

The **Basics** of **Practical** **Optimization**

The **Basics** of **Practical** **Optimization**

Adam B. Levy

Bowdoin College
Brunswick, Maine

siam.

Society for Industrial and Applied Mathematics
Philadelphia

Copyright © 2009 by the Society for Industrial and Applied Mathematics (SIAM)

10 9 8 7 6 5 4 3 2 1

All rights reserved. Printed in the United States of America. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, write to the Society for Industrial and Applied Mathematics, 3600 Market Street, 6th Floor, Philadelphia, PA, 19104-2688 USA.

Trademarked names may be used in this book without the inclusion of a trademark symbol. These names are used in an editorial context only; no infringement of trademark is intended.

Mathematica is a registered trademark of Wolfram Research, Inc.

MATLAB is a registered trademark of The MathWorks, Inc. For MATLAB product information, please contact The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098 USA, 508-647-7000, Fax: 508-647-7001, info@mathworks.com, www.mathworks.com.

Library of Congress Cataloging-in-Publication Data

Levy, Adam B.

The basics of practical optimization / Adam B. Levy.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-898716-79-5

1. Mathematical optimization. I. Title.

QA402.5.L514 2009

519.6—dc22

2009008515

To my wife, Sarah,
and kids, Jonah and Emma.
And to Basha,
who would have gotten a kick out of this.



Contents

List of Figures	xi
List of Tables	xiii
Preface	xv
1 Modeling	1
1.1 An Example with Yes/No Choices	1
1.1.1 Choosing Variables	2
1.1.2 Modeling Constraints	2
1.1.3 The Objective Function and the Final Model	3
1.1.4 Solving the Model	3
1.2 Four Rules of Thumb for Modeling	4
1.3 A Blending Example	5
1.3.1 Choosing Variables	6
1.3.2 Modeling Constraints	6
1.3.3 The Objective Function and the Final Model	7
1.3.4 Solving the Model	7
1.4 A Scheduling Example	8
1.4.1 Choosing Variables	8
1.4.2 Modeling Constraints	9
1.4.3 The Objective Function and the Final Model	9
1.4.4 Solving the Model	10
1.5 An Assignment Example	10
1.5.1 Choosing Variables	10
1.5.2 Modeling Constraints	11
1.5.3 The Objective Function and the Final Model	11
1.5.4 Solving the Model	12
1.6 A Partitioning Example	12
1.6.1 Choosing Variables	12
1.6.2 Modeling Constraints	13
1.6.3 The Objective Function and the Final Model	14
1.6.4 Solving the Model	15
1.7 Modeling Problems	16

2	Impractical Optimization	25
2.1	Algebra Limitations	26
2.2	Size Limitations	29
2.2.1	Integral Optimization	29
2.2.2	Discretization	30
2.3	Conclusions from Chapter 2	33
3	Basic Practical Optimization	35
3.1	Replacement Functions	36
3.1.1	When to Stop	42
3.1.2	Why It's Called Newton's Optimization Method	43
3.2	From Stationary Points to Optimizers	44
3.2.1	But I Learned a Different Second-Derivative Test in Multivariable Calculus	47
3.2.2	Details for Functions of More Than Two Variables	48
3.2.3	More on Stopping	49
3.3	Conclusions from Chapter 3	53
4	Some Practical Modifications	55
4.1	Finding Optimizers	55
4.2	Improving Success	59
4.3	Reducing Expense	61
4.4	Conclusions from Chapter 4	61
5	How Methods Are Ranked	63
5.1	Ranking Newton's Optimization Method	66
5.2	Ranking the Quasi-Newton Methods	67
5.3	Conclusions from Chapter 5	71
6	Constraints	73
6.1	Penalty Methods	75
6.1.1	A Simple Differentiable Penalty Function	75
6.1.2	Improved Penalty Functions	76
6.1.3	Barrier Functions	80
6.1.4	Impassable Barriers?	82
6.2	The Reduction Method	84
6.2.1	Converting Inequalities to Equations	84
6.2.2	Reducing Variables	86
6.3	Conclusions from Chapter 6	88
7	More Practical Modifications	89
7.1	Reduction Modifications	89
7.1.1	The Working-Set Method: Hitting the Boundary	89
7.1.2	Lagrange Multipliers	93
7.1.3	Multipliers for Inequalities	95
7.1.4	Multipliers for Each Constraint	96
7.1.5	Putting This All Together	100

7.2	Special Case: Linear Programming	101
7.2.1	Dual Linear Programs	102
7.2.2	Dual Variables as Multipliers	105
7.2.3	Stopping for Linear Programs	106
7.3	Sequential Quadratic Programming	107
7.3.1	The Lagrangian Function	107
7.3.2	Prelude to the SQP Method	108
7.3.3	The SQP Method	111
7.4	Conclusions from Chapter 7	113
8	Integer Variables	115
8.1	Simple Rounding	116
8.1.1	Trouble with Simple Rounding	117
8.2	The Branching Method	118
8.2.1	Practical Modifications to Branching	121
8.3	The Cutting Method	122
8.4	Automatic Integers: Network Problems	124
8.5	Conclusions from Chapter 8	127
9	Other Methods	129
9.1	Dynamic Programming	129
9.1.1	Multistage Decision Processes	129
9.1.2	The Dynamic Programming Method	130
9.1.3	Comparison and Current Work	132
9.2	What Else?	136
9.2.1	Direct Search Methods	136
9.2.2	Stochastic Search Methods	136
9.2.3	Linear Methods	137
9.2.4	Primal-Dual Methods	137
9.2.5	Nonlinear Methods	137
9.3	A Research Project	138
9.3.1	Annotated Bibliography	138
9.3.2	Problem Setting	138
9.3.3	Method Description	138
9.3.4	Comparison and Current Work	138
	Appendix of Asides	139
	Bibliography	145
	Index	147

List of Figures

1.1	Schematic of the 15 towns in Androscoggin County	13
2.1	Graph of the monthly payment function with fixed interest rate	26
2.2	Two slices of the graph of the interest rate (2.2)	27
2.3	Graph of $f(x) = \cos(x) - x$	28
2.4	Graphs of a few surface-generating $x(t)$	30
2.5	Potential solution to the discretization problem	31
2.6	Solution to Newton's aerodynamics problem	32
3.1	An objective function (in bold) with two consecutive Taylor quadratics and three consecutive guesses	39
3.2	A continuous function $c(x)$	45
3.3	Not reverse continuous everywhere	50
3.4	Graphical implication of a scaling factor	51
4.1	The negative gradient determines uphill and downhill directions	57
4.2	Contour plots of $x^2 + y^2$ and $x^2 + \frac{y^2}{2}$	59
5.1	Zigzagging steepest descent method	68
5.2	Graphs of x^2 and x^4	69
5.3	No zigzagging in Newton's optimization method	70
6.1	Min-type penalty function $p_1(x)$ for $[-1, 2]$	77
6.2	Smoothed penalty function $p_2(x)$ for $[-1, 2]$	78
6.3	Derivative of the penalty function $p_2(x)$ for $[-1, 2]$	78
6.4	Smoothed penalty function $p_3(x)$ for $[-1, 2]$ and its derivative	79
6.5	Graph of $(x + 2)^2 + p_3(x)$	79
6.6	Inverse barrier function $p_4(x)$	81
6.7	Logarithmic barrier function $p_5(x)$	81
6.8	Inverse barrier function $p_4(x)$, wide view	82
6.9	Taylor quadratic for barrier function minimization	83
6.10	Contour plot of $(x + 2)^2 + (y + 1)^2$ with subproblem solutions	85
6.11	Contour plot of $x^2 + y^2$ and graph of curve $y^2 = x - 1$	88

7.1	Line-search from the constraint region $x - y \geq 0$	90
7.2	Working-set method	92
7.3	Constrained optimization	93
7.4	Gradient vectors for the objective function f	94
7.5	Gradient vectors for the constraint function g	94
7.6	Gradient vectors for both f and g	95
7.7	Gradient vectors at the intersection of two constraints	97
7.8	Gradient at a nonminimizer	97
7.9	Contours and gradients of f and g	99
7.10	Plot of f with constraint boundary superimposed	99
7.11	Graphical linear optimization	101
7.12	Objective contour through the minimizer	106
8.1	Integer program solution	117
8.2	Trouble with simple rounding	118
8.3	Integer solution not from simple rounding	118
8.4	Contour plot of $-x - y$ and the constraint region (8.1)	119
8.5	Branching method	120
8.6	Integer-pairs obeying constraints	122
8.7	Cut constraint region	123
8.8	Linear cut	123
8.9	An oil delivery network	124
8.10	Variables for each arc	125
8.11	A cycling network	126
A.1	A continuous function $c(x)$	141
A.2	Graphs of x^2 and x^4	143

List of Tables

1.1	Distances between college towns	2
1.2	Data for steel production	5
1.3	Data on museum renovation	8
1.4	Flight data	10
1.5	Valid districts containing Auburn	14
1.6	Current supporters in each town	15
1.7	Crop data	16
1.8	Clam data	16
1.9	Investment data	18
1.10	Gasoline ingredient data	19
1.11	Gasoline demand and required attributes	19
1.12	Bond data	20
1.13	Transaction data	20
1.14	Property data	21
1.15	Data for lobster processing	21
1.16	Data on horse feed	22
1.17	Registers needed per shift	23
1.18	Productivity on a scale of 1–10	23
1.19	Distribution center daily inputs and outputs	23
1.20	Puzzle manufacturing data	24
2.1	Interest rate packages	27
9.1	Distances for college-visit problem	130

Preface

Optimization is presented in most multivariable calculus courses as an application of the gradient, and while this treatment makes sense for a calculus course, there is much more to the theory of optimization. Moreover, optimization is actually used every day in a way that is much different from what one is led to believe in the typical calculus course. Our world and its societies have for many centuries generated interesting and important optimization problems, and the theory of optimization has grown and developed in response to the challenges presented by these problems. In fact, optimization theory continues to be developed today in response to practical concerns encountered in applications, which makes optimization an ideal topic of study in modern applied mathematics. Through the study of optimization theory, the power and beauty of mathematics can be observed in close connection to interesting and relevant problems of our world.

The goals of this text are to help readers understand how optimization is actually done in practice and, at the same time, to expand their appreciation of the richness of the theory behind the practice. The only prerequisite knowledge for getting the most out of this text is a basic understanding of multivariable calculus. Matrices are one important additional tool for some of this material, but whatever is needed is developed here in a series of “Asides” scattered throughout the text in places where they are relevant. Readers who wish to learn all of these tools at once (or who just need to easily find one or more Aside) can refer to the Appendix, where all the Asides are collected in one place.

One important aspect of practical optimization is the mathematical modeling of real situations in the general format used for optimization. This is an art which is best learned through as much practice as possible, particularly since different real situations often present unique modeling issues. Thus, this book begins in Chapter 1 with the presentation of step-by-step solutions for five prototypical examples which fit the general optimization model. After one example is presented in the first section, there is a section giving four general rules of thumb for modeling. The remaining sections of Chapter 1 present four additional modeling examples with detailed solutions. Readers who wish to focus exclusively on modeling at the start can work carefully through all of Chapter 1 and solve the Modeling Problems presented at its end. When I teach from this text, I move quickly to Chapter 2 and assign several of the Modeling Problems in Chapter 1 to be completed during each week of the course.

Solving real-world problems via optimization models generally involves two steps: constructing an appropriate optimization model of the real-world problem, and then solving the optimization model in order to solve the real-world problem. Model construction can generally be done by hand, but, as will be shown in Chapter 2, model-solving typically requires a computer. The current wide availability of generally useful optimization

software means that one can produce candidate solutions to optimization problems after a little training in how to feed the models into a computer. We take advantage of this here by encouraging readers to use software to “solve” the models they construct throughout the text. However, it can be difficult to interpret the computer’s results without an understanding of how these results are produced. In some cases, the computer can even produce false “solutions” which an uninformed practitioner would have no reason to doubt. By understanding the material presented here on optimization methods, the reader should at least be prepared to make better informed—and therefore more intelligent—use of results produced by commercial optimization software. Moreover, since most of the optimization methods presented in this book were developed relatively recently, there is room not only to improve them but also to develop new optimization methods. Readers who are intrigued enough by this important and developing area should consider further study in it so that they might actually contribute to its development.

As a general principle here, practical issues are dealt with first as a means of motivating the theory. In particular, this means that optimization methods take a front seat and that various general rules from the theory are presented as they are actually used. There are many different optimization methods tailored to different circumstances; however, the focus in this text is on a few core procedures which are essentially prototypical of derivative-based optimization. Chapter 3 begins with the most fundamental of these methods for optimizing functions, and Chapters 4 and 5 cover some of the adjustments that are typically made to the fundamental method in order to accommodate various practical concerns. Chapter 6 presents two fundamental approaches to optimizing functions under constraints, with practical adjustments covered in Chapter 7. An important special case when the variables are integers is covered in Chapter 8.

Lurking in Chapter 7 is an introduction to linear programming, which traditionally has been treated in a much more central way. The special historical status of linear programming is reflected in the label of “nonlinear programming” given to the rest of optimization. The integrated treatment of linear programming in this book reflects the current view of the subject, as is clear from the following quote from Margaret H. Wright [9]:

Today, in complete contrast to the era before 1984, researchers view linear and nonlinear programming from a unified perspective; the magnitude of this change can be seen simply by noting that no one would seriously argue today that linear programming is independent of nonlinear programming.

The recent merging of linear and nonlinear programming has occurred because the methods developed in each domain have been applied with great success across the traditional boundary. For instance, the *simplex method* for solving linear programs extends to nonlinear optimization problems as the *working-set method*, and the *logarithmic barrier method* for solving nonlinear optimization problems was specialized by Narendra Karmarkar as an *interior point method* for solving linear programs. In this book, the methods will be developed for general optimization problems (both linear and nonlinear) and applied to linear programs as a special case. This broader development of optimization not only highlights the current state of the subject but also connects much more naturally to calculus.

