular point of the feasible set $\mathscr{F}$, then the $\boldsymbol{\lambda}^*\in\mathbb{R}^\ell$ of Remark 14.2.3 is such that $\mathbf{v}^\mathsf{T}D^2_{\mathbf{x}}\mathscr{L}(\mathbf{x}^*,\boldsymbol{\lambda}^*)\mathbf{v}\le 0$ whenever $\mathbf{v}\in T_{\mathbf{x}^*}\mathscr{F}$.

(ii) If there exists $\mathbf{x}^* \in \mathbb{R}^n$ and $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ such that Lagrange FONC for maximization holds and $\mathbf{v}^\mathsf{T} D^2_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{v} < 0$ for all nonzero $\mathbf{v} \in T_{\mathbf{x}} \mathscr{F}$, then $\mathbf{x}^*$ is a strict local maximizer.

14.14. Recall the minimization problem in Exercise 14.10. In that problem, we determined the points that satisfied the Lagrange first-order condition. Now use the second-order conditions to determine which, if any, of the points are local maximizers.

14.15. Let $f(x, y, z) = x + y + z$, and define

$$S = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + 2y^2 + 3z^2 = 1\}.$$

Find the maximum and minimum of f on S.

14.16. Find all solutions to the optimization problem

$$\begin{aligned} \text{maximize} \quad & \mathbf{x}^\mathsf{T} A \mathbf{x} \\ \text{subject to} \quad & \|\mathbf{x}\|_2^2 = 1, \end{aligned}$$

where $A = \left[\begin{smallmatrix} 2 & 3 \\ 0 & 2 \end{smallmatrix}\right]$.

14.17. Find all the points that satisfy the first-order KKT conditions for the optimization problem

$$\begin{aligned} \text{maximize} \quad & x_1 x_2 \\ \text{subject to} \quad & x_1 + x_2^2 \leq 2, \\ & x_1, x_2 \geq 0. \end{aligned}$$

14.18. Consider the optimization problem of choosing $(x, y) \in \mathbb{R}^2$ to

$$\begin{aligned} \text{maximize} \quad & -x^2 - 2y^2 \\ \text{subject to} \quad & x + y \leq 4, \\ & xy = 1, \\ & x > 0, y > 0. \end{aligned}$$

(i) Set up the problem in the form (14.1). Hint: Strict inequalities can be incorporated by restricting to an open set Ω rather than as constraints of the form $g_i \leq 0$.

(ii) Write all the first-order KKT conditions that any optimizer must satisfy.

(iii) Find all the candidate points that should be tested for optimality.

(iv) Plot the feasible set and all the points you identified using the KKT conditions.

(v) Identify the global maximizer.

14.19. Find all the points that satisfy the first-order KKT conditions for the optimization problem

$$\begin{aligned} \text{maximize} \quad & x_1^2 + 6x_1 x_2 - 2x_1 - 2x_2 \\ \text{subject to} \quad & x_1^2 + 2x_2 \leq 1, \\ & 2x_1 - 2x_2 \leq 1. \end{aligned}$$

Hint: Do not feel obligated to give exact answers nor to do the entire problem by hand—good numerical approximations are acceptable, and some of the more tedious aspects of this problem are much easier to do using a few lines of code.

14.20. Solve the optimization problem

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{x}\|_2 \\ \text{subject to} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} = b, \\ & \mathbf{x} \succeq \mathbf{0}, \end{aligned}$$

where $\mathbf{c} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are given. Hint: Use the KKT conditions to relate the signs of b, λ, and each x_i, and c_i.

14.21. Consider the problem

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{b} - A\mathbf{x}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{x}\|_1 = 1, \\ & \mathbf{x} \succeq \mathbf{0}, \end{aligned}$$

where $A \in M_{m\times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$.

(i) Explain what it means for a feasible point $\mathbf{x} \in \mathbb{R}^n$ to be a regular point. Are there feasible points that are not regular?

(ii) Write down the first-order KKT conditions.

14.22. ⚠ Write down the KKT first-order conditions for the linear problem

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{b}, \\ & \mathbf{x} \succeq \mathbf{0} \end{aligned}$$

and show that the first two conditions (Theorem 14.4.5, items (i) and (ii)) are equivalent to the dual linear problem (see Definition 13.6.1), while condition (iii) is equivalent to complementary slackness (see Theorem 13.6.10).

14.23.* For Exercise 14.17 write down the KKT second-order conditions and use them to determine which of the points you found from the first-order conditions are optimizers.

14.24.* For Exercise 14.18 write down the KKT second-order conditions and use them to determine which of the points you found from the first-order conditions are optimizers.

14.25.* For Exercise 14.19 write down the KKT second-order conditions and use them to determine which of the points you found from the first-order conditions are optimizers.

14.26.* For Exercise 14.19 write down the KKT second-order conditions and use them to determine which of the points you found from the first-order conditions are optimizers.

14.27.* For Exercise 14.20 write down the KKT second-order conditions and use them to determine which of the points you found from the first-order conditions are optimizers.

14.28.* Provide the details to finish the proof of Theorem 14.5.2.

14.29. Given the points $(1,2)$, $(3,1)$, and $(5,5)$ in $\mathbb{R}^2$,

(i) Assuming that only the y direction is subject to error, write an *unconstrained* linear optimization problem which is equivalent to the ordinary least squares problem of finding the line that best (in terms of the 2-norm) fits these three points.

(ii) Assuming that both the x and y directions are subject to error, write an *unconstrained* linear optimization problem which is equivalent to the total least squares problem of finding the line that best (in terms of the square of the 2-norm) fits these three points.

14.30. Consider the linear optimization problem of finding $\mathbf{x} \in \mathbb{R}^n$ with an equality constraint

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}_1^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A_1\mathbf{x} \preceq \mathbf{b}_1, \\ & \mathbf{p}^\mathsf{T}\mathbf{x} = d. \end{aligned}$$

Show that if $\mathbf{p} \neq \mathbf{0}$, then there exists a linear optimization problem

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}_2^\mathsf{T}\mathbf{y} \\ \text{subject to} \quad & A_2\mathbf{y} \preceq \mathbf{b}_2 \end{aligned}$$

in $n-1$ variables (that is, $\mathbf{y} \in \mathbb{R}^{n-1}$) and an $n \times (n-1)$ matrix M such that if $\mathbf{y}^*$ is a maximizer to the second problem, then $\mathbf{x}^* = M\mathbf{y}^* + \mathbf{x}_0$ is a maximizer to the first problem.

14.31. For any $\mathbf{v} \in \mathbb{R}^n$ and any $A \in M_n(\mathbb{R})$ with $A > 0$, write $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^\mathsf{T} A \mathbf{v}}$. Let $L \in M_n(\mathbb{R})$ and let $W = LL^\mathsf{T} > 0$. Show that the unconstrained optimization problem

$$\text{minimize} \quad \|A\mathbf{x} - \mathbf{b}\|_{W^{-1}}^2$$

has the same solution as the constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{v}\|_2^2 \\ \text{subject to} \quad & A\mathbf{x} + L\mathbf{v} = \mathbf{b}. \end{aligned}$$

14.32. Show that the portfolio minimization problem (14.23) (with short selling allowed) is equivalent to an unconstrained quadratic optimization problem of the form (14.24) as follows:

(i) Write the constraints as $C\mathbf{w} = \mathbf{d}$ and find A, $\mathbf{v}$, and p explicitly in terms of the QR decomposition $C = Q \begin{bmatrix} R_1 & R_2 \end{bmatrix} P^\mathsf{T}$.

(ii) Give the formula for $\mathbf{w}$ in terms of the solution $\mathbf{y}$ of the unconstrained problem.

(iii) Prove that A is positive definite.

14.33. Consider the Markowitz portfolio optimization problem with only two assets of known expected value $\bar{r}_1$ and $\bar{r}_2$, respectively.

(i) Show that the constraints give a unique solution, so the minimum-variance solution is the only solution.

(ii) Show that the variance is always a quadratic function in the expected return $\bar{r}$. (Beware that $E(r^2)$ is a function of $\bar{r}$, so writing $\mathrm{Var}(r) = E(r^2) - \bar{r}^2$ does not, in itself, solve the problem.) Figure 14.9 shows the graph of such a curve.

(iii) The bottom half of the curve in Figure 14.9 is dotted because no one is interested in it. Why not?

14.34. Verify the following claims made in the text:

(i) If $\mathscr{F}$ is convex, and $\mathbf{x}_k, \bar{\mathbf{x}} \in \mathscr{F}$, then $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha(\bar{\mathbf{x}} - \mathbf{x}_k) \in \mathscr{F}$ for all $\alpha \in [0, 1]$.

(ii) If the feasible set is the hyperplane $\mathscr{F} = \{\mathbf{x} \mid \mathbf{a}^\mathsf{T}\mathbf{x} = b\}$, then the projection of $\mathbf{v}$ to $\mathscr{F}$ is given by (14.29).

(iii) If $(\mathbf{x}^*, t^*)$ is a minimizer for (14.34), then $\mathbf{x}^*$ is a minimizer of (14.7.5).

14.35. Consider the problem of choosing $\mathbf{x} \in \mathbb{R}^3$ to

$$\begin{array}{rl} \text{minimize} & f(\mathbf{x}) = \frac{1}{2}(x_1^2 + x_2^2 + 0.1x_3^2) + 0.55x_3 \\ \text{subject to} & x_1 + x_2 + x_3 = 1, \\ & \mathbf{x} \succeq \mathbf{0}. \end{array}$$

(i) Show that the global minimizer is $\mathbf{x}^* = (1/2, 1/2, 0)$.

(ii) Given $\bar{\mathbf{x}}_k$ and $\mathbf{x}_k$, find a closed-form expression for the minimizer $\alpha_k = \operatorname{argmin}_{\alpha \in [0,1]} f(\mathbf{x}_k + \alpha(\bar{\mathbf{x}}_k - \mathbf{x}_k))$.

(iii) Write a computer program implementing the conditional gradient method to solve this problem, using the formula for learning rate computed in the previous step.

(iv) Verify computationally that for any starting point $\mathbf{x}_0 = (x_1, x_2, x_3) \succ \mathbf{0}$ that satisfies $x_1 \neq x_2$, we have

$$\frac{f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)}{f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})} \to 1.$$

Thus, this method does not converge linearly on this problem; see Definition 12.2.1.

14.36. Write a computer program implementing the gradient projection method to solve the optimization problem in Exercise 14.35 with constant $s = 1$ and learning rate $\alpha_k = \operatorname{argmin}_{\alpha \in [0,1]} f(\mathbf{x}_k + \alpha(\bar{\mathbf{x}} - \mathbf{x}_k))$. Compare the convergence rate for various initial points to that of your implementation for the conditional gradient method. Hint: Use (14.29) with $\mathbf{a} = (1, 1, 1)$.

14.37. Write a computer program implementing the gradient projection method to solve the optimization problem of Exercise 14.35 with constant $\alpha = 1$, but with s_k chosen by backtracking along the projection arc to satisfy (14.30). Compare the convergence rate for various initial points to those of your implementations for the conditional gradient method and gradient projection with fixed s and variable α.

14.38. Consider the constrained problem of minimizing $f(\mathbf{x})$ subject to $H(\mathbf{x}) = \mathbf{0}$, where $f \in C^2(\Omega; \mathbb{R})$ and $H \in C^1(\Omega; \mathbb{R}^\ell)$. When applying Newton's method to solve this problem, each iteration requires solving the subproblem

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}_k) + Df(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \tfrac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\mathsf{T} D^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \\ \text{subject to} \quad & H(\mathbf{x}) = \mathbf{0}. \end{aligned} \tag{14.35}$$

Prove that if $\bar{\mathbf{x}} \in \mathbb{R}^n$ and $\boldsymbol{\lambda} \in \mathbb{R}^\ell$ are such that

$$\begin{bmatrix} D^2 f(\mathbf{x}_k) & DH(\mathbf{x}_k)^\mathsf{T} \\ DH(\mathbf{x}_k) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -Df(\mathbf{x}_k) \\ -H(\mathbf{x}_k) \end{bmatrix},$$

then $\bar{\mathbf{x}}$ is a minimizer of (14.35). Thus the Newton subproblem can be solved by solving a linear system of dimension $(n + \ell)$, provided that system is full rank. This shows the temporal cost of each intermediate step in the equality-constrained Newton method is $O((n+\ell)^3)$, where ℓ is the number of equality constraints.

Notes

Much of this chapter was inspired by [CZ01], while the final section was inspired by [Ber16]. We learned of Unexample 14.6.1 from [NW99]. Example 14.4.10 is from [CZ01, Example 20.4], and many of the exercises on the Lagrange and KKT conditions are also from [CZ01]. Example 14.3.6 is from [Jia18].

Other useful sources for these topics include [Bie15, BV04, NW99] and [Ped04]. An interesting history of Lagrange multipliers is given in [Bus03].

15 Convex Analysis and Optimization

If we can formulate a problem as a convex optimization problem, then we can solve it efficiently.... With only a bit of exaggeration, we can say that if you formulate a practical problem as a convex optimization problem, then you have solved the original problem.
—S. Boyd and L. Vandenberghe

A function is convex when any chord of the graph lies on or above the graph (see Figure 15.1). In first-year calculus classes, we usually call such functions *concave up*. In optimization, these functions are extremely nice to work with. If the objective function is smooth and convex and the feasible set is convex, then a local minimum is the global minimum and all that is needed to find it is the FONC (Theorem 14.2.1). There are many important problems spanning nearly every area of applied mathematics that can be formulated and solved as convex optimization problems. For example, linear optimization problems are convex optimization problems.

We begin this chapter by defining convex functions and describing many of their important properties. Among the most useful concepts in the study of convex functions is *Jensen's inequality* and the large inventory of household inequalities that can be derived from it. For example, using Jensen's inequality, we prove all the inequalities in Volume 1, Section 3.6, in particular, Young's inequality, the arithmetic-geometric mean inequality, Hölder's inequality, and Minkowski's inequality. It is staggering how many fundamental inequalities are simple corollaries of Jensen's inequality.

The remainder of the chapter focuses on convex optimization problems and a few of the very powerful numerical methods for solving them. As with linear optimization problems, more general minimization problems have a corresponding dual problem that satisfies a weak duality principle similar to that of Section 13.6. Many convex optimization problems also satisfy strong duality. This means that the dual problem also has a global optimizer and the optimal values of the two problems are equal. As with linear optimization problems, the dual problem is often easier to solve than the original problem. Many of the most powerful numerical methods for convex optimization rely heavily on both the primal and dual formulations and the relation between them.

15.1 Convex Functions

In this section, we define convex functions and describe some of their properties. Throughout this chapter we assume that V is a vector space and $\Omega \subset V$ an open subset.

We first show how to extend the range of a function to include $+\infty$ and $-\infty$. Consider the extended real numbers $\mathbb{R}_{\infty} = \mathbb{R} \cup \{\infty\}$, $\mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$ and $\mathbb{R}_{\pm\infty} = \mathbb{R} \cup \{\infty, -\infty\}$, where the arithmetic rules involving $\pm\infty$ are as follows:

(i) If $x \in \mathbb{R}$, then $x \pm \infty = \pm\infty$.

(ii) If $x > 0$, then $x \cdot (\pm\infty) = \pm\infty$, and if $x < 0$, then $x \cdot (\pm\infty) = \mp\infty$.

(iii) $-\infty - \infty = -\infty$ and $\infty + \infty = \infty$.

Note that some operations such as $\infty - \infty$ are ambiguous and not defined.

Remark 15.1.1. These rules are the same as those involving the special floating point numbers $\pm$ INF. The operations that result in NaN such as $\infty - \infty$ are likewise undefined; see Section 11.1.1.

Remark 15.1.2. For any function $f : \Omega \to \mathbb{R}$, we can extend f to all of V by defining the extension $\overline{f} : V \to \mathbb{R}_{\infty}$ as

$$\overline{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega, \\ \infty & \text{if } \mathbf{x} \notin \Omega. \end{cases}$$

We can also restrict f to the points in the domain where f is finite, as follows.

Definition 15.1.3. *For any function $f : \Omega \to \mathbb{R}_{\pm\infty}$, the* effective domain *of f is the set*

$$\operatorname{effd}(f) = \{\mathbf{x} \in \Omega \mid -\infty < f(\mathbf{x}) < \infty\} \subset \Omega.$$

Remark 15.1.4. If $f : \Omega \to \mathbb{R}_{\pm\infty}$, then we have that $f : \operatorname{effd}(f) \to \mathbb{R}$.

15.1.1 Convex Functions

The concept of a convex function is fundamental to optimization and, indeed, to much of real analysis.

Definition 15.1.5. *Let C be a convex subset of V. A function $f : C \to \mathbb{R}_{\infty}$ is* convex *if for all $\mathbf{x}_1, \mathbf{x}_2 \in C$ and $0 \le \lambda \le 1$, we have*

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \le \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2); \tag{15.1}$$

see Figure 15.1. The function f is strictly convex *if strict inequality holds in* (15.1) *whenever $\mathbf{x}_1 \neq \mathbf{x}_2$ and $0 < \lambda < 1$. The function f is* concave *or* strictly concave *if $-f$ is convex or strictly convex, respectively.*

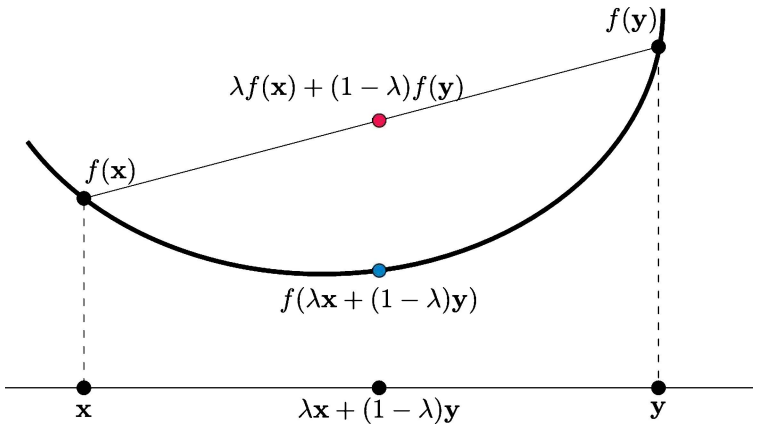


Figure 15.1. *The defining feature of a convex function f is that the line segment connecting the points $(\mathbf{x}_1, f(\mathbf{x}_1))$ and $(\mathbf{x}_2, f(\mathbf{x}_2))$ lies above the function itself for each point of the line segment $\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$, where $0 < \lambda < 1$.*

Example 15.1.6. A norm function $\|\cdot\|$ on a vector space is a convex function. From the triangle inequality we have

$$\|\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2\| \leq \lambda \|\mathbf{x}_1\| + (1-\lambda)\|\mathbf{x}_2\|$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in V$ and $0 \leq \lambda \leq 1$.

Remark 15.1.7. If a function takes on both the values ∞ and $-\infty$, then checking the convexity condition between these two points doesn't make sense, because it involves computing $\lambda \cdot \infty + (1-\lambda)(-\infty) = \infty - \infty$, which is not defined. For this reason, we always require convex functions to take their values in $\mathbb{R}_\infty$. There is no loss of generality by excluding $-\infty$ because if $f : V \to \mathbb{R}_{-\infty}$ has $\operatorname{effd}(f) \neq \emptyset$ and there exists some $\mathbf{x} \in V$ with $f(\mathbf{x}) = -\infty$, then f cannot satisfy (15.1). To see this, consider $\mathbf{z} \in \operatorname{effd}(f)$ and let $\mathbf{y} = 2\mathbf{z} - \mathbf{x}$. If f were convex, we would have

$$-\infty < f(\mathbf{z}) = f\left(\frac{1}{2}\mathbf{x} + \frac{1}{2}\mathbf{y}\right) \leq \frac{1}{2} f(\mathbf{x}) + \frac{1}{2} f(\mathbf{y}) = -\infty,$$

which is a contradiction.

Remark 15.1.8. If f is convex on a convex set $C \subset V$, then we can naturally extend f to a convex $\mathbb{R}_\infty$-valued function $\overline{f}$ on all of V via Remark 15.1.2. Moreover, we have that $\operatorname{effd}(\overline{f}) = C$. This shows that there is no loss of generality to assume that convex $\mathbb{R}_\infty$-valued functions are defined on all of V. We often make this assumption in the rest of this book.

Proposition 15.1.9. *If $f : V \to \mathbb{R}_\infty$ is a convex function, then $\operatorname{effd}(f)$ is a convex set.*

Proof. Assume that f is a convex function with $\mathbf{x}_1, \mathbf{x}_2 \in \operatorname{effd}(f)$. If $0 \leq \lambda \leq 1$, then $f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) < \infty$, and so $\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in \operatorname{effd}(f)$. □

15.1.2 Characterizations of Convex Functions

The following lemma guarantees that a function is convex if and only if it is convex when restricted to any line segment on the domain. This can be used to provide an alternative characterization of a convex function.

Lemma 15.1.10. *Let $f : V \to \mathbb{R}_\infty$ be a function with essential domain $C \subset V$. The following are equivalent:*

(i) *The function f is convex.*

(ii) *For all $\mathbf{x}_1, \mathbf{x}_2 \in C$ the map $g : [0,1] \to \mathbb{R}$ given by $g(t) = f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2)$ is convex.*

(iii) *For all $\mathbf{x}_1, \mathbf{x}_2 \in C$ the map $g : [0,1] \to \mathbb{R}$ given by $g(t) = f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2)$ satisfies*

$$g(t) \leq t g(1) + (1-t) g(0) \tag{15.2}$$

for every $t \in [0,1]$.

Proof.

(i)$\Longrightarrow$(ii): If f is convex, then for any $a, b \in [0,1]$ and any $\lambda \in [0,1]$ we have

$$\begin{aligned} g(\lambda a + (1-\lambda)b) &= f((\lambda a + (1-\lambda)b)\mathbf{x}_1 + (1 - \lambda a - (1-\lambda)b)\mathbf{x}_2) \\ &= f(\lambda(a\mathbf{x}_1 + (1-a)\mathbf{x}_2) + (1-\lambda)(b\mathbf{x}_1 + (1-b)\mathbf{x}_2)) \\ &\leq \lambda f(a\mathbf{x}_1 + (1-a)\mathbf{x}_2) + (1-\lambda) f(b\mathbf{x}_1 + (1-b)\mathbf{x}_2) \\ &= \lambda g(a) + (1-\lambda) g(b). \end{aligned}$$

Therefore, g is convex.

(ii)$\Longrightarrow$(iii): If g is convex, then (15.2) is just a special case of the definition of convexity; that is, taking $a = 1$ and $b = 0$, convexity gives

$$g(t) = g(ta + (1-t)b) \leq t g(a) + (1-t) g(b) = t g(1) + (1-t) g(0).$$

(iii)$\Longrightarrow$(i): Finally, for any $\mathbf{x}_1, \mathbf{x}_2 \in C$ and $\lambda \in [0,1]$ we have

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) = g(\lambda) \leq \lambda g(1) + (1-\lambda) g(0) = \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2),$$

so f is convex on C and can be extended to f on V via Remark 15.1.8. □

Remark 15.1.11. The previous lemma also holds with strict convexity instead of convexity for f and g if the inequality in (15.2) is made strict for all $t \in (0,1)$. The proof is essentially identical.

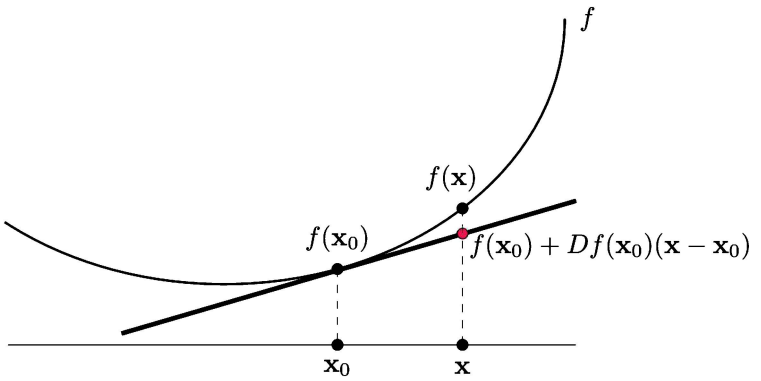


Figure 15.2. *For any differentiable function f on a convex set C, Theorem 15.1.12 shows f is convex if and only if $f(\mathbf{x}) \geq f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$ for all $\mathbf{x}_0, \mathbf{x} \in C$.*

Theorem 15.1.12. *Let f be a real-valued differentiable function on a convex open set $C \subset V$ of a finite-dimensional normed linear*[56] *space $(V, \|\cdot\|)$. The function f is convex if and only if for all $\mathbf{x}_0, \mathbf{x} \in C$, we have*

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0); \tag{15.3}$$

see Figure 15.2. Moreover, f is strictly convex if and only if the strict inequality holds in (15.3) *whenever $\mathbf{x}_0 \neq \mathbf{x}$.*

Proof. If f is convex, then (15.2) holds with $\mathbf{x}_1 = \mathbf{x}_0$ and $\mathbf{x}_2 = \mathbf{x}$ in the definition of $g(t)$. Thus, setting $g(\lambda) = f(\lambda\mathbf{x}_0 + (1-\lambda)\mathbf{x})$, we have

$$(1-\lambda)g(0) - (1-\lambda)g(1) \geq g(\lambda) - g(1).$$

Solving for $g(0)$ gives

$$g(0) \geq g(1) + \frac{g(\lambda) - g(1)}{1-\lambda}.$$

Taking the limit as $\lambda \to 1^-$ yields $g(0) \geq g(1) - g'(1)$, which gives (15.3).

For the converse, choose $\mathbf{x}_1, \mathbf{x}_2 \in C$ and $0 \leq \lambda \leq 1$. Let $\mathbf{x}_0 = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$. Using (15.3) twice gives

$$f(\mathbf{x}_1) \geq f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x}_1 - \mathbf{x}_0) \quad \text{and} \quad f(\mathbf{x}_2) \geq f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x}_2 - \mathbf{x}_0).$$

Multiplying the first inequality by λ and the second by $(1-\lambda)$ and adding yields

$$\begin{aligned}\lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) &\geq f(\mathbf{x}_0) + Df(\mathbf{x}_0)\left(\lambda(\mathbf{x}_1 - \mathbf{x}_0) + (1-\lambda)(\mathbf{x}_2 - \mathbf{x}_0)\right) \\ &= f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 - \mathbf{x}_0) \\ &= f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) + Df(\mathbf{x}_0)(\mathbf{0}).\end{aligned}$$

Thus f is convex. The proof for the strict inequality case is similar. □

[56] The theorem and the proof given here both hold for the more general situation where $(V, \|\cdot\|)$ is a Banach space (see Volume 1, Sections 5.6.2 and 6.3).

The preceding theorem is very useful, but it is still often difficult to verify that a function is convex by using either the definition or this theorem. The next theorem gives a simple method for verifying convexity in many cases.

Theorem 15.1.13. *Let C be a convex open set in a finite-dimensional normed linear*[57] *space $(V, \|\cdot\|)$. A function $f \in C^2(C;\mathbb{R})$ is convex on C if and only if the Hessian $D^2 f(\mathbf{x})$ is positive semidefinite for all $\mathbf{x} \in C$. The function f is strictly convex if $D^2 f(\mathbf{x})$ is positive definite for all $\mathbf{x} \in C$.*

Proof. Taylor's theorem (Theorem 10.3.8) guarantees that for each $\mathbf{x}_0, \mathbf{x} \in C$ we have

$$f(\mathbf{x}) = f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + R_2, \tag{15.4}$$

where

$$R_2 = \int_0^1 (1-t)(\mathbf{x} - \mathbf{x}_0)^{\mathsf{T}} D^2 f(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0))(\mathbf{x} - \mathbf{x}_0)\, dt.$$

If $D^2 f(\mathbf{x}') \geq 0$ for all $\mathbf{x}' \in C$, we have $R_2 \geq 0$ and

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0).$$

Hence, f is convex. In the case that $D^2 f(\mathbf{x}') > 0$, then f is strictly convex.

Conversely, suppose that f is convex but for some $\mathbf{x}_0 \in C$ the Hessian $D^2 f(\mathbf{x}_0)$ is not positive semidefinite. Then there exists some $\mathbf{h} \in V$ such that $\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{x}_0)\mathbf{h} < 0$. Choose ε sufficiently small so that $\mathbf{x} = \mathbf{x}_0 + \varepsilon\mathbf{h} \in C$ and so that $\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{z})\mathbf{h} < 0$ for all $\mathbf{z}$ on the line segment between $\mathbf{x}_0$ and $\mathbf{x}$. The remainder R_2 in (15.4) becomes

$$\varepsilon^2 \int_0^1 (1-t)\mathbf{h}^{\mathsf{T}} D^2 f(\mathbf{x}_0 + t\varepsilon\mathbf{h})\mathbf{h}\, dt < 0,$$

which implies

$$f(\mathbf{x}) < f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0).$$

Thus, f is not convex. □

Example 15.1.14. The following are examples of convex functions:

(i) Let $f : \mathbb{R} \to \mathbb{R}$ be given by $f(x) = e^{ax}$ for fixed $a \neq 0$. The function f is strictly convex because $f''(x) = a^2 e^{ax} > 0$.

(ii) Let $f : (0,\infty) \to (0,\infty)$ be given by $f(x) = x^a$, where either $a > 1$ or $a < 0$. Note that $f''(x) = a(a-1)x^{a-2} > 0$, so f is strictly convex.

(iii) Let $f : (0,\infty) \to \mathbb{R}$ be given by $f(x) = -\log x$. The function f is strictly convex because $f''(x) = \frac{1}{x^2}$.

[57] As in the case of Theorem 15.1.12, this theorem and the proof given here hold for a more general Banach space.

Example 15.1.15. The *LogSumExp* function $f : \mathbb{R}^n \to (0, \infty)$ is given by $f(\mathbf{x}) = \log(e^{x_1} + \cdots + e^{x_n})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. From (10.29) we can show that the Hessian is

$$D^2 f(\mathbf{x}) = \frac{(\mathbb{1}^\mathsf{T}\mathbf{z})\,\mathrm{diag}(\mathbf{z}) - \mathbf{z}\mathbf{z}^\mathsf{T}}{(\mathbb{1}^\mathsf{T}\mathbf{z})^2},$$

where $\mathbf{z} = (e^{x_1}, e^{x_2}, \ldots, e^{x_n})$. If $\mathbf{h} = (h_1, h_2, \ldots, h_n)$, then the Cauchy–Schwarz inequality (with $\mathbf{a} = (\sqrt{z_1}, \ldots, \sqrt{z_n})$ and $\mathbf{b} = (h_1\sqrt{z_1}, \ldots, h_n\sqrt{z_n})$) gives

$$(\mathbf{h}^\mathsf{T}\mathbf{z})^2 = \left(\sum_{k=1}^n e^{x_k} h_k\right)^2 \leq \left(\sum_{k=1}^n e^{x_k}\right)\left(\sum_{k=1}^n e^{x_k} h_k^2\right) = (\mathbb{1}^\mathsf{T}\mathbf{z})(\mathbf{h}^\mathsf{T}\,\mathrm{diag}(\mathbf{z})\mathbf{h}),$$

which implies that $\mathbf{h}^\mathsf{T} D^2 f(\mathbf{x})\mathbf{h} \geq 0$. Therefore f is convex.

15.2 Jensen's Inequality

In this section, we show that the set of all points that lie on or above the graph of a convex function (the *epigraph* of the function) is a convex set. This leads to an easy proof of the finite form of *Jensen's inequality*, which says that any convex combination of a finite collection of points on the graph of f lies on or above the graph.

There is also an integral form of Jensen's inequality, where the finite sum is replaced with an integral (see Theorem 15.2.13). To show this we first prove that every convex function f with a closed epigraph is the supremum of all the affine functions whose graphs lie below the graph of f; see Theorem 15.2.12.

15.2.1 Epigraphs

The set of all points on or above the graph of a function is called the *epigraph* of the function; see Figure 15.3 for an illustration. Theorem 15.2.3 shows that the epigraph is convex if and only if the function is convex.

Definition 15.2.1. *The* graph *of a function $f : V \to \mathbb{R}_\infty$ is the set*

$$\Gamma_f = \{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in \mathrm{effd}(f)\} \subset \mathrm{effd}(f) \times \mathbb{R}.$$

The epigraph of the function f is the set

$$\mathrm{epi}(f) = \{(\mathbf{x}, y) \mid \mathbf{x} \in \mathrm{effd}(f), y \geq f(\mathbf{x})\} \subset \mathrm{effd}(f) \times \mathbb{R}.$$

Remark 15.2.2. Theorem 15.1.12 shows that if f is convex and differentiable at $\mathbf{x}_0$, then the tangent plane $h(\mathbf{x}) = f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$ supports the epigraph of f at $\mathbf{x}_0$ because it contains the point $(\mathbf{x}_0, f(\mathbf{x}_0))$ and the epigraph of the function is in the half space supported by h; see Figure 15.4.

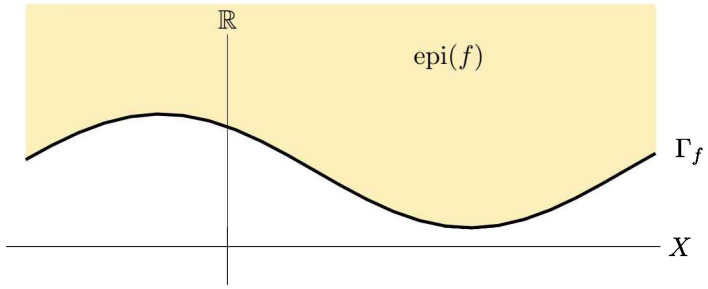


Figure 15.3. *The epigraph* epi(f) *(yellow) and the graph* Γ_f *of a function* f. *Theorem 15.2.3 guarantees that a function is convex if and only if the epigraph of the function is a convex set. The function and the epigraph in this figure are not convex.*

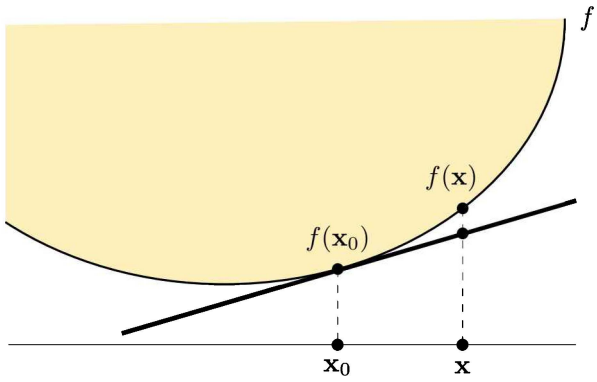


Figure 15.4. *For any differentiable convex function* f, *Theorem 15.1.12 shows that the tangent plane at any point* $(\mathbf{x}_0, f(\mathbf{x}_0))$ *of the graph of* f *is a supporting hyperplane of the epigraph.*

Theorem 15.2.3. *A function* $f : V \to \mathbb{R}_\infty$ *with a nonempty effective domain is convex if and only if the epigraph* epi(f) $\subset V \times \mathbb{R}$ *is a convex set.*

Proof. If f is convex on V, then, since there exists $\mathbf{x} \in \text{effd}(f)$, we have $f(\mathbf{x}) < \infty$, and hence $(\mathbf{x}, f(\mathbf{x})) \in \text{epi}(f)$. Therefore, $\text{epi}(f) \neq \emptyset$. If $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2) \in \text{epi}(f)$, then $f(\mathbf{x}_1) \leq y_1$ and $f(\mathbf{x}_2) \leq y_2$. Thus, for $0 \leq \lambda \leq 1$, we have

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \leq \lambda y_1 + (1-\lambda) y_2,$$

which implies that $\lambda(\mathbf{x}_1, y_1) + (1-\lambda)(\mathbf{x}_2, y_2) \in \text{epi}(f)$.

Conversely, assume that epi(f) is a convex set, that $\mathbf{x}_1, \mathbf{x}_2 \in V$, and that $0 \leq \lambda \leq 1$. If $f(\mathbf{x}_1) = \infty$ or $f(\mathbf{x}_2) = \infty$, then the convexity condition (15.1) holds immediately. Otherwise, $(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)) \in \text{epi}(f)$, and thus we have

$$\lambda(\mathbf{x}_1, f(\mathbf{x}_1)) + (1-\lambda)(\mathbf{x}_2, f(\mathbf{x}_2)) = (\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2, \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)) \in \text{epi}(f),$$

which implies that $f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$. □

Proposition 15.2.4. *Let $f : V \to \mathbb{R}_\infty$ be convex. If $\operatorname{effd}(f) \subset V$ is a closed set and $f : \operatorname{effd}(f) \to \mathbb{R}$ is continuous, then $\operatorname{epi}(f)$ is also a closed set.*

Proof. Assume that $((\mathbf{x}_i, y_i))_{i=0}^\infty \subset \operatorname{epi}(f)$ converges to $(\mathbf{x}^*, y^*) \in V \times \mathbb{R}$. It suffices to show that $(\mathbf{x}^*, y^*) \in \operatorname{epi}(f)$. The projection $p_1 : V \times \mathbb{R} \to V$ mapping $(\mathbf{x}, y) \mapsto \mathbf{x}$ is a continuous function; hence $\mathbf{x}_i = p_1((\mathbf{x}_i, y_i)) \to p_1((\mathbf{x}^*, y^*) = \mathbf{x}^*$. Since $\operatorname{effd}(f)$ is closed, we must have $\mathbf{x}^* \in \operatorname{effd}(f)$. Since each $(\mathbf{x}_i, y_i)$ lies in $\operatorname{epi}(f)$ we have $y_i \geq f(\mathbf{x}_i)$. Taking limits and using the fact that f is continuous on $\operatorname{effd}(f)$ implies that $y^* = \lim_{i\to\infty} y_i \geq \lim_{i\to\infty} f(\mathbf{x}_i) = f(\lim_{i\to\infty} \mathbf{x}_i) = f(\mathbf{x}^*)$. Therefore, $(\mathbf{x}^*, y^*) \in \operatorname{epi}(f)$. □

Remark 15.2.5. Exercise 15.15 shows that every convex function is continuous on the interior of its effective domain, hence the continuity hypothesis of the previous proposition can fail only at the boundary of $\operatorname{effd}(f)$.

15.2.2 Jensen's Inequality—Finite Form

Convexity of the epigraph gives us Jensen's inequality, which is one of the most important inequalities in analysis. An illustration of Jensen's inequality is given in Figure 15.5.