In addition to the Modeling Problems at the end of Chapter 1, there are four other basic forms of learning-by-doing activities incorporated throughout this text:

- Exercises: These are situated throughout the text precisely where they are relevant. They are meant to be attempted immediately when encountered and are short enough to be used as class questions during a lecture on the material.
- Problems: These are more in-depth activities than the Exercises and typically appear toward the end of the relevant sections. Problems too can be used as in-class student activities, but students will need more time to complete them than they will for Exercises.
- Computational Problems: As their title suggests, these activities depend in some way on the use of a computer. These are typically even longer-term activities than the Problems and are best completed out of class or as part of a devoted computer laboratory.
- Implementations: These involve the implementation of the optimization methods presented in this course and therefore depend heavily on the use of a computer. These are also the longest-term activities in the book and so are best completed out of class, possibly as a collaborative activity. Readers with no programming experience will need some support to complete the Implementations, and *Mathematica* notebooks which greatly simplify the programming can be downloaded from

<http://www.siam.org/books/ot114>

This text certainly does not present the complete picture of numerical optimization, nor does it give the historical background of the development of this field. Indeed, one goal here is to stimulate the reader's interest in learning more about numerical optimization, and the bibliography contains several very good resources where the reader can take a next step. In light of this goal, Chapter 9 is devoted to a Research Project through which readers can explore something else in this broad and vibrant field.

Acknowledgments

The content and form of this book have benefited greatly from the careful reading and thoughtful comments of my colleague Steve Wright, as well as the anonymous reviewers coordinated by Sara Murphy at SIAM. My students have also made many important contributions while this text was under development, which I appreciate very much.



Chapter 1

Modeling

To the average American, the word “modeling” evokes images of people representing some sort of ideal of human appearance. This chapter is not about ideals of human appearance, but it is about the practice of representing a real-world situation in an idealized form. The “models” we use here are mathematical ones, and they are constructed to represent certain significant aspects of real-world problems. There are many different kinds of mathematical models (for example, differential equations), but we will focus exclusively on optimization models since they cover plenty of interesting real-world problems and since optimization is the topic of the book. There are three main components in an optimization model:

- **Variables:** These represent components of the model that can be changed to create different possibilities.
- **Constraints:** These represent limitations on the variables.
- **Objective function:** This assigns a value to each different possibility and gets its name from the fact that the objective is to optimize this function.

Mathematical modeling is a skill best learned by doing, so any reader should solve some of the Modeling Problems found at the end of this chapter. To help prepare you for that task, we first present some worked examples where each of the three main components in the model is carefully described.

1.1 An Example with Yes/No Choices

A prospective college student is planning to visit the campuses of three colleges in Maine on one extended trip starting from and returning to the Jetport in Portland. The three schools are in the towns of Brunswick, Lewiston, and Waterville, and the student wants to visit each college town only once while making the round-trip as short as possible. In order to calculate round-trip distances, we need the distances between the towns which are given in Table 1.1.

Table 1.1. *Distances between college towns*

	Town 1	Town 2	Town 3	Town 4
	Portland	Brunswick	Lewiston	Waterville
Portland	0	26	34	78
Brunswick	26	0	18	52
Lewiston	34	18	0	51
Waterville	78	52	51	0

1.1.1 Choosing Variables

The most important step in building a model is deciding on variables that work. In this case, since any round-trip consists of a series of smaller trips between two towns, it makes sense to assign variables to the decision of whether to travel from each town to each other town. Thus, we have a variable $x_{1,2}$ which equals 1 if the student travels from Portland (Town 1) to Brunswick (Town 2) during the round-trip and equals 0 if the student does not. Notice that, since there is no such thing as a trip from a town to itself, we already have the constraints

$$x_{i,i} = 0 \text{ for all } i = 1, 2, 3, 4. \quad (1.1)$$

Once we have chosen our variables, we try to use them to formulate the problem. If we have trouble doing this, we might need to start over and define our variables differently. This is not necessarily a bad thing, as the process of trying to formulate the problem is very often a useful way to guide the choice of variables. This illustrates a fundamental nonlinearity in the process of mathematical modeling, which is one reason why the best way to learn mathematical modeling is to practice it.

1.1.2 Modeling Constraints

We deal now with the requirement in our example problem that each college town be visited only once. In particular, this means that Brunswick (Town 2) appears in the second spot of the trip for exactly one nonzero variable. In other words, $x_{i,2}$ is nonzero for exactly one i (and, moreover, $i \neq 2$ since $x_{2,2} = 0$). What we need for our model is a purely mathematical expression that encodes this information. One such expression that works is

$$x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1$$

since all the variables are constrained already to equal either 0 or 1 (and we already have the constraint (1.1)). There is nothing special in this argument about Brunswick, so we have the constraints

$$\sum_{i=1}^4 x_{i,j} = 1 \text{ for each fixed } j = 1, 2, 3, 4. \quad (1.2)$$

Exercise 1.1.1. *Explain what the case of $j = 1$ says about Portland.*

The constraints (1.1) and (1.2) take care of the requirement “to visit each college town only once,” but they do not necessarily ensure any kind of round-trip. For instance,

the assignment $x_{1,2} = 1$, $x_{1,3} = 1$, $x_{1,4} = 1$, $x_{2,1} = 1$, and all the other variables equal to zero satisfies the constraints (1.1) and (1.2), yet it describes an impossible scheme of visits since Portland is the origin for trips to all three college towns but is the destination only from Brunswick. Evidently, we also need the constraints

$$\sum_{j=1}^4 x_{i,j} = 1 \text{ for each fixed } i = 1, 2, 3, 4 \quad (1.3)$$

to ensure that each town serves as the origin for a trip to exactly one other town. Finally, to get a true round-trip, we have to rule out assignments that describe disconnected groups of “mini” round-trips like $x_{1,2} = x_{2,1} = 1$, $x_{3,4} = x_{4,3} = 1$, and all the other variables equal to zero. We can enforce this with the constraints

$$x_{i,j} + x_{j,i} \leq 1 \text{ for all } i = 1, 2, 3, 4 \text{ and } j = 1, 2, 3, 4. \quad (1.4)$$

Exercise 1.1.2. Explain why we should not instead use the equation constraint $x_{i,j} + x_{j,i} = 1$.

1.1.3 The Objective Function and the Final Model

Evidently, the series of constraints (1.1), (1.2), (1.3), and (1.4) define the allowed round-trips, so now we just need to describe the total distance associated with any allowed round-trip. Since our constraints are described in terms of the variables $x_{i,j}$, we want to describe the total distance in the same terms. Because our variables have only values of 0 or 1, we can simply multiply each variable $x_{i,j}$ by the distance $a_{i,j}$ corresponding to that trip (from the table at the beginning of this example) and add these up:

$$\sum_{i=1}^4 \left(\sum_{j=1}^4 (a_{i,j}) (x_{i,j}) \right).$$

Our mathematical model is to minimize this function over variables $x_{i,j}$ having value 0 or 1 and obeying the constraints (1.1), (1.2), (1.3), and (1.4).

Problem 1.1. Complete the following:

- (a) Describe the changes in the mathematical model that would result from including a fourth college town for the student to visit.
- (b) Describe the changes that would result from also including a fifth college town.

1.1.4 Solving the Model

The college-visit problem has a small enough scale that it could be reasonably solved without modeling it mathematically and using numerical optimization methods to solve the model. Since there are only three significantly different round-trips, the total distance associated with each could be easily computed and the shortest trip chosen. However, an analogous round-trip problem with more college towns could generate an unreasonable

number of allowed round-trips to check individually. For instance, with only 5 towns, there are already 60 significantly different round-trips to compare, and with 10 towns there are 1,814,400. Things get even more out of hand if the college towns are changed to destination cities and the problem is one of airline routing with many tens of cities.

Anyone who has access to optimization software should be able to feed an optimization model of relatively large scale into the computer, press a button, and obtain an answer. Readers with such access should certainly attempt to solve the college-visit problem in this way as well as the other modeling examples and Modeling Problems they encounter in this chapter. All readers should be able to verify that a solution to the original college-visit problem is

Portland → Brunswick → Waterville → Lewiston → Portland

with a total distance of 163 miles.

Exercise 1.1.3. *Give an obvious alternative solution to the original college-visit problem.*

1.2 Four Rules of Thumb for Modeling

There are different ways to model any optimization problem; however, models of the same optimization problem can pose very different model-solving challenges. The following general rules of thumb should help a modeler avoid some model-solving pitfalls.

- **Keep it simple:** If possible, use functions you recognize from calculus, and use the simplest of these. In particular, use linear functions whenever possible since these are the simplest (interesting) functions. For example, the constraint

$$\frac{x_1 + x_2}{x_3 + x_4} \leq 5$$

uses a nonlinear function on the left side (because of the quotient) and is much more challenging for solvers than would be the equivalent constraint

$$x_1 + x_2 \leq 5(x_3 + x_4)$$

using only linear functions. *Caveat:* Try not to lose accuracy of the model as a representation of the real problem by oversimplifying.

- **Keep it lean:** Constraints which can be removed without affecting the solutions to the model can cause difficulties for solvers. For example, the pair of constraints

$$\begin{aligned} x_1 + x_2 &\leq 3, \\ 2x_1 + 2x_2 &\leq 6 \end{aligned}$$

can be replaced by the first constraint $x_1 + x_2 \leq 3$ without affecting the solutions to the model, and solvers may have trouble with the pair of constraints.

- **Keep it compact:** Consolidate constraints when possible. For example, the paired constraint

$$5 \leq x_1 x_2 x_3 \leq 20$$

Table 1.2. *Data for steel production*

Raw material	C%	Si%	Su%	Ph%	Availability	Cost
limonite	3.0	0	0.013	0.015	40	200
taconite	2.5	0	0.008	0.001	30	250
hematite	0	0	0.011	0.05	60	150
magnetite	1.2	0	0.002	0.008	50	220
silicon 1	0	90	0.004	0.002	20	300
silicon 2	0	96	0.012	0.003	30	310
coke	90	0	0.002	0.01	25	165

is sometimes better than the pair of constraints

$$\begin{aligned}x_1 x_2 x_3 &\leq 20, \\ -x_1 x_2 x_3 &\leq -5,\end{aligned}\tag{1.5}$$

which identifies the same points (x_1, x_2, x_3) . Another example involves equations like $x_1 + x_2 = 0$ which are better than the equivalent pair of inequalities

$$x_1 + x_2 \leq 0 \text{ and } -x_1 - x_2 \leq 0.\tag{1.6}$$

Depending on the solver you use, the first constructions in both cases may enable the solver to avoid difficulties which would arise from using the pairs of constraints (1.5) or (1.6).

- **Keep it continuous:** It is better to use continuous variables if possible, and not variables constrained to have integer values or a finite number of values. Sometimes such “discrete” variables are necessary to model a problem appropriately (as in the college-visit problem in Section 1.1), but in those situations we can sometimes get discrete solutions without explicitly requiring that the variables be discrete. As we will see in Chapter 8, special methods are needed to explicitly require discrete variables.

You should have a more complete understanding of the reasons behind these rules after you have worked through the rest of the text, but the rules should be helpful even before that as you attempt to solve the Modeling Problems.

1.3 A Blending Example

An Iron Works needs to build a battleship from 50 tons of steel containing between 0.5% and 1.25% carbon (C), between 0.3% and 0.5% silicon (Si), no more than 0.05% sulfur (Su), and no more than 0.04% phosphorus (Ph). A supplier produces steel from seven raw materials whose grades, availabilities in tons, and costs in \$/ton are given in Table 1.2. The supplier wants to determine the least costly combination of raw materials that it can use to produce the steel that the Iron Works requires.

1.3.1 Choosing Variables

As usual, we choose variables by determining which components of the problem can be changed to create different possibilities. In this case, the supplier can change the amounts of raw materials used to produce the steel, so we could assign a different variable to represent the amount of each raw material:

$$\begin{aligned}x_1 &= \text{tons of limonite,} \\x_2 &= \text{tons of taconite,} \\x_3 &= \text{tons of hematite,} \\x_4 &= \text{tons of magnetite,} \\x_5 &= \text{tons of silicon 1,} \\x_6 &= \text{tons of silicon 2,} \\x_7 &= \text{tons of coke.}\end{aligned}$$

Notice that these variables can reasonably assume a continuum of real values, and so are not discrete like the variables in the college-visit problem.

1.3.2 Modeling Constraints

The variables in this case are naturally bounded below by zero (since negative amounts don't make sense) and bounded above by their availability, so we have the following constraints:

$$\begin{aligned}0 &\leq x_1 \leq 40, \\0 &\leq x_2 \leq 30, \\0 &\leq x_3 \leq 60, \\0 &\leq x_4 \leq 50, \\0 &\leq x_5 \leq 20, \\0 &\leq x_6 \leq 30, \\0 &\leq x_7 \leq 25.\end{aligned}$$

Assuming that any amount of a raw material contributes to the same amount of steel, and knowing that we need to produce at least 50 tons, we have

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \geq 50.$$

Notice that we don't assume that we will produce *exactly* 50 tons since it may be necessary to produce more to meet the other requirements of the problem.

The other constraining feature in this problem is that the steel must contain certain percents of carbon, silicon, sulfur, and phosphorus. To see how these composition requirements translate into constraints on our variables, we'll focus first on the requirement that the steel have between 0.5% and 1.25% carbon composition and expect the requirements for silicon, sulfur, and phosphorus to be similar. From the given data, we know the percent contribution of carbon from each of the raw materials, so we can easily calculate the *amount* of carbon for any choice of variables as $0.03x_1 + 0.025x_2 + 0.012x_4 + 0.9x_7$. However,

since the requirement involves the *percent* of carbon in the steel, we need to divide the amount of carbon by the amount of steel:

$$\% C = 100 \left(\frac{\text{tons of carbon}}{\text{tons of steel}} \right) = \frac{3.0x_1 + 2.5x_2 + 1.2x_4 + 90x_7}{x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7}.$$

Using this, the requirement that the steel contain between 0.5% and 1.25% carbon translates into the paired constraint

$$0.5 \leq \frac{3.0x_1 + 2.5x_2 + 1.2x_4 + 90x_7}{x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7} \leq 1.25. \quad (1.7)$$

Exercise 1.3.1. Write the analogous composition constraints for silicon, sulfur, and phosphorus.

1.3.3 The Objective Function and the Final Model

Since this problem involves finding the least costly combination of raw materials that meets the demand for 50 tons of finished steel, the objective function is simply the cost of the raw materials used:

$$\text{Cost} = 200x_1 + 250x_2 + 150x_3 + 220x_4 + 300x_5 + 310x_6 + 165x_7,$$

where each raw material contributes its own cost to the total. The optimization problem is thus to minimize this cost function over all choices of the variables that satisfy the modeled constraints.

1.3.4 Solving the Model

The scale of this problem is much larger than that of the college-visit problem, where all of the possible scenarios could be relatively easily compared. In particular, now the variables can have a continuum of values, so there are essentially infinitely many possible scenarios. Nonetheless, we can do some thinking about the problem to get insight about a solution. For instance, the objective function to be minimized is obviously decreased whenever any of the variables decreases, and the smallest contributed costs come from the variables with the smaller coefficients (i.e., the raw materials with least per-unit cost).

If we ignore the composition constraints, then the supplier should produce exactly 50 tons of “steel” from the cheapest available raw materials. This would mean using 50 tons of the 60 tons of available hematite (costing \$150 per ton) at a total cost of \$7,500.

Exercise 1.3.2. Explain why now enforcing the composition constraints can’t possibly lead to a lower total cost than \$7,500.

Before attempting to solve the full optimization problem with a computer, we should rewrite the composition constraints in a simpler form (see the first rule of thumb in the preceding section). For instance, the paired constraints (1.7) are nonlinear (since there are variables in the denominator of the fraction) but can be expressed equivalently as paired *linear* constraints by multiplying the denominator through. After simplifying all of the

Table 1.3. *Data on museum renovation*

Task	Weeks	Preceding tasks
1-Relocation of fragile artwork	4	none
2-Site preparation	2	1
3-Excavation	16	2
4-Entrance tunnel	10	3
5-Foundation work	40	3
6-Entrance pavilion	12	4
7-Facade	4	2
8-Grand staircase	6	5
9-Teaching gallery	6	2
10-Terrace repair	3	5,7
11-Electrical system	4	6,9,16,18
12-Climate control system	3	11,14
13-Lighting	5	11
14-Windows	6	7,10
15-Loading zone	4	2
16-Offices	7	2
17-Landscaping	4	6,14,15
18-Storage space	3	2
19-Reinstall artwork	5	7,9,12,13,14,15,16,18

composition constraints in this way and feeding the model into our optimization software, we get the solution

$$x_1 = 13.7888, x_3 = 35.8097, x_5 = 0.166667, x_7 = 0.234818, x_2 = x_4 = x_6 = 0,$$

which translates into exactly 50 tons of steel produced from 13.7888 tons of limonite, 35.8097 tons of hematite, 0.166667 tons of silicon 1, and 0.234818 tons of coke. The total production cost for this solution of \$8,217.96 is clearly above the lower bound of \$7,500 that we got by ignoring the composition constraints. If we had gotten a cost lower than \$7,500, we would have known that something was very wrong with our answer.

1.4 A Scheduling Example

A construction company wants to renovate an art museum in the least possible time, where the renovation can be broken down into a series of 19 tasks. Each task can be completed in a certain number of weeks, and some of the tasks can begin only after other tasks have been completed. Table 1.3 lists the durations in weeks and immediate predecessors of each task.

1.4.1 Choosing Variables

The construction company wants to complete the job as quickly as possible, and the job is completed precisely when all 19 tasks are completed. Since the duration of each task is known, the moment of completion of each task is determined entirely by its starting

time. The starting times are also exactly what the company can choose, so these should be reasonable variables for this problem:

$x_i =$ the starting time of task i .

1.4.2 Modeling Constraints

Assuming that the clock starts at the beginning of the project, the starting-time variables must all be nonnegative $x_i \geq 0$. The only other constraints in this problem have to do with the fact that some tasks can begin only after other tasks are completed. For instance, task 2 can begin only after task 1 is completed. Since task 1 has a duration of 4 weeks, its time of completion is given by $x_1 + 4$. Thus, the requirement that task 2 begin after the completion of task 1 can be modeled by the constraint

$$x_2 \geq x_1 + 4.$$

There are analogous constraints for each task and its predecessors.

Exercise 1.4.1. *Model the requirement that task 17 can begin only after the completion of tasks 6, 14, and 15.*

Notice that we have to create constraints modeling only each task's *immediate* predecessors as listed in Table 1.3 since any earlier predecessors are implicitly covered. For example, task 3 must wait for the completion of task 2, and task 2 must wait for the completion of task 1, so task 3 implicitly must also wait for the completion of task 1. Fortunately, the constraints enforcing the first two explicit relationships will automatically ensure the third, implicit, relationship.

1.4.3 The Objective Function and the Final Model

Since the construction company wants to complete the job as quickly as possible, the objective function should somehow compare different job lengths based on different choices of starting times for the tasks. Because the different tasks can overlap in time, it is difficult to know what combination of variables measures the total time for the job without actually solving the problem first. If there was a “final” task which had to follow all the others, we could make the objective function equal to the start time of the final task. Task 19—Reinstall artwork *sounds* like a final task, but its complete chain of predecessors (deduced from Table 1.3) does not include task 17. Thus, it's possible that task 17 is the true final task.

Exercise 1.4.2. *From the predecessor data given in Table 1.3, list another task (in addition to task 17) which is not necessarily completed in advance of task 19.*

A simple trick that works in this situation is to create a new variable $x_{20} \geq 0$ to represent the starting time of the final task of job completion. Since we are creating this new task, we can assign its duration as 0 and its predecessors as each of the other tasks. Following the second rule of thumb to “keep it lean,” we can reduce the number of explicit predecessors to the final task by tracking the chains of predecessors implicit in Table 1.3 as we did in Exercise 1.4.2 for task 19.

Table 1.4. *Flight data*

Start City	Final City					
	New York	Phila.	Detroit	Chicago	Atlanta	D.C.
Portland	16	12	6	8	3	10
Bangor	35	27	19	22	11	30
Manchester	15	9	10	17	6	13
Hartford	16	5	0	13	2	12
Providence	12	12	7	21	0	15
Hyannis	2	0	0	5	8	9

Exercise 1.4.3. Give three tasks that can be used as explicit predecessors of the final task in order to guarantee that all 19 tasks will be completed before the final task “begins.”

Since our new variable x_{20} measures the starting time of job completion, our objective function is simply this variable. The final model is thus to minimize x_{20} subject to the nonnegativity and predecessor constraints on all the variables (including the new variable x_{20}).

1.4.4 Solving the Model

This model could be solved by hand without too much difficulty; however, it is simpler to feed it to a computer. In any case, there are several different solutions which take the minimum time of 79 weeks, since several of the tasks can be delayed without affecting the final job completion time (for instance, task 4 can start any time after week 22 and before week 43).

1.5 An Assignment Example

An airline uses Logan airport in Boston as a hub through which airplanes from smaller New England cities are routed to their ultimate destinations. Between 9AM and 11AM there is one airplane arriving in Boston from each of Portland, Bangor, Manchester, Hartford, Providence, and Hyannis. The same airplanes are then used for departing flights between 1PM and 3PM to New York, Philadelphia, Detroit, Chicago, Atlanta, and Washington D.C. Table 1.4 gives the numbers of passengers from each arriving flight having each ultimate destination, and the airline wants to redirect the six airplanes so that the fewest number of passengers need to switch airplanes (thus minimizing the need to move baggage).

1.5.1 Choosing Variables

In this case, the airline needs to choose where each arriving airplane goes next. In particular, the airline has to decide whether or not the airplane from each origin city continues on to each of the destination cities. To incorporate these choices, we can use variables like

$$x_{i,j} = \begin{cases} 1 & \text{if the airplane from city } i \text{ continues to city } j, \\ 0 & \text{otherwise,} \end{cases}$$

where Portland corresponds to $i = 1$, Bangor corresponds to $i = 2$, and so on down the column of start cities in Table 1.4; and New York corresponds to $j = 1$, Philadelphia corresponds to $j = 2$, and so on across the row of final cities.

1.5.2 Modeling Constraints

The constraints in this example follow directly from the choice of variables since exactly one airplane leaves each origin city

$$x_{i,1} + x_{i,2} + x_{i,3} + x_{i,4} + x_{i,5} + x_{i,6} = 1 \quad \text{for each } i = 1, \dots, 6$$

and each destination city is visited by exactly one airplane from the origin cities

$$x_{1,j} + x_{2,j} + x_{3,j} + x_{4,j} + x_{5,j} + x_{6,j} = 1 \quad \text{for each } j = 1, \dots, 6.$$

1.5.3 The Objective Function and the Final Model

The airline wishes to minimize the number of passengers needing to switch airplanes, which is the same thing as maximizing the number of passengers that stay on their original airplanes as they travel through to their destination cities. Table 1.4 shows how many passengers on each original flight have each ultimate destination, and it is easy to use this data to maximize the number of passengers staying on the airplane from any single origin city. For instance, the number of passengers staying on the airplane from Portland is maximized when that airplane continues to New York. In terms of our variables, the different possibilities for the numbers of passengers staying on the airplane from Portland can be measured by the function

$$\sum_{j=1}^6 a_{1,j} x_{1,j},$$

where the $a_{1,j}$ are the numbers across the first row of Table 1.4.

There are analogous functions for the other origin cities, and Table 1.4 can be used similarly to determine the maximal number of passengers staying on any single airplane. However, the airline might not be able to simply maximize the number of passengers staying on each individual airplane. For instance, the airplane from Portland is not the only one which maximizes its number of continuing passengers by being redirected to New York; the airplanes from Bangor and Hartford also do this. But the airline can send only one of these airplanes to New York and wants to pick one that maximizes the cumulative number of passengers staying on *all* of the airplanes. Thus, the objective function to be maximized over the variables satisfying the constraints is

$$\sum_{i=1}^6 \sum_{j=1}^6 a_{i,j} x_{i,j},$$

where $a_{i,j}$ is the number from row i and column j of Table 1.4.

1.5.4 Solving the Model

According to the computer, a solution to this problem is $x_{1,2} = 1$, $x_{2,1} = 1$, $x_{3,3} = 1$, $x_{4,6} = 1$, $x_{5,4} = 1$, $x_{6,5} = 1$, and the rest of the variables equal to zero.

Exercise 1.5.1. *What does this solution say about how the airline should redirect the incoming airplanes, and how many passengers stay on their original airplane?*

1.6 A Partitioning Example

“Gerrymandering” is term for the setting of boundaries of districts to gain an advantage for the political party in control. This term gets its name from Eldridge *Gerry*, who, as governor of Massachusetts in the early 1800’s, created a district whose shape resembled a salamander. There are typically two kinds of gerrymandering: “packing” (concentrating the opposition in a few districts so the ruling party can control the rest) and “dilution” (spreading the opposition so that it is relatively weak in all districts).

In this section, we will suppose (fictitiously, as far as I know) that the leaders of Maine’s legislature want to dilution-gerrymander 3 districts from the 15 towns in Androscoggin County so that supporters of the ruling party are not in the minority in any of the three districts. Figure 1.1 shows a schematic of the towns in Androscoggin County with population (in thousands) in parentheses. A valid district consists of a grouping of 8 or fewer towns, where any town in the grouping is connected to any other town in the grouping by traveling across edges (and not just corners) between towns in the same grouping. In addition, a valid district needs to have a population between 32,000 and 38,000.

1.6.1 Choosing Variables

Since the element of choice in this problem is the partitioning of districts, we need to determine the complete set of valid districts. The fancy term for this process of determination is “data preprocessing,” and there are straightforward ways to program a computer to do much of the work. To get an idea of how this might look, we will do some of the work ourselves right now.

Data Preprocessing

To generate all of the valid districts containing the town of Lewiston, we consider which (no more than five) adjacent communities can be added to Lewiston to yield a total population between 32,000 and 38,000. This results in exactly three valid districts: Lewiston (population 35,000), Lewiston/Mechanic Falls (population 38,000), and Lewiston/Minot (population 37,000).

Exercise 1.6.1. *Explain why Lewiston/Greene is not a valid district. Explain why Lewiston/Wales is not a valid district.*

Applying this same process to the other towns, we can proceed systematically to determine that there are 55 valid districts. For example, the results of this preprocessing to find the 27 valid districts containing the town of Auburn appear in Table 1.5.

Livermore Falls (3)		
Livermore (2)		Leeds (2)
Turner (5)		
Minot (2)	Greene (4)	Wales (1.5)
	Lewiston (35)	Sabattus (5)
Mechanic Falls (3)		Lisbon (9)
Auburn (23)		
Lisbon Falls (4.5)		
Poland (5)		Durham (4)

Figure 1.1. *Schematic of the 15 towns in Androscoggin County*

Variables

Once we have the complete set of valid districts, we can create partitions of the county by using the variables

$$x_i = \begin{cases} 1 & \text{if district } i \text{ is used in the partition} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i = 1, \dots, 55. \quad (1.8)$$

1.6.2 Modeling Constraints

In order to enforce the requirement that a partition consists of exactly 3 districts, we simply enforce the constraint

$$\sum_{i=1}^{55} x_i = 3. \quad (1.9)$$

To make sure that each town appears exactly once in a partition, we index the towns alphabetically (so town 1 is Auburn, town 2 is Durham, etc.) and enforce the constraints

$$\sum_{i=1}^{55} a_{i,j} x_i = 1 \quad \text{for } j = 1, \dots, 15, \quad (1.10)$$

Table 1.5. *Valid districts containing Auburn*

District	Towns (in addition to Auburn)	Pop.
4	Poland/Lisbon Falls	32.5
5	Poland/Lisbon Falls/Mechanic Falls	35.5
6	Poland/Lisbon Falls/Durham	36.5
7	Poland/Durham	32
8	Poland/Durham/Mechanic Falls	35
9	Poland/Durham/Mechanic Falls/Minot	37
10	Lisbon Falls/Durham/Mechanic Falls	34.5
11	Lisbon Falls/Durham/Mechanic Falls/Minot	36.5
12	Lisbon Falls/Mechanic Falls/Minot	32.5
13	Lisbon Falls/Mechanic Falls/Minot/Greene	36.5
14	Lisbon Falls/Mechanic Falls/Minot/Turner	37.5
15	Lisbon Falls/Mechanic Falls/Minot/Greene/Wales	38
16	Lisbon Falls/Lisbon	36.5
17	Poland/Mechanic Falls/Minot	33
18	Poland/Mechanic Falls/Minot/Greene	37
19	Poland/Mechanic Falls/Minot/Turner	38
20	Mech. Falls/Minot/Greene	32
21	Mech. Falls/Minot/Greene/Wales	33.5
22	Mech. Falls/Minot/Greene/Wales/Leeds	35.5
23	Mech. Falls/Minot/Greene/Wales/Leeds/Livermore	37.5
24	Mech. Falls/Minot/Greene/Turner	37
25	Mech. Falls/Minot/Turner	33
26	Mech. Falls/Minot/Turner/Leeds	35
27	Mech. Falls/Minot/Turner/Livermore	35
28	Mech. Falls/Minot/Turner/Leeds/Livermore	37
29	Mech. Falls/Minot/Turner/Livermore/Liv. Falls	38
30	Mech. Falls/Minot/Turner/Leeds/Liv. Falls	38

where the coefficients $a_{i,j}$ are defined by

$$a_{i,j} = \begin{cases} 1 & \text{if town } j \text{ is in district } i, \\ 0 & \text{otherwise.} \end{cases}$$

Exercise 1.6.2. *Using Table 1.5, give the values of $a_{i,8}$ for $i = 4, \dots, 30$.*

1.6.3 The Objective Function and the Final Model

In this problem, a partition is sought for which the supporters of the ruling party are not in the minority in any district. Thus, we need data on supporters in each town, as well as some way of indicating when those supporters are not in the minority in a given district. Suppose that the numbers of current supporters in each town are as in Table 1.6. If we set the values $b_i = 1$ when the ruling party is not in the minority in district i and $b_i = 0$ otherwise, the

Table 1.6. *Current supporters in each town*

Town	Current Supporters	Total Population
Auburn	14	23
Durham	2.5	4
Greene	2.5	4
Leeds	1	2
Lewiston	18	35
Lisbon	7	9
Lisbon Falls	2	4.5
Livermore	0.5	2
Livermore Falls	1	3
Mechanic Falls	1	3
Minot	0.5	2
Poland	1	5
Sabattus	2	5
Turner	2	5
Wales	0.5	1.5

objective function

$$\sum_{i=1}^{55} b_i x_i \quad (1.11)$$

tracks the cumulative support for the ruling party in a partition. To determine the values of the b_i , we use Table 1.6 to add the number of supporters in all of the towns composing district i and divide by the population of district i :

$$b_i = \begin{cases} 1 & \text{if } \frac{\text{supporters in district } i}{\text{population of district } i} \geq 0.5, \\ 0 & \text{otherwise.} \end{cases}$$

The final model is thus to maximize the objective function (1.11) subject to the constraints (1.9) and (1.10) over the 55 variables x_i defined in (1.8).