Theorem 15.2.6 (Jensen's Inequality—Finite Form). *Let $f : V \to \mathbb{R}_\infty$ be a convex function. If $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in V$ and $\lambda_1, \lambda_2, \ldots, \lambda_n \in [0, 1]$ with $\sum_{i=1}^n \lambda_i = 1$, then*

$$f(\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2 + \cdots + \lambda_n \mathbf{x}_n) \leq \lambda_1 f(\mathbf{x}_1) + \lambda_2 f(\mathbf{x}_2) + \cdots + \lambda_n f(\mathbf{x}_n). \tag{15.5}$$

Proof. If $f(\mathbf{x}_i) = \infty$ for any i with $\lambda_i > 0$, then (15.5) holds immediately. Assume now that $f(\mathbf{x}_i) < \infty$ for all $i \in \{1, \ldots, n\}$. The points $(\mathbf{x}_1, f(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, f(\mathbf{x}_n))$ all lie on the graph of f, and thus they lie in the epigraph of f. Theorem 15.2.3 guarantees that the epigraph of f is convex, and hence the convex combination

$$\lambda_1(\mathbf{x}_1, f(\mathbf{x}_1)) + \lambda_2(\mathbf{x}_1, f(\mathbf{x}_2)) + \cdots + \lambda_n(\mathbf{x}_n, f(\mathbf{x}_n))$$

must also lie in the epigraph (by Proposition 13.1.6). By definition of the epigraph, this means that (15.5) holds. □

Corollary 15.2.7 (Young's Inequality). *If $a, b \geq 0$ and $\frac{1}{p} + \frac{1}{q} = 1$, where $1 < p, q < \infty$, then*

$$ab \leq \frac{a^p}{p} + \frac{b^q}{q}. \tag{15.6}$$

Proof. Since $x \mapsto e^x$ is convex, we have

$$ab = e^{\log ab} = e^{\frac{1}{p}\log a^p + \frac{1}{q}\log b^q} \leq \frac{1}{p} e^{\log a^p} + \frac{1}{q} e^{\log b^q} = \frac{a^p}{p} + \frac{b^q}{q}. \quad \square$$

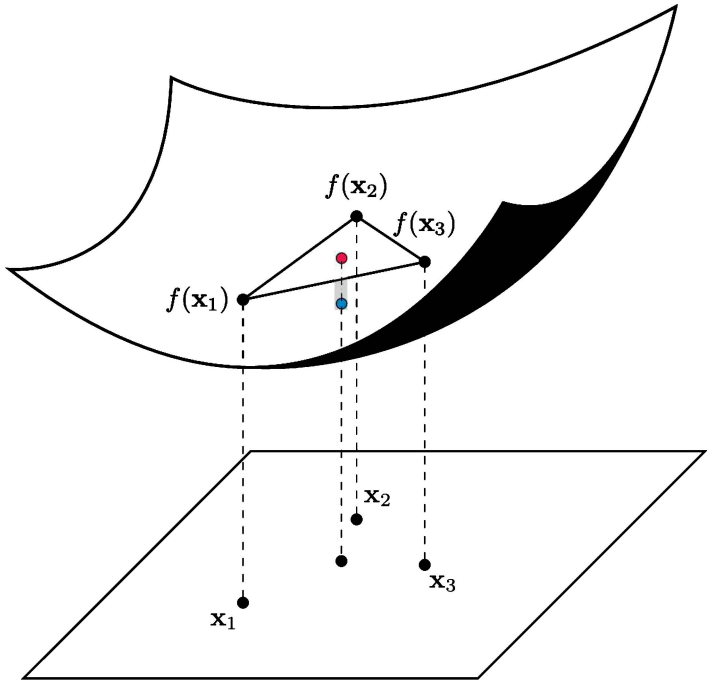


Figure 15.5. *The graph of a convex function f and three points $f(\mathbf{x}_1)$, $f(\mathbf{x}_2)$, $f(\mathbf{x}_3)$ on the graph. Jensen's inequality guarantees that a convex combination $\lambda_1 f(\mathbf{x}_1)+\lambda_2 f(\mathbf{x}_2)+\lambda_3 f(\mathbf{x}_3)$ (red) of the three function values lies above the function evaluated at the convex combination $f(\lambda_1\mathbf{x}_1+\lambda_2\mathbf{x}_2+\lambda_3\mathbf{x}_3)$ (blue), and, in fact, every point in the convex span of these points (the white triangle) lies above the graph (gray).*

Remark 15.2.8. Two other important inequalities that follow from Jensen's inequality are as follows:

(i) The arithmetic-geometric mean inequality: $\frac{1}{n}\sum_{i=1}^n x_i \geq \prod_{i=1}^n x_i^{1/n}$. This follows almost immediately from Jensen's inequality applied to the convex function $-\log(x)$.

(ii) Hölder's inequality: $\sum_{i=1}^n |x_i y_i| \leq (\sum_{i=1}^n |x_i|^p)^{\frac{1}{p}} (\sum_{i=1}^n |y_i|^q)^{\frac{1}{q}}$. This follows (with a little work) from Jensen's inequality applied to the convex function $x^{\frac{1}{p}}$.

Hölder's and Young's inequalities are discussed in more detail in Volume 1, Section 3.6.

15.2.3 Jensen's Inequality—Integral Form

A version of Jensen's inequality holds even when the finite sum is replaced with an integral over an infinite set. To prove this we first show that if a convex function

f has a closed epigraph, then f is the supremum of all the affine functions whose graphs lie on or below the graph of f.

Theorem 15.2.9. *Given a collection $\{f_\alpha\}_{\alpha\in J}$ of convex functions $f_\alpha : V \to \mathbb{R}_\infty$, the function $f(\mathbf{x}) = \sup_{\alpha\in J} f_\alpha(\mathbf{x})$ is convex.*

Proof. Let $\mathbf{x}, \mathbf{y} \in V$ and $\lambda \in [0,1]$. Since f_α is convex for each $\alpha \in J$, we have that

$$\begin{aligned} f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) &= \sup_{\alpha\in J} f_\alpha(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \\ &\leq \sup_{\alpha\in J}(\lambda f_\alpha(\mathbf{x}) + (1-\lambda) f_\alpha(\mathbf{y})) \\ &\leq \sup_{\alpha\in J}(\lambda f_\alpha(\mathbf{x})) + \sup_{\alpha\in J}((1-\lambda) f_\alpha(\mathbf{y})) \\ &= \lambda \sup_{\alpha\in J} f_\alpha(\mathbf{x}) + (1-\lambda) \sup_{\alpha\in J} f_\alpha(\mathbf{y}) \\ &= \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}). \quad \square \end{aligned}$$

Lemma 15.2.10. *If $\{f_\alpha\}_{\alpha\in J}$ is a collection of functions $f_\alpha : V \to \mathbb{R}_\infty$, then*

$$\operatorname{epi}(\sup_{\alpha\in J} f_\alpha) = \bigcap_{\alpha\in J} \operatorname{epi}(f_\alpha). \tag{15.7}$$

Proof. Since $f_\beta(\mathbf{x}) \leq \sup_{\alpha\in J} f_\alpha(\mathbf{x})$ for all $\beta \in J$ and all $\mathbf{x} \in V$, we have $\operatorname{epi}(\sup_{\alpha\in J} f_\alpha) \subset \operatorname{epi}(f_\beta)$ for all $\beta \in J$, which implies that $\operatorname{epi}(\sup_{\alpha\in J} f_\alpha) \subset \bigcap_{\beta\in J} \operatorname{epi}(f_\beta)$. Thus, it suffices to show that $\bigcap_{\beta\in J} \operatorname{epi}(f_\beta) \subset \operatorname{epi}(\sup_{\alpha\in J} f_\alpha)$. If $(\mathbf{x}, y) \in \operatorname{epi}(f_\beta)$ for all $\beta \in J$, then $y \geq f_\beta(\mathbf{x})$ for all $\beta \in J$. This implies that $y \geq \sup_{\alpha\in J} f_\alpha(\mathbf{x})$ for all $\alpha \in J$, and hence that $(\mathbf{x}, y) \in \operatorname{epi}(\sup_{\alpha\in J} f_\alpha)$. $\square$

Lemma 15.2.11. *If $f, g : V \to \mathbb{R}_\infty$ are functions with $\operatorname{epi}(f) = \operatorname{epi}(g)$, then $f = g$.*

Proof. If $f \neq g$, then without loss of generality, there exists an $\mathbf{x} \in V$ for which $f(\mathbf{x}) < g(\mathbf{x})$, but then $(\mathbf{x}, f(\mathbf{x})) \notin \operatorname{epi}(g)$, while $(\mathbf{x}, f(\mathbf{x})) \in \operatorname{epi}(f)$, which is a contradiction. $\square$

Theorem 15.2.12. *If $(V, \langle\cdot,\cdot\rangle)$ is an inner product space, then a convex function $f : V \to \mathbb{R}_\infty$ with a closed epigraph is the pointwise supremum of all hyperplanes that lie below the graph of f; that is, for each $\mathbf{x} \in V$, we have*

$$f(\mathbf{x}) = \sup_{h\in\mathscr{H}} h(\mathbf{x}),$$

where

$$\mathscr{H} = \{h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x}\rangle - b \mid \mathbf{a} \in V, b \in \mathbb{R}, \text{ with } h(\mathbf{x}) \leq f(\mathbf{x}) \text{ for all } \mathbf{x} \in V\}. \tag{15.8}$$

Proof. By Lemma 15.2.11 it suffices to show that $\operatorname{epi}(f) = \operatorname{epi}(\sup_{h\in\mathscr{H}} h)$. By Lemma 15.2.10 we have $\operatorname{epi}(\sup_{h\in\mathscr{H}} h) = \bigcap_{h\in\mathscr{H}} \operatorname{epi}(h)$, and thus it suffices to show that the set

$$H = \bigcap_{h\in\mathscr{H}} \operatorname{epi}(h)$$

is equal to $\operatorname{epi}(f)$. Since $\operatorname{epi}(f) \subset \operatorname{epi}(h)$ for each $h \in \mathscr{H}$, we have $\operatorname{epi}(f) \subset H$, so all that remains is to show that $H \subset \operatorname{epi}(f)$.

Consider $V \times \mathbb{R}$ with the inner product $\langle (\mathbf{v}, \alpha), (\mathbf{w}, \beta) \rangle = \langle \mathbf{v}, \mathbf{w} \rangle + \alpha\beta$. For any point $(\mathbf{x}_0, y_0) \notin \operatorname{epi}(f)$, we have $f(\mathbf{x}_0) > y_0$. Since $\operatorname{epi}(f)$ is closed, the separation lemma (Lemma 13.2.7) guarantees the existence of a half space containing $\operatorname{epi}(f)$ but not containing $(\mathbf{x}_0, y_0)$. This corresponds to a pair $(\mathbf{a}, \alpha) \in V \times \mathbb{R}$ and $b \in \mathbb{R}$ such that $\langle (\mathbf{a}, \alpha), (\mathbf{x}_0, y_0) \rangle = \langle \mathbf{a}, \mathbf{x}_0 \rangle + \alpha y_0 > b$ but $\langle (\mathbf{a}, \alpha), (\mathbf{x}, y) \rangle = \langle \mathbf{a}, \mathbf{x} \rangle + \alpha y \leq b$ for all $(\mathbf{x}, y) \in \operatorname{epi}(f)$. In particular, we have $\langle \mathbf{a}, \mathbf{x}_0 \rangle + \alpha f(\mathbf{x}_0) \leq b$.

Combining the separating conditions gives $b - \alpha f(\mathbf{x}_0) \geq \langle \mathbf{a}, \mathbf{x}_0 \rangle > b - \alpha y_0$, or $\alpha f(\mathbf{x}_0) < \alpha y_0$. But $f(\mathbf{x}_0) > y_0$, so $\alpha < 0$. Dividing the equations through by $|\alpha|$ we may assume that $\alpha = -1$. Letting $h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle - b$ gives $h(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x}$, so $h \in \mathscr{H}$. But $h(\mathbf{x}_0) > y_0$, so $(\mathbf{x}_0, y_0) \notin \operatorname{epi}(h)$; hence, $(\mathbf{x}_0, y_0) \notin H$. Therefore $H \subset \operatorname{epi}(f)$. □

Theorem 15.2.13 (Jensen's Inequality—Integral Form). *Let $C \subset \mathbb{R}^n$ be a convex set and $g : C \to \mathbb{R}$ be a nonnegative integrable function with $\int_C g(\mathbf{x})\, d\mathbf{x} = 1$. If $f : C \to \mathbb{R}$ is a convex function whose epigraph is closed in $V \times \mathbb{R}$, then*

$$f\left(\int_C \mathbf{x} g(\mathbf{x})\, d\mathbf{x}\right) \leq \int_C f(\mathbf{x}) g(\mathbf{x})\, d\mathbf{x}.$$

Proof. Extend f and g to be $\mathbb{R}_\infty$-valued functions on all of $\mathbb{R}^n$. As described in Remark 15.1.8, we have that $\operatorname{effd}(f) = \operatorname{effd}(g) = C$. Let $\mathscr{H}$ be defined as (15.8). For any $h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle - b \in \mathscr{H}$ we have $f(\mathbf{x}) \geq h(\mathbf{x})$ for all $\mathbf{x}$, which gives

$$\begin{aligned} \int_C f(\mathbf{x}) g(\mathbf{x})\, d\mathbf{x} \geq \int_C h(\mathbf{x}) g(\mathbf{x})\, d\mathbf{x} &= \left\langle \mathbf{a}, \int_C \mathbf{x} g(\mathbf{x})\, d\mathbf{x} \right\rangle - b \int_C g(\mathbf{x})\, d\mathbf{x} \\ &= h\left(\int_C \mathbf{x} g(\mathbf{x})\, d\mathbf{x}\right). \end{aligned} \tag{15.9}$$

Since (15.9) holds for all $h \in \mathscr{H}$, it must also hold for the supremum, which yields

$$\int_C f(\mathbf{x}) g(\mathbf{x})\, d\mathbf{x} \geq \sup_{h \in \mathscr{H}} h\left(\int_C \mathbf{x} g(\mathbf{x})\, d\mathbf{x}\right) = f\left(\int_C \mathbf{x} g(\mathbf{x})\, d\mathbf{x}\right),$$

as desired. Here the final equality holds by Theorem 15.2.12. □

Recall that a function $f : V \to \mathbb{R}_\infty$ that is continuous and convex on its effective domain has a closed epigraph if its effective domain is closed; see Proposition 15.2.4.

Corollary 15.2.14. *Let X be a continuous random variable taking values in a closed convex set $C \subset \mathbb{R}^n$. If $\phi : C \to \mathbb{R}$ is a continuous convex function, then*

$$\phi(\mathbb{E}[X]) \leq \mathbb{E}[\phi(X)].$$

Proof. The proof is Exercise 15.14. □

Example 15.2.15. The function $\phi(x) = e^x$ is both continuous and convex on $\mathbb{R}$, so by Jensen's inequality, for any univariate continuous random variable X we have

$$e^{\mathbb{E}[X]} \leq \mathbb{E}[e^X].$$

For $n \in \mathbb{N}$, we generalize to $\phi(x) = x^{2n}$, which is also continuous and convex, and thus $\mathbb{E}[X]^{2n} \leq \mathbb{E}[X^{2n}]$. In the special case of $n = 1$, we have that $\mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathrm{Var}(X) \geq 0$.

15.3 Fundamentals of Convex Optimization

Convexity has important consequences for optimization. Among other things, any local minimizer of a convex optimization problem is also a global minimizer, and that minimizer is unique. Therefore, any technique for finding a local minimizer also yields the global minimizer of a convex function.

There are also some very good numerical methods designed specifically for finding the minimizer of a convex optimization problem (some of these are described in Section 15.6), so for practical purposes, once a problem is formulated as a convex optimization problem it is essentially solved—at least in many cases. As a general rule, with current techniques, most convex optimization problems are relatively easy, whereas nonconvex problems are generally hard.

15.3.1 Definition and Examples

A convex optimization problem is one where the feasible set and the objective function are both convex.

Definition 15.3.1. *Consider an optimization problem in standard form*

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) && \text{(15.10a)}\\ &\textit{subject to} && G(\mathbf{x}) \preceq \mathbf{0}, && \text{(15.10b)}\\ &&& H(\mathbf{x}) = \mathbf{0}, && \text{(15.10c)} \end{aligned}$$

with f, G, H all defined on an open domain $\Omega \subset \mathbb{R}^n$ (f may take values in $\mathbb{R}_\infty$). This is a convex optimization problem *if* (i) *the objective f is a convex function,* (ii) *the set Ω is convex,* (iii) *all the inequality constraint functions $g_1, \ldots, g_m$, given by the components of G, are convex functions, and* (iv) *all of the equality constraint functions $h_1, \ldots, h_\ell$, given by the components of H, are affine functions.*

Remark 15.3.2. It is common to write the affine equality constraint $H(\mathbf{x}) = \mathbf{0}$ as $A\mathbf{x} - \mathbf{b} = \mathbf{0}$ for some $A \in M_{\ell \times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^\ell$.

Proposition 15.3.3. *The feasible set*

$$\mathscr{F} = \{\mathbf{x} \in \Omega \mid f(\mathbf{x}) < \infty, G(\mathbf{x}) \preceq \mathbf{0}, A\mathbf{x} - \mathbf{b} = \mathbf{0}\} \subset \Omega \tag{15.11}$$

of a convex optimization problem (15.10) *is a convex set.*

Proof. The proof is Exercise 15.17. □

Proposition 15.3.4. *Let $h : \Omega \to \mathbb{R}$ be a given function. The functions h and $-h$ are both convex if and only if h is affine.*

Proof. The proof is Exercise 15.18. □

Remark 15.3.5. Any equality constraint of the form $h(\mathbf{x}) = 0$ is equivalent to the two inequality constraints $h(\mathbf{x}) \leq 0$ and $-h(\mathbf{x}) \leq 0$. Proposition 15.3.4 shows that both $h(\mathbf{x})$ and $-h(\mathbf{x})$ are convex if and only if $h(\mathbf{x})$ is affine, so requiring equality constraints to be affine is equivalent to requiring both the corresponding inequality constraints to be convex. We usually find it more convenient to use equality constraints rather than the corresponding pairs of inequality constraints.

Example 15.3.6. In Remark 13.3.4 we showed that a linear optimization problem in standard form (13.4) can always be rewritten as

$$\begin{aligned} &\text{minimize} && \mathbf{c}^\mathsf{T}\mathbf{x} \\ &\text{subject to} && \widetilde{A}\mathbf{x} - \widetilde{\mathbf{b}} \preceq \mathbf{0}. \end{aligned}$$

Since the objective and the constraints are all affine and thus convex, this is a special case of a convex optimization problem.

Example 15.3.7. An important problem in machine learning is the task of classifying points $\mathbf{x} \in \mathbb{R}^d$ into one of two categories—this is called a binary classifier. This can be formulated as finding a function $f : \mathbb{R}^d \to \{\pm 1\}$, where $f(\mathbf{x})$ is $+1$ for one category and -1 for the other.

A simple example of such a classifier is a function of the form $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^\mathsf{T}\mathbf{x} - b)$, which amounts to finding the hyperplane and assigning any point $\mathbf{x}$ on one side of the hyperplane, say, when $\mathbf{w}^\mathsf{T}\mathbf{x} - b < 0$, to -1 and assigning a point on the other side, that is, when $\mathbf{w}^\mathsf{T}\mathbf{x} - b > 0$, to $+1$.

Given the training data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ in $\mathbb{R}^d \times \{\pm 1\}$, we want to find a suitable hyperplane $\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{w}^\mathsf{T}\mathbf{x} = b\}$ for the function f. *Support vector classifiers* provide some methods for doing this. Specifically, the *hard-margin* support vector classifier chooses $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ to

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\mathbf{w}\|_2^2 \\ &\text{subject to} && (1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)) \leq 0 \quad \forall i \in \{1, \ldots, N\}. \end{aligned}$$

The objective is convex and the constraints are all affine in $\mathbf{w}$ and b; therefore, this is a convex optimization problem. The data set D is called *linearly separable* if the feasible set $\mathscr{F}$ is not empty.

Example 15.3.8. If the data set D is not linearly separable, then the hard-margin optimization problem is infeasible. To overcome this, we *relax* the constraints and invoke a penalty for violating the constraints by using a *soft-margin* support vector classifier. This is an unconstrained problem to minimize what is called *regularized hinge loss*:

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x} + b))$$

for some fixed choice of $C > 0$. By Exercise 15.5 this is also a convex optimization problem.

15.3.2 Consequences of Convexity for Optimization

As mentioned above, one very important consequence of convexity is that any local minimizer is also a global minimizer.

Theorem 15.3.9. *If $\mathbf{x}^*$ is a local minimizer of a convex optimization problem* (15.10)*, then it is also a global minimizer.*

Proof. Let $\mathscr{F}$ be the feasible set (15.11) for the convex optimization problem (15.10). Let $\mathbf{x}^* \in X$ be a local minimizer; that is, assume there exists $\delta > 0$ such that

$$f(\mathbf{x}^*) = \min\{f(\mathbf{x}) \mid \mathbf{x} \in \mathscr{F} \cap B(\mathbf{x}^*, \delta)\}.$$

Suppose that $\mathbf{x}^*$ is not a global minimizer, so there exists $\mathbf{y} \in \mathscr{F}$ such that $f(\mathbf{y}) < f(\mathbf{x}^*)$. Choose $\mathbf{z} = (1-\lambda)\mathbf{x}^* + \lambda\mathbf{y}$, where $\lambda = \delta/(2\|\mathbf{x}^* - \mathbf{y}\|)$. Thus,

$$\|\mathbf{x}^* - \mathbf{z}\| = \|\mathbf{x}^* - (1-\lambda)\mathbf{x}^* - \lambda\mathbf{y}\| = \lambda\|\mathbf{x}^* - \mathbf{y}\| = \delta/2.$$

By convexity of $\mathscr{F}$, the point $\mathbf{z}$ must be feasible, and convexity of f implies

$$f(\mathbf{z}) \le (1-\lambda)f(\mathbf{x}^*) + \lambda f(\mathbf{y}) < (1-\lambda)f(\mathbf{x}^*) + \lambda f(\mathbf{x}^*) = f(\mathbf{x}^*).$$

This contradicts $\mathbf{x}^*$ as a local minimizer since $\mathbf{z} \in B(\mathbf{x}^*, \delta)$. Thus $\mathbf{x}^*$ is a global minimizer. ∎

Convexity also gives a first-order sufficient condition.

Theorem 15.3.10. *Assume that the objective function f in the convex optimization problem* (15.10) *lies in $C^1(\Omega;\mathbb{R})$. A point $\mathbf{x}^* \in \mathscr{F}$ is a minimizer if and only if*

$$Df(\mathbf{x}^*)(\mathbf{y} - \mathbf{x}^*) \ge 0 \quad \forall \mathbf{y} \in \mathscr{F}. \tag{15.12}$$

Proof. If (15.12) holds for $\mathbf{x}^* \in \mathscr{F}$, then, since f is convex, Theorem 15.1.12 guarantees that

$$f(\mathbf{y}) \ge f(\mathbf{x}^*) + Df(\mathbf{x}^*)(\mathbf{y} - \mathbf{x}^*) \quad \forall \mathbf{y} \in \mathscr{F}.$$

Combining with (15.12), we have that $f(\mathbf{y}) \geq f(\mathbf{x}^*)$ for all $\mathbf{y} \in \mathscr{F}$. Thus $\mathbf{x}^*$ is a minimizer.

For the converse observe that for every $\mathbf{y} \in \mathscr{F}$, convexity of $\mathscr{F}$ implies that the vector $\mathbf{x} + t(\mathbf{y} - \mathbf{x}^*)$ lies in $\mathscr{F}$ for all $t \in [0, 1]$. Assume, by way of contradiction, that $Df(\mathbf{x}^*)(\mathbf{y} - \mathbf{x}^*) < 0$. For the unit vector $\mathbf{v} = \frac{\mathbf{y}-\mathbf{x}^*}{\|\mathbf{y}-\mathbf{x}^*\|}$ the directional derivative $D_{\mathbf{v}} f(\mathbf{x}^*) = Df(\mathbf{x}^*)\mathbf{v}$ is negative. Since f is C^1, there must be some small ε such that the directional derivative $Df(\mathbf{x})\mathbf{v}$ must be negative for all $\mathbf{x}$ in $B(\mathbf{x}^*, \varepsilon) \cap \mathscr{F}$. Hence the function $f(t(\mathbf{y} - \mathbf{x}^*) + \mathbf{x}^*)$ is decreasing for $t \in [0, \varepsilon)$, and thus $\mathbf{x}^*$ is not a minimizer of f on $\mathscr{F}$ □

Remark 15.3.11. If $\mathbf{x}^*$ is an interior point of $\mathscr{F}$, then $Df(\mathbf{x}^*)(\mathbf{y} - \mathbf{x}^*) \geq 0$ holds for all $\mathbf{y}$ in a neighborhood of $\mathbf{x}^*$, which implies that $Df(\mathbf{x}^*) = \mathbf{0}$. In particular, when the convex optimization problem is unconstrained, we have that $\mathscr{F} = \Omega$ and every point of $\mathscr{F}$ is an interior point.

But if $\mathbf{x}^*$ is not an interior point, then (15.12) implies that for every $\mathbf{y} \in \mathscr{F}$ we have $-Df(\mathbf{x}^*)\mathbf{y} \leq -Df(\mathbf{x}^*)\mathbf{x}^*$. Setting $\mathbf{a} = -Df(\mathbf{x}^*)^\mathsf{T}$ and $b = -Df(\mathbf{x}^*)\mathbf{x}^*$ gives a half space $\{\mathbf{y} \mid \mathbf{a}^\mathsf{T}\mathbf{y} \leq b\}$ that supports $\mathscr{F}$.

15.3.3 Rewriting Problems as Convex Optimization Problems

As mentioned above, there are good algorithms for solving convex optimization problems, but nonconvex problems are generally hard. It is important, therefore, to recognize that many problems are actually convex optimization problems, even when they are not initially formulated as such. Unfortunately, there are no off-the-shelf methods for rewriting general problems as convex problems, but spending a little effort to look for a way to rewrite a problem as a convex problem can yield significant benefits. Among other things, this guarantees a unique minimizer, which must be global, and it means that you can take advantage of the effective numerical methods for solving convex optimization problems. We discuss some of these methods in Section 15.6.

Example 15.3.12. Consider the problem

$$\begin{aligned} \text{minimize} \quad & x^2 + y^2 \\ \text{subject to} \quad & (x+y)^2 = 0, \\ & \tfrac{x}{1+y^2} \leq 0. \end{aligned}$$

Although the objective function is convex, the constraints are not. However, we can recast this as a convex problem in standard form as follows. First, $(x+y)^2 = 0$ is equivalent to $x + y = 0$, and second, $1 + y^2$ is always positive, so $\frac{x}{1+y^2} \leq 0$ is equivalent to $x \leq 0$. Therefore, the original problem is equivalent to the convex optimization problem

$$\begin{aligned} \text{minimize} \quad & x^2 + y^2 \\ \text{subject to} \quad & x + y = 0, \\ & x \leq 0. \end{aligned}$$

The following easy proposition is often used for rewriting optimization problems.

Proposition 15.3.13. *If $D \subset \mathbb{R}$ with $f : \mathbb{R}^n \to D$, and if $\phi : D \to \mathbb{R}$ is a strictly increasing function, then $\mathbf{x}^*$ is a local minimizer for the problem*

$$\begin{aligned} &\text{minimize} && \phi \circ f(\mathbf{x}) \\ &\textit{subject to} && G(\mathbf{x}) \preceq \mathbf{0}, \\ &&& H(\mathbf{x}) = \mathbf{0} \end{aligned}$$

if and only if $\mathbf{x}^*$ is a local minimizer for the problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\textit{subject to} && G(\mathbf{x}) \preceq \mathbf{0}, \\ &&& H(\mathbf{x}) = \mathbf{0}. \end{aligned}$$

Proof. The proof is Exercise 15.19. □

Example 15.3.14. The function $f(x) = \log|3x - 5|$ is not convex, but since $\log|3x - 5| = \frac{1}{2}\log(3x - 5)^2$ and the function $\phi(z) = \frac{1}{2}\log(z)$ is strictly increasing on $[0, \infty)$ we can use Proposition 15.3.13 to recast the optimization problem. Hence the optimization problem

$$\begin{aligned} &\text{minimize} && \log|3x - 5| \\ &\text{subject to} && G(x) \preceq \mathbf{0}, \\ &&& H(x) = \mathbf{0} \end{aligned}$$

has the same minimizer as the problem

$$\begin{aligned} &\text{minimize} && (3x - 5)^2 \\ &\text{subject to} && G(\mathbf{x}) \preceq \mathbf{0}, \\ &&& H(\mathbf{x}) = \mathbf{0}, \end{aligned}$$

which has a convex objective function.

Example 15.3.15. ⚠ An important optimization problem in machine learning is the problem of computing the parameters for *logistic regression*. This arises when we have data of the form $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$, drawn from a distribution with

$$P(Y = 1 \,|\, \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x} + b)}} \quad \text{and} \quad P(Y = -1 \,|\, \mathbf{x}) = \frac{1}{1 + e^{(\mathbf{w}^\top \mathbf{x} + b)}}$$

for some choice of parameters $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. This can be written more cleanly as

$$P(Y = y \,|\, \mathbf{x}) = \frac{1}{1 + e^{-y(\mathbf{w}^\top \mathbf{x} + b)}}.$$

The MLEs are

$$(\widehat{\mathbf{w}}, \widehat{b}) = \operatorname*{argmax}_{\mathbf{w},b} \prod_{i=1}^{N} \frac{1}{1 + e^{-y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)}}.$$

This is not a convex optimization problem. But taking logarithms and putting the new problem in standard form gives

$$(\widehat{\mathbf{w}}, \widehat{b}) = \operatorname*{argmin}_{\mathbf{w},b} \sum_{i=1}^{N} \log(1 + e^{-y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)}).$$

By Exercise 15.6 each summand of the objective is convex, so the objective is also convex. Therefore, this is an unconstrained convex optimization problem.

Remark 15.3.16. Given a solution $(\widehat{\mathbf{w}}, \widehat{b})$ to this problem, we can define a classifier $f : \mathbb{R}^d \to \{\pm 1\}$ by

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } P(Y = 1 \,|\, \mathbf{x}) > 0.5, \\ -1 & \text{if } P(Y = 1 \,|\, \mathbf{x}) < 0.5. \end{cases}$$

It is straightforward to check that this classifier satisfies $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$, which looks very similar to the support vector classifiers of Examples 15.3.7 and 15.3.8. Note, however, that $\widehat{\mathbf{w}}$ and $\widehat{b}$ are computed differently and so the hyperplanes are generally not the same.

Example 15.3.17. The *Chebyshev approximation* for linear regression uses the ∞-norm instead of the 2-norm; that is, given $A \in M_{m\times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$, we have the unconstrained optimization problem

$$\text{minimize} \quad \|A\mathbf{x} - \mathbf{b}\|_\infty.$$

This can be written as a linear (hence convex) optimization problem as follows.

For any $t \geq \|A\mathbf{x} - \mathbf{b}\|_\infty$, if $\mathbb{1} = (1, 1, \ldots, 1)$ is the all-ones vector, then we have $|A\mathbf{x} - \mathbf{b}| \preceq t\mathbb{1}$, so the smallest t satisfying $|A\mathbf{x} - \mathbf{b}| \preceq t\mathbb{1}$ is $\|A\mathbf{x} - \mathbf{b}\|_\infty$. Therefore, the problem is equivalently given as

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & A\mathbf{x} - t\mathbb{1} - \mathbf{b} \preceq \mathbf{0}, \\ & -A\mathbf{x} - t\mathbb{1} + \mathbf{b} \preceq \mathbf{0}. \end{aligned}$$

Since the objective function and the constraints are both affine, this is a convex optimization problem.

Remark 15.3.18. If we use the 1-norm in the regression problem above, instead of the ∞-norm, the problem is instead called *robust regression* or *ℓ^1-minimization*. The robust regression problem can also be formulated as a linear optimization problem; see Exercise 15.22 for details.

15.4 Weak Duality

Just as in the case of weak duality for linear optimization, weak duality in convex optimization gives a lower bound on a given minimization problem in terms of a dual maximization problem. In this section we describe the dual problem and prove that weak duality holds. In the subsequent section we show that *strong duality* also holds in many cases, which means that the optimal value of the dual problem is the same as the optimal value of the primal problem. In some cases this dual problem is easier to solve than the primal problem. The ideas of duality are also important in the *interior point methods* for convex optimization, which we discuss at the end of this chapter.

Throughout this section, unless otherwise stated, we assume functions take values in $\mathbb{R}_{\pm\infty}$.

15.4.1 The Lagrange Dual

Recall from (14.1) the (not necessarily convex) constrained minimization problem

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) && \text{(15.13a)} \\ \text{subject to} \quad & G(\mathbf{x}) \preceq \mathbf{0}, && \text{(15.13b)} \\ & H(\mathbf{x}) = \mathbf{0}, && \text{(15.13c)} \end{aligned}$$

where $f : \Omega \to \mathbb{R}_\infty$ is C^1 on $\operatorname{effd}(f)$, and the constraints $G : \mathbb{R}^n \to \mathbb{R}^m$ and $H : \mathbb{R}^n \to \mathbb{R}^\ell$ form the feasible set $\mathscr{F} \subset \Omega$, where $\Omega \subset \mathbb{R}^n$ is an open set. We denote the inequality constraint functions $g_1, \dots, g_m$ from the components of G and equality constraint functions $h_1, \dots, h_\ell$ from the components of H.

Let $\mathbf{x}^* \in \mathscr{F}$ be the minimizer of f with $p^* = f(\mathbf{x}^*)$. As defined in (14.17), the Lagrangian $\mathscr{L} : \Omega \times \mathbb{R}^\ell \times \mathbb{R}^m \to \mathbb{R}$ can be interpreted as giving a penalty to values of $\mathbf{x}$ that violate the constraints. For example, whenever the constraint $G(\mathbf{x}) \preceq \mathbf{0}$ is violated with $g_i(\mathbf{x}) > 0$, then for any $\boldsymbol{\mu} \succ \mathbf{0}$, we have $\mu_i g_i(\mathbf{x}) > 0$, and the larger μ_i is, the greater the penalty for violating the constraint $g_i(\mathbf{x}) \le 0$. Following this idea, we can use the Lagrangian to make a new unconstrained problem that is equivalent to the original constrained problem, as follows. First, for any $\mathbf{x}$ observe that

$$\sup_{\boldsymbol{\lambda}} \boldsymbol{\lambda}^\mathsf{T} H(\mathbf{x}) = \begin{cases} 0 & \text{if } H(\mathbf{x}) = \mathbf{0}, \\ \infty & \text{otherwise} \end{cases} \tag{15.14}$$

and

$$\sup_{\boldsymbol{\mu} \succeq \mathbf{0}} \boldsymbol{\mu}^\mathsf{T} G(\mathbf{x}) = \begin{cases} 0 & \text{if } G(\mathbf{x}) \preceq \mathbf{0}, \\ \infty & \text{otherwise.} \end{cases} \tag{15.15}$$

If $F(\mathbf{x}) = \sup_{\boldsymbol{\mu} \succeq \mathbf{0}, \boldsymbol{\lambda}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, then the constrained optimization problem (15.13) can be rewritten as the unconstrained optimization problem

$$\text{minimize} \quad F(\mathbf{x}). \tag{15.16}$$

One of the benefits of duality for linear optimization problems is that it gives a bound on the optimal value. In particular, the following *minimax inequality* gives a bound on the optimal value of the nonlinear problem (15.13).

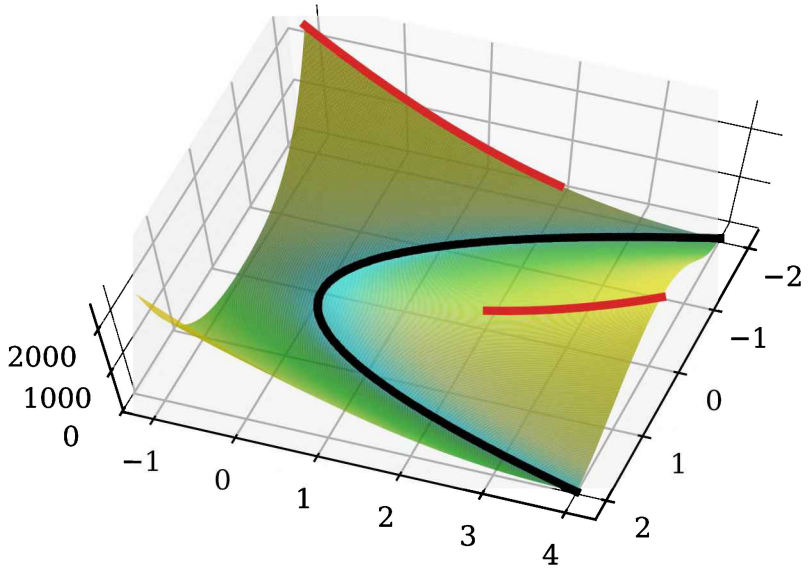


Figure 15.6. *An illustration of the minimax inequality for $\Phi(x, y)$ equal to the Rosenbrock function. For each $x \in [-2, 2]$ (right axis) let $\overline{y}(x) = \operatorname{argmin}_y \Phi(x, y)$. The black curve is the plot of all points of the form $(x, \overline{y}(x), \min_y \Phi(x, y))$. Similarly, for each $y \in [-1, 4]$ (lower left axis) let $\overline{x}(y) = \operatorname{argmax}_x \Phi(x, y)$. The red curve is the plot of all points of the form $(\overline{x}(y), y, \max_x \Phi(x, y))$. The minimax inequality guarantees that the highest point of the black curve is never greater than the lowest point of the red curve.*

Proposition 15.4.1 (Minimax Inequality). *For any sets X, Y and any function $\Phi : X \times Y \to \mathbb{R}$ the following inequality holds:*

$$\sup_{\mathbf{x} \in X} \inf_{\mathbf{y} \in Y} \Phi(\mathbf{x}, \mathbf{y}) \leq \inf_{\mathbf{y} \in Y} \sup_{\mathbf{x} \in X} \Phi(\mathbf{x}, \mathbf{y}). \tag{15.17}$$

See Figure 15.6 for an illustration.

Proof. For every $\mathbf{x} \in X$ and every $\mathbf{y} \in Y$, we have

$$\inf_{\mathbf{y}' \in Y} \Phi(\mathbf{x}, \mathbf{y}') \leq \Phi(\mathbf{x}, \mathbf{y}) \leq \sup_{\mathbf{x}' \in X} \Phi(\mathbf{x}', \mathbf{y}). \tag{15.18}$$

Taking the infimum over all values of $\mathbf{y} \in Y$ on the right gives

$$\inf_{\mathbf{y}' \in Y} \Phi(\mathbf{x}, \mathbf{y}') \leq \inf_{\mathbf{y} \in Y} \sup_{\mathbf{x}' \in X} \Phi(\mathbf{x}', \mathbf{y}).$$

Taking the supremum over all values of $\mathbf{x} \in X$

$$\sup_{\mathbf{x} \in X} \inf_{\mathbf{y}' \in Y} \Phi(\mathbf{x}, \mathbf{y}') \leq \inf_{\mathbf{y} \in Y} \sup_{\mathbf{x}' \in X} \Phi(\mathbf{x}', \mathbf{y}). \quad \square$$

Remark 15.4.2. The minimax inequality applied to the unconstrained optimization problem (15.16) shows that the optimal value $p^* = f(\mathbf{x}^*)$ of this problem satisfies

$$d^* = \sup_{\boldsymbol{\mu} \succeq \mathbf{0}, \boldsymbol{\lambda}} \inf_{\mathbf{x} \in \mathbb{R}^n} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \inf_{\mathbf{x} \in \mathbb{R}^n} \sup_{\boldsymbol{\mu} \succeq \mathbf{0}, \boldsymbol{\lambda}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = p^*. \tag{15.19}$$

This motivates the following definition.