Exercise 1.6.3. *Explain why a choice of variables satisfying the constraints (1.9) and (1.10) for which the value of the objective function (1.11) is 3 corresponds to a successful partitioning of the county from the perspective of the ruling party.*

1.6.4 Solving the Model

Depending on the numbers of supporters in each town listed in Table 1.6, this problem could have no, one, or multiple solutions.

Exercise 1.6.4. *Describe a modified version of Table 1.6 which would guarantee that this problem had no solution.*

Exercise 1.6.5. *Maine currently uses district 9 from Table 1.5, district 1 (Lewiston), and the rest of the towns (as well as one ward of Lewiston) in a third district. Confirm that*

Table 1.7. *Crop data*

crop	yield in bushels/acre	labor/acre	price/bushel
soy	10	5	3
oats	20	7	4

Table 1.8. *Clam data*

	cost/lb	% large	% medium	% small
Supplier 1	\$0.23	40	30	30
Supplier 2	\$0.18	30	35	35
Supplier 3	\$0.17	30	30	40

the current districting—ignoring the one ward of Lewiston—solves the gerrymandering problem posed at the beginning of this section.

1.7 Modeling Problems

Modeling Problem 1. Based on Table 1.7, write and solve an optimization model to answer the following question: If 100 acres of land and 400 hours of labor are available, how many acres of each crop should be produced to maximize revenue under the additional government regulation that at least 30 bushels of soy be produced?

Modeling Problem 2. For a factory that manufactures trucks and cars, each vehicle must go through an assembly stage and a painting stage. If the painting was entirely devoted to cars, 500 per day could be painted. On the other hand, 400 trucks could be painted per day if the painting was devoted to trucks. The analogous numbers for assembly are 80 cars per day and 65 trucks per day. Each car produces \$1300 profit and each truck produces \$1500 profit.

Write and solve an optimization model to answer the following question: How many cars and trucks should the factory produce to maximize profit?

Modeling Problem 3. Consider three paper factories which each pour arsenic and mercury into the same river. It costs \$1450 to process a ton of factory 1 waste, and each ton processed reduces the amount of arsenic by 0.1 ton and the amount of mercury by 0.32 tons. For factory 2, the analogous numbers are \$1210, 0.15 tons, and 0.2 tons, and for factory 3, the numbers are \$2200, 0.3 tons, and 0.3 tons. Regulations stipulate that arsenic needs to be reduced by at least 200 tons and mercury by 300 tons.

Write and solve an optimization model to answer the following question: How can this be accomplished at minimum total cost?

Modeling Problem 4. A restaurant uses three sizes of clams for different dishes on its menu. The restaurant buys clams from three different suppliers whose costs and clam-size mix are recorded in Table 1.8. At least 600 lbs large, 500 lbs medium, and 500 lbs small clams are needed each month, and each supplier can supply at most 800 lbs clams per month.

Write and solve an optimization model to answer the following question: How can the restaurant meet its needs for clams at the lowest cost?

Modeling Problem 5. *A person's daily nutritional requirements include the need for 3 units of vitamin A and 1.5 units of vitamin C. Bananas sell for \$6 per pound, and each pound contains 5 units of vitamin A and 1 unit of vitamin C. Rice sells for \$1 per pound, and each pound contains 1 unit of each vitamin.*

- (a) *Write and solve an optimization model to answer the following question: With only bananas and rice to eat, what is the least expensive scheme that satisfies the person's requirements?*
- (b) *Write and solve an optimization model to answer the following question: What is the least expensive scheme that satisfies the person's requirements exactly?*

Modeling Problem 6. *Write and solve an optimization model to answer the following question: If mining in mine 1 will produce 2 lbs each of gold and silver per day but mining in mine 2 will take twice as long to produce the same amount of gold as mine 1 but only $\frac{1}{3}$ as long to produce the same amount of silver as mine 1, how quickly can a lone miner produce at least 9 lbs of gold and 14 lbs of silver?*

Modeling Problem 7. *A supermarket can be staffed with full- and part-time employees in any way that covers the labor requirements of 400 total person hours on Saturday, 400 on Sunday, 200 on Monday, 200 on Tuesday, 250 on Wednesday, 250 on Thursday, and 300 on Friday. Union rules require that each full-time employee work 5 consecutive full days (8 hours each) followed by 2 days off, and that each part-time employee work 5 consecutive half days (4 hours each) followed by 2 days off. A full-time employee costs the supermarket \$10 per hour and a part-time employee costs \$8 per hour.*

Write and solve an optimization model to answer the following question: If union rules also limit part-time labor to 25% of the weekly labor force, how can the supermarket meet its requirements at least cost?

Modeling Problem 8. *Consider Modeling Problem 7, but now suppose that only full-time employees are used and that the supermarket can require them to work 1 day of overtime each week (with only 1 day off).*

Write and solve an optimization model to answer the following question: If each employee is paid \$80 a day for each of the first 5 days and \$90 for the overtime day (if any), how can the supermarket meet its labor requirements at least cost?

Modeling Problem 9. *A small college in Maine is considering five different investment opportunities. The costs associated with each investment as well as the net present values (NPV) all in millions of dollars are given in Table 1.9. The college has \$50 million available for investment in the first year and expects to have \$25 million available in the second year. The college can purchase any fraction of each investment, where the fraction is then applied to the costs of the investment as well as the NPV.*

Write and solve an optimization model to answer the following question: If any funds left over from the first year cannot be used in the second year, how should the college invest in order to maximize total NPV?

Table 1.9. *Investment data*

	Inv. 1	Inv. 2	Inv. 3	Inv. 4	Inv. 5
Year 1 cost	\$9	\$42	\$6	\$6	\$31
Year 2 cost	\$4	\$3	\$7	\$2	\$23
NPV	\$11	\$30	\$10	\$8	\$42

Modeling Problem 10. *A small college in Maine needs to decide how much investment and debt to undertake during the next year. Each dollar invested reduces the NPV of the college by 15 cents, and each dollar of debt increases the NPV by 70 cents. The college can invest at most one million dollars during the next year, and debt can be at most 50% of investment.*

Write and solve an optimization model to answer the following question: If the college has \$400,000 in cash available and all investments must be paid for from current cash or borrowed money, what should the college do to maximize NPV?

Modeling Problem 11. *A small electronics manufacturer makes DVD players (each selling for \$150 and costing \$65 for labor and \$40 for raw materials) and cell phones (each selling for \$100 and costing \$45 for labor and \$20 for raw materials). On the first of the year, the company has a raw material inventory (valued at \$50,000). Some of this material is appropriate for producing DVD players (no more than 1000) and some is appropriate for producing cell phones (no more than 500). At the same time, the company's other assets consist of \$100,000 in cash and \$35,000 outstanding from customers for previous purchases, while the company owes a bank \$100,000 on a loan.*

Payment for goods sold this year will not be received for two years; however, during this year, the company will collect \$20,000 from customers for previous purchases and must pay off \$10,000 of the outstanding loan as well as \$12,000 in rent. At the end of the year, the company needs to have at least \$45,000 in cash, and, in addition, the company will receive a \$20,000 shipment of raw material which it will pay for one year after that. The bank requires that the company's asset/liability ratio be at least 2 at the beginning of the next year.

Write and solve an optimization model to answer the following question: What should the company produce this year to maximize this year's contribution to profit from production?

Modeling Problem 12. *An oil company has 6,000 barrels of oil 1 (with quality level 10) and 11,000 barrels of oil 2 (with quality level 6) and cannot restock for this selling period. The company sells two products made from a combination of the two oils: gasoline, which must have an average quality level of at least 8, and heating oil, which must have an average quality level of at least 7. Each dollar spent advertising gasoline results in 3 barrels of demand and each dollar spent advertising heating oil results in 7 barrels of demand, and there is no demand without advertising.*

Write and solve an optimization model to answer the following question: If gasoline is sold for \$35 per barrel, and heating oil is sold for \$30 per barrel, what should the oil company sell to maximize profit?

Modeling Problem 13. *A gasoline manufacturer produces regular and premium gasoline from six different ingredients, each having three important attributes: research octane*

Table 1.10. Gasoline ingredient data

Ingredient	Availability	RON	MON	RVP
1	16,113	98.6	91.9	3.22
2	14,505	93.1	88.2	3.37
3	7083	87.4	80.7	11.43
4	2430	96.9	91.3	5.12
5	576	116.7	107.4	4.97
6	68,452	98.3	92.4	57.3

Table 1.11. Gasoline demand and required attributes

	Demand	AKI	RVP
Summer Regular	≥ 7.6	≥ 87 and ≤ 90	≤ 7.8
Summer Premium	≥ 2.1	≥ 91	≤ 7.8
Winter Regular	≥ 7.2	≥ 87 and ≤ 90	≤ 9.0
Winter Premium	≥ 1.9	≥ 91	≤ 9.0

number (RON), motor octane number (MON), and Reid vapor pressure (RVP). The last of these attributes, RVP, affects the volatility (tendency to evaporate) of the resulting gasoline, and the first two attributes, RON and MON, are averaged to produce an antiknock index (AKI), which is the commonly reported octane rating. The daily availability (in gallons) and attributes (per gallon) of each ingredient are given in Table 1.10. The evaporation of gasoline contributes to ground-level ozone, and atmospheric ozone levels are naturally higher during the summer. As a result, the RVP of gasoline is regulated more strongly in the summer than in the winter. The projected demand (in thousands of gallons per day) and required attributes (per gallon) for each type of gasoline in each season are given in Table 1.11.

Write and solve optimization models to answer the following questions:

- If regular gasoline sells for \$3.60 per gallon and premium sells for \$4.00 per gallon, and if neither type of gasoline can have more than 10 percent of its volume come from ingredient 2, how could the oil company maximize its daily revenue during the summer?
- Under the same conditions, how could the oil company maximize its daily revenue during the winter?

Modeling Problem 14. A food manufacturer produces corn syrup, fig paste, and fig bars. There is unlimited demand for these products at the respective selling prices per unit of \$0.25, \$1.2, and \$3.5. One unit of corn syrup requires $\frac{1}{2}$ hour of labor; one unit of fig paste requires 1.5 hours of labor in addition to one unit of corn syrup; and one unit of fig bars requires 2 hours of labor in addition to two units of fig paste.

Write and solve an optimization model to answer the following question: If there is a total of 300 hours of labor available, how can revenues be maximized?

Modeling Problem 15. A violin maker expects the following demands for violins: 5 violins in the first quarter, 4 violins in the second quarter, 3 violins in the third quarter, and 3 violins

Table 1.12. *Bond data*

	Bid Price	Offer Price
Airport bond	98	99
Higher Ed. bond 2	97	98
Pension bond	93	95

Table 1.13. *Transaction data*

Year	Airport bond	Higher Ed. bond	Pension bond
1	10.1	8.4	7.9
2	11.2	9.5	8.3
3	111	113	107

in the fourth quarter. There is an inventory of 6 violins at the beginning of the first quarter. During each quarter, the violin maker can produce 2 violins at a cost of \$200 per violin, and can use overtime employees to make additional violins if necessary, but at a cost of \$300 per violin.

Write and solve an optimization model to answer the following question: If, at the end of each quarter, a holding cost of \$25 per violin is incurred for violins not sold, how should the violin maker schedule production in order to minimize the sum of the production and inventory costs during the four quarters while meeting the expected demand?

Modeling Problem 16. An investor is considering a municipal bond market with three categories of bonds available for purchase and sale: one to support an airport expansion, one for higher education, and one to support pensions. The investor can buy up to 500 units of airport bonds, 1000 units of higher education bonds, and 1200 units of pension bonds at the corresponding “offer” price (the price at which each bond is being offered for sale), and the investor already owns 2000 bonds in each category which can be sold at the corresponding “bid” price (the price which has been bid to buy them). The bid and offer prices of each bond are given in Table 1.12.

As part of any transaction, the seller of the bond agrees to pay the buyer a series of cash payments as in Table 1.13. These payments are discounted, so that a payment of \$1 one year from now is equivalent to a payment of 94 cents now. The investor is required to have a nonnegative cash position (due only to cash payments from bonds and not to purchases or sale of bonds) after each year’s payments are made.

Write and solve an optimization model to answer the following question: How should the investor maximize revenue from selling bonds less payment for buying bonds?

Modeling Problem 17. An automaker advertises with commercial spots on three types of TV programs: dramas (costing \$150,000 per minute and seen by 8 million women and 4 million men), sports (costing \$180,000 per minute and seen by 2 million women and 12 million men), and news (costing \$130,000 per minute and seen by 4 million women and 5 million men).

- (a) Write and solve an optimization model giving a strategy to reach at least 35 million women and 35 million men at least cost.

Table 1.14. *Property data*

	Value	Price
Property 1	\$160,000	\$165,000
Property 2	\$220,000	\$229,000
Property 3	\$310,000	\$315,000
Property 4	\$558,000	\$570,000

Table 1.15. *Data for lobster processing*

Load	1	2	3	4	5	6	7	8	9	10
Loss (lb/hr)	4	3	3.5	2.5	1.5	5	6	4.5	2	3
Edible content (lb)	32	24	14	10	6	40	48	36	16	18

- (b) Now do the same under the further stipulation that only one-minute-long commercial spots can be used.

Modeling Problem 18. An investor is interested in purchasing real estate, and there are four properties for sale. The assessed value and selling price for each property are given in Table 1.14.

- (a) Write and solve an optimization model to maximize value if there is \$1,000,000 available to invest.
- (b) How do your model and its solution change under the additional requirement that property 3 must be purchased if property 2 is purchased?

Modeling Problem 19. A seafood processing plant needs to process 10 50-lb loads of lobster arriving simultaneously at the plant, and the edible content of each load degrades over time. For each load, the hourly loss of edible content and amount of edible content upon arrival are given in Table 1.15. There are two parallel production lines at the plant which can each process any of the loads in one hour, and the processing of any load must be completed before that load loses all of its edible content. After processing, the edible content of the lobster no longer degrades.

Write and solve an optimization model to determine the production schedule that maximizes the amount of processed edible lobster.

Modeling Problem 20. A prospective gold mine has been subdivided into 25 5-ton blocks in the following formation:

Level 1	1	2	3	4	5	6	7	8	9	\$200
Level 2		10	11	12	13	14	15	16		\$300
Level 3			17	18	19	20	21			\$400
Level 4				22	23	24				\$500
Level 5					25					\$600

where the extraction costs per ton at each level are given in the final column. The only exception is block 9, which sits under a wetland and as a result requires an additional \$3,650

Table 1.16. *Data on horse feed*

Raw material	Protein%	Fat%	Crude Fiber%	Availability	Cost
Corn	8	4	2	11900 lb	0.13 \$/lb
Barley	11	2	5	23500 lb	0.17 \$/lb
Molasses	6.6	0	0	750 lb	0.12 \$/lb
Oats	12	5	12	900 lb	0.16 \$/lb
Alfalfa	18	2.6	25	1150 lb	0.21 \$/lb
Timothy hay	9	2.5	30	650 lb	0.18 \$/lb

per ton to extract. The extraction of any block at a lower level requires the extraction of the three nearest blocks above it (so, for example, blocks 11, 12, and 13 must be extracted before block 18 can be extracted). The market values in \$/ton of each block are as follows:

Level 1	100	300	200	20	10	10	100	10	50
Level 2		400	500	300	200	100	100	150	
Level 3			400	300	1000	1500	500		
Level 4				2000	2100	1900			
Level 5					3000				

Write and solve an optimization model to determine which blocks should be extracted to maximize profit. What is the result if the additional per-ton cost of extracting block 9 is \$3,350 instead of \$3,650?

Modeling Problem 21. Horse feed can come in pellets (ground corn, barley, and oats, blended with molasses) and silage (chopped and dried oats, barley, alfalfa, and timothy hay). Horse feed in either form has the following protein, fat, and crude fiber content requirements: protein percentage ≥ 5 , fat percentage ≥ 3 , and crude fiber percentage ≤ 18 . These components are contained in the raw materials at the percentages given in Table 1.16. In addition to the costs of the raw materials given in Table 1.16, there are production costs of 0.35 \$/lb for grinding, 0.07 \$/lb for blending, 0.12 \$/lb for drying, and 0.09 \$/lb for chopping.

Write and solve an optimization model to answer the following question: What mix of raw materials will produce at least 1000 lbs of pellets and 700 lbs of silage at a minimal cost?

Modeling Problem 22. An outdoor gear and apparel store wishes to cover its in-store registers with the fewest number of employees. Table 1.17 gives the minimum number of staffed registers needed for each different 2-hour shift. Each employee works exactly 8 hours per day in alternating shifts: the first shift on a register, the second shift on the floor, the third shift on a register, and a final shift stocking.

Write and solve an optimization model to answer the following question: What is the fewest number of employees that can cover the store's daily register needs?

Modeling Problem 23. Five fast-food workers are to be assigned to five different stations, and Table 1.18 gives the productivity on a scale of 1–10 of each worker at each station.

Table 1.17. *Registers needed per shift*

Shift	Minimum number of registers needed
12PM–2AM	5
2AM–4AM	4
4AM–6AM	6
6AM–8AM	10
8AM–10AM	30
10AM–12AM	30
12AM–2PM	30
2PM–4PM	25
4PM–6PM	30
6PM–8PM	35
8PM–10PM	35
10PM–12PM	10

Table 1.18. *Productivity on a scale of 1–10*

Worker	Station				
	Fries	Burgers	Ice Cream	Register	Drive-Thru
1	1	6	6	5	9
2	3	6	9	4	10
3	2	5	6	3	6
4	3	6	6	2	7
5	4	7	7	2	8

Table 1.19. *Distribution center daily inputs and outputs*

Inputs	ME	VT	NH	MA 1	MA 2
Value of DVD stock (in \$1000's)	1200	1100	1300	1500	1400
Wages	\$2,000	\$1,580	\$2,100	\$3,800	\$4,000
Outputs	ME	VT	NH	MA 1	MA 2
Deliveries	1100	950	1120	1220	1190
Restock requests	100	70	150	210	230

Write and solve an optimization model to answer the following question: If the total productivity of the restaurant is the sum of the productivities of the workers assigned to the stations, what assignment of workers to stations maximizes total productivity?

Modeling Problem 24. An online DVD rental company would like to measure the efficiency of five of its distribution centers in New England: one each in Maine, Vermont, and New Hampshire, and two in Massachusetts. This is done via “data envelopment analysis,” which compares ratios of weighted sums of outputs for a center to weighted sums of inputs for the same center. The efficiency of any distribution center is defined to be the maximum ratio over all choices of (nonnegative) weights for which none of the corresponding five ratios (one for each center) exceeds 1. Table 1.19 shows the average daily inputs and outputs of each distribution center.

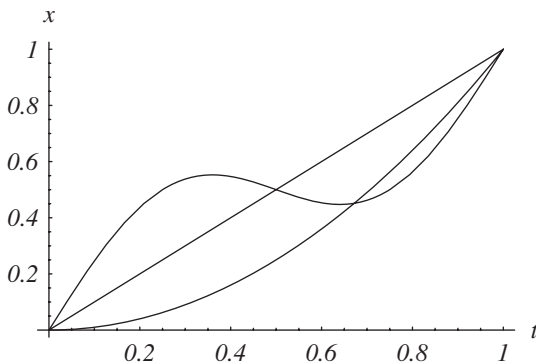
Table 1.20. *Puzzle manufacturing data*

Puzzle Type	1	2	3	4	5
Wood preparation	4	3	2	3	3
Painting	7	8	6	9	7
Cutting	5	5	6	4	4
Sanding	3	2	3	1	3

Write and solve five optimization models to find the efficiency of each of the five distribution centers.

Modeling Problem 25. *A wooden puzzle manufacturer uses four main processes to produce its finished puzzles: wood preparation, painting, cutting, and sanding. The manufacturer uses the same four processing stations to make five different types of puzzles, whose respective times for each stage of processing are given in Table 1.20. Once a puzzle has entered the sequence of processing stations, it cannot be “passed” by a puzzle which entered the sequence later, and puzzles can wait between stations.*

Write and solve an optimization model to find the sequence of processing which minimizes the total time to complete exactly one of each type of puzzle.



Chapter 2

Impractical Optimization

In a typical calculus class, students learn how to optimize a function (i.e., find its maximum or minimum) by finding its “stationary points,” so called because they are where the derivative (or each partial derivative in the case of multivariable functions) equals zero. To remind the reader of the details involved in this procedure, we consider now the following basic optimization problem.

Example: Monthly Payment Function

Most readers are likely at some point in their lives to obtain financing from a bank or a car dealership. If a borrower is considering financing a \$30,000 car at a given interest rate (say 5% compounded annually), they might be interested in minimizing the size of the monthly payments $f(x)$ they would incur for a term of x months:

$$f(x) = \frac{30(1 + 0.05)^{\frac{x}{12}}}{x}, \quad (2.1)$$

where the monthly payment function here has units of thousands of dollars. A naive borrower might assume that the longer the term x the better, since then the total borrowed amount (principal and interest) is divided into more pieces. However, a longer term also means more interest accrued over the term of the loan and thus a larger total borrowed amount. For the same reason, a shorter term does not necessarily mean a lower monthly payment, so there is some work to do to find the minimizer in this case. Since there is only one variable, we could try to find a minimizer by inspecting the graph of the monthly payment function. It is easy to see from Figure 2.1 that there is a minimizing term length x for this monthly payment function, at least among the terms x displayed in Figure 2.1. If there is no lower monthly payment associated with a term outside the frame in Figure 2.1, we call the minimizing term length we see in Figure 2.1 a “global” minimizer.

A **global minimizer** gives the lowest output value among all inputs, whereas a **local minimizer** gives the lowest output value among all *nearby* inputs.

Since Figure 2.1 (or any other graph for that matter) shows only a limited frame of terms, we cannot be certain that we are looking at a global minimizer. However, we can safely

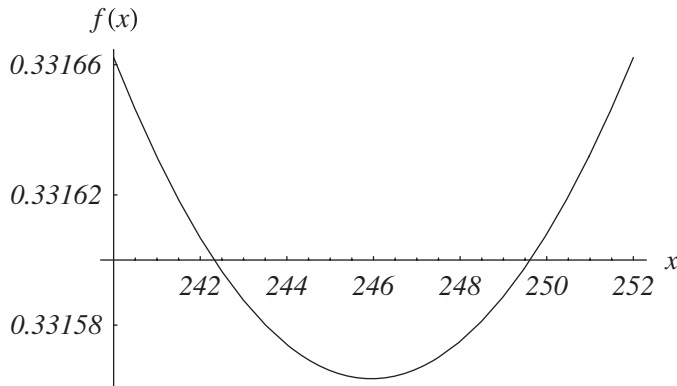


Figure 2.1. Graph of the monthly payment function with fixed interest rate

conclude that the low spot corresponds to a term which is a local minimizer of the monthly payment function with fixed interest rate. As we will see in Problem 2.1, this local minimizer is a good candidate to solve our problem, even if it isn't a global minimizer of the monthly payment function. By zooming in around this point, we could get a better and better approximation of its exact value, but calculus gives us the tools to find the exact value easily.

Problem 2.1. Complete the following:

- Explain why a global minimizer is automatically a local minimizer.
- By coming up with inputs that drive its value toward negative infinity, show that the monthly payment function (2.1) has no global minimizer.
- You should find that the inputs you come up with don't make sense in terms of the monthly payment problem. What restriction on x would make sense in this case?
- Find the stationary point of the monthly payment function (2.1) and consequently the exact value of the minimizing term length x appearing in Figure 2.1.

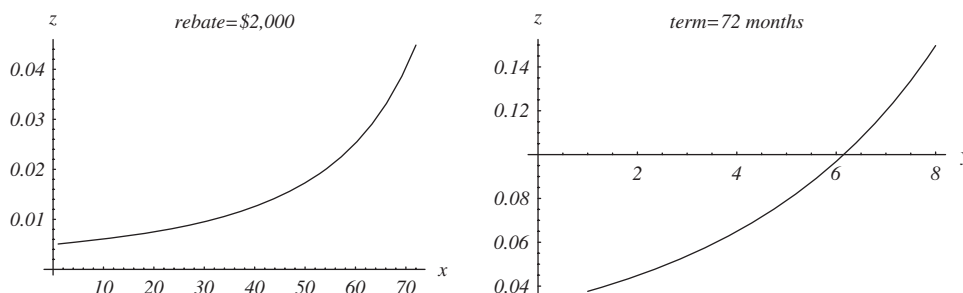
The approach from calculus of optimizing a function by solving for stationary points via algebraic manipulation of the formulas works well for the monthly payment function in this example. However, this approach is rarely used in practice because it almost never works. To understand why this approach fails so often, we now explore a few fundamental practical limitations that were not likely covered in calculus class.

2.1 Algebra Limitations

In order to optimize as in calculus class (and as reviewed above), we need to be able to solve stationary point equations. Unfortunately, due to the limitations of algebraic manipulation, this is rarely possible in practice. We illustrate this limitation by attempting to minimize a slightly modified monthly payment function.

Table 2.1. *Interest rate packages*

term\rebate	\$1,000	\$2,000	\$3,000	\$4,000	\$5,000	\$6,000
36 months	0.94%	1.1%	1.3%	1.6%	2.0%	2.4%
48 months	1.4%	1.6%	1.9%	2.3%	2.8%	3.5%
60 months	2.1%	2.5%	3.0%	3.7%	4.4%	5.5%
72 months	3.8%	4.5%	5.4%	6.5%	7.9%	9.7%

**Figure 2.2.** *Two slices of the graph of the interest rate (2.2)***Example: Variable Interest Rate**

Most car dealerships offer incentives in the form of a variety of packaged rebates and interest rates. A typical incentive package includes both a rebate and an interest rate, with higher rebates linked to higher interest rates. In addition, for any fixed rebate value, the interest rate offered might be higher for longer terms since longer-term loans are riskier. For instance, you might be offered a range of financing packages as shown in Table 2.1.

We would like to find a general relationship between the interest rate z and the corresponding term x and rebate value y (say, in thousands of dollars) that fits this data. There are many different such relationships, but for this discussion we'll use a particular one that gives z explicitly as a function of x and y and also favors rebate values y that avoid 30:

$$z = \frac{10^9}{(108 - x)^2(30 - y)^5}. \quad (2.2)$$

Two slices of the graph of this function, one with $y = 2$ and one with $x = 72$, are shown in Figure 2.2.

We are now in the position to modify our monthly payment function (2.1) to incorporate rebates as well as the variable interest rate described by the interest rate function (2.2):

$$f(x, y) = \frac{(30 - y) \left(1 + \frac{10^9}{(108 - x)^2(30 - y)^5} \right)^{\frac{x}{12}}}{x}. \quad (2.3)$$

Just as in the case of the original monthly payment function (2.1), we want to minimize the function (2.3), only now we need to find both a term x and a rebate y that together generate the minimal monthly payment. Using what worked for the original monthly payment function, we'll try to find the stationary points.