Definition 15.4.3. *The* Lagrange dual function $\widetilde{f} : \mathbb{R}^\ell \times \mathbb{R}^m \to \mathbb{R}_{\pm\infty}$ *is*

$$\widetilde{f}(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x} \in \mathbb{R}^n} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x} \in \mathbb{R}^n} \Big(f(\mathbf{x}) + \boldsymbol{\lambda}^\mathsf{T} H(\mathbf{x}) + \boldsymbol{\mu}^\mathsf{T} G(\mathbf{x}) \Big).$$

Example 15.4.4. Assume that $A \in M_{\ell \times n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^\ell$. Consider the optimization problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\mathbf{x}\|_2^2 \\ &\text{subject to} && A\mathbf{x} = \mathbf{b}. \end{aligned}$$

The Lagrangian is given by

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{2}\|\mathbf{x}\|_2^2 + \boldsymbol{\lambda}^\mathsf{T}(A\mathbf{x} - \mathbf{b}).$$

For any $\boldsymbol{\lambda} \in \mathbb{R}^\ell$ the minimizer of the Lagrangian $\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda})$ occurs where $D_\mathbf{x}\mathscr{L}$ vanishes, so the minimizer $\mathbf{x}$ satisfies

$$D_\mathbf{x}\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{x}^\mathsf{T} + \boldsymbol{\lambda}^\mathsf{T} A = \mathbf{0},$$

which implies $\mathbf{x} = -A^\mathsf{T}\boldsymbol{\lambda}$. Thus, the dual function satisfies

$$\begin{aligned} \widetilde{f}(\boldsymbol{\lambda}) &= \inf_{\mathbf{x} \in \mathbb{R}^n} \left(\frac{1}{2}\|\mathbf{x}\|_2^2 + \boldsymbol{\lambda}^\mathsf{T}(A\mathbf{x} - \mathbf{b}) \right) \\ &= \inf_{\mathbf{x} \in \mathbb{R}^n} \left(\frac{1}{2}(-A^\mathsf{T}\boldsymbol{\lambda})^\mathsf{T}(-A^\mathsf{T}\boldsymbol{\lambda}) + \boldsymbol{\lambda}^\mathsf{T}(A\mathbf{x} - \mathbf{b}) \right) \\ &= -\frac{1}{2}\boldsymbol{\lambda}^\mathsf{T} A A^\mathsf{T}\boldsymbol{\lambda} - \boldsymbol{\lambda}^\mathsf{T}\mathbf{b}. \end{aligned}$$

Example 15.4.5. Let $A \in M_{m \times n}(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. Consider the primal linear optimization problem (see (13.12))

$$\begin{aligned} &\text{minimize} && \mathbf{c}^\mathsf{T}\mathbf{x} \\ &\text{subject to} && A\mathbf{x} \preceq \mathbf{b}, \\ & && \mathbf{x} \succeq \mathbf{0}. \end{aligned} \tag{15.20}$$

The Lagrangian is

$$\mathscr{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}) = \mathbf{c}^\mathsf{T}\mathbf{x} + \mathbf{y}^\mathsf{T}(A\mathbf{x} - \mathbf{b}) - \boldsymbol{\mu}^\mathsf{T}\mathbf{x},$$

where both $\mathbf{y}$ and $\boldsymbol{\mu}$ are nonnegative. The Lagrange dual function is

$$\begin{aligned}\widetilde{f}(\mathbf{y}, \boldsymbol{\mu}) &= \inf_{\mathbf{x}\in\mathbb{R}^n} \left(\mathbf{c}^\mathsf{T}\mathbf{x} + \mathbf{y}^\mathsf{T}(A\mathbf{x} - \mathbf{b}) - \boldsymbol{\mu}^\mathsf{T}\mathbf{x}\right) \\ &= \inf_{\mathbf{x}\in\mathbb{R}^n} \left(-\mathbf{b}^\mathsf{T}\mathbf{y} + (\mathbf{c} + A^\mathsf{T}\mathbf{y} - \boldsymbol{\mu})^\mathsf{T}\mathbf{x}\right) \\ &= \begin{cases} -\mathbf{b}^\mathsf{T}\mathbf{y} & \text{if } \mathbf{c} + A^\mathsf{T}\mathbf{y} - \boldsymbol{\mu} = \mathbf{0}, \\ -\infty & \text{otherwise.} \end{cases}\end{aligned}$$

Remark 15.4.6. We show below that the Lagrange dual function is always concave, even if the original problem was not, since it is the pointwise infimum of affine functions (and may equal $-\infty$ for some values of $\mathbf{x}$).

When expressed in terms of the original function f and its Lagrange dual $\widetilde{f}$, Remark 15.4.2 immediately gives the following important result.

Theorem 15.4.7 (Weak Duality). *For any $\boldsymbol{\lambda} \in \mathbb{R}^\ell$ and $\boldsymbol{\mu} \in \mathbb{R}^m$ with $\boldsymbol{\mu} \succeq \mathbf{0}$, if p^* is the minimal value of f on the feasible set, then $\widetilde{f}(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq p^*$.*

Remark 15.4.8. Weak duality guarantees that solving the *dual optimization problem*

$$\begin{aligned} &\text{maximize} && \widetilde{f}(\boldsymbol{\lambda}, \boldsymbol{\mu}) \\ &\text{subject to} && \boldsymbol{\mu} \succeq \mathbf{0} \end{aligned} \tag{15.21}$$

gives a lower bound for the original problem (15.13), hereafter called the *primal problem*. In particular, if

$$d^* = \sup_{\boldsymbol{\mu}\succeq\mathbf{0},\boldsymbol{\lambda}} \widetilde{f}(\boldsymbol{\lambda}, \boldsymbol{\mu})$$

is the solution to the dual problem, then $d^* \leq p^*$.

Example 15.4.9. Consider again the problem from Example 15.4.4. By the FONC, the maximizer $\boldsymbol{\lambda}^*$ of $\widetilde{f}$ satisfies

$$D_{\boldsymbol{\lambda}}\widetilde{f} = -\boldsymbol{\lambda}^\mathsf{T}AA^\mathsf{T} - \mathbf{b}^\mathsf{T} = \mathbf{0},$$

and thus $\boldsymbol{\lambda}^* = -(AA^\mathsf{T})^{-1}\mathbf{b}$, so the maximal value is $d^* = \frac{1}{2}\mathbf{b}^\mathsf{T}(AA^\mathsf{T})^{-1}\mathbf{b}$. Weak duality implies that $d^* \leq p^*$, but it is straightforward to check that $d^* = p^*$ in this example. Later in this chapter we show that a very broad class of convex optimization problems have the property that $d^* = p^*$.

Example 15.4.10. Consider the problem in Example 15.4.5. Maximizing the Lagrange dual function gives d^*, which is determined by solving

$$\begin{array}{ll} \text{maximize} & -\mathbf{b}^\mathsf{T}\mathbf{y} \\ \text{subject to} & A^\mathsf{T}\mathbf{y} + \mathbf{c} - \boldsymbol{\mu} = \mathbf{0}, \\ & \mathbf{y}, \boldsymbol{\mu} \succeq \mathbf{0}. \end{array} \tag{15.22}$$

This is equivalent to the dual linear optimization problem (13.13). Hence, the notion of duality given here is a generalization of that of Section 13.6 for linear optimization.

Remark 15.4.11. In Example 15.4.10 above, the weak duality inequality $d^* \leq p^*$ becomes the weak duality inequality $-\mathbf{b}^\mathsf{T}\mathbf{y}^* \leq \mathbf{c}^\mathsf{T}\mathbf{x}^*$ for linear optimization as given in Theorem 13.6.5.

Remark 15.4.12. In the following section, we show for a large class of convex optimization problems that the weak duality inequality can be strengthened to the stronger equality $d^* = p^*$. This is a generalization of the strong duality theorem (Theorem 13.6.8) for linear optimization which states that $-\mathbf{b}^\mathsf{T}\mathbf{y}^* = \mathbf{c}^\mathsf{T}\mathbf{x}^*$.

Example 15.4.13. ⚠ Consider again the hard-margin support vector classifier of Example 15.3.7 with linearly separable data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ in $\mathbb{R}^d \times \{\pm 1\}$. The Lagrangian for this optimization problem is

$$\mathscr{L}(\mathbf{w}, b, \boldsymbol{\mu}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^N \mu_i(1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)),$$

where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_N) \succeq \mathbf{0}$. For any given $\boldsymbol{\mu} \succeq \mathbf{0}$ the Lagrangian is a convex function of $\mathbf{w}$ and b, and so the unique minimizer of $\mathscr{L}(\mathbf{w}, b, \boldsymbol{\mu})$ satisfies

$$\mathbf{0} = D_{\mathbf{w}}\mathscr{L}(\mathbf{w}, b, \boldsymbol{\mu}) = \mathbf{w}^\mathsf{T} - \sum_{i=1}^N \mu_i y_i \mathbf{x}_i^\mathsf{T}.$$

This implies that the minimizing $\mathbf{w}$ satisfies $\mathbf{w} = \sum_{i=1}^N \mu_i y_i \mathbf{x}_i$, and the Lagrange dual function is

$$\begin{aligned} \widetilde{f}(\boldsymbol{\mu}) &= \inf_{\mathbf{w}, b} \mathscr{L}(\mathbf{w}, b, \boldsymbol{\mu}) \\ &= \frac{1}{2}\left\|\sum_{i=1}^N \mu_i y_i \mathbf{x}_i\right\|_2^2 - \sum_{i=1}^N \mu_i \left(1 - y_i\left(\sum_{j=1}^N \mu_j y_j \mathbf{x}_j^\mathsf{T}\mathbf{x}_i + b\right)\right). \end{aligned}$$

A little algebra shows that

$$\widetilde{f}(\boldsymbol{\mu}) = \inf_b \left(-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \mu_i \mu_j y_i y_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j + \sum_{i=1}^{n} \mu_i - b \sum_{i=1}^{N} \mu_i y_i \right).$$

But if $\sum_{i=1}^{N} \mu_i y_i \neq 0$, then the infimum is $-\infty$, so the feasible set for the dual problem satisfies $\sum_{i=1}^{N} \mu_i y_i = 0$, and the dual problem is

$$\begin{array}{ll} \text{maximize} & -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \mu_i \mu_j y_i y_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j + \sum_{i=1}^{n} \mu_i \\ \text{subject to} & \sum_{i=1}^{N} \mu_i y_i = 0, \\ & \boldsymbol{\mu} \succeq \mathbf{0}. \end{array} \tag{15.23}$$

Example 15.4.14. An unconstrained problem (to minimize f on the open domain Ω) has an uninteresting Lagrangian $\mathscr{L}(\mathbf{x}) = f(\mathbf{x})$, and the dual function $\widetilde{f} = \inf_{\mathbf{x}} f(\mathbf{x}) = p^*$ is constant. The dual problem is silly: maximize the constant function $\widetilde{f} = p^*$. Weak duality clearly holds, because $f(\mathbf{x}) \geq \widetilde{f} = p^*$ for all $\mathbf{x}$.

15.5 Strong Duality

Weak duality (Theorem 15.4.7) guarantees that $d^* \geq p^*$, where p^* is the minimal value of f and d^* is the maximal value of the Lagrange dual $\widetilde{f}$. In this section we describe conditions under which these two optimal values are equal. The main cases for which this is true are convex optimization problems (see Definition 15.3.1), and we limit ourselves to convex optimization problems in this section.

Definition 15.5.1. *The minimization problem* (15.13) *(called the primal problem)* satisfies strong duality *if it and the dual problem* (15.21) *are both feasible and the minimal value p^* of the primal objective f is equal to the maximal value d^* of the dual objective $\widetilde{f}$.*

Strong duality holds for many convex optimization problems, but not all of them. In this section we discuss some situations where strong duality can be proved to hold.

15.5.1 Strong Duality from Weak Slater

One of the most common conditions that guarantees strong duality is called *Slater's condition.*

Definition 15.5.2. *A minimization problem* (15.3.1) *satisfies the* weak form of Slater's condition *if there exists a feasible* $\mathbf{x}' \in \mathscr{F}$ *from* (15.11) *that lies in the interior of* $\operatorname{effd}(f)$ *and* $g_j(\mathbf{x}') < 0$ *for every* g_j *that is not affine.*

Example 15.5.3. Any feasible linear optimization problem satisfies the weak Slater condition because $\operatorname{effd}(f) = \mathbb{R}^n$ and every constraint is affine. If the problem isn't feasible, then the problem also fails to satisfy the weak Slater condition, since Slater's condition requires a feasible point.

Example 15.5.4. Any feasible convex minimization problem with $\operatorname{effd}(f)$ open and with only equality constraints (no inequality constraints) satisfies the weak Slater condition.

Unexample 15.5.5. Many feasible convex optimization problems satisfy the weak form of Slater's condition. But here is one that does not: let f be the function $f(x, y) = e^{-x}$ if $y > 0$ and ∞ otherwise. Consider the optimization problem

$$\begin{aligned} &\text{minimize} && f(x, y) \\ &\text{subject to} && x^2/y \leq 0. \end{aligned}$$

The condition $x^2/y \leq 0$ implies that either $y < 0$ (in which case f is infinite and hence not minimized) or $x = 0$, and thus the minimal value is $p^* = 1$, and it is achieved by any point in the feasible set $\mathscr{F} = \{(0, y) \mid y > 0\}$ (recall that the feasible set requires $f(x, y) < \infty$; see (15.11)). Moreover, the inequality constraint $x^2/y \leq 0$ is binding on the entire feasible set. Thus this optimization problem does not satisfy the weak Slater condition, yet it is a convex optimization problem.

The next theorem guarantees that strong duality holds for optimization problems of the form (15.13) that are convex and satisfy the weak Slater condition. We prove this in Section 15.5.5.

Theorem 15.5.6 (Weak Slater, Strong Duality). *Assume the optimization problem* (15.13) *is a convex optimization problem (see Definition 15.3.1) with a finite infimum* $p^* = \inf_{\mathbf{x} \in \mathscr{F}} f(\mathbf{x})$. *If there exists a point* $\mathbf{x}'$ *satisfying the weak Slater condition, then the dual problem has a feasible maximizer and the maximum* d^* *of the dual problem is equal to the infimum* p^* *of the primal problem.*

Remark 15.5.7. The hypothesis of the previous theorem does not require that a feasible primal minimizer exists (the infimum might not be realized on $\mathscr{F}$); nevertheless, the theorem guarantees the existence of a feasible dual maximizer.

Unexample 15.5.8. Recall the optimization problem in Unexample 15.5.5. The Lagrangian is

$$\mathscr{L}(x,y,\mu) = \begin{cases} e^{-x} + \mu x^2/y & \text{if } y > 0, \\ \infty & \text{otherwise,} \end{cases}$$

and the Lagrange dual function is

$$\widetilde{f}(\mu) = \inf_{\substack{x\in\mathbb{R} \\ y>0}} (e^{-x} + \mu x^2/y) \geq 0.$$

If $\mu = 0$, then $\lim_{x\to\infty} \mathscr{L}(x,y,\mu) = \lim_{x\to\infty} e^{-x} = 0$. If $\mu > 0$, then for any $\varepsilon > 0$ choosing x large enough to make $e^{-x} < \varepsilon/2$ and choosing y large enough to make $2\mu x^2/y < \varepsilon/2$ shows that $\widetilde{f}(\mu) < \varepsilon$ for all $\varepsilon > 0$, so we have

$$\widetilde{f}(\mu) = \begin{cases} 0 & \text{if } \mu \geq 0, \\ -\infty & \text{if } \mu < 0. \end{cases}$$

Therefore, the dual problem of maximizing $\widetilde{f}$ has maximal value $d^* = 0$, and strong duality fails for this convex problem because there is a *duality gap* of $p^* - d^* = 1$.

15.5.2 Strong Duality Implies KKT First Order

In Section 14.4 the KKT first-order conditions are proved to hold for regular points. But if strong duality holds, then the KKT first-order conditions also hold for singular points.

Theorem 15.5.9. *Given a minimization problem* (15.13) *(not necessarily convex) with dual problem* (15.21)*, assume there exists a primal feasible value* $\mathbf{x}^* \in \mathbb{R}^n$ *and dual feasible values* $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$ *and* $\boldsymbol{\mu}^* \in \mathbb{R}^m$ *with* $\boldsymbol{\mu}^* \succeq \mathbf{0}$*, such that*

$$p^* = f(\mathbf{x}^*) = \widetilde{f}(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = d^*.$$

If $\mathscr{L}$ *is differentiable at* $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *and* $\mathbf{x}^*$ *lies in the interior of* $\operatorname{effd}(\mathscr{L}(\cdot, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*))$*, then the KKT first-order conditions (see Theorem 14.4.5) hold for* $\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*$*:*

(i) $D_{\mathbf{x}}\mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$*, and*

(ii) $\mu_i^* g_i(\mathbf{x}^*) = 0$ *for all* $i \in \{1, \ldots, m\}$.

Proof. By (15.16) we have

$$\begin{aligned} p^* = f(\mathbf{x}^*) = \sup_{\boldsymbol{\lambda},\boldsymbol{\mu}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) &\geq \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \\ &\geq \inf_{\mathbf{x}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \widetilde{f}(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = d^*, \end{aligned} \tag{15.24}$$

but strong duality gives $p^* = d^*$, so

$$f(\mathbf{x}^*) = \inf_{\mathbf{x}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*),$$

and the (unconstrained) FONC gives

$$D_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0},$$

which is the first KKT condition.

Moreover, since all the inequalities in (15.24) must be equalities, we have

$$f(\mathbf{x}^*) = \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*) + (\boldsymbol{\lambda}^*)^{\mathsf{T}} H(\mathbf{x}^*) + (\boldsymbol{\mu}^*)^{\mathsf{T}} G(\mathbf{x}^*),$$

but the fact that $\mathbf{x}^*$ is feasible gives $H(\mathbf{x}^*) = \mathbf{0}$. Hence, we have

$$\mathbf{0} = (\boldsymbol{\mu}^*)^{\mathsf{T}} G(\mathbf{x}^*) = \sum_{i=1}^{m} \mu_i^* g_i(\mathbf{x}^*).$$

Since $G(\mathbf{x}) \preceq 0$ and $\boldsymbol{\mu}^* \succeq \mathbf{0}$, no term of the sum $\sum_{i=1}^m \mu_i^* g_i(\mathbf{x}^*)$ is positive, and therefore every term in the sum must vanish. Thus,

$$\mu_i^* g_i(\mathbf{x}^*) = 0 \qquad \forall i,$$

which is the KKT complementary slackness condition. □

15.5.3 Convex and KKT Imply Optimality and Strong Duality

The KKT first-order conditions are necessary conditions for a regular point to be a minimizer. However, in the case of a convex optimization problem, the KKT conditions are also sufficient, and strong duality automatically holds.

Theorem 15.5.10. *Given a convex minimization problem of the form* (15.13) *with dual problem* (15.21)*, if there exist feasible points* $\mathbf{x}^* \in \mathbb{R}^n$*,* $\boldsymbol{\lambda}^* \in \mathbb{R}^\ell$*, and* $\boldsymbol{\mu}^* \in \mathbb{R}^m$ *with* $\boldsymbol{\mu}^* \succeq \mathbf{0}$*, satisfying the first-order KKT conditions (see Theorem 14.4.5), then*

$$p^* = f(\mathbf{x}^*) = \widetilde{f}(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = d^*.$$

In other words, strong duality holds for this problem, the primal optimizer is $\mathbf{x}^*$*, and the dual optimizer is* $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$*.*

Proof. By definition, we have

$$\widetilde{f}(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + (\boldsymbol{\lambda}^*)^{\mathsf{T}} H(\mathbf{x}) + (\boldsymbol{\mu}^*)^{\mathsf{T}} G(\mathbf{x}) \right).$$

The sum $\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}) + (\boldsymbol{\lambda}^*)^{\mathsf{T}} H(\mathbf{x}) + (\boldsymbol{\mu}^*)^{\mathsf{T}} G(\mathbf{x})$ is a convex function of $\mathbf{x}$ because f and all g_i are convex, $H(\mathbf{x})$ is affine, and $\boldsymbol{\mu}^* \succeq \mathbf{0}$.

Since $\mathbf{x}^* \in \mathscr{F}$ satisfies the first-order KKT condition $D_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$, Theorem 15.3.10 (sufficiency of first-order condition for convex problems) and Theorem 15.3.9 (local minimizers are global minimizers) combine to guarantee that $\mathbf{x}^*$ is a global minimizer for this problem, so

$$\inf_{\mathbf{x}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathscr{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*).$$

Strong duality now follows from a straightforward computation:

$$\begin{aligned}\widetilde{f}(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \inf_{\mathbf{x}} \left(f(\mathbf{x}) + (\boldsymbol{\lambda}^*)^\mathsf{T} H(\mathbf{x}) + (\boldsymbol{\mu}^*)^\mathsf{T} G(\mathbf{x}) \right) \\ &= f(\mathbf{x}^*) + (\boldsymbol{\lambda}^*)^\mathsf{T} H(\mathbf{x}^*) + (\boldsymbol{\mu}^*)^\mathsf{T} G(\mathbf{x}^*) \\ &= f(\mathbf{x}^*),\end{aligned}$$

where the last line follows from the feasibility of $\mathbf{x}^*$ and complementary slackness, so both $(\boldsymbol{\mu}^*)^\mathsf{T} G(\mathbf{x}^*) = 0$ and $H(\mathbf{x}^*) = \mathbf{0}$.

For any $\mathbf{z} \in \mathscr{F}$ with $\mathbf{z} \neq \mathbf{x}^*$, weak duality gives $f(\mathbf{z}) \geq \widetilde{f}(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*)$, so $\mathbf{x}^*$ must be a global minimizer of f. □

Unexample 15.5.11. In the case of Unexample 15.5.5, no minimizers $(0, y)$ are regular points. This is because the constraint $h(x, y) = x^2/y$ is active at these points, and $Dh(x, y) = \begin{bmatrix} 2x/y & -x^2/y^2 \end{bmatrix}$. Any point of the form $(0, y)$ has $Dh(0, y) = \begin{bmatrix} 0 & 0 \end{bmatrix}$, which is not of maximal rank. Therefore, the first-order KKT conditions do not apply.

15.5.4 Equivalent Primals May Have Different Duals

As discussed earlier there may be many ways to reformulate an optimization problem as a different, equivalent problem. Each of these different reformulations has its own dual, and often these dual problems are very different.

Example 15.5.12. Any unconstrained optimization problem has a constant dual function (see Example 15.4.14). But we can often rewrite an unconstrained problem as a constrained problem, and the new problem may have a dual that is easier to solve or interesting in other ways.

Consider the logistic regression problem of Example 15.3.15 to minimize $\sum_{i=1}^N \log(1 + e^{-y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)})$, without constraints. This can be rewritten as the equivalent constrained problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^N \log(1 + e^{z_i}) \\ \text{subject to} \quad & z_i + y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) = 0 \quad \forall i \in \{1, \ldots, N\}. \end{aligned}$$

The Lagrangian of this equivalent problem is

$$\mathscr{L}(\mathbf{w}, b, \mathbf{z}, \boldsymbol{\lambda}) = \sum_{i=1}^N \log(1 + e^{z_i}) + \sum_{i=1}^N \lambda_i (z_i + y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)),$$

and the dual function is

$$\widetilde{f}(\boldsymbol{\lambda}) = \inf_{\mathbf{w}, b, \mathbf{z}} \left(\sum_{i=1}^N \log(1 + e^{z_i} + \lambda_i z_i) + \mathbf{w}^\mathsf{T} \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i + b \sum_{i=1}^N \lambda_i y_i \right).$$

The function $\widetilde{f}(\boldsymbol{\lambda})$ is $-\infty$ unless $\sum_{i=1}^N \lambda_i y_i = 0$ and $\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i = \mathbf{0}$, and otherwise it is $\inf_{\mathbf{z}} \sum_{i=1}^N (\log(1+e^{z_i}) + \lambda_i z_i)$. Exercise 15.34 shows that this is also $-\infty$ unless every λ_i is bounded by $-1 < \lambda < 0$, in which case it takes the value $\widetilde{f}(\boldsymbol{\lambda}) = \sum_{i=1}^N ((1-\lambda_i)\log(1-\lambda_i) - \lambda_i \log(\lambda_i))$. Changing the sign of $\boldsymbol{\lambda}$ gives the dual problem

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i=1}^N \lambda_i \log(-\lambda_i) - (1+\lambda_i)\log(1+\lambda_i) \\ \text{subject to} & 0 \le \lambda_i \le 1 \quad \forall i \in \{1,\dots,N\}, \\ & \sum_{i=1}^N \lambda_i y_i = 0, \\ & \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i = \mathbf{0}. \end{array}$$

Example 15.5.13. ⚠ As with the previous example, the soft-margin support vector classifier of Example 15.3.8 is an unconstrained optimization problem, and hence it has an uninteresting dual. The original problem

$$\text{minimize} \quad f(\mathbf{w},b) = C\sum_{i=1}^N \max(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)) + \frac{1}{2}\|\mathbf{w}\|_2^2$$

can be rewritten as

$$\begin{array}{ll} \text{minimize} & \displaystyle C\sum_{i=1}^N \xi_i + \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} & 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \le \xi_i \quad \forall i \in \{1,\dots,N\}, \\ & 0 \le \xi_i \qquad \forall i \in \{1,\dots,N\}. \end{array} \tag{15.25}$$

Writing $\boldsymbol{\mu} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ gives

$$\mathscr{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = C\sum_{i=1}^N \xi_i + \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^N \alpha_i(\xi_i - 1 + y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)) - \sum_{i=1}^N \beta_i \xi_i.$$

Exercise 15.35 shows that the dual problem is almost identical to that for the hard margin:

$$\begin{array}{ll} \text{maximize} & \displaystyle -\tfrac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j + \sum_{i=1}^n \alpha_i \\ \text{subject to} & \displaystyle \sum_{i=1}^N \alpha_i y_i = 0, \\ & 0 \le \alpha_i \le C \qquad \forall i \in \{1,\dots,N\}. \end{array}$$

Moreover, Slater's conditions hold, so the KKT conditions hold for the unique minimizer. Therefore we have

$$\alpha_i(\xi_i - 1 + y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)) = 0 \quad \forall i \in \{1,\dots,N\}.$$

This is called the *soft-margin* support vector classifier.

15.5.5 *Proof of Strong Duality from Weak Slater

We conclude this section by proving Theorem 15.5.6, that strong duality holds if the weak Slater conditions are satisfied.

Proof. Any equality constraint in an optimization problem can be rewritten as two inequality constraints; therefore, we may assume that the primal optimization problem is of the form

$$\begin{aligned} \underset{\mathbf{x}\in\Omega}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_1(\mathbf{x}) \leq 0, \\ & \vdots \\ & g_m(\mathbf{x}) \leq 0, \end{aligned}$$

where the domain $\Omega \subset \mathbb{R}^n$ is open and convex, and the functions f and g_i are all convex. Slater's condition gives an $\mathbf{x}' \in \mathscr{F}$ such that if g_i is not affine, then $g_i(\mathbf{x}') < 0$. Without loss of generality, assume that $g_1(\mathbf{x}'), \ldots, g_k(\mathbf{x}')$ are all strictly negative and $g_{k+1}, \ldots, g_m$ are all affine, with $g_{k+1}(\mathbf{x}') = \cdots = g_m(\mathbf{x}') = 0$. For convenience, we write $\widetilde{G} = (g_1, \ldots, g_k)$.

Define the set

$$V = \{(\mathbf{u}, w) \in \mathbb{R}^k \times \mathbb{R} \mid \exists \mathbf{x} \in \Omega, \widetilde{G}(\mathbf{x}) \preceq \mathbf{u}, f(\mathbf{x}) \leq w\}.$$

By Exercise 15.37 the set V is convex and not empty. Moreover, given any $(\mathbf{u}, w) \in V$ and any $(\mathbf{u}', w') \succeq (\mathbf{u}, w)$ we must have $(\mathbf{u}', w') \in V$, by definition of V.

The point $(\mathbf{0}, p^*)$ is not in the interior of V because otherwise there would be a point of the form $(\mathbf{0}, p^* - \varepsilon)$ in V for some $\varepsilon > 0$, and this would contradict the minimality of p^* on $\mathscr{F}$. Therefore $(\mathbf{0}, p^*)$ lies on the boundary of V. By the supporting hyperplane theorem (Theorem 13.2.8) there exists a hyperplane $h((\mathbf{u}, w)) = \boldsymbol{\mu}^\mathsf{T}\mathbf{u} + \mu_0 w + b$, with either $\boldsymbol{\mu} \neq \mathbf{0}$ or $\mu_0 \neq 0$, such that $h((\mathbf{0}, p^*)) \leq 0$ and $h((\mathbf{u}, w)) \geq 0$ for all $(\mathbf{u}, w) \in \overline{V}$. This gives

$$\boldsymbol{\mu}^\mathsf{T}\mathbf{u} + \mu_0 w \geq \mu_0 p^* \tag{15.26}$$

for all $(\mathbf{u}, w) \in \overline{V}$. If any entry μ_i of $\boldsymbol{\mu}$ were negative, then for any $(\mathbf{u}, w) \in V$ we could increase the corresponding component u_i of $\mathbf{u}$ arbitrarily and still remain in V, but this would violate (15.26) for u_i sufficiently large. Therefore $\boldsymbol{\mu} \succeq \mathbf{0}$. A similar argument shows that $\mu_0 \geq 0$.

If $\mu_0 = 0$, then (15.26) shows that $\inf_{(\mathbf{u},w)\in V} \boldsymbol{\mu}^\mathsf{T}\mathbf{u} \geq 0$, but $\boldsymbol{\mu} \succeq \mathbf{0}$ combined with $\boldsymbol{\mu} \neq \mathbf{0}$ imply

$$\inf_{(\mathbf{u},w)\in V} \boldsymbol{\mu}^\mathsf{T}\mathbf{u} \leq \boldsymbol{\mu}^\mathsf{T} g_i(\mathbf{x}') < 0$$

because $g_i(\mathbf{x}') < 0$ for all $i \leq k$. Therefore $\mu_0 > 0$.

Since $\mu_0 > 0$ we may divide (15.26) by μ_0 to get

$$\inf_{(\mathbf{u},w)\in V} \left(\widetilde{\boldsymbol{\mu}}^\mathsf{T}\mathbf{u} + w\right) \geq p^*,$$

where $\widetilde{\boldsymbol{\mu}} = \frac{1}{\mu_0}\boldsymbol{\mu}$. Letting $\widehat{\boldsymbol{\mu}} = (\widetilde{\boldsymbol{\mu}}, \mathbf{0}) \in \mathbb{R}^k \times \mathbb{R}^{m-k}$ gives the following bound on the Lagrange dual function:

$$\widetilde{f}(\widehat{\boldsymbol{\mu}}) = \inf_{\mathbf{x}}(f(\mathbf{x}) + (\widehat{\boldsymbol{\mu}})^\mathsf{T} G(\mathbf{x})) = \inf_{(\mathbf{u},w)\in V} \left(\widetilde{\boldsymbol{\mu}}^\mathsf{T}\mathbf{u} + w\right) \geq p^*.$$

Taking the supremum of $\widetilde{f}(\boldsymbol{\mu})$ yields

$$d^* = \sup_{\boldsymbol{\mu}} \widetilde{f}(\boldsymbol{\mu}) \geq \widetilde{f}(\widehat{\boldsymbol{\mu}}) \geq p^*,$$

but weak duality implies $d^* \leq p^*$; therefore $d^* = p^*$ and $\widehat{\boldsymbol{\mu}}$ is the maximizer. □

15.6 Interior Point Methods I: The Barrier Method

Interior point methods originally began as an alternative to the simplex method for solving linear optimization problems, but many of them also work very well for general convex optimization problems. These algorithms are iterative, moving through the interior of the feasible set and converging to an optimizer, rather than moving among the vertices of the boundary of the feasible set

The simplex method is very effective on a large class of linear optimization problems, but there are pathological examples where the temporal complexity of the simplex method is exponential in the number of variables. Specifically, there are examples where the simplex method has to visit exponentially many vertices (exponential in the number of variables) before reaching the minimizer; see Section 13.5.4 for more on this. It was a long-standing problem to determine whether there was any algorithm for solving linear optimization problems that was guaranteed to have polynomial complexity.

The first interior point method was the *ellipsoid method* due to Shor, Nemirovski, Yudin, and Khachiyan, who showed that their method was provably polynomial in complexity. In particular, the temporal complexity is bounded by $O(n^6 L)$ time, where L is the number of bits of input in the problem (see Remark 13.5.8).

While the development of this algorithm was an important advance in theory, the algorithm itself was impractical for solving real problems, due in part to a high cost for each iteration and also due to poor numerical stability. A few years later, Karmarkar presented the first practical algorithm, also an interior point method, that could be proved to solve linear optimization problems in polynomial time. The complexity of his algorithm is bounded by $O(n^{3.5} L)$. Of course, interior point methods have continued to improve since Karmarkar and have been generalized to handle many convex optimization problems—not just linear ones.

A nice feature of interior point methods is that they can give relatively accurate approximations of the optimal point in very few iterations. Moreover, the number of iterations required to give an accurate solution increases very slowly with the dimension of the problem. At present, interior point methods are the only good option for very large linear optimization problems (say with a million or more constraints and variables), and they remain competitive when dealing with smaller linear problems.

15.6.1 Formulation

Barrier methods provide a powerful class of methods for numerically computing solutions of convex optimization problems of the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) && \text{(15.27a)} \\ \text{subject to} \quad & G(\mathbf{x}) \preceq \mathbf{0}, && \text{(15.27b)} \\ & A\mathbf{x} - \mathbf{b} = \mathbf{0}, && \text{(15.27c)} \end{aligned}$$

where f and the components $g_1, \dots, g_m$ of G are convex functions that are C^2 on $\mathbb{R}^n$, where $A \in M_{\ell \times n}(\mathbb{R})$, and $\mathbf{b} \in \mathbb{R}^\ell$. Moreover, the feasible set

$$\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid G(\mathbf{x}) \preceq \mathbf{0} \text{ and } A\mathbf{x} - \mathbf{b} = \mathbf{0}\}$$

must have a nonempty interior $\mathscr{F}^\circ$. As in the case of unconstrained optimization algorithms, these methods can often still work if the domain $\mathbb{R}^n$ is replaced with an open convex set $\Omega \subset \mathbb{R}^n$, but applied naïvely, there is a risk that they will step outside the domain. One way to deal with this situation is to express the set $\overline{\Omega}$ in terms of additional convex inequalities $g_{m+1}(\mathbf{x}) \leq 0, \dots, g_{m+k}(\mathbf{x}) \leq 0$, and then add these inequalities to the list of constraints in G. Of course, a drawback to this approach is that it can produce minimizers outside the feasible set (on the boundary $\partial\Omega = \overline{\Omega} \setminus \Omega$).

In theory, and occasionally in practice, it is sufficient to consider the reduced problem

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & G(\mathbf{x}) \preceq \mathbf{0}, \end{aligned} \tag{15.28}$$

where the affine equality constraints (15.27c) are removed using the methods of Section 14.6. However, many numerical packages follow the general form (15.27).

The idea of barrier methods is to add a small multiple of a convex *barrier function* b to the objective f, where $b(\mathbf{x})$ is finite for $\mathbf{x} \in \mathscr{F}^\circ$, but with $b(\mathbf{x}) \to \infty$ as $g_i(\mathbf{x}) \to 0^-$ for any $i \in \{1, \dots, m\}$. In this section, we study the *logarithmic barrier* function given by

$$b(\mathbf{x}) = -\sum_{i=1}^{m} \log(-g_i(\mathbf{x})). \tag{15.29}$$

It is straightforward to check that this $b(\mathbf{x})$ is convex if each $g_i(\mathbf{x})$ is; see Exercise 15.38. The optimization problem (15.28) is replaced with a new problem,

$$\begin{aligned} \text{minimize} \quad & f_\varepsilon(\mathbf{x}) = f(\mathbf{x}) + \varepsilon b(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathscr{F}^\circ \end{aligned} \tag{15.30}$$

for some small $\varepsilon > 0$. Because $b(\mathbf{x})$ and $f(\mathbf{x})$ are convex, the function $f_\varepsilon(\mathbf{x})$ is also convex. For each ε let $\mathbf{x}^*(\varepsilon)$ be the unique minimizer of (15.30) on $\mathscr{F}^\circ$. The path defined by $\mathbf{x}^*(\varepsilon)$ for $\varepsilon \in (0, \infty)$ is called the *central path.*

Example 15.6.1. Consider the problem of minimizing $f(x) = x^2$, subject to $x \geq 0$. Adding a logarithmic barrier to the objective gives the new problem of minimizing $f_\varepsilon(x) = x^2 - \varepsilon \log(x)$ on the set $\{x \in \mathbb{R} \mid x > 0\}$. In this case the minimizer for the new problem is $x^*(\varepsilon) = \sqrt{\varepsilon/2}$, and as $\varepsilon \to 0^+$ we have $x^*(\varepsilon) \to 0$, which is the minimizer of the original problem. Theorem 15.6.3, below, shows that this is a general phenomenon.

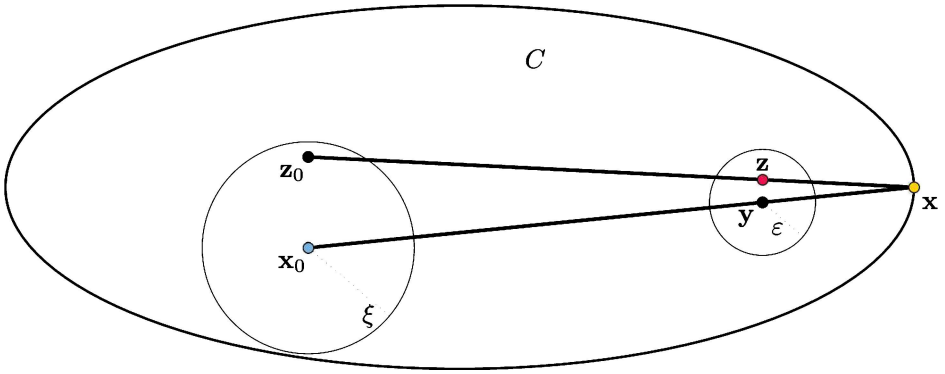


Figure 15.7. *Illustration of the situation in Lemma 15.6.2. Given a point* $\mathbf{x}$ *(yellow) in* C *and an interior point* $\mathbf{x}_0 \in C^\circ$ *(blue), let* $\mathbf{y} = t\mathbf{x}_0 + (1-t)\mathbf{x}$ *be any point on the segment connecting* $\mathbf{x}$ *and* $\mathbf{x}_0$*. The point* $\mathbf{y}$ *lies in the interior of* C*, because any point* $\mathbf{z}$ *(red) in* $B(\mathbf{y}, t\xi)$ *can be written as* $\mathbf{z} = t\mathbf{z}_0 + (1-t)\mathbf{x} \in C$ *for some* $\mathbf{z}_0 \in B(\mathbf{x}_0, \xi)$*.*

15.6.2 Limiting Behavior

We generalize the example by showing that the minimizer $\mathbf{x}^*(\varepsilon)$ of (15.30) converges to the minimizer $\mathbf{x}^*$ of (15.29) as $\varepsilon \to 0^+$. First we have the following lemma.