Problem 2.2. Complete the following:

- Identify all the appropriate interest rates given in Table 2.1 with corresponding coordinates on the graphs in Figure 2.2.
- Explain how the interest rate function (2.2) favors rebate values y that avoid 30. Why is this desirable?
- Explain the first factor $(30 - y)$ in the monthly payment function (2.3) with variable interest rate.
- Try to find the stationary points of the monthly payment function (2.3). You should be able to compute the partial derivatives in this case but be unable to solve the resulting pair of equations for a stationary point.

Though a typical math student's early experience does not suggest it, the vast majority of equations cannot be solved with algebraic manipulations. There are basically two things that can go wrong in this situation:

- *Unsolvable:* The equation has no solution among the variables of interest.
- *Implicit:* The equation has a solution among the variables of interest, but algebraic manipulations can't expose it explicitly.

One example of an unsolvable equation is $x^2 = -1$, which evidently has no solutions among real numbers x . Of course, if *complex* numbers are allowed, this equation can be solved for $x = \pm i$. If stationary points for an optimization problem are defined by an unsolvable equation, we usually conclude that there are no stationary points (or optimizers) since complex numbers typically do not make sense in optimization problems.

Examples of implicit equations are actually quite easy to find once you start thinking about them. For instance, $\cos(x) - x = 0$ cannot be solved for x using algebra (try it), but it definitely has a solution, as we can see from the graph in Figure 2.3. The worst scenario from our current point of view would be to have unsolvable implicit equations defining our stationary points. As we will see later, the equations defining the stationary points in Problem 2.2 are implicit equations that do have a solution among the variables of interest,

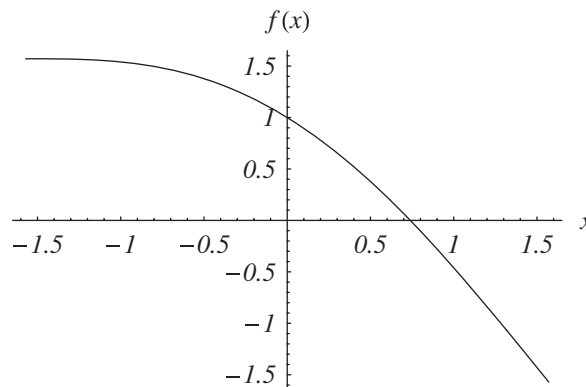


Figure 2.3. Graph of $f(x) = \cos(x) - x$

and it turns out that this is by far the most typical situation in practice. Fortunately, there is something we can do in this situation, but we'll need more than the approach from calculus. In Chapter 3, we will introduce what is actually done in practical optimization.

Another optimization rule from calculus that isn't used the way it is typically presented is the Lagrange multiplier rule. You might recall that this rule applied when the variables in the optimization problem were constrained. In the monthly payment example with variable interest rates, the relationship (2.2) defining interest rate z as a function of term x and rebate y can be thought of as such a constraint on the monthly payment function of three variables:

$$f(x, y, z) = \frac{(30 - y)(1 + z)^{\frac{x}{12}}}{x}. \quad (2.4)$$

Students who learn about constrained optimization in calculus are typically left with the sense that solving these problems involves somehow solving the system of equations that results from applying the Lagrange multiplier rule. Since these equations suffer the same problems as stationary point equations, this approach is also very rarely used in practice. In fact, systems of linear equations are the only systems whose explicit analytic solution we can reliably find in general. As a result, many of the practical methods for optimization involve some form of “linearization” (i.e., approximating nonlinear equations with linear ones). Chapter 6 introduces what is typically done in practice to solve constrained optimization problems, and Chapter 7 includes a discussion of how the Lagrange multiplier rule is actually used in practice.

2.2 Size Limitations

Even when the relevant equations can be solved, it is not practical to do so when there are more than a few of them. Since there is one partial derivative for each variable, the number of simultaneous equations defining the stationary points is the same as the number of variables in the optimization problem. Thus, only optimization problems with relatively few variables can be solved through a reasonable amount of direct algebraic manipulation; this explains why it is typical to see only optimization problems of this type in calculus class. However, many realistic problems involve more than just a few variables (for example, an airline's routing plan can involve hundreds of flights), so their stationary points are defined by more than just a few simultaneous equations. Even if these simultaneous equations can be solved via algebraic manipulations, a computer is needed to do it without an unreasonable effort. In addition to problems like airline routing that naturally come with many variables, *large-scale* optimization problems result from integral optimization problems through a process called “discretization.”

2.2.1 Integral Optimization

One kind of optimization problem that appears relatively often in practice but rarely in a calculus class is the optimization of an integral. The following is one famous example of an integral optimization problem studied by Newton in 1687 (cf. [2]).

Example: Newton's Aerodynamics Problem

Newton was interested in comparing the resistances to downward motion of different surfaces of revolution. Using the t - x plane, a surface is generated by revolving (about the

x -axis) the graph of a function $x(t)$ passing through the points $(0,0)$ and $(1,1)$. Figure 2.4 shows graphs of several such surface-generating functions $x(t)$. Newton found that the resistance of such a surface of revolution was measured by (a constant multiple of) the integral

$$\int_0^1 \frac{x(t)x'(t)^3}{1+x'(t)^2} dt \quad (2.5)$$

in terms of the corresponding surface-generating function $x(t)$.

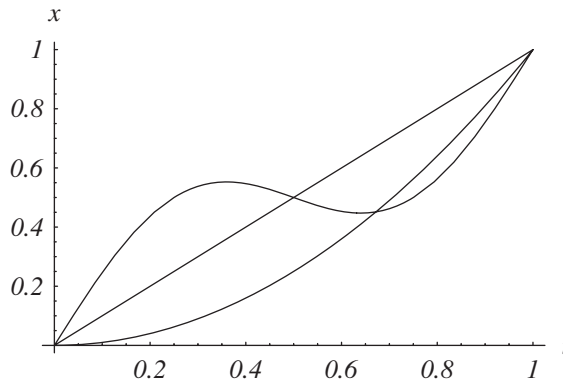


Figure 2.4. Graphs of a few surface-generating $x(t)$

Problem 2.3. Complete the following:

- Sketch the three surfaces of revolution corresponding to the three surface-generating functions shown in Figure 2.4.
- Give the equations $x(t) = ?$ for two surface-generating functions, and compute the value of Newton's integral (2.5) for each.

Newton sought a surface with minimal resistance, which evidently minimizes the integral (2.5) over all choices of surface-generating functions $x(t)$. This kind of optimization problem is very different from anything covered in calculus class, since here the variables are the surface-generating functions $x(t)$. Integral optimization problems like this therefore involve choosing a continuum of variable values $x(t)$ corresponding to each value of t , instead of some finite number of variable values x , or (x, y) , or (x, y, z) , etc., as is the case in any calculus class. In fact, the usual calculus does not work with function variables $x(t)$, and the fields of “calculus of variations” and (more recently) “optimal control” were developed specifically to deal with integral optimization problems. A discussion of these fields is beyond the scope of this book; however, the process of discretization can be applied to integral optimization problems to convert them to a format which this book does cover.

2.2.2 Discretization

The idea behind discretization is simple: Instead of choosing a continuum of variable values $x(t)$ corresponding to each value of t , pick a finite number of special values of t at

which to choose a corresponding variable value $x(t)$. This is quite similar to the integral approximation methods (left-endpoint rectangles, right-endpoint rectangles, midpoint rectangles, trapezoids, etc.) covered in most calculus classes. The reason this process is called “discretization” is because a problem having a continuum of variables is converted to one with a finite number of variables. To illustrate how this works, we will again consider Newton’s integral (2.5) and choose the special t -values of 0, $1/2$, and 1. The fancy name for what we will use is a *linear spline interpolation*, which in this case means restricting our attention to surface-generating functions $x(t)$ that are made of linear pieces (a.k.a. linear “splines”).

Example: Newton’s Aerodynamics Problem Revisited

From the statement of the problem, we know the value of $x(t)$ at both ends: $x(0) = 0$ and $x(1) = 1$. The only variable value that remains to be chosen is thus $x(1/2)$. Graphically, this variable gives the height above $t = 1/2$ of the corner connecting two line segments as in Figure 2.5.

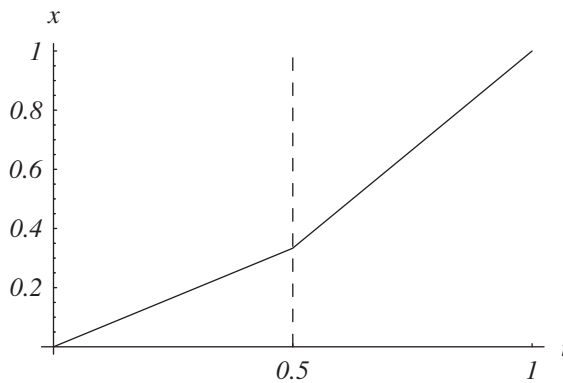


Figure 2.5. Potential solution to the discretization problem

There are many ways to approximate the integral (2.5) that would be valid for this discretization. One such approximation echoes the right-endpoint integral approximation method from calculus class: We use the value of $x(1/2)$ to replace $x(t)$ over the t -interval $[0, 1/2]$ and the value of $x(1) = 1$ to replace $x(t)$ over the t -interval $(1/2, 1]$. However, since Newton’s integral (2.5) also depends on the function $x'(t)$, we need to replace that as well with its value at the two right endpoints. We can do this simply with the following difference quotient approximations:

$$\begin{aligned} x'(1/2) &\leftarrow \frac{x(1/2) - x(0)}{1/2} = 2x(1/2), \\ x'(1) &\leftarrow \frac{x(1) - x(1/2)}{1/2} = 2(1 - x(1/2)). \end{aligned} \tag{2.6}$$

By labeling $x(1/2)$ simply as the variable x , we can then discretize Newton's integral (2.5) to get the following function of x :

$$\text{Newton}(x) = x \left(\frac{\left(\frac{x-0}{1/2}\right)^3}{1 + \left(\frac{x-0}{1/2}\right)^2} \right) \frac{1}{2} + 1 \left(\frac{\left(\frac{1-x}{1/2}\right)^3}{1 + \left(\frac{1-x}{1/2}\right)^2} \right) \frac{1}{2}. \quad (2.7)$$

Note that the sum essentially replaces the integral and that the final factor of $1/2$ in each term of the sum replaces the dt . For a discretization having a different space Δt between t -values, that number Δt plays the role of $1/2$ in this example.

Problem 2.4. Complete the following:

- (a) Explain how the difference quotient approximations (2.6) are consistent with Figure 2.5.
- (b) Without solving the discretization problem of minimizing $\text{Newton}(x)$ defined in (2.7) over all values of x , explain why its solution can't be the solution to the original integral optimization problem whose graph appears in Figure 2.6.
- (c) Give the discretization problem that results from using left endpoints instead of right endpoints.
- (d) Give the stationary point equation for the corresponding "left-endpoint" discretization function.

Later, we'll compute a solution to the discretization problem, but even without that computation, we know that the solution does not look exactly like the solution to the original integral optimization problem whose graph appears in Figure 2.6. In fact, the solution to any discretization problem is generally only an approximation to the solution to the original integral optimization problem. To make a better approximation, we can do the same thing we do in calculus to improve a rectangle approximation to an integral: increase the number of sample points. In our case, this means using more t -values at which to find corresponding x -values.

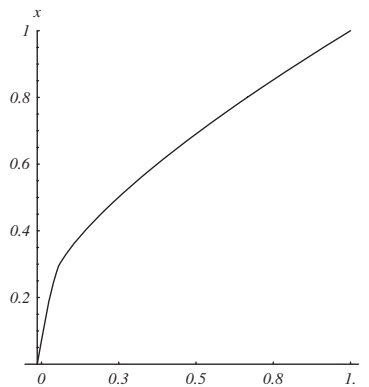


Figure 2.6. Solution to Newton's aerodynamics problem

Exercise 2.2.1. *Complete the following:*

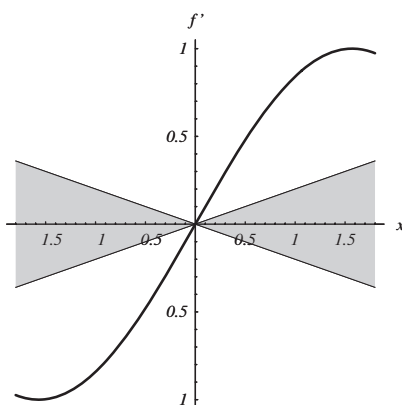
- (a) *Consider any integral optimization problem over the t -interval $[0, 1]$ with $x(0) = 0$ and $x(1) = 1$. How many variables are there in a discretization problem using the t -values of 0, $1/4$, $1/2$, $3/4$, and 1?*
- (b) *Explain why the solution to this new discretization problem will definitely be at least as good an approximation to the solution of the original integral optimization problem as the discretization solution from the sampling at the t -values of 0, $1/2$, and 1.*

Evidently, improvement in the approximation comes at the price of discretization problems with greater numbers of variables. Depending on the circumstances, one might need to sample quite a few t -values to be assured of a good enough approximation, which means discretization problems with quite a few variables. Large-scale optimization problems like these not only make the approaches to optimization taught in calculus impractical but also pose challenges for the practical optimization approaches that we present later in this book. A more thorough discussion of the special issues involved in large-scale optimization is beyond the scope of this book, but the reader can explore this topic further in some of the resources found in the Bibliography.

2.3 Conclusions from Chapter 2

In this chapter, we learned about two major reasons why the standard optimization approach from calculus is rarely used in practice:

- Algebra limitations: The stationary point equations often cannot be solved explicitly.
- Size limitations: Since there is a stationary point equation for each variable, only optimization problems with relatively few variables are practical.



Chapter 3

Basic Practical Optimization

At first glance, the field of optimization can seem imposing. It includes a vast array of different methods, each identified by confusing acronyms and described with special terminology. However, every one of these methods boils down to the same basic procedure for finding optimizers of functions:

Repeatedly guess an optimizer until satisfied or frustrated.

This is really so simple that no one has any right to be intimidated; after all, anyone can repeatedly guess an optimizer. What makes this simple procedure interesting, and consequently lends optimization its many flavors, is how the guessing is done. Each different optimization method basically involves a different rule for making educated guesses of optimizers.

In this text, we will focus on a class of methods that are based on familiar calculus results, so much of optimization is already well within the grasp of anyone who has had multivariable calculus. The basic procedure for such “derivative-based” optimization is usually stated more formally in terms something like the following:

Basic Optimization Method

1. **Initialize** the current guess.
2. **Update**: Determine a new guess from the current guess.
3. **Stop or repeat**: If the stopping criterion is not satisfied and progress is being made, use the new guess as the current guess and repeat the process from step 2.

It is important to note that derivative-based methods are not universally useful. In particular, they rely on derivative information being obtained with reasonable effort, which might not be the case if there is no accurate explicit formula for the objective function. Therefore, any reader who wishes to have a complete arsenal of tools for doing optimization will need to study methods beyond the focus of this text. Some of these are listed in Chapter 9, where a Research Project is modeled and described that will allow the reader to study one of these other methods in depth. Readers should begin the Research Project

as soon as possible (right now would be fine) and not wait until the other chapters of this book have been completed. Instructors can facilitate this by assigning the subsections of the Research Project throughout the term (the subsections are ordered intentionally to allow this). Ideally, groups reading this text together can tackle different research topics and share their results with each other so that the entire group gets as broad an exposure to the field as possible. Instructors can facilitate this by arranging some form of student presentations of their Research Projects.

3.1 Replacement Functions

It has already been argued in Chapter 2 that the optimization strategies from calculus cannot be usefully applied to most real optimization problems. One basic approach that *is* used in practice is to replace the objective function with a simpler function that looks like the objective function near the current guess and whose stationary points are easily found. A stationary point of the simpler function becomes the new guess, and if it is close enough to a stationary point of the objective function, the procedure is a success. If not, then a different (presumably better) replacement function is used that looks like the objective function near the new guess. After enough iterations, one either achieves success or declares failure. This method fits the pattern described previously: “Repeatedly guess an optimizer until satisfied or frustrated,” where now the repeated guessing is guided by the replacement functions.

The key to this method then is the replacement functions. They need to be simple enough so that their stationary points are easy to find but complicated enough to capture the relevant features of the original functions whose stationary points are sought. The simplest replacement functions that satisfy these requirements are quadratic functions.

Aside: Quadratic Functions

For this discussion, the letter Q will be used to denote a generic quadratic function. Also, for simplicity, the focus will be on a function of two variables x and y since these are the simplest quadratic functions that exhibit all of the important properties of quadratic functions of any number of variables. Moreover, the properties of quadratic functions of more variables can be understood via obvious extensions of the results for two-variable quadratics. Any quadratic function of two variables x and y can be written as

$$Q(x, y) = \frac{a}{2}x^2 + bxy + \frac{c}{2}y^2 + dx + gy + h$$

for some constants $a, b, c, d, g,$ and h . It's easy to find the stationary points of Q because it's easy to compute the gradient (vector) of Q :

$$\nabla Q(x, y) = \begin{bmatrix} ax + by + d \\ bx + cy + g \end{bmatrix}.$$

The stationary points of Q are then simply the solutions (x, y) to the two equations obtained by setting the $\nabla Q(x, y)$ equal to the zero-vector:

$$\begin{aligned} ax + by + d &= 0, \\ bx + cy + g &= 0. \end{aligned} \tag{3.1}$$

It is relatively easy to solve a pair of linear equations like this for values of x and y .

Exercise 3.1.1. *Complete the following:*

- (a) *Solve the pair of equations (3.1).*
- (b) *Choose values of the constants for which there are multiple solutions.*
- (c) *Choose values of the constants for which there is no solution.*

You should notice in Exercise 3.1.1 that either the trio of constants a , b , and c leads to equations that can be uniquely solved for all choices of d and g , or the trio of constants a , b , and c gives multiple solutions for certain values of d and g and no solutions for other values of d and g . In other words, the presence of degeneracy in the form of multiple or no solutions is dependent entirely on the trio of constants a , b , and c , whereas the constants d and g merely determine the type of degeneracy.

Aside: Matrix Notation and Multiplication

A matrix is a grouping of numbers into rows and columns. Vectors can be thought of as matrices with only one column, and matrices in general can be thought of as a group of column vectors. Matrices come in all different sizes, but for now the focus will be on matrices A with two rows and two columns:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

where a , b , c , and d are any numbers. For any such matrix A , its multiplication against a vector whose components are x and y is defined by

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}. \quad (3.2)$$

Notice that the basic rule for generating each row of the result is to consecutively multiply the corresponding row elements of A against the column elements of the vector, adding the results as you go (this same rule applies for vectors having any number of components as long as A has that same number of columns). You might recognize this procedure as just a pair of dot products generated by each row (vector) of A against the vector whose components are x and y .

You can also multiply a vector against a matrix against another vector, which we call a *vector-matrix-vector multiplication*. For instance, the vector-matrix-vector multiplication

$$\begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix}$$

represents the dot product of the first vector $\begin{bmatrix} u & v \end{bmatrix}$ with the vector that results from the matrix multiplication (3.2) and thus is given by the formula

$$\begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix} = u(ax + by) + v(cx + dy).$$

Notice in particular that the result of such a vector-matrix-vector multiplication is a number.

Exercise 3.1.2. Use matrix notation to express

- (a) a general quadratic $Q(x, y)$ of two variables in terms of the vector $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$,
- (b) the gradient of Q in terms of the vector $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$.

Recall the criteria for choosing replacement functions: They need to be simple enough so that their stationary points are easy to find but complicated enough to capture the relevant features of the actual functions whose stationary points are sought. Evidently, quadratics satisfy the first criterion, so it remains to be seen only if they also satisfy the second criterion. Certainly, any old quadratic is not going to satisfactorily capture the relevant features of a particular function f to be optimized. One choice that satisfies the second criterion is the Taylor quadratic associated with f at the base point (\bar{x}, \bar{y}) equal to the current guess:

$$TQ(x, y) = f(\bar{x}, \bar{y}) + \nabla f(\bar{x}, \bar{y}) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix} \nabla^2 f(\bar{x}, \bar{y}) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \quad (3.3)$$

in terms of the *second-derivative matrix*

$$\nabla^2 f(\bar{x}, \bar{y}) = \begin{bmatrix} f_{xx}(\bar{x}, \bar{y}) & f_{xy}(\bar{x}, \bar{y}) \\ f_{xy}(\bar{x}, \bar{y}) & f_{yy}(\bar{x}, \bar{y}) \end{bmatrix}.$$

Note that the superscript “2” on the ∇ is meant to indicate that this matrix is a second-derivative analogue for functions of two variables, the subscripts on the f ’s denote partial differentiation with respect to the corresponding variables, and all of these partial derivatives are assumed to be continuous functions. (Other texts may refer to the second-derivative matrix as the *Hessian matrix*.)

If you look carefully at the formula for the Taylor quadratic (3.3), you should see that there are two different kinds of (x, y) -pairs present: the pair (x, y) represents true variables (x, y) , and the pair (\bar{x}, \bar{y}) represents the fixed base point. To compute the Taylor quadratic in any situation, these different kinds of pairs are treated very differently; the variables (x, y) remain as variables, while the elements (\bar{x}, \bar{y}) are replaced by the base point of interest. See Figure 3.1 for a visual representation of a single-variable example using the stationary points of Taylor quadratics to improve a first guess and then to improve the resulting second guess. To see how the Taylor quadratic works out in a particular example, we consider the following discretization problem.

Example: A Two-Variable Discretization Problem

We construct the discretization at the t -values 0, 1/3, 2/3, and 1 of the integral optimization problem

$$\min \int_0^1 x(t)^2 x'(t) dt \text{ with } x(0) = 0 \text{ and } x(1) = 1.$$

In this case, the discretization has two variables $x(1/3)$ and $x(2/3)$ since the values $x(0)$ and $x(1)$ are specified. For simplicity, we’ll use the variables x and y to denote $x(1/3)$

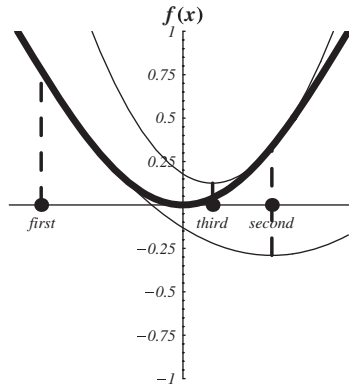


Figure 3.1. An objective function (in **bold**) with two consecutive Taylor quadratics and three consecutive guesses

and $x(2/3)$, respectively. Using right-endpoint replacements for the x -values and the appropriate difference quotient replacements for $x'(1/3)$, $x'(2/3)$, and $x'(1)$, we discretize the integral to get the function

$$f(x, y) = \left[x^2 \left(\frac{x-0}{1/3} \right) + y^2 \left(\frac{y-x}{1/3} \right) + 1 \left(\frac{1-y}{1/3} \right) \right] \frac{1}{3} \\ = x^3 - x y^2 + y^3 - y + 1. \quad (3.4)$$

To compute the Taylor quadratic associated with this function, we need its first and second partial derivatives, which we can easily find in this case:

$$\begin{aligned} f_x(x, y) &= 3x^2 - y^2, \\ f_y(x, y) &= -2xy + 3y^2 - 1, \\ f_{xy}(x, y) &= -2y, \\ f_{xx}(x, y) &= 6x, \\ f_{yy}(x, y) &= -2x + 6y. \end{aligned} \quad (3.5)$$

If the base point happens to be $(1, 2)$, say, then the partial derivatives evaluated at this base point are

$$\begin{aligned} f_x(1, 2) &= -1, \\ f_y(1, 2) &= 7, \\ f_{xy}(1, 2) &= -4, \\ f_{xx}(1, 2) &= 6, \\ f_{yy}(1, 2) &= 10, \end{aligned}$$

so the Taylor quadratic is

$$TQ(x, y) = 4 - 1(x - 1) + 7(y - 2) + 3(x - 1)^2 - 4(x - 1)(y - 2) + 5(y - 2)^2.$$

This is certainly a quadratic function, which you can see by multiplying out all the terms and collecting elements. In fact, the general Taylor quadratic (3.3) is always a quadratic function.

Exercise 3.1.3. *Verify that the general Taylor quadratic $TQ(x, y)$ defined by (3.3) is a quadratic function*

$$TQ(x, y) = \frac{a}{2}x^2 + bx + \frac{c}{2}y^2 + dx + gy + h$$

with

$$\begin{aligned} a &= f_{xx}(\bar{x}, \bar{y}), \\ b &= f_{xy}(\bar{x}, \bar{y}), \\ c &= f_{yy}(\bar{x}, \bar{y}), \\ d &= f_x(\bar{x}, \bar{y}) - f_{xx}(\bar{x}, \bar{y})\bar{x} - f_{xy}(\bar{x}, \bar{y})\bar{y}, \\ g &= f_y(\bar{x}, \bar{y}) - f_{yy}(\bar{x}, \bar{y})\bar{y} - f_{xy}(\bar{x}, \bar{y})\bar{x}. \end{aligned}$$

What is the value of the constant h in this case?

The Taylor quadratic satisfies the second criterion for choosing replacement functions because its first and second derivatives at the base point agree with f 's first and second derivatives there. In fact, for each point (x, y) the value of the function $f(x, y)$ is the same as the value of a different quadratic function that looks almost like the Taylor quadratic: There is some point (ξ_x, ξ_y) between (x, y) and the base point (\bar{x}, \bar{y}) such that

$$f(x, y) = f(\bar{x}, \bar{y}) + \nabla f(\bar{x}, \bar{y}) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix} \nabla^2 f(\xi_x, \xi_y) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix}. \quad (3.6)$$

The quadratic function in (3.6) is sometimes called the *remainder form* of the Taylor quadratic, and the only difference between it and the usual Taylor quadratic $TQ(x, y)$ defined in (3.3) is that now the second-derivative matrix is constructed at (ξ_x, ξ_y) . For the two-variable discretization function (3.4), we can use the formulas (3.5) for the partial derivatives to give the remainder form of the Taylor quadratic at $(1, 2)$:

$$\begin{aligned} &4 - 1(x - 1) + 7(y - 2) + 3\xi_x(x - 1)^2 - 2\xi_y(x - 1)(y - 2) \\ &+ \frac{-2\xi_x + 6\xi_y}{2}(y - 2)^2. \end{aligned}$$

The remainder form is not as practically useful as the usual Taylor quadratic $TQ(x, y)$ defined in (3.3) since the remainder form involves the mystery point (ξ_x, ξ_y) , and this mystery point generally changes with the variables (x, y) (that's why the subscripts are used on the ξ 's). However, as will soon become apparent, the remainder form is really helpful for generating mathematical rules that can then be put to practical use.

Newton's optimization method is a famous optimization method whose essence is to solve the equations (3.1) where the constants are given in Exercise 3.1.3, which in matrix form is just

$$\nabla^2 f(\bar{x}, \bar{y}) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} + \nabla f(\bar{x}, \bar{y}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad (3.7)$$

Exercise 3.1.4. Give the matrix equation (3.7) which corresponds to the two-variable discretization function (3.4) with $(\bar{x}, \bar{y}) = (1, 2)$.

The pair of equations encoded in (3.7) could be solved directly using basic algebra; however, this process would be relatively complicated to carry out. When there are more variables, there are more equations encoded by the corresponding matrix equation, and the complications of solving these systems of equations are even greater. To understand what is actually done in practice, we will explore an elegant reformulation of the matrix equation (3.7) using the concept of inversion for matrices.

Aside: Matrix Inversion

Solving an equation like $ax = b$ for the variable x is easy; simply multiply both sides by the inverse of the number a to get the solution $x = a^{-1}b$. What is needed now is a generalization of this to matrix equations $A\mathbf{x} = \mathbf{b}$, where \mathbf{x} and \mathbf{b} are vectors with two components and A is a matrix with two rows and two columns. The idea is to come up with another matrix A^{-1} , the inverse of A , which multiplies both sides of the matrix equation to give the solution $\mathbf{x} = A^{-1}\mathbf{b}$.

Not all matrices A have inverses (just as the number 0 has no inverse), and this happens precisely when the linear equations encoded by $A\mathbf{x} = \mathbf{b}$, for any right-side vector \mathbf{b} , have either no solution or infinitely many solutions. When the encoded linear equations can be solved uniquely for every \mathbf{b} , the matrix encoding the equations has an inverse, and vice versa. If a matrix of the form

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

has an inverse, it is given by

$$A^{-1} = \begin{bmatrix} \frac{d}{ad-bc} & \frac{-b}{ad-bc} \\ \frac{-c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix}. \quad (3.8)$$

The lack of an inverse for A then corresponds exactly to the situation when $ad - bc = 0$, in which case the formula (3.8) does not produce a matrix with finite-valued entries.

Exercise 3.1.5. Choose any particular values of a , b , c , and d for which the formula (3.8) does not produce finite-valued entries, and try to directly solve the pair of equations corresponding to the matrix equation $A\mathbf{x} = \mathbf{b}$ for any choice of vector \mathbf{b} .