Lemma 15.6.2. *If* $C \subset V$ *is a convex set in an inner product space* $(V, \|\cdot\|)$*, then for every* $\mathbf{x} \in C$ *and every* $\mathbf{x}_0$ *in the interior* C°*, the point* $\mathbf{y} = t\mathbf{x}_0 + (1-t)\mathbf{x}$*,* $t \in (0,1]$*, lies in* C°*; see Figure 15.7. Moreover, if* $C^\circ \neq \emptyset$*, then* $C \subset \overline{C^\circ}$*.*

Proof. Let $\mathbf{x} \in C$, $\mathbf{x}_0 \in C^\circ$, and $\mathbf{y} = t\mathbf{x}_0 + (1-t)\mathbf{x}$ for some $t \in (0,1]$. To show that $\mathbf{y} \in C^\circ$, it suffices to prove that the open ball $B(\mathbf{y}, \varepsilon) \subset C$ for some $\varepsilon > 0$. Since $\mathbf{x}_0 \in C^\circ$, there exists $\xi > 0$ with $B(\mathbf{x}_0, \xi) \subset C$. For any $\mathbf{z} \in B(\mathbf{y}, \varepsilon)$, with $\varepsilon = t\xi$, the point $\mathbf{z}_0 = \frac{1}{t}(\mathbf{z} + (t-1)\mathbf{x})$ lies in the ball $B(\mathbf{x}_1, \xi)$, because

$$\|\mathbf{z}_0 - \mathbf{x}_0\| = \frac{1}{t}\|\mathbf{z} + (t-1)\mathbf{x} - t\mathbf{x}_0\| = \frac{1}{t}\|\mathbf{z} - \mathbf{y}\| < \xi.$$

This shows that $\mathbf{z}_0 \in B(\mathbf{x}_0, \xi) \subset C$ and thus $\mathbf{z} = t\mathbf{z}_0 + (1-t)\mathbf{x} \in B(\mathbf{y}, \varepsilon) \subset C$.

We conclude the proof by showing $\mathbf{x} \in \overline{C^\circ}$, that is, for any $\delta > 0$, there exists $0 < t \leq 1$ such that $y \in B(\mathbf{x}, \delta)$. Setting $t < \min(1, \delta/\|\mathbf{x} - \mathbf{x}_0\|)$ gives

$$\|\mathbf{y} - \mathbf{x}\| = \|t\mathbf{x}_0 + (1-t)\mathbf{x} - \mathbf{x}\| = t\|\mathbf{x}_0 - \mathbf{x}\| < \delta. \quad \square$$

Theorem 15.6.3. *Let* $b : \mathscr{F}^\circ \to \mathbb{R}$ *be a convex function with the property that for any point* $\bar{\mathbf{x}}$ *in the boundary* $\partial\mathscr{F} = \overline{\mathscr{F}} \setminus \mathscr{F}^\circ$ *we have* $\lim_{k\to\infty} b(\mathbf{x}_k) = \infty$ *for any sequence* $(\mathbf{x}_k)_{k\in\mathbb{N}}$ *in* $\mathscr{F}^\circ$ *that converges to* $\bar{\mathbf{x}}$*. Fix a positive sequence* $(\varepsilon_k)_{k\in\mathbb{N}}$ *converging to* 0*. For each* $j \in \mathbb{N}$ *let* $\mathbf{x}_j$ *be the minimizer of* (15.30) *for* $\varepsilon = \varepsilon_j$*. Any limit point of* $(\mathbf{x}_j)_{j\in\mathbb{N}}$ *in* Ω *lies in* $\mathscr{F}$ *and is a global minimizer of* f *on* $\mathscr{F}$*.*

Proof. Let $\bar{\mathbf{x}} \in \Omega$ be a limit point of $(\mathbf{x}_j)_{j\in\mathbb{N}}$. Passing to a subsequence, if necessary, we may assume that $\mathbf{x}_j \to \bar{\mathbf{x}}$. If $\bar{\mathbf{x}} \in \mathscr{F}^\circ$, then $|b(\bar{\mathbf{x}})| < \infty$ and $\lim_{k\to\infty} \varepsilon_k b(\mathbf{x}_k) = 0$. If $\bar{\mathbf{x}} \notin \mathscr{F}^\circ$, then by hypothesis, $b(\mathbf{x}_k) \to \infty$, and thus for any sufficiently large k we have

$$\varepsilon_k b(\mathbf{x}_k) \geq 0.$$

This implies that

$$f(\mathbf{x}_k) + \varepsilon_k b(\mathbf{x}_k) \geq f(\mathbf{x}_k). \tag{15.31}$$

Assume by way of contradiction that $\bar{\mathbf{x}}$ is not a global minimizer, so that there exists $\mathbf{x}^* \in \mathscr{F}$ such that $f(\mathbf{x}^*) < f(\bar{\mathbf{x}})$. By continuity of f there exists $\delta > 0$ such that $f(\mathbf{y}) < f(\bar{\mathbf{x}})$ for all $\mathbf{y} \in B(\mathbf{x}^*, \delta)$. By Lemma 15.6.2 we may assume that $\mathbf{y} \in \mathscr{F}^\circ$. The definition of $\mathbf{x}_k$ as a minimizer on $\mathscr{F}^\circ$ and (15.31) combine to give

$$f(\mathbf{y}) + \varepsilon_k b(\mathbf{y}) \geq f(\mathbf{x}_k) + \varepsilon_k b(\mathbf{x}_k) \geq f(\mathbf{x}_k)$$

for all sufficiently large $k \in \mathbb{N}$. Taking limits gives $f(\mathbf{y}) \geq f(\bar{\mathbf{x}})$, a contradiction. □

Example 15.6.4. Consider the optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x,y) = x - y^2 \\ \text{subject to} \quad & 1 + x - y^2 \geq 0, \end{aligned}$$

where the minimum of the function is -1. The log barrier problem is to minimize the function

$$f_\varepsilon(x,y) = x - y^2 - \varepsilon \log(1 + x - y^2) - \varepsilon \log(y).$$

This is an unconstrained problem, which we can solve by computing the FONC given by

$$\begin{aligned} 1 - \frac{\varepsilon}{1 + x - y^2} &= 0, \\ -2 + 2\varepsilon \frac{y}{1 + x - y^2} - \varepsilon \frac{1}{y} &= 0. \end{aligned}$$

We simplify to get

$$\begin{aligned} y^2 - y - \varepsilon/2 &= 0, \\ 1 + x - y^2 &= \varepsilon. \end{aligned}$$

This gives the solution

$$\begin{bmatrix} x^*(\varepsilon) \\ y^*(\varepsilon) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 3\varepsilon - 1 + \sqrt{1 + 2\varepsilon} \\ 1 + \sqrt{1 + 2\varepsilon} \end{bmatrix}.$$

Letting $\varepsilon \to 0^+$ gives the solution $(x^*, y^*) = (0, 1)$.

Example 15.6.5. Consider the optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x,y) = x^3 + y^3 \\ \text{subject to} \quad & 1 \le x \le 4, \\ & 2 \le y \le 5. \end{aligned} \tag{15.32}$$

Replace the objective with $f_\varepsilon(x,y) = f(x,y) + \varepsilon b(x,y)$, where

$$b(x,y) = -\log(4-x) - \log(x-1) - \log(5-y) - \log(y-2).$$

Take a sequence $\varepsilon_k \to 0^+$ and for each k apply a few steps of Newton's method to f_{ε_k}. This gives an approximation $\mathbf{x}_k$ of the optimizer $\mathbf{x}^*(\varepsilon_k)$ for each ε_k. This is illustrated in Figure 15.8.

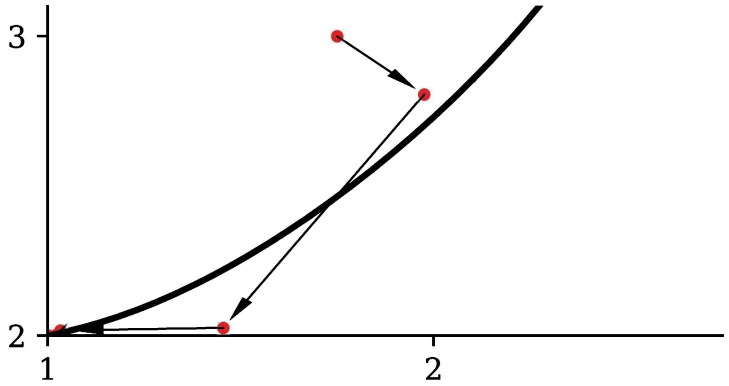


Figure 15.8. *The logarithmic barrier method applied to $f(x,y)$ in (15.32). The black curve is the central path, corresponding to the minimizer $\mathbf{x}^*(\varepsilon)$ for all $\varepsilon > 0$. This path approaches the minimizer $\mathbf{x}^* = (1,2)$ as $\varepsilon \to 0^+$. The red dots correspond to iterates of the method for a decreasing sequence $\varepsilon_k \to 0^+$. Rarely does it make sense to actually find the minimizer $\mathbf{x}^*(\varepsilon_k)$—that would correspond to red dots all lying on the black central path. Instead, it is usually best to apply Newton's method for ε_k only until the Newton steps are sufficiently small, and then update to ε_{k+1} and start another stage of Newton's method, moving toward $\mathbf{x}^*(\varepsilon_{k+1})$.*

15.6.3 Naïve Implementation

Many barrier methods follow the basic idea of choosing a sequence $\varepsilon_k \to 0^+$ and then, for each k, approximately solving for $\mathbf{x}^*(\varepsilon_k)$. The important choices in such a method are

(i) which method to use for approximating $\mathbf{x}^*(\varepsilon_k)$;

(ii) when to stop the search for $\mathbf{x}^*(\varepsilon_k)$;

(iii) how to update ε_k to ε_{k+1}.

One common barrier method uses (i) a Newton-type method on the logarithmic barrier (15.30), (ii) stopping the search for $\mathbf{x}^*(\varepsilon_k)$ when $D^2 f_\varepsilon(\mathbf{x})^{-1} Df_\varepsilon(\mathbf{x})^\mathsf{T} \leq \tau$ for some fixed $\tau > 0$, and (iii) updating by $\varepsilon_{k+1} = \theta\varepsilon_k$ for some fixed $\theta \in (0,1)$. This is given in Algorithm 15.1.

Begin with an initial point $\mathbf{x}_0 \in \mathscr{F}$, a value of $\theta \in (0,1)$ (to decrease ε_k), a value $\tau > 0$ (to decide when to stop searching for $\mathbf{x}(\varepsilon_k)$), an initial $\varepsilon_1 > 0$, and a final value $\varepsilon > 0$. Now proceed as follows:

(i) Set $k = 1$.

(ii) Perform a Newton search (possibly using exact line search or backtracking in the Newton direction) for (15.30) in order to approximate $\mathbf{x}^*(\varepsilon_k)$.

(iii) Terminate the search for $\mathbf{x}^*(\varepsilon_k)$ once $D^2 f_\varepsilon(\mathbf{x})^{-1} Df_\varepsilon(\mathbf{x})^\mathsf{T} \leq \tau$ and set $\mathbf{x}_{k+1}$ equal to the approximate solution.

(iv) Update $\varepsilon_{k+1} = (1-\theta)\varepsilon_k$.

(v) If $\varepsilon_k > \varepsilon$, then increment k to $k+1$ and go to (ii). Otherwise return $\mathbf{x}_{k+1}$ and stop.

Algorithm 15.1. *Outline of the naïve barrier method to solve the optimization problem* (15.29). *It can be shown that this algorithm solves linear and quadratic (quadratic objective with linear constraints) optimization problems in* $O(\sqrt{n}L\log(L))$ *time. In the next section we examine the more general approach and a modification that speeds things up considerably.*

15.7 Interior Point Methods II: The Primal-Dual Method

Primal-dual methods are a modification of barrier methods that provide a faster and more powerful approach to solving convex optimization problems. The general idea is to solve both the primal and dual problems simultaneously by relaxing one of the constraints in the barrier method. We begin with a more general implementation of the barrier method.

15.7.1 General Implementation of the Barrier Method

Consider the convex optimization problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) && \text{(15.33a)}\\ &\text{subject to} && G(\mathbf{x}) \preceq \mathbf{0}, && \text{(15.33b)}\\ &&& A\mathbf{x} - \mathbf{b} = \mathbf{0}, && \text{(15.33c)} \end{aligned}$$

where f and the components $g_1, \ldots, g_m$ of G are convex functions that are C^2 on a convex open domain $\Omega \subset \mathbb{R}^n$, with $A \in M_{\ell\times n}(\mathbb{R})$, and $\mathbf{b} \in \mathbb{R}^\ell$. The feasible set is given by

$$\mathscr{F} = \{\mathbf{x} \in \mathbb{R}^n \mid G(\mathbf{x}) \preceq \mathbf{0} \text{ and } A\mathbf{x} - \mathbf{b} = \mathbf{0}\} \subset \Omega.$$

We assume that strong duality holds and that the problem is solvable, that is, there exists a feasible optimizer $\mathbf{x}^*$.

The Lagrangian is

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\mu}^\mathsf{T} G(\mathbf{x}) + \boldsymbol{\lambda}^\mathsf{T}(A\mathbf{x} - \mathbf{b}).$$

The KKT necessary conditions for a minimizer (Theorem 14.4.5) are as follows:

(i) Primal feasibility: $A\mathbf{x}^* - \mathbf{b} = \mathbf{0}$ and $G(\mathbf{x}^*) \preceq \mathbf{0}$.

(ii) Dual feasibility: $\boldsymbol{\mu}^* \succeq \mathbf{0}$ and

$$Df(\mathbf{x}^*) + (\boldsymbol{\mu}^*)^\mathsf{T} DG(\mathbf{x}^*) + (\boldsymbol{\lambda}^*)^\mathsf{T} A = \mathbf{0}. \tag{15.34}$$

(iii) Complementary slackness: $\mu_i^* g_i(\mathbf{x}^*) = 0$ for all $i \in \{1, \dots, m\}$.

It is difficult to solve these conditions directly. In particular, complementary slackness often causes some numerical difficulty due to the sharpness of the constraints. Instead, we use the barrier method and solve an equality-constrained optimization problem that relaxes complementary slackness.

For $\varepsilon > 0$, consider instead the *relaxed* problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) - \varepsilon \sum_{i=1}^m \log(-g_i(\mathbf{x})) \\ &\text{subject to} && A\mathbf{x} - \mathbf{b} = \mathbf{0}. \end{aligned} \tag{15.35}$$

The Lagrangian is

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \varepsilon \sum_{i=1}^m \log(-g_i(\mathbf{x})) + \boldsymbol{\lambda}^\mathsf{T}(A\mathbf{x} - \mathbf{b}).$$

The FONC is

$$Df(\mathbf{x}) - \varepsilon \sum_{i=1}^m \frac{Dg_i(\mathbf{x})}{g_i(\mathbf{x})} + \boldsymbol{\lambda}^\mathsf{T} A = \mathbf{0}, \tag{15.36}$$

together with the constraint $A\mathbf{x} - \mathbf{b} = \mathbf{0}$. We can solve this by finding a zero of the function $r : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$ given by

$$r(\mathbf{x}, \boldsymbol{\lambda}) = \begin{bmatrix} Df(\mathbf{x})^\mathsf{T} - \varepsilon \displaystyle\sum_{i=1}^m \frac{Dg_i(\mathbf{x})^\mathsf{T}}{g_i(\mathbf{x})} + A^\mathsf{T}\boldsymbol{\lambda} \\ A\mathbf{x} - \mathbf{b} \end{bmatrix}.$$

This can be done with Newton's method. To avoid notational clutter, let $x \leftarrow x + a$ indicate that the value of the variable x is set to the old value of x plus a. We update

$$\begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} + \alpha \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix}$$

by solving the linear system

$$Dr(\mathbf{x}, \boldsymbol{\lambda}) \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = -r(\mathbf{x}, \boldsymbol{\lambda}),$$

where

$$Dr(\mathbf{x}, \boldsymbol{\lambda}) = \begin{bmatrix} D^2 f(\mathbf{x}) - \varepsilon \displaystyle\sum_{i=1}^{m} \left(\frac{D^2 g_i(\mathbf{x})}{g_i(\mathbf{x})} - \frac{Dg_i(\mathbf{x})^\mathsf{T} Dg_i(\mathbf{x})}{g_i(\mathbf{x})^2} \right) & A^\mathsf{T} \\ A & 0 \end{bmatrix},$$

and $\alpha > 0$ is judiciously chosen to maintain feasibility.

Remark 15.7.1. One choice for the learning rate α is to fix some $\beta \in (0, 1)$ (an upper bound for the learning rate), and set the learning rate $\alpha > 0$ as

$$\alpha = \beta \cdot \min \left\{ 1, \min \left\{ \frac{-x_i}{\Delta x_i} \,\middle|\, \Delta x_i < 0 \right\} \right\}. \tag{15.37}$$

Note that this choice ensures that the feasibility condition $\mathbf{x} + \alpha \Delta \mathbf{x} \succ \mathbf{0}$ holds.

Remark 15.7.2. The first-order necessary condition for barrier methods (15.36) can be written as (15.34) by setting

$$\boldsymbol{\mu}^\mathsf{T} = \begin{bmatrix} \dfrac{-\varepsilon}{g_1(\mathbf{x})} & \dfrac{-\varepsilon}{g_2(\mathbf{x})} & \cdots & \dfrac{-\varepsilon}{g_m(\mathbf{x})} \end{bmatrix}. \tag{15.38}$$

This is equivalent to enforcing a perturbed complementary slackness constraint $\mu_i g_i(\mathbf{x}) = -\varepsilon$ for each $i = 1, 2, \ldots, m$, which, in the limit as $\varepsilon \to 0$, gives the original complementary slackness condition. In other words, the barrier method is simply a relaxed form of the KKT conditions given above.

Example 15.7.3. Consider the linear optimization problem in standard form

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\mathsf{T} \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{b}, \\ & \mathbf{x} \succeq \mathbf{0}, \end{aligned} \tag{15.39}$$

where $A \in M_{m \times n}(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. By adding slack variables, this can be written as

$$\begin{aligned} \text{minimize} \quad & \widetilde{\mathbf{c}}^\mathsf{T} \mathbf{z} \\ \text{subject to} \quad & \widetilde{A}\mathbf{z} = \mathbf{b}, \\ & \mathbf{z} \succeq \mathbf{0}, \end{aligned} \tag{15.40}$$

where $\widetilde{\mathbf{c}} = \begin{bmatrix} \mathbf{c}^\mathsf{T} & \mathbf{0}^\mathsf{T} \end{bmatrix}$, $\widetilde{A} = \begin{bmatrix} A & I \end{bmatrix}$, and $\mathbf{z} \in \mathbb{R}^{m+n}$. Thus, the barrier method becomes the equality-constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & \widetilde{\mathbf{c}}^\mathsf{T} \mathbf{z} - \varepsilon \sum_{i=1}^{m+n} \log(z_i) \\ \text{subject to} \quad & \widetilde{A}\mathbf{z} = \mathbf{b}. \end{aligned} \tag{15.41}$$

An implementation of the barrier method on this problem is given in Algorithm 15.2, using the learning rate described in Remark 15.7.1.

Example 15.7.4. Consider the linear optimization problem in Example 13.3.3. The problem can be modified to fit the form of (15.40) by setting

$$\widetilde{A} = \begin{bmatrix} 4 & -1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad \widetilde{\mathbf{c}}^{\mathsf{T}} = \begin{bmatrix} -5 & -1 & -6 & 0 & 0 \end{bmatrix}.$$

Using the barrier method described in this section with (15.41), as implemented in Algorithm 15.2, yields the optimizer $\mathbf{x}^* = (0, 7, 5)$ very rapidly.

15.7.2 Formulation of the Primal-Dual Method

In the barrier method, each iteration of the Newton step requires that (15.38) hold. As it turns out, we can often get an algorithmic improvement by keeping the relaxed complementary slackness condition (15.38), but solving the original dual condition (15.34), that is, solve the following system of equations:

(i) Primal feasibility: $A\mathbf{x}^* - \mathbf{b} = \mathbf{0}$ and $G(\mathbf{x}^*) \preceq \mathbf{0}$.

(ii) Dual feasibility: $Df(\mathbf{x}^*) + (\boldsymbol{\mu}^*)^{\mathsf{T}} DG(\mathbf{x}^*) + (\boldsymbol{\lambda}^*)^{\mathsf{T}} A = \mathbf{0}$ and $\boldsymbol{\mu}^* \succeq \mathbf{0}$.

(iii) Relaxed complementary slackness: $\mu_i^* g_i(\mathbf{x}^*) = -\varepsilon$ for all $i \in \{1, \ldots, m\}$.

We can solve this by finding a zero of the function $r : \mathbb{R}^{n+m+\ell} \to \mathbb{R}^{n+m+\ell}$ given by

$$r(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \begin{bmatrix} Df(\mathbf{x})^{\mathsf{T}} + DG(\mathbf{x})^{\mathsf{T}} \boldsymbol{\mu} + A^{\mathsf{T}} \boldsymbol{\lambda} \\ \operatorname{diag}(\boldsymbol{\mu}) G(\mathbf{x}) + \varepsilon \mathbb{1} \\ A\mathbf{x} - \mathbf{b} \end{bmatrix}. \tag{15.42}$$

This can be done with Newton's method where we update

$$\begin{bmatrix} \mathbf{x} \\ \boldsymbol{\mu} \\ \boldsymbol{\lambda} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\mu} \\ \boldsymbol{\lambda} \end{bmatrix} + \alpha \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\mu} \\ \Delta\boldsymbol{\lambda} \end{bmatrix}$$

by solving the linear system

$$Dr(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\mu} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = -r(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}), \tag{15.43}$$

where

$$Dr(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \begin{bmatrix} D^2 f(\mathbf{x}) + \varepsilon \sum_{i=1}^{m} \mu_i D^2 g_i(\mathbf{x}) & DG(\mathbf{x})^{\mathsf{T}} & A^{\mathsf{T}} \\ \operatorname{diag}(\boldsymbol{\mu}) DG(\mathbf{x}) & \operatorname{diag}(G(\mathbf{x})) & 0 \\ A & 0 & 0 \end{bmatrix},$$

and then choosing $\alpha > 0$ judiciously.

```python
import numpy as np
import numpy.linalg.solve as solve

def linear_barrier(A,b,c,y,eps=0.1):
    """Use the logarithmic barrier method to minimize
    c^T x subject to A x = b and x >= 0.
    """

    theta=0.9     # Decay rate for eps
    beta=0.9      # Maximum learning rate
    tol=1e-13     # Stop once |Newton dir| <= tol
    max_iter=1000
    m,n = A.shape

    counter = 0  # which iteration
    d = 1        # d = Newton direction, but init to 1
    while np.linalg.norm(d) > tol:  # Execute barrier method
        x = y[0:n]; lamb = y[n:n+m]
        r = np.block([-c+eps/x + A.T @ lamb, A @ x - b])
        Dr = np.block([[-eps*np.diag(1/x**2),A.T],
                       [A,np.zeros((m,m))]])
        d = -solve(Dr,r) # Newton direction
        ratio = x / d[0:n] # Used to compute lrning rate alpha
        alpha = beta*np.min(np.block([1,-ratio[ratio<0]]))
        y = y + alpha * d     # Newton step
        eps = theta * eps     # Shrink epsilon
        counter = counter + 1
        if counter >= max_iter:
            print('Does not converge')
            break
    return y, counter
```

Algorithm 15.2. *Python implementation of the equality-constrained barrier method for linear optimization problems of the form* (15.40)*, as described in Example 15.7.3. Given an initial choice of* `eps` *and an initial guess* `y = (x, lamb)`*, where* `x` *is the primal variable and* `lamb` *is the dual variable, use the logarithmic barrier method to solve the linear problem by approximating the zeros of* $r(\mathtt{y}, \mathtt{eps}) = \mathbf{c}^\mathsf{T} - \mathtt{eps}\sum 1/x_i + A^\mathsf{T}\mathtt{lamb}$ *as* $\mathtt{eps} \to 0$*. Here we use the learning rate described in Remark 15.7.1. Note that in Python* `@` *denotes matrix multiplication.*

Remark 15.7.5. In this case, the inequality constraint $\boldsymbol{\mu} + \alpha\Delta\boldsymbol{\mu} \succ \mathbf{0}$ needs to be preserved. Given $0 < \beta < 1$, set the learning rate $\alpha > 0$ to satisfy

$$\alpha = \beta \cdot \min\left\{1, \min\left\{\frac{-\mu_i}{\Delta\mu_i} \,\middle|\, \Delta\mu_i < 0\right\}\right\}. \tag{15.44}$$

It is common to also use backtracking to make sure that $G(\mathbf{x}) \preceq \mathbf{0}$ holds and that $\|r(\mathbf{x}, \mu)\|$ decreases.

Example 15.7.6. Consider a linear optimization problem in the form (15.40). Here we have $f(\mathbf{z}) = \widetilde{\mathbf{c}}^{\mathsf{T}}\mathbf{z}$ and $G(\mathbf{z}) = -\mathbf{z}$. Thus, $Df(\mathbf{z})^{\mathsf{T}} = \mathbf{c}$, $DG(\mathbf{z}) = -I$. It follows from (15.43) that

$$\begin{bmatrix} 0 & -I & A^{\mathsf{T}} \\ -\operatorname{diag}(\boldsymbol{\mu}) & -\operatorname{diag}(\mathbf{x}) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\mu} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \mathbf{c} - \boldsymbol{\mu} + \widetilde{A}^{\mathsf{T}}\boldsymbol{\lambda} \\ -\operatorname{diag}(\boldsymbol{\mu})\operatorname{diag}(x)\mathbb{1} + \varepsilon\mathbb{1} \\ A\mathbf{z} - \mathbf{b} \end{bmatrix}.$$

Putting these into code and applying the learning rate given in Remark (15.7.5) gives Algorithm 15.3.

Remark 15.7.7. In the barrier method (Algorithm 15.2), our terminal condition is based on the size of the step $\|\mathbf{d}\|$. By contrast, the terminal condition of the primal dual method (Algorithm 15.3) is based on the norm of the residual r. This is because the barrier method can converge to a different point as the algorithm achieves the solution due to the asymptotic relationship between the numerator and denominators in (15.38).

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

```python
import numpy as np

def rpd(z, eps):
    """
    Return r and Dr for the primal-dual method
    """

    x = z[0:n]; mu = z[n:2*n]; lamb = z[2*n:2*n+m]
    r = np.block([c-mu+np.dot(A.T,lamb),
                  -x*mu + eps*np.ones(n),
                  np.dot(A,x)-b])
    Dr = np.block([[np.zeros((n,n)),-np.eye(n),A.T],
                   [-np.diag(mu),-np.diag(x),np.zeros((n,m))],
                   [A,np.zeros((m,m+n))]])
    return r, Dr

def primal_dual(z, eps):
    """
    Primal dual iteration via Newton's method
    """

    counter = 0
    r = 1
    while np.linalg.norm(r) > 1e-6:
        r, Dr = rpd(z,eps)
        d = -np.linalg.lstsq(Dr,r,rcond=None)[0]
        ratio = z[0:2*n] / d[0:2*n]
        alpha = beta*np.min(np.block([1,-ratio[ratio<0]]))
        z = z + alpha * d
        eps = gamma * eps
        counter = counter + 1
        if counter >= N:
            print('Does not converge')
            break
    return z, counter
```

Algorithm 15.3. *Python implementation of the primal dual method for linear optimization problems of the form* (15.40) *as described in Example 15.7.6.*

15.1. Prove that any nonnegative combination of convex functions is convex. That is, for any convex set C, for any convex functions $f_1, \ldots, f_k$ taking C to $\mathbb{R}$, and for any $\lambda_1, \ldots, \lambda_k \in [0, \infty)$, the function

$$f(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i f_i(\mathbf{x})$$

is convex.

15.2. Prove that if $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then $\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) \leq c\} \subset \mathbb{R}^n$ is a convex set for every $c \in \mathbb{R}$.

15.3. Prove that if $f : \mathbb{R}^m \to \mathbb{R}$ is convex, $A \in M_{m\times n}(\mathbb{R})$, and $\mathbf{b} \in \mathbb{R}^m$, then the function $g : \mathbb{R}^n \to \mathbb{R}$ given by $g(\mathbf{x}) = f(A\mathbf{x} + \mathbf{b})$ is convex.

15.4. Prove that if $g : C \to D \subset \mathbb{R}$ is convex and $f : D \to \mathbb{R}$ is convex and increasing, then $f \circ g : C \to \mathbb{R}$ is a convex function. Give an example of a pair of convex functions $f, g : \mathbb{R} \to \mathbb{R}$ such that $f \circ g$ is not convex.

15.5. ⚠ For any fixed value of $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{\pm 1\}$ the *hinge loss function* is $h(\mathbf{w}, b) = \max(0, 1 - y(\mathbf{w}^\mathsf{T}\mathbf{x} + b))$. Show that $h : \mathbb{R}^{d+1} \to \mathbb{R}$ is a convex function of $(\mathbf{w}, b)$.

15.6. ⚠ For any choice of $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{\pm 1\}$, the *logistic loss function* is $\ell(\mathbf{w}, b) = \log(1 + e^{-y(\mathbf{w}^\mathsf{T}\mathbf{x}+b)})$ for $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Prove that the function ℓ is a convex function of $(\mathbf{w}, b) \in \mathbb{R}^{d+1}$. Hint: Observe that $\log(1 + e^t)$ is a special case of the LogSumExp function (Example 15.1.15) and that $-y(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$ is an affine function of $(\mathbf{w}, b)$.

15.7.* If $f : [a, b] \to \mathbb{R}$ is convex, show that

(i) $f(x) \leq \frac{b-x}{b-a} f(a) + \frac{x-a}{b-a} f(b) \quad \forall x \in (a, b)$,

(ii) $\frac{f(x)-f(a)}{x-a} \leq \frac{f(b)-f(a)}{b-a} \leq \frac{f(b)-f(x)}{b-x} \quad \forall x \in (a, b)$.

15.8.* Let K denote the positive orthant $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \succeq \mathbf{0}\}$. Consider the map $f : K \to \mathbb{R}$ given by $f(x_1, x_2, \ldots, x_n) = -\left(\Pi_{i=1}^n x_i\right)^{1/n}$. Show that f is convex by the following steps:

(i) Let $\mathbf{q} = (\frac{1}{x_1}, \ldots, \frac{1}{x_n})$. Show that the Hessian of f can be written as

$$D^2 f(\mathbf{x}) = \frac{\prod_{i=1}^n x_i^{1/n}}{n^2} \left[n \operatorname{diag}\left(\frac{1}{x_1^2}, \frac{1}{x_2^2}, \ldots, \frac{1}{x_n^2}\right) - \mathbf{q}\mathbf{q}^\mathsf{T} \right].$$

(ii) Use the Cauchy–Schwarz inequality to prove that for any vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ we have

$$n \sum_{i=1}^n \frac{v_i^2}{x_i^2} \geq \left(\sum_{i=1}^n \frac{v_i}{x_i} \right)^2.$$

(iii) Show that $D^2 f(\mathbf{x}) \geq 0$ by directly checking that $\mathbf{v}^\mathsf{T} D^2 f(\mathbf{x}) \mathbf{v} \geq 0$ for every $\mathbf{v} \in \mathbb{R}^n$.

15.9.* † Exercise 13.6 shows that the set $\mathrm{PSD}_n(\mathbb{R})$ of positive semidefinite matrices in $M_n(\mathbb{R})$ is convex. Prove the following:

(i) Prove that the set $\mathrm{PD}_n(\mathbb{R})$ of positive definite matrices in $M_n(\mathbb{R})$ is convex.

(ii) The function $f(X) = -\log(\det(X))$ is convex on $\mathrm{PD}_n(\mathbb{R})$. To prove this, show the following:

(a) The function f is convex if for every $A, B \in \mathrm{PD}_n(\mathbb{R})$ the function $g(t) : [0, 1] \to \mathbb{R}$ given by $g(t) = f(tA + (1 - t)B)$ is convex.

(b) Use the fact that positive definite matrices are normal to show that there is an S such that $S^{\mathsf{T}}S = A$ and

$$\begin{aligned} g(t) &= -\log(\det(S^{\mathsf{T}}(tI + (1-t)(S^{\mathsf{T}})^{-1}BS^{-1})S)) \\ &= -\log(\det(A)) - \log(\det(tI + (1-t)(S^{\mathsf{T}})^{-1}BS^{-1})). \end{aligned}$$

(c) Show that

$$g(t) = -\sum_{i=1}^{n} \log(t + (1-t)\lambda_i) - \log(\det(A)),$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of $(S^{\mathsf{T}})^{-1}BS^{-1}$.

(d) Prove that $g''(t) \geq 0$ for all $t \in [0, 1]$.

15.10. Prove that if $f : \mathbb{R}^n \to \mathbb{R}$ is convex and bounded above, then f is constant.

15.11. Give an example of a convex function on a convex set in $\mathbb{R}^n$ whose epigraph is not closed and which is not the supremum of the hyperplanes lying below its graph (so the conclusion of Theorem 15.2.12 does not hold).

15.12. Let $\alpha_1, \alpha_2, \alpha_3$ be the interior angles of a triangle. Prove that $\sum \sin(\alpha_i) \leq \frac{3\sqrt{3}}{2}$. Hint: Consider using Jensen's inequality.

15.13. Let $x_1, x_2, \ldots, x_n \geq 0$ and $\lambda_1, \lambda_2, \ldots, \lambda_n \geq 0$ with $\sum_{k=1}^{n} \lambda_k = 1$.

(i) Generalize the arithmetic-geometric mean inequality by showing that

$$\prod_{k=1}^{n} x_k^{\lambda_k} \leq \sum_{k=1}^{n} \lambda_k x_k.$$

If $\lambda_1, \lambda_2, \ldots, \lambda_n > 0$, prove that equality holds if and only if $x_1 = x_2 = \cdots = x_n$. Hint: For the equality, consider using Lagrange multipliers.

(ii) Use the arithmetic-geometric mean inequality to show that the hyperbolic set $\{(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \mid \prod_{k=1}^{n} x_k \geq 1\}$ is convex.

15.14. Prove Corollary 15.2.14.

15.15. ⚠ Prove, using the following steps, that if $U \subset \mathbb{R}^n$ is open and $f : U \to \mathbb{R}$ is convex, then f is continuous on U.

(i) For any $\mathbf{x}_0 \in U$, prove that the function $g(\mathbf{x}) = \|f(\mathbf{x}) - f(\mathbf{x}_0)\|$ is convex on U.

(ii) For any $\mathbf{x}_0 \in U$ there exists a closed ball $\overline{B(\mathbf{x}_0, r)} \subset U$. Prove that there exist n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in B(\mathbf{x}_0, r)$ and a real number $\rho > 0$ such that the closed ball $\overline{B(\mathbf{x}_0, \rho)}$ is contained in the convex hull $\operatorname{conv}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

(iii) Show that $g(\mathbf{x})$ is bounded on $\overline{B(\mathbf{x}_0, \rho)}$ by some $M < \infty$. Hint: Use Jensen's inequality.

(iv) Show that for every $\varepsilon > 0$ if $\delta = \frac{\varepsilon\rho}{M}$, then $g(\mathbf{x}) \leq \varepsilon$. Explain why this proves that f is continuous.

Give an example to show that if $f : U \to \mathbb{R}_\infty$ is convex, then f need not be continuous. Give an example of a convex set C and a convex function $h : C \to \mathbb{R}$ such that h is not continuous.

15.16.* Assume that both $C \subset \mathbb{R}^n$ and $f : C \to \mathbb{R}$ are convex. Show that a vertical supporting hyperplane of $\operatorname{epi}(f)$ cannot contain an interior point of C (i.e., a point of the form $(\mathbf{c}, 0)$ where $\mathbf{c}$ is an interior point of C). Beware that $C \subset \mathbb{R}^n$ but $\operatorname{epi}(f) \subset \mathbb{R}^{n+1}$ and all the supporting hyperplanes also lie in $\mathbb{R}^{n+1}$, not in $\mathbb{R}^n$. Hint: Do the problem first for $n = 1$. For the general case think about a line segment near $\mathbf{c}$ in the direction orthogonal to the hyperplane.

15.17. Prove Proposition 15.3.3.

15.18. Prove Proposition 15.3.4.

15.19. Prove Proposition 15.3.13.

15.20. For each of Exercises 14.17–14.21, identify whether the problem is a convex optimization problem (or can be rewritten as a convex optimization problem).

15.21. Show that both the objective function and the constraint in the problem

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{x^2+1} \\ \text{subject to} \quad & x^2 \geq \tfrac{1}{4}, \\ & 0 \leq x \leq 10 \end{aligned}$$

are not convex. Make a change of variables to rewrite this as a constrained convex optimization problem, and prove that the new objective and constraints are convex.

15.22. Show that the ℓ^1 *linear regression problem*

$$\text{minimize} \quad \|A\mathbf{x} - \mathbf{b}\|_1$$

can be expressed as a linear optimization problem. That is, for a matrix $A \in M_{m\times n}(\mathbb{R})$ and a vector $\mathbf{b} \in \mathbb{R}^m$, write a linear (hence convex) optimization problem whose optimizer $\mathbf{y}^*$ can be used to find a vector $\mathbf{x}^* \in \mathbb{R}^n$ that solves the optimization problem

$$\text{minimize} \quad \|A\mathbf{x} - \mathbf{b}\|_1. \tag{15.45}$$

Identify how to obtain the solution $\mathbf{x}^*$ from $\mathbf{y}^*$.

15.23.* Prove that the Markowitz portfolio optimization problem (see Section 14.6.4) is a convex optimization problem. Prove that it remains a convex optimization problem even if short selling is prohibited.

15.24.* Let $\mathbf{b} \in \mathbb{R}^n$ and $c \in \mathbb{R}$, and let $Q \in M_n(\mathbb{R})$ be a symmetric matrix. The unconstrained optimization problem

$$\text{minimize} \quad f(\mathbf{x}) = \mathbf{x}^\mathsf{T} Q\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$$

is not convex if Q is not positive definite. Recast it as a constrained convex optimization problem as follows:

(i) Find a change of variables $\mathbf{x} = \phi(\mathbf{y})$ that converts the objective function into the form

$$f(\phi(\mathbf{y})) = \mathbf{y}^\mathsf{T} D\mathbf{y} - \mathbf{r}^\mathsf{T}\mathbf{y} + c, \tag{15.46}$$

where $\mathbf{r} \in \mathbb{R}^n$ and $D = \operatorname{diag}(d_1, \ldots, d_n)$ is diagonal (but the d_i are not necessarily positive).

(ii) Show that if $\mathbf{r} = (r_1, \ldots, r_n)$ and if $\mathbf{y}^* = (y_1, \ldots, y_n)$ is any optimal point of (15.46), then for each i we have $r_i y_i \leq 0$. Hint: If not, show that changing the sign of y_i gives a smaller value for the objective function.

(iii) Show that in a neighborhood of the optimal point $\mathbf{y}^*$ we can change variables to $\mathbf{z} = \begin{bmatrix} z_1 & \ldots & z_n \end{bmatrix}^\mathsf{T} \succeq \mathbf{0}$ with $y_i = \operatorname{sign}(r_i)\sqrt{z_i}$.

(iv) Show that the new objective function is a convex function of $\mathbf{z}$ and that the constraints on $\mathbf{z}$ are also convex, so that the new problem expressed in terms of $\mathbf{z}$ is a convex optimization problem.

15.25.* Consider the problem of fitting an ellipse to a set $\{(x_i, y_i)\}_{i=1}^N$ of points in $\mathbb{R}^2$.

(i) The general form of the ellipse can be written as

$$ax^2 + bxy + cy^2 + dx + ey = 1. \tag{15.47}$$

Define the residual ε_i at the point (x_i, y_i) by

$$\varepsilon_i = ax_i^2 + bx_i y_i + cy_i^2 + dx_i + ey_i - 1. \tag{15.48}$$

Find a matrix A and a vector $\mathbf{b}$ such that the residual vector $\boldsymbol{\varepsilon}$ satisfies

$$\boldsymbol{\varepsilon} = A\mathbf{w} - \mathbf{b}, \tag{15.49}$$

where $\mathbf{w} = (a, b, c, d, e)$ is the parameter vector.

(ii) Write the problem of finding the ellipse (the parameters a, b, c, d, and e) that minimizes the errors $\boldsymbol{\varepsilon}$ relative to the norm $\|\cdot\|_1$ as an optimization problem in standard form (this need not be a convex problem).

(iii) Using techniques similar to those used in Exercise 15.22, rewrite the previous optimization problem as a linear problem.

15.26.* A *posynomial* is a function of the form

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{N} c_i \prod_{j=1}^{n} x_j^{a_{ij}},$$

where each a_{ij} is real and each c_i is positive. A *geometric program* is an optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & G(\mathbf{x}) \preceq \mathbb{1}, \\ & \mathbf{x} \succ \mathbf{0}, \\ & H(\mathbf{x}) = \mathbb{1}, \end{aligned}$$

where f and each g_i is a posynomial and each h_i has the form $cx_1^{a_1} \cdots x_n^{a_n}$, with each $a_i \in \mathbb{R}$ and $c > 0$. Show that taking the logarithm of the new objective and new constraints transforms this into a convex optimization problem. Hint: Example 15.1.15 and Exercise 15.3 may be useful.

Remark 15.7.8. The maximum likelihood estimator for logistic regression (see Example 15.3.15) is a geometric program. Generally speaking, geometric programs are not convex, but the change of variables $y_i = \log(x_i)$ transforms a geometric program into a convex problem.

15.27. Prove that the maximum value of $-\frac{1}{2}\boldsymbol{\lambda}^{\mathsf{T}} A A^{\mathsf{T}} \boldsymbol{\lambda} - \boldsymbol{\lambda}^{\mathsf{T}} \mathbf{b}$ is $\frac{1}{2}\mathbf{b}^{\mathsf{T}}(AA^{\mathsf{T}})^{-1}\mathbf{b}$, as claimed in Example 15.4.9. This proves that strong duality holds and hence $d^* = p^*$.

15.28. Consider the problem

$$\begin{aligned} \text{minimize} \quad & x^2 + y^2 \\ \text{subject to} \quad & x + y \geq 4, \\ & x \geq 0, \\ & y \geq 0. \end{aligned}$$

Find the Lagrangian, and the Lagrange dual function. Solve both the primal problem and the dual problem and compare the results.

15.29. Consider the problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} - \sum_{i=1}^{n} \log(x_i) \\ \text{subject to} \quad & \mathbf{r}^{\mathsf{T}}\mathbf{x} = s, \\ & \mathbf{x} \succ \mathbf{0}, \end{aligned}$$

where $\mathbf{r}, \mathbf{x} \in \mathbb{R}^n$. Find the Lagrange dual function. Hint: Since the inequality constraints are strict, you need not include the inequality constraints in the Lagrangian, but the infimum used to compute the dual is taken only over strictly positive values of $\mathbf{x}$.

15.30. Let $W \in M_n(\mathbb{R})$ be positive definite. Consider the problem of choosing $\mathbf{x} \in \mathbb{R}^n$ to minimize $\mathbf{x}^{\mathsf{T}} W \mathbf{x}$, subject to $x_i \in \{-1, 1\}$ (which can be rewritten as $x_i^2 = 1$) for every i. Let p^* be the minimum value. For every choice of $\boldsymbol{\lambda} \in \mathbb{R}^n$ with $(W + \operatorname{diag}(\boldsymbol{\lambda})) \succeq \mathbf{0}$ prove that $p^* \geq -\sum_{i=1}^{n} \lambda_i$.

15.31. In a lot of optimization textbooks, the standard form of a linear optimization problem is to

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \succeq \mathbf{0}. \end{aligned} \tag{15.50}$$

(i) Find the Lagrange dual function.

(ii) Find the dual problem for this problem.

15.32. Consider the optimization problem

$$\begin{aligned} \text{minimize} \quad & S(\mathbf{x}) = \begin{cases} \sum_{i=1}^{n} x_i \log(x_i) & \text{if } \mathbf{x} \succ \mathbf{0}, \\ \infty & \text{otherwise} \end{cases} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{b}, \\ & \mathbb{1}^{\mathsf{T}}\mathbf{x} = 1. \end{aligned}$$

(i) Find the Lagrange dual of S and the corresponding dual optimization problem.

(ii) Under what circumstances would the weak Slater condition fail to hold for this problem?

15.33. For each of Exercises 14.17–14.21, identify whether it satisfies the weak Slater condition.

15.34. For the logistic regression problem of Examples 15.3.15 and 15.5.12 show that the dual function

$$\widetilde{f}(\boldsymbol{\lambda}) = \inf_{\mathbf{w},b,\mathbf{z}} \left(\sum_{i=1}^{N} (\log(1+e^{z_i}) + \lambda_i z_i) + \mathbf{w}^{\mathsf{T}} \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i + b \sum_{i=1}^{N} \lambda_i y_i \right)$$

is $-\infty$ unless every λ_i is bounded by $-1 \leq \lambda_i \leq 0$, in which case it takes the value $\widetilde{f}(\boldsymbol{\lambda}) = \sum_{i=1}^{N} (\lambda_i \log(-\lambda_i) - (1+\lambda_i)\log(1+\lambda_i))$. Note: Assume that $t\log(t) = 0$ when $t = 0$.

15.35. Prove the claim in Example 15.5.13 that the dual problem (15.25) of the (reformulated) soft-margin linear support vector machine primal is to choose $\mathbf{w}, b, \boldsymbol{\xi}$ in order to

$$\begin{aligned} \text{maximize} \quad & -\tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \\ \text{subject to} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0, \\ & \mathbf{0} \preceq \boldsymbol{\alpha} \preceq C\mathbb{1}. \end{aligned}$$

15.36. Consider again the linear optimization problem (15.50) of Exercise 15.31.

(i) Find the dual of the dual.

(ii) Show that (15.50) satisfies strong duality.

15.37.* Prove the claim in the proof of Theorem 15.5.6 that V is convex and not empty.

15.38. Assuming that each g_i is convex, prove that the logarithmic barrier (15.29) is also convex. Hint: It suffices to show that each $-\log(-g(\mathbf{x}))$ is convex.

15.39. Compute the derivative and the Hessian of f_ε as defined in (15.30). More precisely, show that

(i)

$$Df_\varepsilon(\mathbf{x}) = Df(\mathbf{x}) - \varepsilon \sum_{i=1}^{m} \frac{Dg_i(\mathbf{x})}{g_i(\mathbf{x})}, \tag{15.51}$$

(ii)

$$D^2 f_\varepsilon(\mathbf{x}) = D^2 f(\mathbf{x}) - \varepsilon \sum_{i=1}^{m} \left(\frac{D^2 g_i(\mathbf{x})}{g_i(\mathbf{x})} + \frac{Dg_i(\mathbf{x})^{\mathsf{T}} Dg_i(\mathbf{x})}{g_i(\mathbf{x})^2} \right). \tag{15.52}$$

From (15.52) show that if f is convex and each g_i is also convex, then so is f_ε.

15.40. Show that the gradient (15.51) of f_ε is equivalent to the FONC from the KKT conditions (Theorem 14.4.5(i)) for a special choice of $\boldsymbol{\mu} \succeq \mathbf{0}$.

15.41. Consider the minimization problem

$$\begin{aligned} \text{minimize} \quad & f(x) = x^2 + 1 \\ \text{subject to} \quad & 2 \leq x \leq 4. \end{aligned}$$

What is f_ε? Show that the only root of $f_\varepsilon'(x)$ that is in the feasible set is slightly larger than $x = 2$ when ε is slightly larger than zero. Hence, the minimizer $x^*(\varepsilon)$ converges to $x = 2$ (which is the solution).

15.42. Consider the problem in Exercise 15.32. Implement the logarithmic barrier method to solve this optimization problem as follows:

(i) Start with $\varepsilon_0 = 1$ and $\mathbf{x}_0 = (3.5, 4.5)$. For each Newton step, given $\mathbf{x}_j$ choose the learning rate α by backtracking from 1 until the next step $\mathbf{x}' = \mathbf{x}_j - \alpha D^2 f_{\varepsilon_k}(\mathbf{x}_j)^{-1} Df_{\varepsilon_k}(\mathbf{x}_j)^\mathsf{T}$. This satisfies $\mathbf{x}' \in \mathscr{F}$ and $f(\mathbf{x}') + \varepsilon_k b(\mathbf{x}') < f(\mathbf{x}_j) + \varepsilon_k b(\mathbf{x}_j)$. Once such an α has been found, set $\mathbf{x}_{j+1} = \mathbf{x}'$ with that choice of α. Take also $\tau = 0.01$ and $\theta = 0.25$. Terminate the algorithm when $\varepsilon_k < 10^{-6}$.

(ii) Plot in the plane all the values of $\mathbf{x}_k$ that you found in the previous step.

15.43. Consider the simple minimization problem

$$\begin{aligned} \text{minimize} \quad & x^3 + 2 \\ \text{subject to} \quad & 1 \leq x \leq 4. \end{aligned}$$

Set up the primal-dual algorithm by completing the following steps:

(i) Write the KKT conditions for this problem in the form $r(x, \boldsymbol{\mu}) = \mathbf{0}$ as given in (15.42).

(ii) Find $Dr(x, \boldsymbol{\mu})$.

(iii) Plot the region in $\mathbb{R}^3$ corresponding to all $x, \boldsymbol{\mu}$ satisfying the primal and dual feasibility constraints. Plot all the points $(x^*, \boldsymbol{\mu}^*)$ that satisfy the KKT conditions. These are the points to which we expect the algorithm to converge.

(iv) Identify and plot the points of the feasible set satisfying complementary slackness, that is, $\mu_i g_i(x) = 0$ for each $i = 1, 2$.

15.44. Continuing with Exercise 15.43, complete one step of the primal-dual algorithm, as outlined here. Hint: The algebra may become unpleasant, so feel free to use numerical or symbolic computing tools, or your own code whenever convenient.

(i) Plot the point $(3, 1, 1)$ on the feasible set of Exercise 15.43 corresponding to the starting point $x = 3$ and $\boldsymbol{\mu} = (1, 1)$.

(ii) For this starting point and small $\varepsilon > 0$, compute the steps Δx and $\Delta \boldsymbol{\mu}$.

(iii) Compute α using Remark 15.7.5 for $\beta = 0.9$.

(iv) Show that the residual $\|r(x, \boldsymbol{\mu})\|$ decreases as a result of this step.

15.45. Implement the two methods in Algorithms 15.2 and 15.3, applying them to the problem in Example 15.7.4.

15.46. Consider the quadratic optimization problem

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\mathbf{x}^{\mathsf{T}} Q \mathbf{x} + \mathbf{c}^{\mathsf{T}} \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{b}, \end{aligned} \tag{15.53}$$

where $Q \in M_n(\mathbb{R})$ is positive definite and $A \in M_{m \times n}(\mathbb{R})$.

(i) Prove that (15.53) is a convex optimization problem.

(ii) By taking the "relaxed" KKT conditions (15.42), determine $r(\mathbf{x}, \boldsymbol{\mu})$ and $Dr(\mathbf{x}, \boldsymbol{\mu})$.

(iii) Write a program to compute the minimizer.

15.47. Apply the code from the previous exercise to solve the following problem:

$$\begin{aligned} \text{minimize} \quad & 2x_1^2 + x_2^2 - 2x_1 x_2 - 5x_1 - 2x_2 \\ \text{subject to} \quad & 3x_1 + 2x_2 \leq 20, \\ & -5x_1 + 3x_2 \leq 4, \\ & x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

Notes

Our treatment of convexity and convex optimization was partially inspired by [BV04, Ber09]. Other references consulted on convex optimization include [Ber79, Bie15, BL06, CV13, Nes04, Bec14]. Exercise 15.24 is modeled after [Bec14, Section 8.2.7]. Important references for the theory of duality include [BV04, CV13]. Our proof of strong duality from weak Slater is inspired by [Tan15]. Our main source for the logarithmic barrier is [Ber09]. For a detailed analysis of the logarithmic barrier see [dH94].

16 Dynamic Optimization

Let them eat cake.
—Marie Antoinette[58]

The previous chapters on optimization discuss how to make a single decision given a well-formulated problem. This chapter treats sequential decision making, or dynamic optimization, where a sequence of decisions are made at various points in time. Naturally, the decisions made at one point in time affect the later decisions, and so the goal in sequential decision making is to optimize over the entire time horizon of the problem, not just the immediate decision before us.

We begin by considering finite-horizon problems, where decisions are made at a finite number of points in time. Then we move to infinite-horizon problems, where time marches on forever. One of the hallmark achievements in the theory of dynamic optimization is Blackwell's theorem, which gives a necessary condition for the unique solution to a general class of infinite-horizon dynamic optimization problems.

Many dynamic optimization problems fall under the category of investment–consumption problems, where an economic agent has capital that is consumed at fixed periods over time. At each period, a portion of the capital is consumed, giving the consumer a certain amount of *utility*; whatever is not yet consumed is usually invested so that it can grow, giving the agent more to consume later on. In variations of investment–consumption models, there can be an income, or inflow of capital, the ability to borrow and repay money, and sophisticated investment options with a portfolio of choices having uncertain (or probabilistic) outcomes. There can even be hazards, such as medical events or large capital purchases, corresponding to instantaneous costs or shifts in utility or income. Some models even allow for insurance and other derivative securities to be purchased. Indeed, many modern and widely used economic and financial models can be considered as variations of investment–consumption models.

[58] Some historians contend that it is unlikely that Marie Antoinette ever actually said this, but the allegations that she did fueled revolutionaries and ultimately contributed to her demise. Had she been aware of Blackwell's theorem, perhaps things would have gone better for her.

We begin by making considerable simplifications and idealizations of behavior. First, we assume that economic agents are rational,[59] meaning that they will choose the sequence of decisions that maximizes their overall utility. To make this work, we make some assumptions on the mathematical properties of utility functions.

We assume the agent's utility is a smooth function u that takes as input the amount c of capital consumed. The output value $u(c)$ represents the amount of utility enjoyed. This is quantified by a fictitious unit called *utils*. We assume that the more you consume, the happier you are, and so we require that $u'(c) > 0$. However, that happiness also diminishes as consumption increases, so twice the consumption does not give you twice the happiness. This observation is called the *law of diminishing returns*, and it essentially requires that $u''(c) < 0$. In other words, the utility function is increasing and concave; see Figure 16.1 for examples.

Another assumption is that, all things being equal, people prefer immediate gratification over delayed gratification, that is, I'd rather consume now (while my mouth is watering) than later. Mathematically this is expressed as saying that there is some $\beta \in (0, 1]$ for which $u(c)$ utils tomorrow is as desirable as $\beta u(c)$ utils today. The value β is called the *discount factor* and it varies from person to person depending on how well they can delay gratification. A value of $\beta = 1$ corresponds to someone who has no preference for anything today versus tomorrow, which is the same as not discounting at all. A value of $\beta = 0$ would correspond to someone who prefers to consume everything now, with no interest in having anything tomorrow (a hopeless addict), whereas values of β greater than 1 would correspond to people who actually prefer to wait until tomorrow (Scrooge). Since these pathological extremes ($\beta = 0$ or $\beta > 1$) will result in starvation, we usually assume $\beta \in (0, 1]$.

Finally, to prevent the agents from starving themselves, it is common to assume that $u'(c) = \infty$ as $c \to 0^+$, which, in economics language, says that the marginal utility of consuming at least a tiny amount gets arbitrarily large as $c \to 0^+$. The readers should convince themselves that this assumption guarantees it is never optimal to consume nothing.

16.1 Finite-Horizon Cake Eating

We begin the discussion of dynamic optimization by examining the *cake-eating problem*, which is a highly idealized model that contains many of the salient features of an investment–consumption problem. At each point in time $t \in \{0, 1, \ldots, T\}$, the agent has w_{t-1} units of cake going into the period and w_t units of cake coming out of the period. The difference $c_t = w_t - w_{t-1}$ is the amount of cake consumed during the period, reaping a utility of $u(c_t)$ during that period as a result; see Figure 16.1. We constrain c_t to be nonnegative, corresponding to the assumption that we can't eat a negative amount of cake.

As indicated in the introduction, we assume that u is smooth, strictly increasing, and strictly concave. We also assume that $u'(c) \to \infty$ as $c \to 0^+$.

A sequence of decisions $\mathbf{c} = (c_0, \ldots, c_T)$ on how much cake to eat at each period in time is called a *policy*. The *value* of the policy is the *present value* (discounted

[59] There is some debate about whether this is a rational assumption to make.

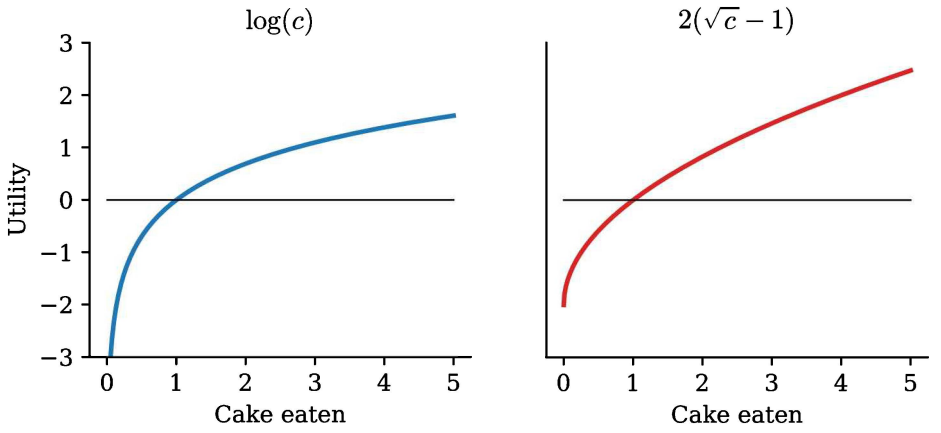


Figure 16.1. *Plot of the two utility functions used in Section 16.1.2. On the left is the logarithmic utility* $u(c) = \log(c)$ *and on the right is the utility for constant relative risk aversion with* $\gamma = \frac{1}{2}$. *For both of these, the utility function* $u(c)$ *is smooth, increasing, and concave, with* $\lim_{c \to 0^+} u'(c) = \infty$.

lifetime utility) resulting from a given policy, that is,

$$V(\mathbf{c}) = \sum_{t=0}^{T} \beta^t u(c_t).$$

Our goal is to find the policy $\mathbf{c}$ with the most value for a given amount of cake w. In other words, we seek to solve the optimization problem

$$\begin{aligned} \text{maximize} \quad & \sum_{t=0}^{T} \beta^t u(c_t) \\ \text{subject to} \quad & \sum_{t=0}^{T} c_t = w, \\ & c_t \geq 0, \quad t = 0, 1, \ldots, T, \end{aligned} \tag{16.1}$$

Reformulating this as the minimization problem

$$\begin{aligned} \text{minimize} \quad & -\sum_{t=0}^{T} \beta^t u(c_t) \\ \text{subject to} \quad & \sum_{t=0}^{T} c_t = w, \\ & c_t \geq 0, \quad t = 0, 1, \ldots, T, \end{aligned} \tag{16.2}$$

shows the cake-eating problem is a convex optimization problem (see Exercise 16.1), and hence it has a unique solution.

16.1.1 Euler Conditions

The Lagrangian for (16.2) is given by

$$\mathscr{L}(\mathbf{c}, \lambda, \boldsymbol{\mu}) = -\sum_{t=0}^{T} \beta^t u(c_t) + \lambda \left(\sum_{t=0}^{T} c_t - w \right) - \boldsymbol{\mu}^\mathsf{T} \mathbf{c}.$$

Since $\lim_{c \to 0+} u'(c) = \infty$, the optimizer will satisfy $c_t > 0$ for all t; hence by complementary slackness, each μ_t vanishes. Thus, we can use the simpler Lagrangian

$$\mathscr{L}(\mathbf{c}, \lambda) = -\sum_{t=0}^{T} \beta^t u(c_t) + \lambda \left(\sum_{t=0}^{T} c_t - w \right). \tag{16.3}$$

Taking the derivative of (16.3) with respect to c_t and setting it equal to zero gives

$$\frac{\partial \mathscr{L}}{\partial c_t} = -\beta^t u'(c_t) + \lambda = 0.$$

In other words, $\lambda = u'(c_0) = \beta u'(c_1) = \beta^2 u'(c_2) = \cdots = \beta^T u'(c_T)$. This gives

$$u'(c_t) = \beta u'(c_{t+1}) \qquad \forall t \in \{0, 1, \ldots, T-1\}. \tag{16.4}$$

These equations are called *Euler's conditions*, and they are necessary conditions for the maximizing solution. Since this problem is convex, the KKT first-order conditions are also sufficient (see Theorem 15.5.10), so a policy $\mathbf{c}$ is a maximizer if and only if it satisfies (16.4).

16.1.2 *Two Canonical Examples

Most dynamic optimization problems do not have a nice closed-form solution and need to be solved numerically. Here are two special cases which do have closed-form solutions.

Logarithmic Utility

A commonly used utility function is $u(c) = \log c$; see Figure 16.1. We solve it for different values of $T \in \mathbb{N}$.

Case $T = 0$. If there is only one period, then it is optimal to eat all the cake at once. Thus, $c_0 = w$.

Case $T = 1$. If there are two periods, then we eat all of the cake in two periods. So we have $c_0 + c_1 = w$. By Euler's condition, we also have that $u'(c_0) = \beta u'(c_1)$, which implies that

$$\frac{1}{c_0} = \frac{\beta}{c_1}.$$

Simplifying gives the two linear equations

$$c_0 + c_1 = w \qquad \text{and} \qquad c_1 = \beta c_0$$

with two unknowns c_0 and c_1. Hence, we have the unique solution

$$c_0 = \frac{w}{1+\beta} \quad \text{and} \quad c_1 = \frac{\beta w}{1+\beta}.$$

Case of general T. The same type of argument as in the previous cases shows that the solution for general T is

$$c_0 = \frac{w}{\sum_{t=0}^{T} \beta^t} \qquad \text{and} \qquad c_t = \beta^t c_0 = \frac{\beta^t w}{\sum_{\tau=0}^{T} \beta^\tau} = \frac{\beta^t(1-\beta)w}{1-\beta^{T+1}}. \tag{16.5}$$

Constant Relative Risk Aversion

A more general utility function used in many economic and financial models is the *constant relative risk aversion* utility given by

$$u(c) = \frac{c^{1-\gamma} - 1}{1-\gamma}, \tag{16.6}$$

where $\gamma > 0$ is a constant that represents risk aversion. Figure 16.1 shows a plot of this utility function for $\gamma = \frac{1}{2}$. The case of $\gamma = 0$ corresponds to $u(c) = c - 1$, or no risk aversion (called *risk neutral*). As γ becomes larger, $u(c)$ becomes more concave (since $u''(c) = -\gamma c^{-\gamma-1}$), corresponding to greater risk aversion. And when $\gamma \to 1$, l'Hôpital's rule shows that (16.6) becomes $\log(c)$.

Using Euler's condition's one can show that the optimal solution for constant relative risk aversion is

$$c_0 = \frac{w}{\sum_{t=0}^{T} \beta^{t/\gamma}} \qquad \text{and} \qquad c_t = \beta^{t/\gamma} c_0 = \frac{\beta^{t/\gamma}(1-\beta^{1/\gamma})w}{1-\beta^{(T+1)/\gamma}}. \tag{16.7}$$

See Exercise 16.6 for details.

16.1.3 The Optimality Principle

Since closed-form solutions are special, we need tools that work more generally. We turn to the Bellman optimality principle, which plays an important role in Dijkstra's algorithm (see Section 4.2.4) and many other optimization methods.

Recall that Bellman's optimality principle says that from any point along an optimal path, the remaining path is optimal for the corresponding problem initiated at that point. If the shortest route from Salt Lake City to Los Angeles passes through Las Vegas, then the last part of that route, from Las Vegas to Los Angeles, must be the shortest route from Las Vegas to Los Angeles. We state this in cake-eating terms with the following definition and proposition.

Proposition 16.1.1 (Finite-Horizon Optimality Principle). *Let $V(a,b,w)$ denote the value for the optimal policy of the cake-eating problem on $\{a, a+1, \ldots, b\}$*

$$\begin{aligned} \text{maximize} \quad & \sum_{t=a}^{b} \beta^t u(c_t) \\ \textit{subject to} \quad & \sum_{t=a}^{b} c_t = w, \\ & c_t \geq 0 \quad \forall t \in \{a, a+1, \ldots, b\}. \end{aligned} \tag{16.8}$$

For any $k \in \{0, \dots, T-1\}$ we have

$$V(0,T,w) = \sup_{y \in [0,w]} \left(V(0,k,y) + V(k+1,T,w-y) \right). \tag{16.9}$$

Proof. Given any $y \in (0,w)$, let $c_0^*, \dots, c_k^*$ be the optimal policy for $V(0,k,y)$, and let $c_{k+1}^*, \dots, c_T^*$ be the optimal policy for $V(k+1,T,w-y)$. The tuple $c_0^*, \dots, c_T^*$ satisfies

$$\sum_{t=0}^{T} c_t^* = \sum_{t=0}^{k} c_t^* + \sum_{t=k+1}^{T} c_t^* = y + w - y = w$$

and

$$\sum_{t=0}^{T} \beta^t u(c_t^*) = \sum_{t=0}^{k} \beta^t u(c_t^*) + \sum_{t=k+1}^{T} \beta^t u(c_t^*) = V(0,k,y) + V(k+1,T,w-y).$$

This implies

$$V(0,T,w) = \sup_{\mathbf{c} \succeq \mathbf{0}} \sum_{t=0}^{T} \beta^t u(c_t) \geq \sup_{y \in [0,w]} \left(V(0,k,y) + V(k+1,T,w-y) \right),$$

where the supremum is taken over all policies $\mathbf{c} \succeq \mathbf{0}$ with $\sum_{t=0}^{T} c_t = w$. Conversely, given any $\mathbf{c} \succeq \mathbf{0}$ with $\sum_{t=0}^{T} c_t = w$, let $\tilde{y} = \sum_{t=0}^{k} c_t$. We have $\sum_{t=k+1}^{T} c_t = w - \tilde{y}$, which yields

$$\sum_{t=0}^{T} \beta^t u(c_t) \leq V(0,k,\tilde{y}) + V(k+1,T,w-\tilde{y}) \leq \sup_{y \in [0,w]} \left(V(0,k,y) + V(k+1,T,w-y) \right).$$

Taking the supremum on the left gives

$$V(0,T,w) \leq \sup_{y \in [0,w]} \left(V(0,k,y) + V(k+1,T,w-y) \right). \quad \square$$

The following corollary is an important special case of the Bellman principle.

Corollary 16.1.2. *For any $T \in \mathbb{N}$ and any $w > 0$ the maximum value of the objective function $V(0,T,w)$ in problem (16.8) satisfies*

$$V(0,T,w) = \sup_{y \in [0,w]} \quad u(y) + \beta V(0,T-1,w-y). \tag{16.10}$$

Proof. The proof is Exercise 16.2. $\square$

Section 16.2 describes how to use the optimality principle to compute a solution to the cake-eating problem.

16.1.4 Optimal Growth

Suppose that, rather than cake, you have an investment portfolio that grows over time. You must balance consuming some of your assets now (say, to pay for food,

housing, and other comforts) against the need to save and/or invest for future consumption. This is the *optimal growth problem.*

The optimal growth problem is like the cake-eating problem, but where cake is replaced by capital. At the beginning of each period you have $x_t \geq 0$ units of capital. Assume that you cannot borrow, so that $x_t \in [0, \infty)$. Also assume that investing x_t units of capital at the beginning of the period gives a return of $f(x_t)$ units at the end of the period, after which a consumption decision is made. Here we assume the function f is continuously differentiable, strictly increasing ($f' > 0$), and concave ($f'' < 0$). Further, assume that $f(0) = 0$.

After the investment outcome is realized, you may choose to consume c_t units, where $0 \leq c_t \leq f(x_t)$. After consumption, you have $x_{t+1} = f(x_t) - c_t$ for the next period. Finally, the utility function $u(c)$ depends only on the amount of capital consumed. As before, assume that u is continuously differentiable, strictly increasing ($u' > 0$), and strictly concave ($u'' < 0$) with $\lim_{c \to 0^+} u'(c) = \infty$.

Assuming an initial capitalization of x_0, solve the dynamic optimization problem

$$\begin{aligned} \underset{\mathbf{c}}{\text{maximize}} \quad & \sum_{t=0}^{T} \beta^t u(c_t) \\ \text{subject to} \quad & 0 \leq c_t \leq f(x_t) \qquad \forall t \in \{0, 1, \dots, T\}, \\ & x_{t+1} = f(x_t) - c_t \qquad \forall t \in \{0, 1, \dots, T-1\}. \end{aligned} \tag{16.11}$$

Specifying the amount c_t to consume is equivalent to specifying the amount x_{t+1} to invest at the next period, so we can reformulate the problem as that of choosing $\{x_1, \dots, x_T\}$ to

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \sum_{t=0}^{T} \beta^t u(f(x_t) - x_{t+1}) \\ \text{subject to} \quad & 0 \leq x_{t+1} \leq f(x_t) \qquad \forall t \in \{0, 1, \dots, T-1\}. \end{aligned} \tag{16.12}$$

Reformulating the problem as a minimization problem gives

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & -\sum_{t=0}^{T} \beta^t u(f(x_t) - x_{t+1}) \\ \text{subject to} \quad & 0 \leq x_{t+1} \leq f(x_t) \qquad \forall t \in \{0, 1, \dots, T-1\}. \end{aligned}$$

Example 16.1.3. If $f(x) = x$, then this model is the same as the cake-eating problem. In this case, unconsumed capital does not grow or shrink.

The Lagrangian is

$$\mathscr{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\nu}) = -\sum_{t=0}^{T} \beta^t u(f(x_t) - x_{t+1}) + \sum_{t=0}^{T} \mu_t (x_{t+1} - f(x_t)) - \sum_{t=0}^{T} \nu_t x_{t+1}.$$

Since $\lim_{c \to 0^+} u'(c) = \infty$, it is never optimal to choose $c_t = 0$ (or $x_{t+1} = f(x_t)$). Moreover, since $f(0) = 0$, choosing $x_{t+1} = 0$ means that $x_{t+k} = 0$ for all $k > 0$; this would be equivalent to setting $T = t$. Finally, since $u'(c) > 0$ for all $c > 0$, it

is never optimal to allow $x_{T+1} > 0$. So we always assume that $0 < x_{t+1} < f(x_t)$ for $t \in \{0, \dots, T-1\}$ and $x_{T+1} = 0$. Therefore, by complementary slackness, we have $\mu_t = 0 = \nu_t$ for all t, and the first-order KKT conditions reduce to the usual unconstrained FONC, which gives[60]

$$0 = \beta^{s-1} u'(f(x_{s-1}) - x_s) - \beta^s f'(x_s) u'(f(x_s) - x_{s+1}) \quad \forall s \in \{1, \dots, T\},$$

where $x_{T+1} = 0$. These give the Euler conditions

$$\beta f'(x_s) u'(f(x_s) - x_{s+1}) = u'(f(x_{s-1}) - x_s) \qquad \forall s \in \{1, \dots, T\}.$$

Expressed in terms of x and c, the Euler equations are usually called the *envelope condition*:

$$\beta f'(x_{s+1}) u'(c_{s+1}) = u'(c_s) \qquad \forall s \in \{0, \dots, T-1\}, \tag{16.13}$$

where $c_T = x_T$ (at the end, all remaining capital must be consumed).

Translated into economics jargon, the envelope condition states that the marginal benefit $u'(c_s)$ of extra consumption today is equal to the discounted marginal cost $\beta f'(x_{s+1}) u'(c_{s+1})$ in terms of lost production (and hence consumption) tomorrow. In some cases the Euler equations or envelope condition can be solved explicitly, but in most cases we must turn to the Bellman principle.

For the optimal growth problem, one case of Bellman optimality (the analogue of Corollary 16.1.2) is the following.

Proposition 16.1.4. *Define $V(a, b, w)$ to be the value of the optimal policy for the optimal growth problem on $\{a, a+1, \dots, b\}$*

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{maximize}} & \displaystyle\sum_{t=a}^{b} \beta^t u(f(x_t) - x_{t+1}) \\ \textit{subject to} & 0 \le x_{t+1} \le f(x_t), \quad t \in \{a+1, \dots, b\}, \\ & x_a = w. \end{array}$$

Thus for any $k \in \{0, \dots, T-1\}$ we have

$$V(0, T, w) = \sup_{y \in [0, f(w)]} \left(u(f(w) - y) + \beta V(0, T-1, y) \right), \tag{16.14}$$

and the maximum value $V(0, T, w)$ is realized with the policy $x_1^*, \dots, x_T^*$ if and only if the maximum value $V(0, T-1, y)$ is realized with the policy $x_2^*, \dots, x_T^*$ and the supremum is realized with $y = x_1^*$.

Proof. The proof is Exercise 16.5. □

The next section describes how to use the optimality principle to compute a solution to the optimal growth problem.

[60] Note that terms corresponding to $\frac{\partial \mathscr{L}}{\partial x_0}$ and $\frac{\partial \mathscr{L}}{\partial x_{T+1}}$ do not show up in these equations because x_0 is the initial capitalization, hence fixed, and $x_{T+1} = 0$.

16.2 Dynamic Optimization Problems and Value Iteration

In this section, we describe a general setting for a large class of dynamic optimization problems and show how Bellman's principle can be applied to solve these problems, using a method called *value iteration*.

16.2.1 The General Framework

As discussed in the introduction of this chapter, there are many variations of investment–consumption problems. Here we create a framework that includes many of these variations and many other types of dynamic optimization problems. In all cases, we seek the policy that maximizes the present value.

Many dynamic optimization problems can be described in the following terms:

- A discrete set $\mathbb{T}$ of time periods (often called *decision epochs*). In this section (and the previous section) we use $\mathbb{T} = \{0, 1, \ldots, T\}$, but in Section 16.3 we also consider $\mathbb{T} = \mathbb{N}$.
- A set S of states (for example, the amount of cake remaining).
- A set A_s of allowable actions for each state $s \in S$ (for example, the amount of cake one can choose to eat).
- A *law of motion*, or *transition function*
$$s_{t+1} = g(s_t, a_t)$$
that describes how the state changes, depending on the previous state and the action.
- A time discount factor $\beta \in (0, 1]$.
- A reward $u_t(s, a)$ for taking action a while being in state s at time t. Often u only depends on a and the state s affects u only by its effect on the set of allowable actions.

Given an initial state s_0 and a law of motion g, dynamic optimization is about choosing a policy of action $\mathbf{a} = (a_t)_{t \in \mathbb{T}}$ to

$$\begin{aligned} \underset{\mathbf{a}}{\text{maximize}} \quad & \sum_{t \in \mathbb{T}} \beta^t u(s_t, a_t) \\ \text{subject to} \quad & s_{t+1} = g(s_t, a_t) \qquad \forall t \in \mathbb{T}. \end{aligned}$$

Problems in this general framework will have a set of Euler equations. The Bellman optimality principle can be applied to give an algorithm (called *value iteration*, which we describe in Section 16.2.5) for solving the problem.

Example 16.2.1. In the cake-eating problem with an initial amount of cake equal to w_0, the state w_t at time t is the amount of cake available, and the state space S is $[0, w_0] \subset \mathbb{R}$. For a given state (amount of cake) w, the allowable actions are to eat some amount of cake $c \in [0, w]$, so $A_w = [0, w]$. Finally, the law of motion is

$$w_{t+1} = w_t - c_t. \tag{16.15}$$

Example 16.2.2. In the optimal growth problem we take $\mathbb{T} = \{0, \dots, T\}$, and the state of the system at each period is the amount of capital x_t. We can assume that $x_t \in [0, \infty)$, so the state space is $S = [0, \infty)$. The set of allowable actions A_x for state x is $A_x = [0, f(x)]$. After consumption, we have $x_{t+1} = f(x_t) - c_t$ units, so the transition function is $g(x, c) = f(x) - c$.

Remark 16.2.3. These models can account for multiple states at once (say we have both cake and pie to eat) by taking S to be a higher-dimensional space like $\mathbb{R}^n$.

16.2.2 Example: Human Capital

Consider an individual entering the workforce whose pay at time period t is based on the product of the hours $h_t \geq 0$ worked during that period and her level $s_t > 0$ of skill. Assuming that she retires after T periods, the present value of her lifetime earnings is given by $\sum_{t=0}^{T} \beta^t h_t s_t$.

Assume that the worker can increase her skill level by spending some hours training instead of working. The state of this system at time t is the worker's skill level $s_t \in S = [0, \infty)$. We take the ratio of skill increase $a_t = s_{t+1}/s_t$ as the action at time t. Assume that if she spends all her time working and not training, then her skill level depreciates at a rate of $\delta > 0$, so $a_t \geq 1 - \delta$. She can grow her skill level at a rate of $\lambda > 0$ if she devotes all her time to training, so $a_t \leq 1 + \lambda$. Thus, the set of allowable actions $A_s = A = [1 - \delta, 1 + \lambda]$ is the same for every state s, and the transition function is $s_{t+1} = g(s_t, a_t) = a_t s_t$.

Assume that the hours available to work in a given period are given by a C^1, decreasing function ϕ of the ratio $a_t = s_{t+1}/s_t$. Therefore, the reward for taking action a at state s is $u(s, a) = s\phi(a)$. We normalize so that $\phi(1 - \delta) = 1$ and $\phi(1 + \lambda) = 0$.

We write this as the dynamic optimization problem

$$\begin{aligned} \underset{(s_t)_{t=0}^T}{\text{maximize}} \quad & \sum_{t=0}^{T} \beta^t s_t \phi\left(\frac{s_{t+1}}{s_t}\right) \\ \text{subject to} \quad & (1 - \delta) s_t \leq s_{t+1} \leq (1 + \lambda) s_t \qquad \forall t \in \{0, \dots, T\}, \end{aligned} \tag{16.16}$$

where the initial skill level is $s_0 > 0$. As in the case of optimal growth, we have written this problem so that the state variables s_t are also the decision variables.

16.2.3 Motion on a Grid

Dynamic optimization has many applications beyond economics problems. One simple example is the problem of motion on a grid. Consider a robot that moves around on the following grid until it reaches the yellow square, at which point it stops moving:

1	2	3
4	5	6
7	8	9

The robot must move at each time step $0, \ldots, T$ unless it reaches the yellow square, and it may move either horizontally or vertically to an adjacent square. Each move has a cost of 1, but entering the red square also gives a reward of 1.7. The goal is to choose a sequence of actions that maximize the total reward.

The state space of this problem is the set of the nine squares of the grid, and the allowable actions for a given state are the horizontally and vertically adjacent squares.

$$\begin{aligned} A_1 &= \{2,4\}, & A_2 &= \{1,3,5\}, & A_3 &= \{2,6\}, \\ A_4 &= \{1,5,7\}, & A_5 &= \{2,4,6,8\}, & A_6 &= \{3,5,9\}, \\ A_7 &= \{4,8\}, & A_8 &= \{5,7,9\}, & A_9 &= \emptyset. \end{aligned}$$

The transition function is simply $g(s,a) = a$. The reward $u(a)$ for action a is

$$u(a) = \begin{cases} -1 & \text{if } a \neq 1, \\ 0.7 & \text{if } a = 1. \end{cases} \tag{16.17}$$

And, of course, the reward for no action (in state 9) is 0.

To write the Bellman equation, let $V(t_0, t_1, s)$ be the maximal reward achievable from time t_0 to time t_1 starting in state s. We have

$$V(0,T,s) = \sup_{a \in A_s} \left(u(a) + \beta V(0, T-1, a) \right).$$

16.2.4 *Example: Inventory Management

Consider a merchant who can sell, at a fixed price p, up to one unit each day of a certain product (she can buy or sell fractions of a unit). If less than one unit is in inventory in a given day, she will sell all her remaining inventory that day. In other words, if she has $x \geq 0$ units in stock, she will sell $\min\{x, 1\}$ units each day.

Assume that she can order y units for inventory at the beginning of a day, paying immediately and then receiving delivery at the end of the day. We assume that the cost of that order is $my + b$, where $b > 0$ is the transaction cost[61] and $m > 0$ is the marginal cost.[62]

She doesn't want to order too often because of the transaction costs, and she doesn't want to order too much at once because of the storage costs involved in holding inventory—assume she must pay rx for storing x units for one day, at some fixed rate $r > 0$.

[61] Transaction costs are costs paid regardless of the size of the order—for example, the cost paid to a broker for facilitating the purchase, or the cost of paying a truck driver to go pick up the order in your truck.

[62] Marginal costs are the price paid per unit in addition to the fixed transaction costs.

We further assume that $0 < m < p/(1+r)$, because if $m \geq p/(1+r)$, then no orders will ever be placed, and if $m = 0$, then the optimal choice would be a one-time infinite order. Assume also that $b > 0$; otherwise the optimal policy is to order one unit per day.

Here the state of the system at time t is the amount of inventory, measured by the state variable x_t, and the decision to make each time is the amount to order, measured by the decision variable y_t. So the state space is $S = [0, \infty)$, and the set of allowable actions $A = [0, \infty)$ is the same for all states. The transition function is the amount of inventory available at the beginning of time $t+1$:

$$x_{t+1} = x_t + y_t - \min\{x_t, 1\}.$$

The reward in a given period t is the profit:

$$u(x_t, y_t) = p\min\{1, x_t\} - rx_t - \begin{cases} b + my_t & \text{if } y_t > 0, \\ 0 & \text{if } y_t = 0. \end{cases}$$

See Exercise 16.11 for more details on this problem.

16.2.5 Value Iteration

The optimality principle gives a general method for solving finite-horizon dynamic optimization problems called *value iteration*, namely, first solving the problem for $T = 0$, and then for $T = 1$, and so forth, until reaching the desired value of T. In other words, the solution is computed using bottom-up dynamic programming (see Section 4.1).

Example 16.2.4. In the cake-eating problem, the first step of the algorithm is to find $V(0, 0, w)$ for every w. Since it is not optimal to leave any cake remaining, we have $c_0^* = w$ and $V(0, 0, w) = u(w)$. The second step is to compute $V(0, 1, w)$ for every w, using (16.10).

$$V(0,1,w) = \sup_{0 \leq y \leq w} \left(u(y) + \beta V(0,0,w-y)\right) = \sup_{0 \leq y \leq w} \left(u(y) + \beta u(w-y)\right).$$

The problem becomes more complicated as more time steps are considered. To make it more computable, we discretize w; that is, we assume that the cake is initially cut into N equal pieces and that each decision involves choosing a whole number of pieces to consume or leave for later. This means the state space is $\left\{0, \frac{w}{N}, \ldots, N\frac{w}{N}\right\}$, and the set of allowable actions for state $k\frac{w}{N}$ is $A_k = \left\{0, \frac{w}{N}, \ldots, k\frac{w}{N}\right\}$. Thus, the problem becomes that of computing

$$V\left(0, 1, k\frac{w}{N}\right) = \max_{n \in \{0,\ldots,k\}} \left(u\left(n\frac{w}{N}\right) + \beta V\left(0, 0, (k-n)\frac{w}{N}\right)\right)$$

for each $k \in \{0, 1, \ldots, N\}$. This is a straightforward discrete optimization problem that can be solved by brute force—just compute

$$u\left(n\frac{w}{N}\right) + \beta V\left(0, 0, (k-n)\frac{w}{N}\right)$$

for every $n \in \{0, \ldots, k\}$ and choose the largest. Continuing in the same manner for each consecutive $t \in \{2, \ldots, T\}$ gives

$$V\left(0, t, k\frac{w}{N}\right) = \max_{n \in \{0,\ldots,k\}} u\left(n\frac{w}{N}\right) + \beta V\left(0, t-1, (k-n)\frac{w}{N}\right)$$

for each $k \in \{0, \ldots, N\}$. At each step, after computing all the $V(0, t, k\frac{w}{N})$, the values of $V(0, t-1, k\frac{w}{N})$ are no longer needed and may be discarded.

Remark 16.2.5. Value iteration is formulated in terms of the optimal value, rather than in terms of the policy that achieves that optimal value. But each step of the algorithm involves solving the optimization problem

$$V(0, T, w) = \sup_y \left(u(y) + \beta V(0, T-1, w-y)\right),$$

and the optimizer y can be saved at each step to identify the overall optimizing policy.

Nota Bene 16.2.6. As mentioned in Nota Bene 13.3.2, the word *programming* is used to mean many things in applied mathematics (we could say it is *overloaded*). Optimization problems are often called *programs*, and *programming* is often used to mean solving these programs. For example, linear optimization problems are often called *linear programs*, and the simplex method is often called *linear programming*.

Dynamic optimization problems are also often called *dynamic programs*, and the various methods (especially value iteration) for solving a dynamic optimization problem are often called *dynamic programming*. The confusion is heightened by the fact that using value iteration to solve dynamic optimization problems is an example of dynamic programming in the sense of Section 4.1. Conversely, many important algorithms that use dynamic programming in the sense of Section 4.1 rely, at heart, on Bellman's optimality principle, and the problems they solve can often be reformulated as dynamic optimization problems.

Of course, not all solutions to dynamic optimization problems rely on dynamic programming. For example, the analytic solutions to the cake-eating problems described in Section 16.1.2 do not use dynamic programming. Also, Blackwell's theorem, described in Section 16.3, leads to two other methods, called *successive approximation* and *policy iteration*, for computing the optimizer.

16.2.6 Example: Value Iteration for Motion on a Grid

The problem of optimal movement on a grid can be solved with value iteration. For simplicity of exposition we assume that $\beta = 1$ for the rest of this section (so the

robot feels no loss from delayed consumption), but the case of general $\beta \in (0,1)$ is no harder to compute.

If $T = 0$, then there is only one move, and $V(0,0,s) = \max_{a \in A_s} u(a)$, so the optimal choice is to move into the red square if possible, earning a reward of $1.7 - 1 = 0.7$. All other moves have reward of -1. Thus, if the robot is in squares 1, 3, or 5, then its optimal move is 1 and its reward for that move is 0.7. If it is in the yellow square, then it has no moves, and the reward is 0. If it is in any other square, all moves have reward -1. Below we have labeled each square s with the value $V(0,0,s)$:

0.7	-1	0.7
-1	0.7	-1
-1	-1	0

Considering the case of $T = 1$, we have

$$
\begin{aligned}
V(0,1,1) &= \max_{a \in \{2,4\}} (u(a) + V(0,0,a)) = \max(0.7 - 1, -1 - 1) = -0.3, \\
V(0,1,2) &= \max_{a \in \{1,3,5\}} (u(a) + V(0,0,a)) = -1 + 0.7 = -0.3, \\
&\vdots \\
V(0,1,7) &= \max_{a \in \{4,8\}} (u(a) + V(0,0,a)) = -2, \\
V(0,1,8) &= \max_{a \in \{5,7,9\}} (u(a) + V(0,0,a)) = -0.3, \\
V(0,1,9) &= 0.
\end{aligned}
$$

Labeling each square s with $V(0,1,s)$ gives

-0.3	-0.3	-0.3
-0.3	-0.3	-0.3
-2	-0.3	0

Similarly, for $V(0,2,s)$ we have

0.4	-1.3	0.4
-1.3	0.4	-1.0
-1.3	-1	0

And for $V(0,3,s)$ we have

-0.6	-0.6	-0.6
-0.6	-0.6	-0.6
-2	-0.6	0

Continuing in this manner gives $V(0,T,s)$ for every T and s.

Vista 16.2.7. Many dynamic optimization problems, especially those with a random aspect (see the next chapter), are called *reinforcement learning* problems. These are an important topic in machine learning. When the ideas of this chapter and the next are combined with other methods of machine learning, like deep neural networks, they become even more powerful. For example, such reinforcement learning methods have been successful in training robots, such as self-driving cars, to navigate complex environments, as well as training a computer to dominate humans and other computer systems in chess, go, and other complex strategy games.

16.2.7 *Variants and Applications of Investment–Consumption

Dynamic investment–consumption problems can go in many directions. The problems could includes uncertain or risky growth, in which case they are called *stochastic* investment–consumption problems. We discuss these in Section 17.1. There are

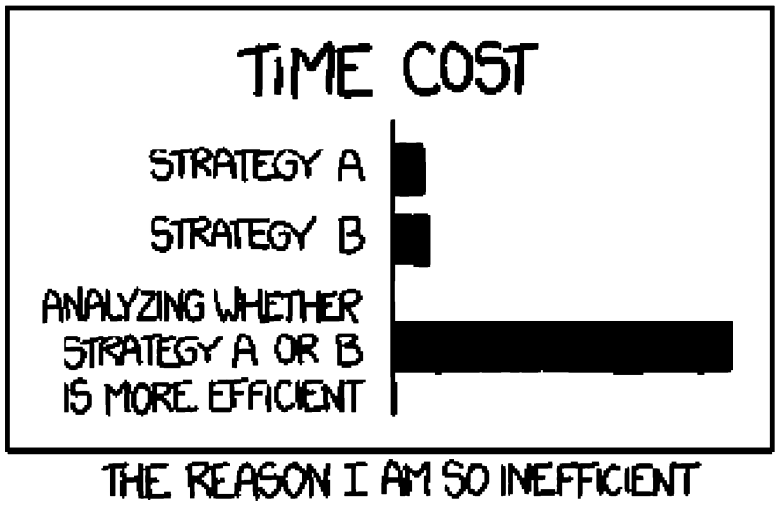


Figure 16.2. *Efficient decision making. Source: XKCD, Randall Munroe.* `http://xkcd.com/1445/`

also situations where a person may have a one-time decision to make, and so the solution to the investment–consumption problem tells them when to pull the trigger on that decision. For example, you might have an annuity or pension that you can start drawing an income from. The longer you wait, the more the monthly payout, but the less time you will have to enjoy that payout, since you will be that much closer to death. These are called *optimal starting* or *optimal stopping* problems. There are also problems with uncertain horizons, such as when you are going to die. You don't want to outlive your wealth, since you can't live comfortably without money. But you can't take it with you, so you also want to enjoy as much as you can before you die.

Investment–consumption problems are pervasive in finance and economics. For example, a government can use these same ideas to think about how to reallocate money through taxation to maximize social welfare. And companies can use these ideas to make decisions about using marketing and capital investment budgets to increase sales.

16.3 Infinite-Horizon Dynamic Optimization

The previous two sections considered dynamic optimization problems where the horizon is finite, signifying that the problems have an end. In some situations, however, we want an infinite horizon. For example a corporation, a government, or a large nonprofit foundation may want to make decisions that are optimal in perpetuity. In these cases, the present value (or discounted lifetime utility) is an infinite series instead of a finite sum. What's remarkable about these problems is that they are often simpler to solve than finite-horizon problems. In this section, we examine a few infinite-horizon problems and prove Blackwell's theorem, which gives general conditions for when an infinite-horizon solution exists.

16.3.1 Cake Eating

Consider the infinite-horizon cake-eating problem

$$\begin{aligned} \text{maximize} \quad & \sum_{t=0}^{\infty} \beta^t u(c_t) \\ \text{subject to} \quad & \sum_{t=0}^{\infty} c_t = w, \\ & c_t \geq 0 \qquad \forall t \in \mathbb{N}. \end{aligned} \tag{16.18}$$

This is just (16.1) in the limit that $T \to \infty$. Here we define the *value function* of the infinite-horizon cake-eating problem to be $V(w) = \lim_{T\to\infty} V(0, T, w)$. This is the maximum discounted infinite-horizon utility that comes from an initial quantity w of cake.

Reindexing (16.18) gives

$$\sum_{t=0}^{\infty} \beta^t u(c_t) = u(c_0) + \beta \sum_{t=1}^{\infty} \beta^{t-1} u(c_t) = u(c_0) + \beta \sum_{t=0}^{\infty} \beta^t u(\tilde{c}_t),$$

where $\tilde{c}_t = c_{t+1}$. Note that

$$w - c_0 = \sum_{t=1}^{\infty} c_t = \sum_{t=0}^{\infty} \tilde{c}_t.$$

Thus, by the principle of optimality, the infinite-horizon version of the Bellman equation can be expressed as

$$V(w) = \sup_{c \in [0,w]} (u(c) + \beta V(w - c)). \tag{16.19}$$

Note that the same result occurs if we let the finite horizon T go to infinity in the finite-time Bellman equation (16.10).

Remark 16.3.1. It is often convenient to replace today's consumption with the difference between today's cake and tomorrow's cake. Specifically, we can denote tomorrow's inventory of cake as $w' = w - c$ and rewrite (16.19) as

$$V(w) = \sup_{w' \in [0,w]} (u(w - w') + \beta V(w')). \tag{16.20}$$

Remark 16.3.2. The expressions (16.19) and (16.20) cover a broad class of problems, not just cake eating. Hence one should consider what follows to be of broad use in dynamic optimization even though the primary focus is the cake-eating problem.

Remark 16.3.3. A policy for a finite-horizon dynamic optimization problem is a choice of values for each of the control variables in the problem. But in the case of a cake-eating problem with infinite horizon, the Bellman equation shows that the optimal policy boils down to deciding how much cake to consume now, given w units of cake. This means that a policy is determined by a function $\pi : \mathbb{R}^+ \to \mathbb{R}^+$, where $\pi(w) \in [0, w]$ is the amount of cake this policy dictates should be consumed if the current amount of cake is w.

16.3.2 *A Canonical Example Revisited

In some special cases the Bellman equation can be solved explicitly. Here are two examples of how that can be done in the special case of cake eating with logarithmic utility.

Limiting Solution as $T \to \infty$

Consider the dynamic optimization problem (16.19) with the log-utility function $u(c) = \log c$. This can be thought of as the limiting case as $T \to \infty$ in the finite-horizon problem in Section 16.1.2. Taking this limit gives $c_t = \beta^t(1-\beta)w$, and thus

the total utility is

$$\begin{aligned} V(w) &= \sum_{t=0}^{\infty} \beta^t u(c_t) = \sum_{t=0}^{\infty} \beta^t \log\left(\beta^t (1-\beta) w\right) \\ &= \sum_{t=0}^{\infty} \beta^t \left(t \log \beta + \log(1-\beta) + \log w\right) \\ &= \left(\sum_{t=0}^{\infty} t\beta^t\right) \log \beta + \left(\sum_{t=0}^{\infty} \beta^t\right) \log(1-\beta) + \left(\sum_{t=0}^{\infty} \beta^t\right) \log w \\ &= \frac{\beta}{(1-\beta)^2} \log \beta + \frac{1}{1-\beta} \log(1-\beta) + \frac{1}{1-\beta} \log w. \end{aligned} \tag{16.21}$$

Note that we weren't really precise in the analysis here. We implicitly passed the limit through the sum by taking the limiting solution for the consumption and then summing it infinitely many times. This can be done rigorously, but even without being rigorous about these limiting operations, it is straightforward to check that the solution we found satisfies the infinite-horizon optimality principle (16.19). Blackwell's theorem (Theorem 16.3.4) shows that this is the unique solution to that equation, so it must be the optimal solution.

Undetermined Coefficients

Consider again the dynamic optimization problem (16.19) with the log-utility function $u(c) = \log c$. Another approach to finding a solution is to guess the form using the *method of undetermined coefficients* (sometimes called the method of inspired guessing), where we make an assumption (called an *ansatz*[63]) about the form of the solution, and solve for the coefficients. It is not always clear how to choose an ansatz (hence the inspired guessing), but in this case it is not unreasonable to guess that since the utility of eating w units of cake now is $\log(w)$, then the optimal utility might also be of the form $V(w) = a + b \log w$ for some positive values of a and b. This is our ansatz. If there exist a and b that satisfy the equation

$$a + b \log w = \sup_{c \in [0,w]} \left(\log c + \beta\left(a + b \log(w - c)\right)\right), \tag{16.22}$$

then we have a solution. Take the derivative of the right-hand side (without the sup) with respect to c and set it equal to zero. This gives

$$0 = \frac{1}{c} - \frac{\beta b}{w - c},$$

which yields $c = \frac{w}{1+\beta b}$. Plugging the optimal c back into (16.22) gives

$$a + b \log w = \log \frac{w}{1+\beta b} + \beta\left(a + b \log \frac{\beta b w}{1+\beta b}\right).$$

Expanding and simplifying gives

$$a + b \log w = \log w - \log(1+\beta b) + \beta a + \beta b \log \beta b + \beta b \log w - \beta b \log(1+\beta b).$$

[63] The word *ansatz* (pronounced *ON-zahts*) is German, meaning *an initial setup*, *a starting point*, or *an approach*.

Since this must hold for all w, we assume that

$$b \log w = \log w + \beta b \log w,$$

or equivalently that $b = 1 + \beta b$, which simplifies to $b = \frac{1}{1-\beta}$. Consider now the terms that are independent of w:

$$\begin{aligned} a(1-\beta) &= -\log(1+\beta b) + \beta b \log \beta b - \beta b \log(1+\beta b) \\ &= \beta b \log \beta b - b \log b, \end{aligned}$$

which simplifies, after substituting $b = \frac{1}{1-\beta}$, to give

$$a = \frac{\beta}{(1-\beta)^2} \log \beta + \frac{1}{1-\beta} \log(1-\beta).$$

Thus, the ansatz works and we have a solution

$$V(w) = \frac{\beta}{(1-\beta)^2} \log \beta + \frac{1}{1-\beta} \log(1-\beta) + \frac{1}{1-\beta} \log w,$$

which is the same as (16.21).

16.3.3 Blackwell's Theorem

The previous two examples give a candidate solution to the infinite-horizon cake-eating problem, but they do not guarantee that this candidate is actually optimal. *Blackwell's theorem* guarantees that such a solution is unique and hence it must be optimal. In fact, Blackwell's theorem gives fairly general conditions for proving both existence and uniqueness of a solution. Moreover, it provides another general approach, called *successive approximation*, for finding the solution numerically. The proof follows from the contraction mapping principle (see Volume 1, Section 7.1).

Theorem 16.3.4 (Blackwell's Theorem). *For any set $X \subset \mathbb{R}^n$, let $L^\infty(X;\mathbb{R})$ denote the set of all bounded functions $f : X \to \mathbb{R}$ with sup-norm $\|f\|_\infty := \sup_{\mathbf{x}\in X} |f(\mathbf{x})|$. If $T : L^\infty(X;\mathbb{R}) \to L^\infty(X;\mathbb{R})$ is an operator satisfying*

(i) *(monotonicity) If $f, g \in L^\infty(X;\mathbb{R})$ satisfy $f \leq g$, then $T[f] \leq T[g]$.*

(ii) *(discounting) There exists some $\beta \in (0,1)$ such that for all $a \geq 0$ and all $f \in L^\infty(X;\mathbb{R})$ we have $T[f+a] \leq T[f] + \beta a$.*

Then T is a contraction mapping with constant β.

Proof. Given the hypothesis, for any $f, g \in L^\infty(X;\mathbb{R})$, we have $f \leq g + \|f-g\|_\infty$. Thus,

$$T[f] \leq T[g + \|f-g\|_\infty] \leq T[g] + \beta \|f-g\|_\infty,$$

which implies $T[f] - T[g] \leq \beta \|f-g\|_\infty$. Interchanging f and g gives the other direction, and thus

$$\|T[f] - T[g]\|_\infty \leq \beta \|f-g\|_\infty. \quad \square$$

Recall that $L^\infty(X;\mathbb{R})$ is a Banach space (Volume 1, Theorem 5.7.6), and so the contraction mapping principle (Volume 1, Theorem 7.1.7) guarantees that any contraction mapping has a unique fixed point. Combined with Blackwell's theorem, this proves there is a unique solution to the Bellman equation (16.19).

Theorem 16.3.5. *The Bellman equation for the infinite-horizon cake-eating problem* (16.20) *on a compact interval* $X = [0, M]$ *(that is, for any* $w \in [0, M]$*) has a unique solution* $V \in L^\infty(X; \mathbb{R})$.

Proof. Following (16.20), consider the map $T : L^\infty(X; \mathbb{R}) \to L^\infty(X; \mathbb{R})$ given by

$$T[f](w) = \sup_{w' \in [0,w]} (u(w - w') + \beta f(w')). \tag{16.23}$$

We call this the *Bellman operator*; see Exercise 16.13 for details. Note that a function V is a fixed point of T if and only if it satisfies

$$V(w) = \sup_{w' \in [0,w]} (u(w - w') + \beta V(w')); \tag{16.24}$$

that is, it is a fixed point of T if and only if it satisfies the optimality principle.

We claim that T is a contraction, and thus it has a unique fixed point V. It suffices to show that T is monotonic and discounting.

(i) Monotonicity: If $f_1(w) \leq f_2(w)$ for all w, then

$$\sup_{w' \in [0,w]} (u(w' - w) + \beta f_1(w')) \leq \sup_{w' \in [0,w]} (u(w' - w) + \beta f_2(w'))$$

for all w; that is, $T[f_1](w) \leq T[f_2](w)$ for all w.

(ii) Discounting: Note that

$$\begin{aligned} T[f + a](w) &= \sup_{w' \in [0,w]} (u(w' - w) + \beta(f(w') + a)) \\ &= \sup_{w' \in [0,w]} (u(w' - w) + \beta f(w') + \beta a) \\ &= T[f](w) + \beta a. \quad \square \end{aligned}$$

16.3.4 Successive Approximation

Blackwell's theorem and Theorem 16.3.5 suggest an iterative method, to maximize the infinite-horizon utility V in the cake-eating problem (16.20). We call this *successive approximation*, but it is also the infinite-horizon version of value iteration. Start with any guess $V_0 \in L^\infty(X; \mathbb{R})$ and define $V_1(w) = T[V_0](w)$. By iterating, we have

$$V_{k+1}(w) = T[V_k](w). \tag{16.25}$$

In the limit, we have the unique solution $V(w) = \lim_{k \to \infty} V_k(w)$. The limit V is guaranteed to exist by the contraction mapping principle (Volume 1, Theorem 7.1.7), and $V \in L^\infty(X; \mathbb{R})$ by completeness.

Performing successive approximation by hand can get really gross really fast and when there's a nice closed-form solution, it's usually easier to solve (16.24) directly than it is to carry out successive approximations by hand. However, successive approximation does lend itself nicely to numerical approximation. One easy way to approximate the sequence $(V_k)_{k \in \mathbb{N}}$ numerically is by discretizing the state space $X = [0, M]$. We do this by choosing a number N and dividing X into N intervals of length M/N. The optimization problems defining T and the final optimal policy are now reduced to searching over a set of N values of the form $w = kM/N$ for $k \in \{0, \ldots, N - 1\}$. Generally, this approximation method converges linearly, at a rate equal to the contraction coefficient β.

Example: Cake Eating by Successive Approximation

Consider the infinite-horizon cake-eating problem with utility $u(c) = \sqrt{c}$. Let the initial guess be $V_0 \equiv 0$. Iterating gives

$$V_1(w) = \sup_{w' \in [0, w]} \left(u(w - w') + 0 \right) = \sqrt{w}$$

with a maximizer of $w' = 0$. The Bellman operator (16.23) on this function is

$$V_2(w) = T[V_1](w) = \sup_{w' \in [0, w]} \left(u(w - w') + \beta V_1(w') \right) = \sup_{w' \in [0, w]} \left(\sqrt{w - w'} + \beta \sqrt{w'} \right).$$

A little calculus shows that the maximizer is

$$w' = \frac{\beta^2}{1 + \beta^2} w, \tag{16.26}$$

which yields

$$V_2(w) = T[V_1](w) = \sqrt{1 + \beta^2} \sqrt{w}.$$

Iterating again (see Exercise 16.14) gives

$$V_3(w) = T[V_2](w) = \sup_{w' \in [0, w]} \left(u(w - w') + \beta V_2(w') \right) = \sqrt{1 + \beta^2 + \beta^4} \sqrt{w}$$

with a maximizer of

$$w' = \frac{\beta^2 + \beta^4}{1 + \beta^2 + \beta^4} w. \tag{16.27}$$

Indeed we can show that if

$$V_k(w) = \left(\sum_{t=0}^{k-1} \beta^{2t} \right)^{1/2} \sqrt{w},$$

then

$$V_{k+1}(w) = T[V_k](w) = \sup_{w' \in [0, w]} \left(u(w - w') + \beta V_k(w') \right) = \left(\sum_{t=0}^{k} \beta^{2t} \right)^{1/2} \sqrt{w},$$

where the maximizer is

$$w' = \frac{\beta^2 + \beta^4 + \cdots + \beta^{2k}}{1 + \beta^2 + \beta^4 + \cdots + \beta^{2k}} w. \tag{16.28}$$

In the limit, we have

$$V(w) = \lim_{k \to \infty} V_k(w) = \left(\sum_{t=0}^{\infty} \beta^{2t} \right)^{1/2} \sqrt{w} = \sqrt{\frac{w}{1 - \beta^2}}.$$

16.3.5 Policy Iteration

From Remarks 16.3.1 and 16.3.3, we know a value function V has a corresponding policy π satisfying

$$\pi(w) = \underset{y \in [0,w]}{\operatorname{argmax}} \quad (u(w-y) + \beta V(y)), \tag{16.29}$$

which gives

$$V(w) = u(w - \pi(w)) + \beta V(\pi(w)). \tag{16.30}$$

As with the infinite-horizon value iteration (given by successive approximation) we can also iterate on the policy function and get convergence.

Starting with an initial guess π_0, we solve for the value function V_0, which is given by

$$\begin{aligned} V_0(w) &= u(w - \pi_0(w)) + \beta V_0(\pi_0(w)) \\ &= u(w - \pi(w)) + \beta \left(u(w - \pi_0(\pi_0(w))) + \beta V(\pi_0(\pi_0(w)))\right) \\ &\vdots \\ &= \sum_{t=0}^{\infty} \beta^t u(\pi_0^t(w) - \pi_0^{t+1}(w)), \end{aligned} \tag{16.31}$$

where π_0^t denotes the t-fold composition $\pi_0 \circ \pi_0 \circ \cdots \circ \pi_0$. Since the policy π_0 is unlikely to be optimal the value function V_0 is also unlikely to be optimal. Hence we can identify a better policy π_1 by solving

$$\pi_1(w) = \underset{y \in [0,w]}{\operatorname{argmax}} \quad (u(w-y) + \beta V_0(y)).$$

We then find V_1 by either computing

$$V_1(w) = \sum_{t=0}^{\infty} \beta^t u(\pi_1^t(w) - \pi_1^{t+1}(w)) \tag{16.32}$$

or by computing

$$V_1(w) = T[V_0](w) = \sup_{y \in [0,w]} \quad (u(w-y) + \beta V_0(y)). \tag{16.33}$$

Repeating gives a sequence of policy functions $(\pi_k)_{k \in \mathbb{N}}$ and value functions $(V_k)_{k \in \mathbb{N}}$. Blackwell's theorem does not apply directly to policy iteration, but these sequences can be shown generally to converge to the optimal policy π and corresponding value function V.

Example: Cake Eating by Policy Iteration

Assume as before that $u(c) = \sqrt{c}$. Given the initial guess $\pi_0 \equiv 0$, we have that (16.31) simplifies to

$$V_0(w) = \sqrt{w}.$$

We improve the policy by solving

$$\pi_1(w) = \underset{y \in [0,w]}{\operatorname{argmax}} \quad (u(w-y) + \beta V_0(y)),$$

which we can show is

$$\pi_1(w) = \frac{\beta^2}{1+\beta^2} w.$$

Hence we update the value function (see Exercise 16.15), using either (16.32) or (16.33), to get

$$V_1(w) = \sqrt{1+\beta^2}\sqrt{w}.$$

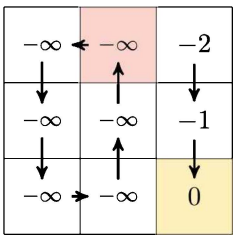
This example of policy iteration is the same as value iteration, except that the indices are offset by one (for example, π_1 is the same as (16.26)). In deterministic problems this can be the case, in particular if the initial guess with policy iteration π_0 corresponds to the initial guess with value iteration V_0, but in stochastic cases or if the initial policy is randomly selected, they are usually different, and policy iteration often converges much faster.

Example: Motion on a Grid

Consider again the case of a robot moving on a 3×3 grid described in Section 16.2.3. Each move gives a reward of -1, but entering the red square gives a reward of 1.7. Once the robot enters the yellow square, it stops moving, but until then it must move with every time step.

The infinite-horizon version of this problem can be attacked in at least two ways. The first is to use finite-time value iteration repeatedly, for longer and longer horizons, until the system stabilizes and gives the same answers with each iteration. In this specific example, the system stabilizes at step $t = 19$, meaning that the optimal value function is the same for all time $t \geq 19$.

Alternatively, we can use policy iteration with random initialization, which works well for this problem. As indicated, let π_0 be randomly initialized, where the policy at each square is to move in the direction of the unique arrow pointing out of the square. The value $V_0(s)$ is also indicated in this diagram at each square s. Note that there is no discounting for this problem (so $\beta = 1$).



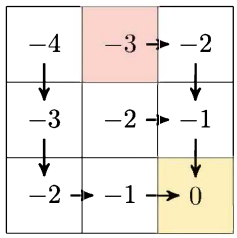
The value function $V_0(s)$ was computed here for each s by simply starting at s and following the policy $\pi_0(s)$, recording the reward $u(\pi_0(s))$ as the value, and then taking action $\pi_0(\pi_0(s))$ and adding $u(\pi_0(\pi_0(s)))$ to the value, and so on, to get

$$V_0(s) = \sum_{k=1}^{\infty} u(\pi_0^k(s)).$$

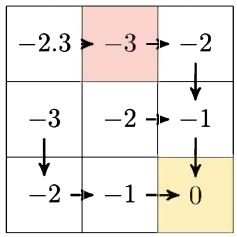
Now compute the new policy $\pi_1(s) = \operatorname{argmax}_{a \in A_s}\ (u(a) + V_0(s))$, as follows. The policy for states 1, 4, and 7 does not change because every action for these states gives the value $-\infty$. But for some states a change in policy improves the value:

$$\pi_1(2) = \operatorname*{argmax}_{a \in \{1,3,5\}} (-1 + \beta V_0(a)) = 3,$$
$$\pi_1(5) = \operatorname*{argmax}_{a \in \{2,4,6,8\}} (-1 + \beta V_0(a)) = 6,$$
$$\pi_1(8) = \operatorname*{argmax}_{a \in \{5,7,9\}} (-1 + \beta V_0(a)) = 9.$$

Here we show the new policy π_1, together with the new value function V_1:



Repeating the process gives a new policy π_2 and a new value function V_2:



It is straightforward to check that $V_2 = T[V_1] = T[V_2]$, so V_2 is a fixed point of T and thus must satisfy the Bellman equation.

Since we used $\beta = 1$, Blackwell's theorem does not guarantee that the value function obtained by this policy is the unique solution of the Bellman equation, but it can be verified that it agrees with the optimal value found using value iteration. Note that policy iteration converges to an optimal policy in only two steps in this example, while value iteration for the same problem takes 19 steps to converge.

Nota Bene 16.3.6. Although the optimal value is unique when $\beta \in (0, 1)$, the optimizing policy need not be unique. Under discounting, Blackwell's theorem guarantees that the optimal value function is the unique fixed point of T, but it does not uniquely determine which policy achieves that value. In the motion-on-a-grid example there are many different policies that produce the same optimal value function. For example, the arrow from 4 to 7 could be replaced with an arrow from 4 to 5 and still yield the same optimal value function.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

16.1. Assuming that a given smooth utility function u satisfies $u' > 0$ and $u'' < 0$, with $\lim_{c \to 0^+} u'(c) = \infty$, reformulate the finite-horizon cake-eating problem as a convex optimization problem, and prove that it is convex.

16.2. Prove Corollary 16.1.2.

16.3. Let $u(c) = \frac{c-1}{c}$, let $w \geq 0$, and let $\beta \in (0, 1)$. Use the Euler conditions to solve the cake-eating problem for this u and for general T.

16.4. Assume you are given a general smooth utility function u satisfying $u' > 0$ and $u'' < 0$, with $\lim_{c \to 0^+} u'(c) = \infty$, and a smooth technology function f (that is strictly increasing ($f' > 0$) and concave ($f'' < 0$), with $f(0) = 0$ as described in Section 16.1.4). Reformulate the finite-horizon optimal growth problem (16.11) as a convex optimization problem, and prove that it is convex.

16.5. Prove Proposition 16.1.4.

16.6.* Carefully write out the details of the proof of (16.7) and verify that letting $\gamma \to 1$ gives (16.5).

16.7. Find the Euler conditions for the human capital problem (16.16) and write the Bellman optimality equation (the analogue of (16.10)) for this problem. To simplify the problem, you may assume that $(1 - \delta)s_t \leq s_{t+1} \leq (1 + \lambda)s_t$ for all $t < T$ and that $s_{T+1} = (1 - \delta)s_t$.

16.8. Suppose that utility in period t of the cake-eating problem depended on the consumption in both the current and the previous time period; that is, suppose the utility is given by $u(c_t, c_{t-1})$. Describe how this could be fit into the general form of Section 16.2.1. Identify the state space, allowable actions, transition function, and reward function. Hint: Consider a two-dimensional state space. What is the corresponding Bellman equation (the analogue of (16.10))?

16.9. Verify all of the details of the value-iteration example in Section 16.2.6: For each of the nine states s in the table, explicitly compute $V(0, T, s)$ for each

$T \in \{0, \dots, 3\}$. Also compute the optimal policy for each initial state and each choice of $T \in \{0, \dots, 3\}$.

16.10. Code up the value iteration method for finding the optimal value function $V(0, T, s)$ in the 3×3 grid motion problem. Your code should accept a time T and a dictionary u of rewards for each action and return the value function $V(0, T, s)$ for every $s \in \{1, \dots, 9\}$. Apply your method to the example in the text with utility given by (16.17). Compare your results to those in the text, and find the value of T where $V(0, t, s) = V(0, t+1, s)$ for all $t \geq T$.

16.11.* Formulate the inventory management problem as a dynamic optimization problem (like (16.16) or (16.11)). Write out the Euler conditions and the Bellman equation (the analogue of (16.10)) for this problem.

16.12. Solve the infinite-horizon version of the cake-eating problem with the utility function given in Exercise 16.3.

16.13. Prove that the Bellman operator T in Theorem 16.3.5 actually takes bounded functions to bounded functions; that is, show that $T : L^\infty(X; \mathbb{R}) \to L^\infty(X; \mathbb{R})$.

16.14. Consider the infinite-horizon cake-eating problem with $u(c) = \sqrt{c}$. Compute the following steps using successive approximation:

(i) $V_2(w) = \sqrt{1 + \beta^2}\sqrt{w}$.

(ii) $V_3(w) = \sqrt{1 + \beta^2 + \beta^4}\sqrt{w}$.

(iii) If $V_k(w) = \sqrt{\sum_{t=0}^{k-1} \beta^{2t}}\sqrt{w}$, then $V_{k+1}(w) = \sqrt{\sum_{t=0}^{k} \beta^{2t}}\sqrt{w}$.

(iv) Show that $V(w) = \lim_{k\to\infty} V_k(w) = \sqrt{\frac{w}{1-\beta^2}}$ satisfies (16.23).

(v) Show that $\pi(w) = \lim_{k\to\infty} \pi_k(w) = \beta^2 w$.

16.15. Compute V_1 for the infinite-horizon cake-eating problem using both methods (16.32) and (16.33).

16.16. Prove that the Bellman equation for the infinite-horizon version of the human capital problem has a unique solution whenever the state space S is bounded.

16.17.* Code up the policy iteration method for the 3×3 grid motion problem. Your code should accept a dictionary u of rewards for each action and a dictionary π giving the initial policy for each state $s \in \{1, \dots, 9\}$. It should return the optimal value function and an optimal policy.

Notes

Exercise 16.8 was inspired by [AC03], which is a great resource for economic models using dynamic optimization. More advanced books are [SLP89, LS18].

17 Stochastic Dynamic Optimization

It's not hard to make decisions when you know what your values are.
—Roy E. Disney

The previous chapter treats deterministic dynamic optimization problems, which means that the outcomes of each decision are known and certain. In this chapter, we consider the case where the outcomes of each decision are unknown and uncertain. Such problems and situations are often called *stochastic*.

A crucial factor in the study of dynamic optimization is the agent's knowledge of the state of a problem. Even if the actual state is known by someone else, the agent may not know it. For example, in a poker game, I want to know whether the player across from me has a full house. She knows what she has, but I don't. To her it's deterministic and to me its stochastic. One of the major challenges in stochastic dynamic optimization is accounting for what is known to the decision maker at the time the decision is to be made. This gets even more complicated when one has to balance the cost of obtaining knowledge (exploration) with the benefit of using knowledge to optimize a utility (exploitation). This trade-off between exploration and exploitation is a major theme of stochastic dynamic optimization.

17.1 Markov Decision Processes

In this section, we consider uncertainty in dynamic optimization. A very natural setting for many such problems is the *Markov decision process*, which we describe here, along with several examples.

17.1.1 Markov Decision Processes

A *discrete-time Markov process* or *Markov chain* is a discrete-time stochastic process (a sequence of random variables) $X_0, X_1, \ldots$, where each X_{n+1} depends only on X_n. That is to say, the conditional probability given all earlier states is the same as the conditional probability given only the preceding state X_n; so, for all sequences $x_0, \ldots, x_n, x_{n+1}$, we have

$$P(X_{n+1} = x_{n+1} | X_0 = x_0, X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x_{n+1} | X_n = x_n).$$

A Markov decision process can be thought of as a Markov chain combined with decisions and rewards. It is a stochastic dynamic optimization problem where the states of the problem evolve according to a Markov chain and where decisions are made along the way to maximize a utility function.

Example 17.1.1. The board game *Monopoly* is a Markov decision process. The players' movements around the board according to sequential dice rolls can be considered a Markov chain. But the players also decide whether to buy properties, houses, and hotels, and they pay or receive money depending on who owns the property and how many houses or hotels are on the property. The additional aspect of decision making with rewards and penalties makes this game a Markov decision process.

Definition 17.1.2. *A* discrete-time Markov decision process (MDP) *is a tuple* $(\mathbb{T}, S, A_s, p_t(s' \mid s, a), r_t(s, s', a), \beta)$*, where*

- $\mathbb{T}$ *is a set of discrete time periods (sometimes called* decision epochs*);*
- S *is a set of states;*
- *for each* $s \in S$*, the set* A_s *is the set of allowable actions;*
- $\beta \in (0, 1]$ *is a discount factor;*
- *for each* $t \in \mathbb{T}$*,* $s, s' \in S$*, and* $a \in A_s$*, the* transition probability $p_t(s' \mid s, a)$ *is the probability of ending in state* s'*, given that the process is in state* s *and action* a *is taken;*
- $r_t(s, s', a)$ *is the reward or expected reward for ending in state* s' *if the process is currently in state* s *and action* a *is taken.*

Finally, we require that $\sum_{s' \in S} p_t(s' \mid s, a) = 1$*, corresponding to the fact that these are probabilities.*

The process is called *Markov* because the probabilities $p_t(s' \mid s, a)$ do not depend on any previous states or actions—only the current state s and the current action a. Of course the current state s may be a result of previous states and actions, but the process is still Markov as long as any effect of those previous states and actions on current transition probabilities is completely captured by the current state s.

For convenience we normally take $\mathbb{T} = \{0, 1, 2, \ldots, T\}$ or $\mathbb{T} = \mathbb{N}$. When $\mathbb{T}$ is finite, we say that the problem has a *finite horizon*. When $\mathbb{T}$ is infinite, we say that the problem has an *infinite horizon*.

Remark 17.1.3. Although we require $\mathbb{T}$ to be discrete, the state space S and the set A_s need not be discrete.

For any MDP the main problem is to find the optimal policy, which is a sequence of actions that maximize the present value of expected rewards. The dynamic optimization methods of the previous sections can be used to solve these problems. Consider the case where the transition probabilities are not time dependent. Let $R(t, s)$ be the maximum present value of the expected reward after t time periods, starting in state s. Bellman's optimality principle applies and takes the form

$$R(t+1, s) = \max_{a \in A_s} \sum_{s' \in S} (p(s' \mid s, a) r_t(s, s', a) + \beta R(t, s')). \tag{17.1}$$

In the case of a finite horizon, we can solve this recursively. In the case of an infinite horizon, Blackwell's theorem applies, and we may start with any initial guess for $R(t, s)$ and iterate the Blackwell contraction mapping to converge on the unique solution.

17.1.2 Example: Stochastic Optimal Growth

Suppose that, as in Section 16.1.4, we have a production function $f(x_t)$, but at each time step this is now also multiplied by a random variable Z_t to give a total output of $Z_t f(x_t)$. The variable Z_t is often called a *random shock* and is meant to account for both good and bad outcomes that might affect output. Examples include technological advances, crop failures, weather, and political strife. We assume that the sequence of shocks is a Markov chain (this could also include the case where they are i.i.d.).

To formulate this as an MDP, let the state $s_t = (x_t, z_t) \in S = [0, \infty) \times [0, \infty)$. In state (x, z) an allowable action is to consume c units, where $0 \le c \le z f(x)$, so the set of allowable actions $A_{(x,z)}$ for state (x, z) is $A_{(x,z)} = [0, z f(x)]$. After consumption at time t, we have $x_{t+1} = z_t f(x_t) - c_t$, so the transition probability is

$$p_t((x', z') \mid (x, z), c) = \begin{cases} 0 & \text{if } x' \neq z f(x) - c, \\ P(Z_{t+1} = z' \mid Z_t = z) & \text{if } x' = z f(x) - c. \end{cases}$$

Finally, the reward is given by the utility function $u(c_t)$, which depends only on the amount c_t of capital consumed. As before we assume that u is continuously differentiable, strictly increasing ($u' > 0$), and strictly concave ($u'' < 0$) with $\lim_{c \to 0^+} u'(c) = \infty$.

We wish to maximize the expected utility over time. The problem is to find the optimal policy $c_t = c(x_t, z_t)$ that determines the choice of how much to consume at each stage, given this period's capital x_t and this period's shock $Z_t = z_t$. That is, assuming that the initial state (x_0, z_0) is given, the problem is to choose the policy $(c_t)_{t \in \mathbb{T}}$ in order to

$$\begin{aligned} \text{maximize} \quad & \mathbb{E}_t \left[\sum_{t=0}^{T} \beta^t u(c_t) \right] \\ \text{subject to} \quad & 0 \le c_t \le z_t f(x_t) \qquad \forall t \in \mathbb{T}, \\ & x_{t+1} = z_t f(x_t) - c_t \qquad \forall t \in \{0, \dots, T-1\}. \end{aligned} \tag{17.2}$$

The notation $\mathbb{E}_t$ here means the expected value, given that we know everything up to time t, including $Z_t = z_t$ and x_t. Equivalently, we can formulate the problem as that of finding an optimal $x_{t+1} = x(x_t, z_t)$ and then compute $c_t = z_t f(x_t) - x_{t+1}$.

As with the deterministic case (16.13), we get the Euler equations (the envelope condition):

$$u'(c_t) = \beta \mathbb{E}_t \left[u'(c_{t+1}) Z_{t+1} f'(x_{t+1}) \right] \tag{17.3}$$

for all $t \in \{0, \dots, T-1\}$. The envelope condition is proved in Exercise 17.6.

Example 17.1.4. A *Cobb–Douglas technology function* is a technology function of the form $f(x_t) = x_t^\alpha$. Assume a Cobb–Douglas technology function and a log-utility $u(c_t) = \log c_t$. The envelope condition gives

$$\frac{1}{c_t} = \beta \mathbb{E}_t \left[\frac{1}{c_{t+1}} Z_{t+1} \alpha x_{t+1}^{\alpha-1} \right]. \tag{17.4}$$

This can be solved analytically with the following ansatz:

$$x_{t+1} = \theta z_t x_t^\alpha \qquad \text{and} \qquad c_t = (1-\theta) z_t x_t^\alpha$$

for some θ to be determined. Plugging into (17.4) gives

$$\frac{1}{(1-\theta) z_t x_t^\alpha} = \beta \mathbb{E}_t \left[\frac{1}{(1-\theta) Z_{t+1} x_{t+1}^\alpha} Z_{t+1} \alpha x_{t+1}^{\alpha-1} \right] = \alpha\beta \mathbb{E}_t \left[\frac{1}{(1-\theta) x_{t+1}} \right].$$

Since x_{t+1} is determined after Z_t is realized, there's no random variable in the expectation, which gives

$$\frac{1}{(1-\theta) z_t x_t^\alpha} = \alpha\beta \mathbb{E}_t \left[\frac{1}{(1-\theta) x_{t+1}} \right] = \alpha\beta \frac{1}{(1-\theta) x_{t+1}} = \alpha\beta \frac{1}{(1-\theta)\theta z_t x_t^\alpha}.$$

Thus, when $\theta = \alpha\beta$, we have equality. It follows that the optimal policy is given by

$$x_{t+1} = \alpha\beta z_t x_t^\alpha \qquad \text{and} \qquad c_t = (1-\alpha\beta) z_t x_t^\alpha.$$

Remark 17.1.5. Following (17.1), we also have the Bellman formulation

$$V(x_t, z_t) = \sup_{c_t, x_{t+1}} \left(u(c_t) + \beta \mathbb{E}_t \left(V(x_{t+1}, Z_{t+1}) \right) \right)$$
$$\text{subject to } c_t + x_{t+1} = z_t f(x_t).$$

As in the deterministic case, this can be used to compute optimal solutions using value iteration and policy iteration, but we don't dive into this here. For details, see the computer lab manual that accompanies this text.

17.1.3 Example: Unemployment and Partial Insurance

Consider a variant of the cake-eating problem where the cake is capital (money) and the cake eaters are workers who not only have some initial amount of capital to consume but also earn a wage each period. Since this is money and not cake, we also

assume they can earn some interest by investing the money. Finally, since the job market is not entirely stable, there is a chance of losing one's job. Luckily, many workers have some access to unemployment insurance that pays a little benefit during periods of unemployment, but the benefit is usually not sufficient, so it makes sense to save something as an emergency fund for times of unemployment. The goal is to find the optimal amount to consume and to save each period, in order to maximize expected utility over all time $\mathbb{T} = \{0, 1, \dots, T\}$.

The first step of this problem is to formulate this as an MDP. The evolution of this process depends both on current wealth and current employment status, suggesting the choice of $S = [0, \infty) \times \{1, 0\}$ as the state space, where 1 represents the state of being employed and 0 the state of being unemployed. We assume that the income for each period is w when employed and b (the benefit) when unemployed. Any savings grow at a rate of r per period, so if x_t is the total amount of money available at the beginning of time t, then before we must decide the amount c_t to consume, the money first grows to $(1 + r)x_t$ and the wage or benefit is paid. Assuming no borrowing, the amount consumed must be nonnegative and no more than the total $(1+r)x_t + w$, if employed, or $(1+r)x_t + b$, if unemployed. Thus, the set A_s of available actions for a state $s \in S$ is

$$A_s = \begin{cases} [0, (1+r)x + w] & \text{if } s = (x, 1), \\ [0, (1+r)x + b] & \text{if } s = (x, 0). \end{cases}$$

Employment status is assumed to be a Markov chain, so the probability of being employed or unemployed next period depends on current employment status. Let the transition probability matrix for employment status be

$$\begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix},$$

where the probability of staying employed is p_{11}, the probability of getting a new job if unemployed is p_{10}, and so forth. Further assume that the transition from one employment status at time t to another at time $t+1$ occurs after wage or benefit has been paid and after the decision c_t has been made. Thus the transition probability $p(s' \mid s, c)$ for moving from state s to state s', after taking action c, is

$$p(s' \mid s, c) = \begin{cases} p_{00} & \text{if } s = (x, 0) \text{ and } s' = ((1+r)x + b - c, 0), \\ p_{01} & \text{if } s = (x, 1) \text{ and } s' = ((1+r)x + w - c, 0), \\ p_{10} & \text{if } s = (x, 0) \text{ and } s' = ((1+r)x + b - c, 1), \\ p_{11} & \text{if } s = (x, 1) \text{ and } s' = ((1+r)x + w - c, 1), \\ 0 & \text{otherwise.} \end{cases}$$

The reward at each time step is determined only by the utility $u(c)$, satisfying the same assumptions as in the original cake-eating problem.

Let $V^1(x_t)$ denote the maximum expected utility attainable if the worker is employed at time t and has wealth x_t, and let $V^0(x_t)$ denote the maximum expected utility if the worker is unemployed at time t. The Bellman equation for this situation

breaks into two cases:

$$V^1(x_t) = \sup_{x_{t+1}} \big(u(c_t) + \beta(p_{11}V^1(x_{t+1}) + p_{10}V^0(x_{t+1}))\big)$$
$$\text{with } c_t = (1+r)x_t + w - x_{t+1},$$

$$V^0(x_t) = \sup_{x_{t+1}} \big(u(c_t) + \beta(p_{01}V^1(x_{t+1}) + p_{00}V^0(x_{t+1}))\big)$$
$$\text{with } c_t = (1+r)x_t + b - x_{t+1}.$$

17.1.4 Example: Uncertain Robot Motion

Consider a variant of the grid-motion problem of Sections 16.2.3 and 16.3.5, where a robot moves around on the grid until it reaches the yellow square, at which point it stops moving.

1	2	3
4	5	6
7	8	9

The robot may attempt to move horizontally or vertically to an adjacent square, but, for each of the other adjacent squares, it could move to that square with probability p instead. Each move has a cost of 1, but entering the red square gives a reward of 1.7. The goal is to choose a policy that will maximize the lifetime reward of moving around the grid.

In this setting $\mathbb{T} = \mathbb{N}$ because there is no upper bound on the number of steps that will be taken before reaching the yellow square. The state space of this MDP is the set of the nine squares of the grid, and the allowable actions for a given state are the horizontally and vertically adjacent squares:

$$\begin{aligned} A_1 &= \{2,4\}, & A_2 &= \{1,3,5\}, & A_3 &= \{2,6\}, \\ A_4 &= \{1,5,7\}, & A_5 &= \{2,4,6,8\}, & A_6 &= \{3,5,9\}, \\ A_7 &= \{4,8\}, & A_8 &= \{5,7,9\}, & A_9 &= \emptyset. \end{aligned}$$

The transition probabilities are

$$p(s' \mid s, a) = \begin{cases} p & \text{if } s' \in A_s \setminus \{a\}, \\ 1 - (|A_s| - 1)p & \text{if } s' = a, \\ 0 & \text{otherwise.} \end{cases}$$

This problem is amenable to the same solution techniques as the deterministic infinite-horizon problems of the previous chapter.

17.2 Bandit Problems

The bandit problem was formulated during the [Second World] War, and efforts to solve it so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany, as the ultimate instrument of intellectual sabotage.
—Peter Whittle

An important class of Markov decision processes is the class of *bandit problems*. These are problems that can be modeled as a row of slot machines.[64] Each machine has its own initially unknown payout distribution. Which machines should gamblers play to maximize their total expected reward? To do this optimally they must balance between exploration and exploitation. In other words, the gamblers must learn while also trying to maximize their return. These types of problems arise in many settings, ranging from testing of pharmaceuticals to Internet advertising.

17.2.1 Multiarmed Bernoulli Bandits

In a bandit problem the states of the system are usually not physical states but rather information states—the state of our current knowledge about the reward distribution of each machine. Possible actions are the choice of which machine's lever (arm) to pull.

First consider the case of a single machine with a Bernoulli payout distribution of fixed, unknown probability θ. That is, the machine pays a fixed amount J with a fixed, but unknown, probability θ, and it pays nothing with probability $1 - \theta$. The state of our knowledge is the number of wins and losses already observed, so the state space is $\mathbb{N} \times \mathbb{N} = \{(a, b) \mid a, b \in \mathbb{N}\}$.

Each successful pull of the machine's arm results in an update from state (a, b) to state $(a + 1, b)$, while each failed pull updates to state $(a, b + 1)$. Thus, the transition probability is given by

$$p((a+1,b) \mid (a,b)) = \theta \qquad \text{and} \qquad p((a,b+1) \mid (a,b)) = 1 - \theta.$$

We can estimate the value of θ using the MLE:

$$\hat{\theta} = \frac{a}{a+b}.$$

Of course if there have been no pulls, that is, if the state is $(0, 0)$, then the estimate is undefined. If the machine is in state (a, b), the estimated expected payout of the next pull is $J\hat{\theta}$. We also include discounting, so the estimated expected present value of pulling the arm at time t is $r = \beta^{t-1} J\hat{\theta}$, where $\beta \in (0, 1]$ is the discount factor.

For a collection of n independent Bernoulli machines, where machine i has actual payout J_i and unknown probability θ_i, the state space for the whole collection of machines is the product

$$S = (\mathbb{N} \times \mathbb{N}) \times (\mathbb{N} \times \mathbb{N}) \times \cdots \times (\mathbb{N} \times \mathbb{N}) = (\mathbb{N} \times \mathbb{N})^n.$$

[64] A slot machine is sometimes called a one-armed bandit.

The set of possible actions is $A = \{1, \ldots, n\}$, corresponding to the choices of which machine to play.

The expected (undiscounted) reward for decision i in state $(s_1, \ldots, s_n)$ is the expected reward for playing machine i in state $s_i = (a_i, b_i)$ at time t; so,

$$r((s_1, s_2, \ldots, s_n), i) = r_i(s_i) = \hat{\theta}(s_i) J_i.$$

Transitions change only the state of (our knowledge of) the one machine whose arm was pulled, and in that case the transition probability is determined by that one machine. So, given action i, the transition probabilities of going from state $(s_1, \ldots, s_n)$ to state $(s'_1, \ldots, s'_n)$ are

$$p((s'_1, \cdots, s'_n) \mid (s_1, \cdots, s_n), i) = \begin{cases} p(s'_i \mid s_i) & \text{if } s'_m = s_m \text{ for all } m \neq i, \\ 0 & \text{if } s'_m \neq s_m \text{ for any } m \neq i. \end{cases}$$

Application 17.2.1. Multiarmed Bernoulli bandit problems have been used in a medical setting to improve clinical trials. In a typical clinical trial application, there are n treatments (arms). These treatments can be given to a patient repeatedly, in any order. Each treatment results in either success or failure. Discounting applies here because if a treatment succeeds, the reward of being healed is affected by timing: patients prefer to be healed sooner than later, and if they spend more time suffering and more money on treatments over a longer period of time, this reduces their total utility of being healed. Thus a success results in a reward of the form β^{t-1}, where $\beta \in (0, 1]$ is the discount factor. If the treatment fails, obviously the patient isn't healed and has no reward.

One policy for a multiarmed bandit problem is to pull the arm for which the current expected reward is largest. An arm $i \in \{1, \ldots, n\}$ in state $s_i = (a_i, b_i)$ has expected reward $\hat{\theta}_i J_i = \frac{a_i}{a_i + b_i} J_i$. Thus, this policy pulls arm j with

$$j = \operatorname*{argmax}_{i \in \{1, \ldots, n\}} \hat{\theta}_i J_i. \tag{17.5}$$

While this policy may seem good, it is not optimal because it is myopic—it does not address the possibility of greater future rewards from identifying a better machine. If there is a tie, where the means of two arms are equal and maximal, but the estimated variance of one is much greater than the other, then it is advantageous to select the arm with greater estimated variance, since the outcome of that pull will give more information than pulling the low-variance arm. If the expected value of the lower-variance arm is only slightly higher than that of the higher-variance arm, then it should still be advantageous to select the arm with greater variance, but the previous policy does not account for that.

A better policy would be to formulate this as a recursive dynamic optimization problem. In the case that $n = 2$, setting $R(a_1, b_1, a_2, b_2, J_1, J_2, N)$ to be the maximal

total reward obtainable starting in state $((a_1, b_1), (a_2, b_2))$ over N periods, we seek to solve

$$\begin{aligned} R(a_1, b_1, a_2, b_2, J_1, J_2, N) = \max \Bigg\{ & \frac{a_1}{a_1 + b_1}[J_1 + \beta R(a_1 + 1, b_1, a_2, b_2, N - 1)] \\ & + \frac{b_1}{a_1 + b_1}\beta R(a_1, b_1 + 1, a_2, b_2, N - 1), \\ & \frac{a_2}{a_2 + b_2}[J_2 + \beta R(a_1, b_1, a_2 + 1, b_2, N - 1)] \\ & + \frac{b_2}{a_2 + b_2}\beta R(a_1, b_1, a_2, b_2 + 1, N - 1) \Bigg\}. \end{aligned} \tag{17.6}$$

This can be used to recursively compute the value of $R(a_1, b_1, a_2, b_2, J_1, J_2, N)$ in terms of four values of R, where N is replaced by $N - 1$, and each of those requires four values of R, where $N - 1$ is replaced by $N - 2$, and so on, ultimately requiring computation of $O(4^N)$ values of R. Computing iteratively (bottom-up dynamic programming), instead, requires computing values of $R(a_1 + i_1, b_1 + j_1, a_2 + i_2, b_2 + j_2, J_1, J_2, t)$ for all values of $t \in \{1, 2, \ldots, N\}$ and all values of i_1, i_2, j_1, j_2 such that $i_1 + j_1 + i_2 + j_2 = t$, thus requiring the computation of $\sum_{t=1}^{N} p_4(t) = O(N^4)$ values of R, where $p_4(t)$ is the number ways of writing t as a sum of four nonnegative integers. More generally, for a bank of bandits with n arms, the corresponding iterative computation requires $O(N^{2n})$ subcomputations. This becomes very difficult, computationally, as $N \to \infty$ even for relatively small values of n. Thus, for a very long or infinite horizon we need another approach.

17.2.2 Indexing

Indexing gives an alternative algorithm for deciding the optimal next action for a general Bernoulli bandit process with n arms with infinite horizon. The idea of the index is to compare a bandit in state (a, b) to a simple bandit with known, fixed payout. We call a one-armed bandit a *simple bandit with reward* r if it has an (undiscounted) expected value of r. Here we mean that r is the true expected value and not an estimate of its value. The expected present value (that is, the value now, after discounting) for an infinite lifetime of playing this simple bandit is

$$\sum_{t=0}^{\infty} r\beta^t = \frac{r}{1 - \beta}.$$

Consider now the special case of a two-arm Bernoulli bandit, where arm 1 has a payout of J_1 and an (unknown) success probability θ, which currently we estimate to be $\hat{\theta} = a/(a + b)$, having observed a successes and b failures. Arm 2 is a simple bandit with reward J_2. Let $R(a, b, J_1, J_2)$ denote the optimal expected reward from playing this bandit system with infinite horizon, and with starting state (a, b) on the first machine. Note that $R(a, b, J_1, J_2) = J_1 R(a, b, 1, J_2/J_1)$, so setting $r = J_2/J_1$ and finding $R(a, b, 1, r)$ for all a, b, r will give the general case just by multiplying by J_1. From now on we use the notation $R(a, b, r) = R(a, b, 1, r)$.

The Bellman equation for this situation is

$$R(a,b,r) = \max\left\{\frac{a(1+\beta R(a+1,b,r)) + b\beta R(a,b+1,r)}{a+b}, \frac{r}{1-\beta}\right\}. \tag{17.7}$$

A rough argument for why the Bellman equation takes this form is given at the end of this section.

If r is very small, then the second term $\frac{r}{1-\beta}$ inside the maximum in (17.7) is very small, while the first term is at least $\frac{a}{a+b}$; so the first term is larger, and pulling the first arm is the better choice. Alternatively, if r is very large, then it should never make sense to pull the first arm, so $R(a,b,r)$ should be equal to the expected reward $\frac{r}{1-\beta}$ of always pulling the second arm. If $R(a,b,r)$ is a continuous function of r, then there must be some value of r where the two arms give the same expected reward, that is, when the two inner terms of (17.7) are equal:

$$\frac{r}{1-\beta} = \frac{a}{a+b}[1+\beta R(a+1,b,r)] + \frac{b}{a+b}\beta R(a,b+1,r). \tag{17.8}$$

It can be shown that there is a unique value of r that satisfies (17.8). This value is called the *Gittins index.*

Definition 17.2.2. *The* Gittins index $\nu(a,b,1)$ *of a Bernoulli bandit in state* (a,b) *with a winning payout of* 1 *is defined to be the value of* r *that makes* (17.8) *hold:*

$$\begin{aligned}\nu(a,b,1) = \Bigg(&\frac{a(1-\beta)}{a+b}[1+\beta R(a+1,b,\nu(a,b,1))] \\ &+ \frac{b(1-\beta)}{a+b}\beta R(a,b+1,\nu(a,b,1))\Bigg).\end{aligned} \tag{17.9}$$

Define $\nu(a,b,J) = J\nu(a,b,1)$ *for all* J.

It follows immediately from the definition that the present value of an arm in state (a,b) with payout J is the same as the present value of a simple bandit with reward $\nu(a,b,J)$.

The Algorithm

Assuming the Gittins index can be calculated in a reasonable amount of time, this leads to an algorithm for deciding the optimal next action for a general Bernoulli bandit process with n arms. Assume that for each i the ith arm is in state (a_i,b_i) and has winning payout J_i. For each arm i compute the index $\nu(a_i,b_i,J_i)$ and then select the arm with the greatest index. This is the optimal policy because arm i is equivalent in expected present value to a simple bandit with reward $\nu(a_i,b_i,J_i)$, and thus the bandit with the highest index is the choice with the highest expected value.

We approximate the Gittins index as follows. First, if $a+b$ is large enough, then the MLE $\hat{\theta} = \frac{a}{a+b}$ is a good estimate of the true probability θ, so it is reasonable, in that case, to assume that the first arm is a simple bandit with known probability $\hat{\theta}$. Doing this amounts to replacing the first part of (17.7) with the expected present value $\frac{a}{a+b}\frac{1}{1-\beta}$. Thus, it is reasonable to choose a large value M and assume that

$$R(a,b,r) = \frac{1}{1-\beta}\max\left\{\frac{a}{a+b}, r\right\} \qquad \text{if } a+b \geq M. \tag{17.10}$$

Finally, $\nu(a,b,1)$ must be approximated when $a+b<M$. This can be done with a variant of the bisection algorithm from Section 12.2 to find a value of r that comes close to giving equality in (17.8), but where each function evaluation $R(a+1,b,r)$ or $R(a,b+1,r)$ is computed recursively, using (17.7) and (17.10).

Alternatively $\nu(a,b,1)$ can be computed with an exhaustive (brute force) search, as follows: Choose a fine grid of K possible values $r \in (0,1)$, and for each $a,b \in \mathbb{N}\times\mathbb{N}$ with $a+b \leq M$ use (17.7) and otherwise use (17.10) to compute $R(a,b,r)$, recursively. Then approximate $\nu(a,b,1)$ as the value of r in the grid that comes closest to giving equality in (17.8).

The brute force index algorithm requires a constant number of computations per choice of a,b,r, so it has total complexity $O(KM(M+1))$ to approximate the Gittins index, which is generally much cheaper to compute than (17.6) for large N.

Remark 17.2.3. If the computation of the Gittins index is only needed for arms in states (a_i,b_i) with $a_i+b_i \geq m$, then there is no need to compute the values $R(a,b,r)$ for $a+b<m$.

Example 17.2.4. Gittins index computation for $M=4$.

We show how to approximate the Gittins indices $\nu(a,b,1)$ for all states (a,b) with $a+b \leq M=4$ for a bandit process having discount factor $\beta = 0.9$.

First select a grid of r-values, say, $\{0.01, 0.02, \ldots, 0.98, 0.99\}$. One must iterate through all the choices of r, but for purposes of this example, assume that the current choice is $r=0.35$. This gives

$$\frac{r}{1-\beta} = \frac{0.35}{1-0.9} = 3.5.$$

To simplify notation for this example, since $r=0.35$ is fixed here, write $R(a,b) = R(a,b,r)$. To initialize, approximate $R(a,b)$ for $a+b=4$ by using (17.10).

$$\begin{aligned}
R(0,4) &= \max\{0, 3.5\} = 3.5,\\
R(1,3) &= \max\{2.5, 3.5\} = 3.5,\\
R(2,2) &= \max\{5, 3.5\} = 5,\\
R(3,1) &= \max\{7.5, 3.5\} = 7.5,\\
R(4,0) &= \max\{10, 3.5\} = 10.
\end{aligned}$$

Now calculate for each $a+b=3$ using (17.7):

$$R(0,3)=\max\left\{\frac{0}{3}[1+.9(3.5)]+\frac{3}{3}.9(3.5),3.5\right\}=\max\{3.15,3.5\}=3.5,$$

$$R(1,2)=\max\left\{\frac{1}{3}[1+.9(5)]+\frac{2}{3}.9(3.5),3.5\right\}=\max\{3.93,3.5\}=3.93,$$

$$R(2,1)=\max\left\{\frac{2}{3}[1+.9(7.5)]+\frac{1}{3}.9(5),3.5\right\}=\max\{6.66,3.5\}=6.66,$$

$$R(3,0)=\max\left\{\frac{3}{3}[1+.9(10)]+\frac{0}{3}.9(7.5),3.5\right\}=\max\{10,3.5\}=10.$$

And for $a+b=2$,

$$R(0,2)=\max\left\{\frac{0}{2}[1+.9(3.93)]+\frac{2}{2}.9(3.5),3.5\right\}=\max\{3.5,3.15\}=3.5,$$

$$R(1,1)=\max\left\{\frac{1}{2}[1+.9(6.66)]+\frac{1}{2}.9(3.93),3.5\right\}=\max\{3.5,5.27\}=5.27,$$

$$R(2,0)=\max\left\{\frac{2}{2}[1+.9(10)]+\frac{0}{2}.9(6.66),3.5\right\}=\max\{3.5,10\}=10.$$

And finally for $a+b=1$,

$$R(0,1)=\max\{.9R(0,2),3.5\}=3.5,$$
$$R(1,0)=\max\{1+.9R(2,0),3.5\}=10.$$

The case of $(a,b)=(0,0)$ does not make sense in the formulas and is not useful anyway, because the state $(0,0)$ gives no information about the process.

The algorithm is finished by performing these same calculations for all r values and all (a,b) with $1\le a+b\le 4$. For each combination of a and b, determine the value of r that most nearly satisfies (17.8)—this value of r is the (approximate) Gittins index $\nu(a,b,1)$.

17.2.3 *Rough Argument for Bellman

We give a heuristic argument, using (17.6), for why the Bellman equation (17.7) holds. First, if $N=\infty$, then $N-1=N$, and a change in the number of wins or losses on the second (known probability) machine will not change our knowledge of r, so we have $R(a_1,b_1,a_2,b_2,N)=R(a_1,b_1,r)$. Next, wherever the MLE estimate $\hat{\theta}_2=\frac{a_2}{a_2+b_2}$ occurs in (17.6), replace it with the actual value $\theta_2=p$. Finally, the expected value of arm 2 is $r=pJ_2$. Making these substitutions gives

$$\begin{aligned} R(a,b,r) &= \max\Big\{\frac{a}{a+b}[1+\beta R(a+1,b,r)] + \frac{b}{a+b}\beta R(a,b+1,r), \\ &\qquad p[J_2 + \beta R(a,b,r)] + (1-p)\beta R(a,b,r)\Big\} \\ &= \max\Big\{\frac{a}{a+b}[1+\beta R(a+1,b,r)] + \frac{b}{a+b}\beta R(a,b+1,r), \\ &\qquad r + \beta R(a,b,r)\Big\}. \end{aligned} \tag{17.11}$$

If it is optimal to pull arm 2 at any time, then it must be optimal to pull arm 2 every time after that, since there will be no new information about arm 1, keeping the first part of (17.11) the same. Therefore, if the maximum is attained by the second term in (17.11), then we have $R(a,b,r) = r + \beta R(a,b,r)$ and $R(a,b,r) = r/(1-\beta)$. This implies, regardless of which arm is optimal, that (17.7) holds.

17.3 Thompson Sampling

Although solving bandit problems using the Gittins index is much more efficient than value iteration, it is still costly for large-scale problems. In this section we describe a fast and powerful heuristic method, motivated by Bayesian estimation, for approximating the solution to a bandit problem. This method is sometimes called *Thompson sampling* and is widely used to identify optimal choices of which web pages to present to users, but it has many other applications as well.

17.3.1 The Bayesian Framework

We do not know the true values of the probabilities θ_i, but given a history of successful and unsuccessful pulls, we can say something about what range we think they might be in. If you have had one success and two failures on machine i, the MLE for θ_i is $\hat{\theta}_i = \frac{1}{3}$. But this estimate is probably not very useful, especially when it is based on so little information. Instead of choosing a single value $\hat{\theta}_i$, it is better to quantify our current knowledge in terms of a probability distribution for the value of θ_i. This is the Bayesian framework (see Section 6.5).

In this setting, it is natural to use a prior distribution of the form $\text{Beta}(a,b)$ for θ_i, since Beta is the conjugate prior for the Bernoulli distribution. New information is incorporated by updating the prior to the appropriate posterior distribution. This framework allows us to fluidly incorporate new information as we seek to determine the true nature of θ_i.

Recall from Section 5.6.2 that beta distributions $\text{Beta}(a,b)$ are defined on $[0,1]$ and parametrized by a pair (a,b). The distribution $\text{Beta}(a,b)$ has most of its mass concentrated around its mean $\hat{\theta}_i = a/(a+b)$, and it becomes more concentrated around $a/(a+b)$ as $a+b$ gets larger. Moreover, the distribution $\text{Beta}(1,1)$ is the uniform distribution $\text{Uniform}([0,1])$, so it is a popular choice for a prior when there is no other information about the distribution of θ_i.

Starting with the prior distribution $\text{Beta}(a,b)$, if the next pull is a success, the posterior distribution becomes $\text{Beta}(a+1,b)$. And if the next pull is a failure, then the posterior distribution is $\text{Beta}(a,b+1)$.

As we get more information, the sum $a+b$ becomes larger, and the peak of $\text{Beta}(a,b)$ gets narrower, corresponding to a lower variance. In Figure 17.1, all

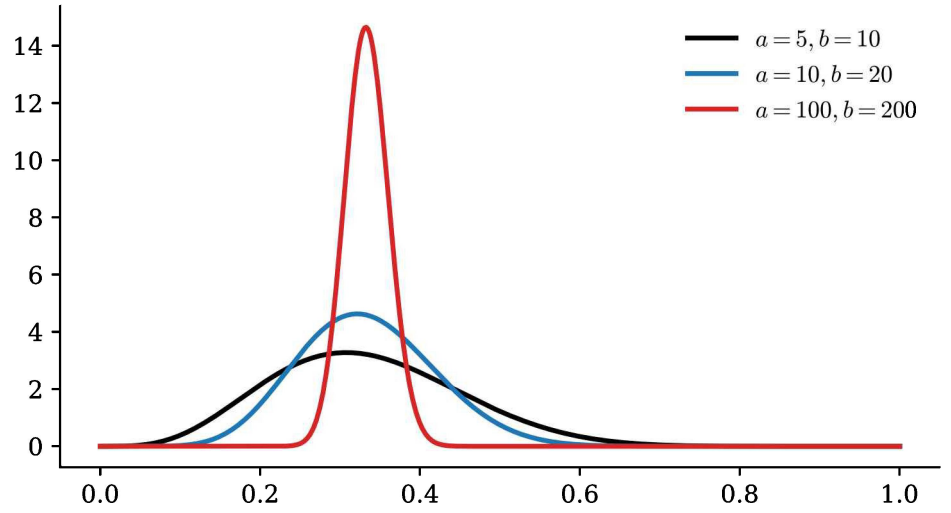


Figure 17.1. *Beta distributions* $\text{Beta}(a, b)$ *for various choices of* a *and* b*. All of these distributions have expected value* $\frac{1}{3}$*, but the variance decreases as* $a + b$ *increases.*

three distributions have expected value $\frac{1}{3}$ and they become tighter around that value as we gain more information.

17.3.2 Thompson Sampling

The Thompson sampling algorithms is to take a sample from the current distribution for each arm and then pull the arm with the largest sample. Based on the outcome of the pull, the parameters are updated and the process is repeated. Thompson sampling is an old idea (dating to 1933) that has been rediscovered several times, but until recently it had not received much attention as a way to solve bandit problems because of a lack of good analysis about its performance. Empirical evidence suggests that it generally outperforms other solution methods. In some restricted cases its performance has also been analyzed and shown to be equal to or better than other known competitors. For an implementation of Thompson sampling, see Algorithm 17.1.

Example 17.3.1. In the case of a two-armed Bernoulli bandit problem in state $((a_1, b_1), (a_2, b_2))$, with a beta distribution $\text{Beta}(a_i, b_i)$ for the parameter θ_i, simply draw a sample θ_1 from $\text{Beta}(a_1, b_1)$ and a sample θ_2 from $\text{Beta}(a_2, b_2)$, and then choose the arm whose sample value is greater. After making the pull, update the distributions and repeat the process.

If we started with uniform priors $\text{Beta}(1, 1)$ for each arm and have drawn 98 times from the first arm with 39 successes and 7 times from the second arm with 2 successes, then the two current distributions are $\text{Beta}(40, 60)$ and $\text{Beta}(3, 6)$, respectively; see Figure 17.2.

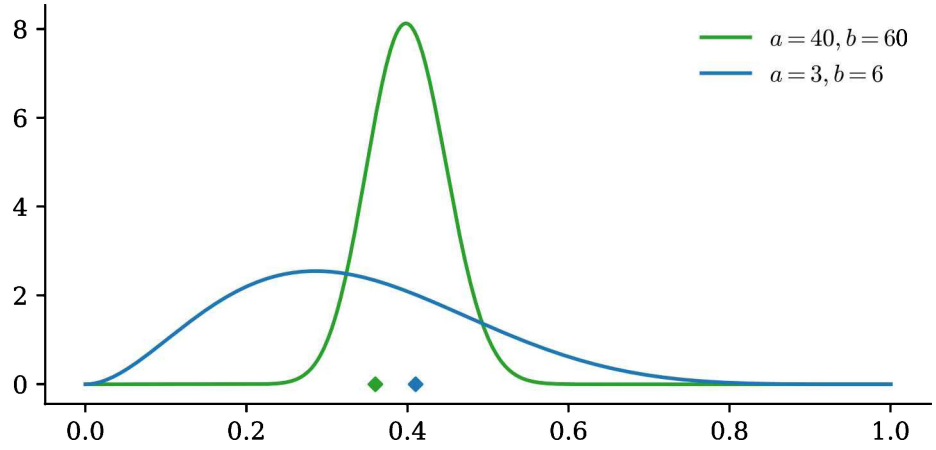


Figure 17.2. *Samples drawn from the two distributions* Beta$(40, 60)$ *and* Beta$(3, 6)$. *Although the mode of the blue distribution is much less than that of the green, the high variance of the blue and low variance of the green mean that there is still a good chance of drawing a blue sample (blue diamond) that is greater than the green sample (green diamond).*

Example 17.3.2. Consider a situation where a projector can take replacement bulbs from various manufactures, and the manufacturers all charge the same price for a bulb. The bulbs are expensive, so you want to buy your bulbs from the manufacturer whose bulbs last the longest. This can be modeled as a variant of the bandit problem, where the greedy approach would be to buy from the manufacturer whose bulbs have the longest expected life, based on what you have observed in the past, but it could be that some other manufacturer actually has bulbs that last longer, on average, and you just haven't gathered enough data to observe that.

We assume that bulb life has an exponential distribution Gamma$(1, \lambda)$, which has an expected value of $1/\lambda$. The conjugate prior for the exponential distribution is $\lambda \sim$ Gamma(a, b), with expected value $\frac{a}{b}$ (see (5.26)). Thus, for the ith manufacturer, we associate a distribution of the form Gamma(a_i, b_i) for some choice of $a_i > 0$ and $b_i > 0$.

If all the bulb manufacturers claim their bulbs have the same lifetime ℓ, then before sampling any bulbs it is natural to take a_i and b_i such that $\frac{a_i}{b_i} = \frac{1}{\ell}$ for the prior. The more believable the manufacturers' claims of lifetime are, the larger a_i and b_i should be (corresponding to a smaller variance $\frac{a_i}{b_i^2}$). Thus, one choice of prior, if we are not very certain of the manufacturers' claims, would be Gamma$(1, \ell)$, whereas if we believe the claims more strongly, we could take Gamma$(10, 10\ell)$.

Each time a bulb from manufacturer i burns out, having lasted for time t, we update the prior Gamma(a_i, b_i) to the posterior Gamma$(a_i + 1, b_i + t)$

(see Section 6.5.4). To use Thompson sampling in order to choose which manufacturer to buy from next, simply draw t_i from each of the updated distributions and select the manufacturer i with the largest value of t_i. Once the next bulb burns out, if it was made by manufacturer j and lasted for time t, then update the corresponding a_j and b_j to the posterior $a_j + 1$ and $b_j + t$ and repeat the process.

```
1  def thompson(theta,N):
2      """
3      Thompson sample to choose the arm, then simulate a pull,
4      then update. Repeat N times.
5
6      theta : array of true probabilities for the arms
7      N     : total number of pulls to make
8
9      Return percentage of successes up to each time step
10     """
11
12     # Initialize
13     n = len(theta)    # Number of arms
14     a = np.ones(n)    # Initial `a` hyperparameters
15     b = np.ones(n)    # Initial `b` hyperparameters
16     X = np.random.random(N) # Draw from [0,1] to simulate pulls
17     traj = np.zeros(N)      # Initial trajectory
18
19     for k in range(N):
20         draw = beta.rvs(a,b)  # Thompson sample for all arms
21         index = np.argmax(draw)     # Identify arm to pull
22         if X[k] <= theta[index]:    # If pull is a success
23             a[index] +=1   # Update posterior with success
24             traj[k] = traj[k-1] + 1 # Update trajectory
25         else:
26             b[index] +=1   # Update posterior with failure
27             traj[k] = traj[k-1]     # Update trajectory
28     return traj/np.arange(1,N+1)    # Percentage successes
```

Algorithm 17.1. *Algorithm for using Thompson sampling to choose arms in* `N` *simulated pulls of a multiarmed Bernoulli bandit with true parameters* `theta` *and returning the percentage of successes after each pull. Here a pull on arm i (a draw from* Bernoulli(θ_i)*) at stage k is accomplished by taking $X[k]$ from* Uniform$(0, 1)$ *(Line* 16*) and then returning success if $X[k] \leq \theta_i$ (Line* 22*). Running this code* 200 *times with* `N=5000` *and* `theta=[0.30, 0.45, 0.23, 0.4]` *gives the results plotted in Figure 17.3. After a brief dip, while exploring, the average payout approaches the largest Bernoulli probability* 0.45*, demonstrating a successful balance of exploration and exploitation.*

Vista 17.3.3. Thompson sampling can also be used to minimize driving time for a commuter with multiple possible routes to work. We can formulate this problem as a stochastic graph optimization problem, where each route from home to work is composed of various edges in the graph, and the weights (average driving times) for each edge are unknown. Thompson sampling applied to the unknown parameters of the weight distributions can be used to find an advantageous balance between exploring new edges and exploiting what is already known. See [RRKO17] for more on this application.

Sometimes it may be more convenient to take several samples from the distributions before updating. Empirical evidence suggests that Thompson sampling is very robust even with this delayed updating.

One could continue the experiment indefinitely, but it is often useful to have criteria for stopping. First, we may want to run the process for a minimum period of time, regardless of results, just to ensure that we have enough data to make a decision and that the results are not overly influenced by a small number of random draws. A second stopping criterion could be that there be a certain probability (typically 95%) that one of the variations is the best.

It may seem that these two criteria should be enough, but in some cases the experiment could last a very long time using just these criteria. For example, consider the case that two variations have nearly the same value of θ. In this case it is very difficult to determine which is best, but it is also not very important to decide between them, since the results are so similar. Thus it makes sense to use additional criteria that quantify the potential value remaining.

17.3.3 Application: Web Page Experiments

Bandit problems provide a way to balance exploration and exploitation when experimenting with variations of a web page. Suppose a business wants to test new versions of a web page. The goal of the page might be to get the user to click a certain link or to make a purchase. When the user does this, we call it a *conversion*. The *conversion rate*, or *CvR*, is the proportion of web page visits that results in a conversion. The site designer wants to determine which variation of the web page has the best CvR.

We can formulate this situation as a bandit problem by considering each page as a different arm. Each page has some unknown probability (the CvR) that a user will perform the desired action. The company then wants to experiment with giving different users different versions of the page in order to determine which variation is most successful.

A more classical approach is to explore first and then exploit by using a method called *A/B testing*. This involves splitting traffic between each variation for a long enough period of time to give sufficient data to determine the best arm with some level of confidence.

The bandit approach has significant advantages over a classical A/B test. First, the bandit method generally converges more quickly. Second, an A/B requires that

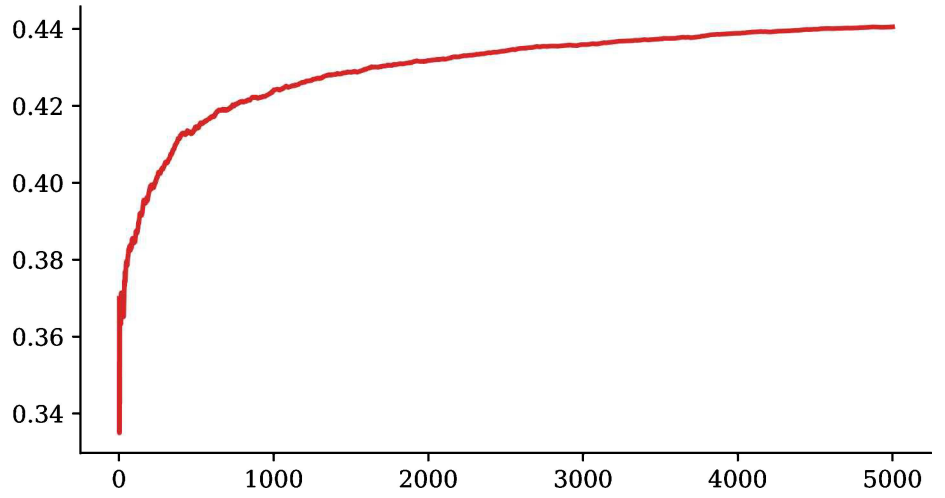


Figure 17.3. *Plot of average payout over* 5000 *steps for Thompson sampling for a Bernoulli bandit with true probabilities* $\theta_0, \ldots, \theta_3 = (0.30, 0.45, 0.23, 0.4)$*, starting with a uniform (*Beta$(1,1)$*) distribution for each* θ_i*, as in Algorithm 17.1. This plot shows the result of repeating the experiment* 200 *times and averaging the results at each time step. After a brief dip while exploring, the average payout approaches the largest Bernoulli probability of* 0.45*.*

one generate enough data to have substantial confidence (say 95%) that there is a real difference between the options.

Another advantage of the bandit approach is that it balances exploitation with exploration, whereas the A/B test primarily focuses on collecting data (exploration without exploitation) for the initial, often very long period, before it uses that data (exploitation without further exploration). As the bandit process proceeds and we gain more information, we allocate more visits to the variation that we believe has a better CvR, hence gaining more conversions.

Finally, an important benefit of the Thompson sampling approach to the bandit problem is that we can incorporate previous knowledge or beliefs into the model. Traditionally decisions about these sorts of problems might have been made by a marketer or a web page editor who would rely on intuition or experience to guess which page would have a better CvR. In Thompson sampling these opinions can still be incorporated as a prior—by choosing larger values of a and b for the arm the marketer or editor believes to be best. If the marketers or editors are correct, they will be rewarded with better initial CvR than starting with a naïve Beta$(1,1)$, but, of course, over time the data swamp the prior, so, regardless of initial beliefs, the system should converge to the optimal solution.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the

first and second lines are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

17.1. Formulate the following problem as a Markov decision process. What are the states, actions, transition probabilities, and reward function? Assume the following:

- You have a monopoly on widgets. In month i there is demand for d_i widgets. Demand changes each month as follows: If in any month i the demand falls to zero ($d_i = 0$), then in all subsequent months, there will be no demand ($d_j = 0$ for all $j \geq i$). Otherwise, the demand d_i either grows by one with probability p_+, remains the same with probability p_0, or shrinks by one with probability p_-. Initially, the demand is $d_0 = 100$.
- You have a factory with m_i widget-making machines in month i. You can make each machine produce either zero or one widget each month. In addition you can buy up to K new machines each month or remove up to K old machines, and this will determine the number m_{i+1} of machines available for the subsequent month. Initially, you have no machines ($m_0 = 0$).
- Each new machine costs b dollars to buy (one time cost), k dollars to maintain per month, and r dollars to remove (one time cost).
- Each month, you choose to produce w_i widgets (which can be no more than your production capacity m_i). Each widget costs c dollars to make and sells for f dollars. Of course, you can't sell more than the demand d_i or more than you actually produce w_i. For simplicity, assume $f > c$ and that you cannot store unsold widgets for the next month.
- Your profit u_i each month is the revenue from sales minus total operating costs. Your goal is to maximize the present value of the profit $\sum_{i=0}^{\infty} \beta^i u_i$ for a given discount factor $\beta \in (0, 1]$.

17.2. Write the Bellman optimality equation for the previous problem.

17.3. Formulate the following problem as a Markov decision process. What are the states, actions, transition probabilities, and reward function? Assume the following:
Let C be a set of cities which are joined by a set of roads R, where $R_{ij} \in R$ means you can take a road from city i to city j. However, the traffic can prevent travel. Specifically, if you're in city i and try to go to city j, you'll

succeed with probability p_{ij} and fail with probability $1 - p_{ij}$, in which case you stay in city i, and the probability of success does not depend on previous trials. If you fail, it takes $u_{ij} > 0$ time, and if you succeed, it takes $v_{ij} > u_{ij}$ time. Your goal is to go from city $c \in C$ to city $c' \in C$ in the minimum expected time (your utility is total commute time).

17.4. Write the Bellman optimality equation for the previous problem.

17.5. Explain (give an informal proof of) why the general Bellman equation (17.1) holds for every MDP.

17.6. Prove the envelope condition (17.3) for the stochastic growth problem. Hint: Consider using the Bellman equation.

17.7. Consider the problem of a pair ($n = 2$) of Bernoulli bandits having winning payouts of J_1 and J_2, respectively.

(i) If you are allowed only one pull, what is the optimal choice, starting in state $((a_1, b_1), (a_2, b_2))$?

(ii) What is the optimal expected reward $R(a_1, b_1, a_2, b_2, 1)$ in this case?

17.8. Use the results of the previous problem and (17.6) to give a formula (with no unsolved terms of the form $R(a, b, a', b', N)$) for the optimal expected reward $R(a_1, b_1, a_2, b_2, 2)$ when you are allowed two pulls.

17.9. Code up a simulation of a single pull on an n-armed bandit system, as follows. Write a method called `pull` that takes the following input:

- An array of n probabilities $\theta_1, \ldots, \theta_n$ corresponding to the true probability of success for each arm,
- An array of payouts $(J_1, \ldots, J_n)$ for each arm.
- An action i indicating that arm number i should be pulled.

This should return the amount won and a pair $(\Delta a_i, \Delta b_i) \in \{(1, 0), (0, 1)\}$, with $(1, 0)$ corresponding to a success of the ith arm (the one that was pulled), and $(0, 1)$ corresponding to a failure. Hint: To draw from a Bernoulli distribution with probability θ, you can draw u from a uniform distribution on $[0, 1]$ and then return success if $u \leq \theta$. Alternatively, Bernoulli is a special case of binomial, and many computational systems already have methods for drawing from a binomial distribution.

17.10. Code up a solution to a general Bernoulli bandit process using the Gittins index algorithm discussed in the text.

(i) Write a method `compute_R` that accepts as input an integer M and floats r, β and returns an $(M + 1) \times (M + 1)$ array `R_values` with `R_values[a,b]` $= R(a, b, r)$ for all $a + b \leq M$ (the remaining entries in the array can be set to zero).

(a) Initialize using the assumption of (17.10) for $a + b = M$.

(b) Use the recursion (17.7) to find the other values for $1 \leq a + b < M$.

(c) Apply your code to the situation of Example 17.2.4 and compare the results to that example.

(ii) Write a method `gittins` to approximate the Gittins index of each of the different arms. Your code should accept as input an array of floats

$(J_1, \ldots, J_n)$, an array of states $((a_1, b_1), \ldots, (a_n, b_n))$, an integer M, and an integer K. For each r in the grid of K points compute $R(a, b, r)$ and then for each a, b find the r which most nearly satisfies (17.8). Your method should return an array of floats $\nu((a_1, b_1), J_1), \ldots, \nu((a_n, b_n), J_n)$ corresponding to the Gittins index of each arm.

17.11. Combine your code from the previous two problems to make a simulation for a Bernoulli bandit process/solution where one arm is pulled, the results are recorded, the next optimal arm (with the largest Gittins index) is then chosen and pulled, and the results are recorded, and so on for T iterations. Your code should accept

- an array of n probabilities $\theta_1, \ldots, \theta_n$ corresponding to the true probability of success for each arm;
- an array of payouts $(J_1, \ldots, J_n)$ corresponding to the payouts for each arm;
- an integer K number of grid points in $[0, J_i]$ to take each r from;
- an integer T number of iterations to repeat the process; and
- an integer $M > T$ at which to initialize the method `compute_R`.

With these data, your code should do the following:

(i) Set the initial state as $(1, 1)$ for each arm.

(ii) For each iteration,

(a) use `gittins` to choose an action j;

(b) simulate the next pull on arm j using `pull` and record the resulting success or failure;

(c) update the state vector.

Your code should return the estimated probabilities $\hat{\theta}_1, \ldots, \hat{\theta}_n$ and the total payout gained by the actions taken.
Hint: In order to get this simulation to run in a reasonable amount of time, you may want to memoize results from `compute_R`.

17.12. Run both the Thompson sampling and the Gittins index simulations for $T = 100$ iterations for a three-armed Bernoulli bandit process with equal payouts (all 1) and with true probabilities $(\theta_1, \theta_2, \theta_3) = (0.2, 0.5, 0.7)$. Compare the run times and the outcomes of each. Repeat the comparisons 20 times. Also compare results for other values of T.

17.13. Write a program that performs A/B testing. Have each arm tested m times to estimate each $(\theta_1, \ldots, \theta_n)$ with the MLE estimator. Then choose the largest θ_i and use the remaining $N - nm$ pulls (where N is the total number of pulls) to try to maximize the average payoff. Compare the average payout with Thompson sampling in Algorithm 17.1.

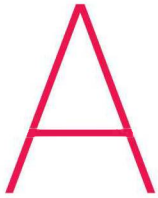
17.14. As an alternative to A/B testing, try randomly choosing arms (with replacement) m times and give MLE estimates for each $(\theta_1, \ldots, \theta_n)$. Then choose the largest θ_i and use the remaining $N - m$ pulls to try to maximize the average payoff. Compare the average payoff with A/B testing and Thompson sampling in Algorithm 17.1.

17.15. Instead of evaluating algorithms based on average payout over time, reapply A/B testing and Thompson sampling but assess the quality with the discounted utility function $\sum_{i=1}^{N} \beta^i u_i$, where u_i is the payoff. By doing multiple runs, compute the expected utility and use that as the basis of comparison for deciding whether Thompson sampling is better than A/B testing.

17.16. Generalize Algorithm 17.1 by coding up a simulation of a Bernoulli bandit process/solution where Thompson sampling can also accommodate an array of payouts $(J_1, \ldots, J_n)$ for each arm.

Notes

The example on unemployment insurance is from [Bar12]. For more about the Gittins index see [GGW11]. Google's use of Thompson sampling is described in [Sco13]. For a detailed treatment of Thompson sampling, see [RRKO17, Liu18].



The Greek Alphabet

I fear the Greeks even when they bring gifts.
—Virgil

Capital	Lower	Variant	Name
A	α		Alpha
B	β		Beta
Γ	γ		Gamma
Δ	δ		Delta
E	ϵ	ε	Epsilon
Z	ζ		Zeta
H	η		Eta
Θ	θ	ϑ	Theta
I	ι		Iota
K	κ	$\varkappa$	Kappa
Λ	λ		Lambda
M	μ		Mu
N	ν		Nu
Ξ	ξ		Xi
O	o		Omicron
Π	π	ϖ	Pi
P	ρ	ϱ	Rho
Σ	σ	ς	Sigma
T	τ		Tau
Υ	υ		Upsilon
Φ	ϕ	φ	Phi
X	χ		Chi
Ψ	ψ		Psi
Ω	ω		Omega

Bibliography

[Abb15] Stephen Abbott. *Understanding analysis.* Undergraduate Texts in Mathematics. Springer, New York, second edition, 2015. [xv]

[AC03] Jerome Adda and Russell W. Cooper. *Dynamic economics: Quantitative methods and applications*, Volume 1. MIT Press, Cambridge, MA, 2003. [742]

[Alb16] Laura Albert. Chocolate chip cookies are Poisson distributed. `https://punkrockor.com/2016/12/07/chocolate-chip-cookies-are-poisson-distributed/`, 2016. Last accessed 21 Sept. 2017. [243]

[AMH10] Awad H. Al-Mohy and Nicholas J. Higham. The complex step approximation to the Fréchet derivative of a matrix function. *Numer. Algorithms*, 53(1):113–148, 2010. [516]

[Art64] Emil Artin. *The gamma function.* Translated by Michael Butler. Athena Series: Selected Topics in Mathematics. Holt, Rinehart and Winston, New York, 1964. [106]

[Art91] Michael Artin. *Algebra.* Prentice-Hall, Englewood Cliffs, NJ, 1991. [3]

[AVL62] G. M. Adelson-Velsky and E. M. Landis. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*, 146:263–266, 1962. [139]

[Bar12] Jorge A. Barro. Lecture notes on dynamic optimization. `https://pdfs.semanticscholar.org/5c7b/a183b7f777863bea3fe8f02c714f4804fe11.pdf`, 2012. Last accessed 5 Aug. 2019. [764]

[BB08] Jonathan Borwein and David Bailey. *Mathematics by experiment: Plausible reasoning in the 21st Century.* A K Peters, Ltd., Wellesley, MA, second edition, 2008. [106]

[BB14] A. R. Benson and G. Ballard. A framework for practical parallel fast matrix multiplication. *ArXiv e-prints*, Sept. 2014. [83]

[BBC+99] Évelyne Barbin, Jacques Borowczyk, Jean-Luc Chabert, Michel Guillemot, and Anne Michel-Pajus. *A history of algorithms: From the pebble to the microchip.* Springer-Verlag, Berlin, 1999. Translated from the 1994 French original by Chris Weeks. [83]

[BCR91] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms. I. *Comm. Pure Appl. Math.*, 44(2):141–183, 1991. [403]

[Bec14] Amir Beck. *Introduction to nonlinear optimization: Theory, algorithms, and applications with MATLAB.* MOS-SIAM Series on Optimization. SIAM, Philadelphia, Mathematical Optimization Society, Philadelphia, 2014. [716]

[Ber79] D. P. Bertsekas. Convexification procedures and decomposition methods for nonconvex optimization problems. *J. Optim. Theory Appl.*, 29(2):169–197, 1979. [716]

[Ber09] Dimitri P. Bertsekas. *Convex optimization theory.* Athena Scientific, Nashua, NH, 2009. [716]

[Ber16] Dimitri P. Bertsekas. *Nonlinear programming.* Athena Scientific Optimization and Computation Series. Athena Scientific, Belmont, MA, third edition, 2016. [666]

[BH15] Joseph K. Blitzstein and Jessica Hwang. *Introduction to probability.* Texts in Statistical Science Series. CRC Press, Boca Raton, FL, 2015. [243, 280, 319]

[BHS+78] J. L. Bentley, D. Haken, and J. B. Saxe. *A general method for solving divide-and-conquer recurrences.* Carnegie-Mellon University, Computer Science Department, Defense Technical Information Center, 1978. [83]

[Bie15] Michel Bierlaire. *Optimization: Principles and algorithms.* EPFL Press, Lausanne; distributed by CRC Press, Boca Raton, FL, 2015. [619, 666, 716]

[BL06] Jonathan M. Borwein and Adrian S. Lewis. *Convex analysis and nonlinear optimization: Theory and examples.* Volume 3 of *CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC*, Springer, New York, second edition, 2006. [716]

[Bli13] Joseph Blitzstein. Conditioning is the soul of statistics. YouTube Talk: Harvard Thinks Big 4, `https://www.youtube.com/watch?v=dzFf3r1yph8`, 2013. Last accessed 16 Aug. 2017. [243]

[BN09] Albert Boggess and Francis J. Narcowich. *A first course in wavelets with Fourier analysis.* John Wiley & Sons, Hoboken, NJ, second edition, 2009. [403]

[Bog14] I. Bogaert. Iteration-free computation of Gauss–Legendre quadrature nodes and weights. *SIAM J. Sci. Comput.*, 36(3):A1008–A1026, 2014. [449]

[BT04] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Rev.*, 46(3):501–517, 2004. [417, 459]

[Bus03] P. Bussotti. On the genesis of the Lagrange multipliers. *J. Optim. Theory Appl.*, 117(3):453–459, 2003. [666]

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, UK, 2004. [536, 619, 666, 716]

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001. [6, 83, 139, 184]

[Con90] John B. Conway. *A course in functional analysis*. Volume 96 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1990. [619]

[Con16] Keith Conrad. Stirling's formula. `http://www.math.uconn.edu/~kconrad/blurbs/analysis/stirling.pdf`, 2016. Last accessed 23 May 2018. [106]

[Coo14] John Cook. Five tips for floating point programming. `http://www.codeproject.com/Articles/29637/Five-Tips-for-Floating-Point-Programming`, 2014. Last accessed 29 July 2019. [516]

[CV13] Erhan Cinlar and Robert J. Vanderbei. *Real and convex analysis*. Undergraduate Texts in Mathematics. Springer, New York, 2013. [716]

[CZ01] E.K.P. Chong and S.H. Żak. *An Introduction to Optimization*. Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, Hoboken, NJ, second edition, 2001. [536, 574, 666]

[Dan67] James W. Daniel. Convergence of the conjugate gradient method with computationally convenient modifications. *Numer. Math.*, 10:125–131, 1967. [574]

[Dan70] James W. Daniel. A correction concerning the convergence rate for the conjugate gradient method. *SIAM J. Numer. Anal.*, 7(2):277–280, 1970. [574]

[DD10] Kenneth R. Davidson and Allan P. Donsig. *Real analysis and applications: Theory in practice*. Undergraduate Texts in Mathematics. Springer, New York, 2010. [403]

[DDM+14] Erik D. Demaine, Martin L. Demaine, Yair N. Minsky, Joseph S. B. Mitchell, Ronald L. Rivest, and Mihai Pătraşcu. Picture-hanging puzzles. *Theory Comput. Syst.*, 54(4):531–550, 2014. [184]

[Dem96] James Demmel. Basic issues in floating point arithmetic and error analysis. `http://www.cs.berkeley.edu/~demmel/cs267/lecture21/lecture21.html`, 1996. Last accessed 11 Aug. 2015. [516]

[Dem97] James W. Demmel. *Applied numerical linear algebra*. SIAM, Philadelphia, PA, 1997. [487, 516, 574]

[dH94] D. den Hertog. *Interior point approach to linear, quadratic and convex programming: Algorithms and complexity.* Volume 277 of *Mathematics and Its Applications.* Kluwer, Dordrecht, 1994. [716]

[DKH12] Ding-Zhu Du, Ker-I Ko, and Xiaodong Hu. *Design and analysis of approximation algorithms.* Volume 62 of *Springer Optimization and Its Applications.* Springer, New York, 2012. [184]

[DL05] Erik D. Demaine and Charles E. Leiserson. Introduction to algorithms: Problem set 1. `http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/assignments/ps1.pdf`, 2005. [83]

[DS19] Yann Disser and Martin Skutella. The simplex algorithm is NP-mighty. *ACM Trans. Algorithms*, 15(1):Art. 5, 2019. [619]

[Fav35] J. Favard. Sur les polynomes de Tchebicheff. *Comptes Rendus de l'Académie des Sciences*, 200:2052–2053, 1935. [425]

[Fel71] William Feller. *An introduction to probability theory and its applications*, volume II. John Wiley & Sons, New York, second edition, 1971. [265]

[FS15] John Fearnley and Rahul Savani. The complexity of the simplex method. In *STOC'15—Proceedings of the 2015 ACM Symposium on Theory of Computing*, pages 201–208. ACM, New York, 2015. [619]

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, 34(3):596–615, 1987. [184]

[GGW11] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices.* John Wiley & Sons, Chichester, UK, 2011. Second edition; with a foreword by Peter Whittle. [764]

[Gol94] Donald Goldfarb. On the complexity of the simplex method. In *Advances in optimization and numerical analysis (Oaxaca, 1992).* Volume 275 of *Mathematics and Its Applications*, pages 25–38. Kluwer, Dordrecht, 1994. [619]

[GS92] A. Greenbaum and Z. Strakos. Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM J. Matrix Anal. Appl.*, 13(1):121–137, 1992. [574]

[GS03] Charles M. Grinstead and J. Laurie Snell. *Introduction to probability.* American Mathematical Society, Providence, RI, 2003. [280]

[GV15] Jonas Gomes and Luiz Velho. *From Fourier analysis to wavelets.* Volume 3 of *IMPA Monographs.* Springer, Cham, 2015. [403]

[GYZ] Gregory Gutin, Anders Yeo, and Alexey Zverovich. Traveling salesman should not be greedy: Domination analysis of greedy-type heuristics for the TSP. *Discrete Appl. Math.*, 117:81–86. [184]

[Hea49] Thomas Heath. *Mathematics in Aristotle.* Clarendon Press, Oxford, UK, 1949. [83]

[Hig96] Nicholas J. Higham. *Accuracy and stability of numerical algorithms.* SIAM, Philadelphia, 1996. [487, 498, 516]

[Hig18] Nicholas J. Higham. Differentiation with(out) a difference. *SIAM News*, June 2018. [516]

[HJE17] Jeffrey Humpherys, Tyler J. Jarvis, and Emily J. Evans. *Foundations of applied mathematics: Volume* 1*, mathematical analysis.* SIAM, Philadelphia, 2017. [xv]

[HLHG00] Ulrich Hoffrage, Samuel Lindsey, Ralph Hertwig, and Gerd Gigerenzer. Communicating statistical information. *Science*, 290:2261–2262, 2000. [203]

[IEE08] IEEE Task P754. *IEEE 754-2008, Standard for floating-point arithmetic.* IEEE, New York, 2008. [516]

[Jar30] Vojtěch Jarník. O jistém problému minimálním. (z dopisu panu o. borøuvkovi) (Czech) [on a certain problem of minimization]. *Práce moravské přírodovědecké společnosti*, 6(4):57–63, 1930. [184]

[Jia18] Yan-Bin Jia. Lagrange multipliers, notes. `http://web.cs.iastate.edu/~cs577/handouts/lagrange-multiplier.pdf`, 2018. Last accessed 12 July 2019. [666]

[JK15] K. Jansen and S. E. J. Kraft. A faster FPTAS for the unbounded knapsack problem. *ArXiv e-prints*, 2015. [184]

[Kle08] Achim Klenke. *Probability theory: A comprehensive course.* Universitext. Springer-Verlag, London, 2008. Translated from the 2006 German original. [265, 266]

[KM72] Victor Klee and George J. Minty. *How good is the simplex algorithm?*, In Inequalities III, pages 159–175. Academic Press, New York, 1972. [606, 619]

[KT05] Jon Kleinberg and Eva Tardos. *Algorithm design.* Addison-Wesley, Boston, MA, 2005. [83, 184]

[Kur15] Will Kurt. Six neat tricks with Monte Carlo simulations. `https://www.countbayesie.com/blog/2015/3/3/6-amazing-trick-with-monte-carlo-simulations`, 2015. Last accessed 4 Nov. 2017. [280, 319]

[Leu04] Joseph Y.-T. Leung, editor. *Handbook of scheduling: Algorithms, models, and performance analysis.* Chapman & Hall/CRC Computer and Information Science Series. Chapman & Hall/CRC, Boca Raton, FL, 2004. [180]

[LeV07] Randall J. LeVeque. *Finite difference methods for ordinary and partial differential equations: Steady-state and time-dependent problems.* SIAM, Philadelphia, 2007. [516]

[Liu18] Che-Yu Liu. Thompson sampling for bandit problems, PhD thesis, Princeton University, Princeton, NJ, 2018. [764]

[LM67] J. N. Lyness and C. B. Moler. Numerical differentiation of analytic functions. *SIAM J. Numer. Anal.*, 4:202–210, 1967. [516]

[LS18] Lars Ljungqvist and Thomas J Sargent. *Recursive macroeconomic theory.* MIT Press, Cambridge, MA, 2018. [742]

[Mac18] J. MacCormick. *What can be computed? A practical guide to the theory of computation.* Princeton University Press, Princeton, NJ, 2018. [184]

[Mat15] Oxford dictionaries: Language matters. Which letters in the alphabet are used most often? `http://www.oxforddictionaries.com/us/words/which-letters-are-used-most`, 2015. Last accessed 11 Aug. 2015. [184]

[May11] Philip Z. Maymin. Markets are efficient if and only if $P = NP$. *Algorithmic Finance*, 1(1):1–11, 2011. [184]

[MH03] J. C. Mason and D. C. Handscomb. *Chebyshev polynomials.* Chapman & Hall/CRC, Boca Raton, FL, 2003. [459]

[Nem10] Gergő Nemes. On the coefficients of the asymptotic expansion of $n!$. *J. Integer Seq.*, 13(6):Art. 10.6.6, 2010. [106]

[Nes04] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course.* Volume 87 of *Applied Optimization.* Kluwer, Boston, MA, 2004. [716]

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical optimization.* Springer Series in Operations Research. Springer-Verlag, New York, 1999. [574, 619, 666]

[OSS+16] N. Ohuchi, A. Suzuki, T. Sobue, M. Kawai, S. Yamamoto, Y.-F. Zheng, Y.-N. Shiono, H. Saito, S. Kuriyama, E. Tohno, T. Endo, A. Fukao, I. Tsuji, T. Yamaguchi, Y. Ohashi, M. Fukuda, and T. Ishida. Sensitivity and specificity of mammography and adjunctive ultrasonography to screen for breast cancer in the Japan strategic anti-cancer randomized trial (J-START): A randomised controlled trial. *Lancet*, 387(10016):341–348, 2016. [202]

[Pat17] Rich Pattis. AVL trees. `https://www.ics.uci.edu/~pattis/ICS-46/lectures/notes/avl.txt`, 2017. Last accessed 9 Aug. 2018. [139]

[Ped04] Pablo Pedregal. *Introduction to optimization.* Volume 46 of *Texts in Applied Mathematics.* Springer-Verlag, New York, 2004. [619, 666]

[Pro08] S. David Promislow. *A first course in functional analysis.* Pure and Applied Mathematics (Hoboken). Wiley-Interscience, Hoboken, NJ, 2008. [619]

[Pru17] Kirk Pruhs. Dynamic programming homework problems. `http://people.cs.pitt.edu/~kirk/cs1510/homework/dynproghw.pdf`, 2017. Last accessed 9 Aug. 2017. [184]

[PS75] Thomas Porter and Istvan Simon. Random insertion into a priority queue structure. *IEEE Trans. Software Engrg.*, SE-1(3):292–298, 1975. [139]

[PSZ09] Konstantinos Paparrizos, Nikolaos Samaras, and Dimitrios Zissopoulos. In *Linear programming: Klee–Minty examples*, pages 1891–1897. Springer, Boston, MA, 2009. [619]

[RN03] Stuart J. Russell and Peter Norvig. *Artificial intelligence: A modern approach.* Pearson Education, London, second edition, 2003. [319]

[Ros07] Sheldon M. Ross. *Introduction to probability models.* Elsevier/Academic Press, Amsterdam, ninth edition, 2007. [280]

[Ros14] Sheldon Ross. *A first course in probability.* Pearson, London, ninth edition, 2014. [280]

[RRKO17] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, and Ian Osband. A tutorial on Thompson sampling. *CoRR*, abs/1707.02038, 2017. [759, 764]

[Rud91] Walter Rudin. *Functional analysis.* International Series in Pure and Applied Mathematics. McGraw-Hill, New York, second edition, 1991. [619]

[Sco13] Steven L. Scott. Multi-armed bandit experiments. `http://analytics.blogspot.com/2013/01/multi-armed-bandit-experiments.html`, 2013. Last accessed 9 April 2019. [764]

[She94] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, edition $1\frac{1}{4}$. `https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`, 1994. Last accessed 9 Feb. 2016. [574]

[Shi84] A. N. Shiryayev. *Probability.* Volume 95 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1984. Translated from the Russian by R. P. Boas. [265]

[SLP89] N. L. Stokey, R. E. Lucas, and E. C. Prescott. *Recursive methods in economic dynamics.* Harvard University Press, Cambridge, MA, 1989. [742]

[SS07] Y. N. Srikant and Priti Shankar. *The compiler design handbook: Optimizations and machine code generation.* CRC Press, Boca Raton, FL, 2nd edition, 2007. [83]

[ST98] William Squire and George Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Rev.*, 40(1):110–112, 1998. [516]

[Str93] Gilbert Strang. Wavelet transforms versus Fourier transforms. *Bull. Amer. Math. Soc. (N.S.)*, 28(2):288–305, 1993. [403]

[Tan15] Vincent Y. F. Tan. Simplified proof of Slater's theorem for strong duality. `https://www.ece.nus.edu.sg/stfpage/vtan/ee5138/slater.pdf`, 2015. Last accessed 19 June 2019. [716]

[Tan17] James Tanton. Tanton's take on the Poisson distribution. http://www.jamestanton.com/wp-content/uploads/2012/03/Curriculum-Essay_December-2017_Poisson-Distribution.pdf, 2017. Last accessed 4 Sept. 2018. [243]

[TB97] Lloyd N. Trefethen and David Bau, III. *Numerical linear algebra.* SIAM, Philadelphia, 1997. [83, 487, 516]

[TdD14] Sid Touati and Benoit de Dinechin. *Advanced backend code optimization.* Wiley-IEEE Press, New York, 2014. [83]

[Tho07] Mikkel Thorup. Equivalence between priority queues and sorting. *J. ACM*, 54(6):Art. 28, 2007. [139]

[TMS10] Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Why are nonlinear fits to data so challenging? *Phys. Rev. Lett.*, 104(2010), Art. 060201. [574]

[Tre08] Lloyd N. Trefethen. Is Gauss quadrature better than Clenshaw–Curtis? *SIAM Rev.*, 50(1):67–87, 2008. [459]

[Tre13] Lloyd N. Trefethen. *Approximation theory and approximation practice.* SIAM, Philadelphia, 2013. [452, 459]

[Van14] Robert J. Vanderbei. *Linear programming: Foundations and extensions.* Volume 196 of International Series in Operations Research & Management Science, Springer, New York, fourth edition, 2014. [606, 619]

[Van18] Lieven Vandenberghe. Nonlinear least squares. http://www.seas.ucla.edu/~vandenbe/133A/lectures/nlls.pdf, 2018. Last accessed 4 Jan. 2019. [574]

[vdW83] B. L. van der Waerden. *Geometry and algebra in ancient civilizations.* Springer-Verlag, Berlin, 1983. [83]

[vRB12] Henry J.J. van Roessel and John C. Bowman. *Lecture notes for math 538: Asymptotic methods.* University of Alberta Edmonton, Canada. http://www.math.ualberta.ca/~bowman/m538/m538.pdf, 2012. Last accessed 23 May 2018. [106]

[Wal06] Jörg Waldvogel. Fast construction of the Fejér and Clenshaw-Curtis quadrature rules. *BIT*, 46(1):195–202, 2006. [459]

[Was04] Larry Wasserman. *All of statistics: A concise course in statistical inference.* Springer Texts in Statistics. Springer-Verlag, New York, 2004. [280, 319]

[Wik15] Wikipedia. NP-hard. https://en.wikipedia.org/w/index.php?title=NP-hard&oldid=669029008, 2015. Last accessed 10 Aug. 2015. [184]

[Wik17] Wikipedia. Abraham Wald. https://en.wikipedia.org/wiki/Abraham_Wald, 2017. Last accessed 16 Aug. 2017. [243]

[Wik18a] Wikipedia. Stirling's approximation. `https://en.wikipedia.org/?title=Stirling%27s_approximation`, 2018. Last accessed 23 May 2018. [106]

[Wik18b] Wikipedia. Wallis's integrals. `https://en.wikipedia.org/wiki/Wallis%27_integrals`, 2018. Last accessed 31 May 2018. [106]

[WS11] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, Cambridge, UK, 2011. [184]

[WY12] Z. Wei and K. Yi. Equivalence between priority queues and sorting in external memory. *ArXiv e-prints*, 2012. [139]

Index

y Humpherys is a research professor at the rsity of Utah School of Medicine, the former Chair of the SIAM Activity Group on Applied ematics Education, and a two-term member of AM Education Committee. He is the recipient ational Science Foundation CAREER award. search spans a wide range of topics in applied omputational mathematics, from nonlinear l differential equations to network sciences to ne learning.

J. Jarvis is a professor of mathematics at am Young University whose research has rily been in geometric problems arising from cs. He is the recipient of a National Science dation CAREER award and the MMA's Deborah ranklin Tepper Haimo Award for Distinguished rsity Teaching of Mathematics.

Design by Sarah Kay Miller

of Summation — 1.7 Products ar
Proof of the Master Theorem —
— 2.2 The Beta Function and
3.1 Theory of Graphs — 3.2 Trees
Search Trees — 3.4 Priority Queue
Dynamic Programming — 4.2 Gr
Hard Problems — 5 Probability
Paradoxes, and Pitfalls — 5.4 D
5.6 Continuous Random Variable
Estimation — 6.2 The Law of La
— 6.5 Bayesian Statistics — 7 Ra
Hashing — 7.4 Simulated Annealin
Series — 8.4 Convergence of Fou
— 8.8 Haar Wavelets — 8.9 Discret
9 Polynomial Approximation and
Approximation — 9.4 Interpolatio
— 9.7 Clenshaw–Curtis and Gau
10.3 Implicit Function Theorem a
Floating-Point Arithmetic — 11.2 A
— 12 Unconstrained Optimizati
12.4 Newton and Quasi-Newton
Convergence of the Conjugate-G
and Separation — 13.3 Fundamen
— 14 Nonlinear Constrained Optim
Condition — 14.3 Lagrange's Sec
Second-Order KKT — 14.6 Remo
Optin l Convex Fun
Dualit g Duality — 15
16 Dy ation — 16.1

ISBN 978-1-611976-05-2
90000
9781611976052