The update in Newton's optimization method is defined by solving the matrix equation (3.7) for the new guess $(x_{\text{new}}, y_{\text{new}}) = (x, y)$ in terms of the current guess $(x_{\text{old}}, y_{\text{old}}) = (\bar{x}, \bar{y})$, which can be expressed compactly with matrix notation as

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} + \nabla^2 f(x_{\text{old}}, y_{\text{old}})^{-1} [-\nabla f(x_{\text{old}}, y_{\text{old}})]. \quad (3.9)$$

Graphically, this update formula says that the new guess vector is obtained from the old by adding the vector

$$\mathbf{d} = \nabla^2 f(x_{\text{old}}, y_{\text{old}})^{-1} [-\nabla f(x_{\text{old}}, y_{\text{old}})]$$

that results from multiplying the inverse of the second-derivative matrix against the negative gradient vector. A way to describe this update formula that fits with the terminology we'll use later is that the new guess vector is obtained from the old by taking a full-step along the step-direction \mathbf{d} .

Newton's optimization method is thus the version of the basic optimization method described at the beginning of this chapter where the update formula (3.9) is used in step 2.

Newton's optimization method

1. **Initialize** the current guess $\begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix}$.
2. **Update:**

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} + \nabla^2 f(x_{\text{old}}, y_{\text{old}})^{-1} [-\nabla f(x_{\text{old}}, y_{\text{old}})].$$

3. **Stop or repeat.**

Note that there is an implicit assumption here about the existence of an inverse for the second-derivative matrix associated with the current guess. This is a reasonable assumption because as long as the Taylor quadratic at the current guess actually has a stationary point to use for the next guess, there will be a corresponding inverse matrix (see the discussion immediately following Exercise 3.1.1 as well as the Aside on Matrix Inversion). Some other scheme must be used in the case when the Taylor quadratic at the current guess does not have a stationary point.

Problem 3.1. Complete the following:

- (a) Write the update (3.9) for the case of the two-variable discretization function (3.4) with $(x_{\text{old}}, y_{\text{old}}) = (1, 2)$.
- (b) What does the update (3.9) look like when the function f has only one variable x ?
- (c) Give an example of a Taylor quadratic that does not have a stationary point.

3.1.1 When to Stop

The decision to stop any optimization method is typically guided by some criterion that gives a reasonable amount of confidence in the answer, at least up to some acceptable level of accuracy. In practice, such *stopping criteria* can be a little tricky since computers represent numbers up to a fixed number of decimal points. This means that two different numbers will look the same to a computer as long as their differences occur beyond the decimal places which the computer stores. Moreover, computations done using computers often introduce small errors (which can develop into larger errors), so the guesses generated by a computer are always somewhat suspect. Finally, even when the desired level of accuracy is low enough that it can be reliably obtained by the computer, it might take an intolerable amount of time to obtain. Thus, Newton's optimization method will typically be terminated at a nearly stationary point whose gradient is close to the zero-vector. We will see more later about stopping criteria, but for now you should decide on your own in the following exercise.

Implementation 1. *Implement Newton's optimization method for (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3).*

3.1.2 Why It's Called Newton's Optimization Method

Recall from Problem 2.2 that the difficulty was solving exactly where the gradient was zero. Interestingly, repeatedly using Taylor quadratics as replacement functions turns out to be the same thing as applying Newton's method to the gradient-equals-zero equations. Readers should be familiar with Newton's method for solving the equation $F(x) = 0$ for functions F of one variable. Basically, the idea is to replace F with its tangent line approximation at the current guess x_{old} and to let the next guess x_{new} equal the point where the tangent line equals zero:

$$x_{\text{new}} = x_{\text{old}} - \frac{F(x_{\text{old}})}{F'(x_{\text{old}})} = x_{\text{old}} - F'(x_{\text{old}})^{-1} F(x_{\text{old}}). \quad (3.10)$$

Newton's method for finding roots (i.e., the inputs that yield zero output) can be used to find stationary points of any function f since they are roots of the derivative. In this case, then, the function F is replaced with the derivative f' to get

$$x_{\text{new}} = x_{\text{old}} - f''(x_{\text{old}})^{-1} f'(x_{\text{old}}) \quad (3.11)$$

as a rule for determining next guesses for stationary points from current guesses.

Exercise 3.1.6. *Complete the following:*

- (a) *Verify the formula (3.10) for Newton's method by finding the root of the tangent line.*
- (b) *Compare the formula (3.11) for finding stationary points of $f(x)$ to the update (3.9) in Newton's optimization method applied to functions of one variable as in Problem 3.1.*

For functions of more than one variable, there is an analogous Newton's method for solving the vector equation

$$\nabla f(x, y) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Of course, this vector equation represents the pair of equations $f_x(x, y) = 0$ and $f_y(x, y) = 0$, where evidently each equation is of the form $F(x, y) = 0$ for a different function F of two variables. Just as in the case of Newton's method for finding roots, Newton's method for solving the vector equation replaces each such function F with its Taylor linear approximation and solves the resulting pair of linear equations. Carrying this out relies on the Taylor linear approximation associated with any function F of two variables about the current guess (\bar{x}, \bar{y}) (just remove the second-derivative terms from the Taylor quadratic):

$$F(\bar{x}, \bar{y}) + F_x(\bar{x}, \bar{y})(x - \bar{x}) + F_y(\bar{x}, \bar{y})(y - \bar{y}). \quad (3.12)$$

Applying this separately to each of the equations $f_x(x, y) = 0$ and $f_y(x, y) = 0$ and setting the results equal to zero yields the pair of equations

$$\begin{aligned} f_x(\bar{x}, \bar{y}) + f_{xx}(\bar{x}, \bar{y})(x - \bar{x}) + f_{xy}(\bar{x}, \bar{y})(y - \bar{y}) &= 0, \\ f_y(\bar{x}, \bar{y}) + f_{yx}(\bar{x}, \bar{y})(x - \bar{x}) + f_{yy}(\bar{x}, \bar{y})(y - \bar{y}) &= 0. \end{aligned} \quad (3.13)$$

This pair of equations is equivalent to the matrix equation (3.7) that was solved in Newton's optimization method.

Computational Problem 3.1. *Again consider the stationary-point equations that you were unable to solve in Problem 2.2. Now apply Newton's method to them by repeatedly finding the solutions to the corresponding pair of equations (3.13) with (\bar{x}, \bar{y}) replaced by the current guess and making the new guess the solution (x, y) . Compare the sequence of guesses that result from this approach with the sequence you generated in Implementation 1 for the monthly payment function (2.3) using the same initial guess.*

3.2 From Stationary Points to Optimizers

Newton's optimization method for finding stationary points of functions has already been covered, but ultimately it is the actual optimizers of these functions that are desired. From your calculus class you are probably familiar with the second-derivative test for optimizers of functions of one variable: $f''(\bar{x}) > 0$ at a stationary point \bar{x} guarantees that \bar{x} is a local minimizer and likewise $f''(\bar{x}) < 0$ guarantees that \bar{x} is a local maximizer. In the unhappy situation when $f''(\bar{x}) = 0$, nothing can be safely concluded from the second-derivative test.

The remainder form of the Taylor quadratic (3.6) can be used to develop an analogous test for optimizers of functions of more than one variable. If the guess (\bar{x}, \bar{y}) is a stationary point, then the gradient $\nabla f(\bar{x}, \bar{y})$ is the zero-vector, and the remainder form (3.6) reduces to

$$f(x, y) = f(\bar{x}, \bar{y}) + \frac{1}{2} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix} \nabla^2 f(\xi_x, \xi_y) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix},$$

where (ξ_x, ξ_y) is some mystery point between (x, y) and (\bar{x}, \bar{y}) . From this, it is clear that the stationary point (\bar{x}, \bar{y}) is a minimizer (maximizer) if the vector-matrix-vector multiplication

$$\begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix} \nabla^2 f(\xi_x, \xi_y) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \quad (3.14)$$

is always nonnegative (nonpositive).

Exercise 3.2.1. *Verify this test for optimizers.*

This test for stationary points as optimizers is not really very practical since it depends on some mystery point (ξ_x, ξ_y) between (x, y) and (\bar{x}, \bar{y}) , and it's generally impossible to evaluate the second-derivative matrix at a mystery point. One practical solution is to instead evaluate the second-derivative matrix at the stationary point (\bar{x}, \bar{y}) (which is not mysterious at all) and then test the sign of the vector-matrix-vector multiplication:

$$\begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix} \nabla^2 f(\bar{x}, \bar{y}) \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix}. \quad (3.15)$$

It turns out that since the partial derivatives making up the second-derivative matrix are continuous, as long as the mystery point (ξ_x, ξ_y) is close enough to the guess (\bar{x}, \bar{y}) , the vector-matrix-vector multiplication in (3.14) is always positive (negative) when the one in (3.15) is.

Aside: Continuity

You can get some intuition for the fact that the sign of the vector-matrix-vector multiplication in (3.15) determines the sign of the analogous vector-matrix-vector multiplication at nearby points by considering any continuous function of one variable.

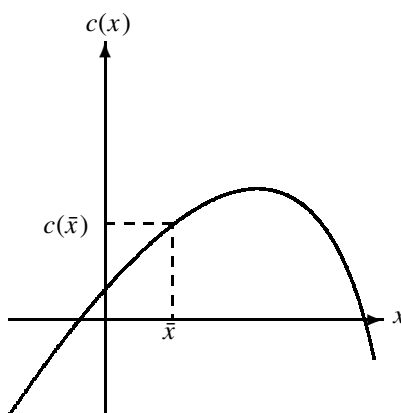


Figure 3.2. A continuous function $c(x)$

Figure 3.2 shows the graph of a particular continuous function with a particular base point \bar{x} , but the idea is the same for any continuous function and any base point. Notice in Figure 3.2 that for a point like \bar{x} with positive value $c(\bar{x})$, the points x nearby \bar{x} on the x -axis have values that are also positive and that get closer to the value of $c(\bar{x})$ as x gets closer to \bar{x} . For general functions of any number of variables, the property of being continuous ensures that the difference between output values shrinks to nothing as the distance between the corresponding input points is shrunk to nothing. The vector-matrix-vector multiplication in (3.15) defines a continuous function of (\bar{x}, \bar{y}) as long as the partial derivatives which define the second-derivative matrix are continuous functions, which we are assuming for all the functions we're considering.

All that is needed to make the new test useful is some way of guaranteeing that the mystery point (ξ_x, ξ_y) is close enough to the guess (\bar{x}, \bar{y}) . This is done essentially by relaxing the standards and allowing *local* optimizers as the goal.

The point (\bar{x}, \bar{y}) is a **local minimizer (maximizer)** for f if for points (x, y) sufficiently close to (\bar{x}, \bar{y}) , the value $f(x, y)$ is no less than (greater than) $f(\bar{x}, \bar{y})$.

Recall that the only thing known for certain about the mystery point is that it lies between its associated (x, y) and the guess (\bar{x}, \bar{y}) .

Problem 3.2. Complete the following:

- (a) Explain why a zero result from the vector-matrix-vector multiplication (3.15) would not necessarily say anything about the sign of vector-matrix-vector multiplication in (3.14).
- (b) Explain how you know that a stationary point (\bar{x}, \bar{y}) is a local minimizer as long as the vector-matrix-vector multiplication in (3.15) is positive whenever $(x, y) \neq (\bar{x}, \bar{y})$.

Aside: Definite Matrices and Eigenvalues

There is terminology for the sign of a vector-matrix-vector multiplication like that in (3.15) or (3.14) being always positive or negative, and it actually applies to any matrix

$$A = \begin{bmatrix} a & b \\ b & d \end{bmatrix} \quad (3.16)$$

which is *symmetric* above and below its diagonal.

A symmetric matrix A is called **positive-definite** if the vector-matrix-vector multiplication

$$\begin{bmatrix} x & y \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix}$$

definitely gives a positive result in all cases when $(x, y) \neq (0, 0)$, and A is called **negative-definite** if the same vector-matrix-vector multiplication definitely gives a negative result in all cases when $(x, y) \neq (0, 0)$.

There is also a practical way to verify the definiteness of a symmetric matrix (3.16) which doesn't involve actually testing the sign of the vector-matrix-vector multiplication over and over with every possible vector (of course, such direct verification is impossible). The idea is to instead find the *eigenvalues* of A and test their sign.

A matrix with two rows and two columns has at most two distinct **eigenvalues**, which are simply the scalars ϵ solving the matrix equation

$$A\mathbf{v} = \epsilon\mathbf{v}$$

for some special vector \mathbf{v} (called an **eigenvector** corresponding to that eigenvalue ϵ).

Notice that an eigenvalue ϵ is thus the number for which matrix multiplication of the matrix A against the eigenvector \mathbf{v} is identical to scalar multiplication of the vector \mathbf{v} by ϵ . It turns out that the eigenvalues of a symmetric matrix (3.16) can be found by solving for ϵ in the quadratic equation

$$(a - \epsilon)(d - \epsilon) = b^2.$$

Problem 3.3. Complete the following:

- Explain why the eigenvalues of a symmetric matrix are reciprocals of the eigenvalues of its inverse matrix (assuming its inverse exists).
- Find the eigenvalues of the second-derivative matrix from Exercise 3.1.4 for the two-variable discretization function (3.4) with $(\bar{x}, \bar{y}) = (1, 2)$.

Most mathematical software these days (e.g., *Mathematica* or *MATLAB*®) has built-in eigenvalue finders so you don't have to do these computations by hand. In many cases, finding the eigenvalues of a matrix is relatively straightforward, so testing their signs is too. The relationship between eigenvalues and the definiteness of symmetric matrices is as follows:

- Both eigenvalues of A are positive if and only if A is positive-definite.
- Both eigenvalues of A are negative if and only if A is negative-definite.

When the eigenvalues of the matrix are reasonable to compute, this relationship gives a very practical test of definiteness (as opposed to direct verification, which is impossible). For very large scale problems (e.g., with tens of thousands of variables) where the computation of eigenvalues can become too expensive, other methods are used to test for positive-definiteness. A nice side-effect of definiteness is that it ensures the existence of the inverse matrix (which shares the same definiteness property via Problem 3.3).

Using this matrix terminology, the optimality rule can be compactly stated as the following “theorem” (math lingo for an important fact).

Theorem 1 (optimality of stationary points). *If the second-derivative matrix is positive-definite (negative-definite) at a stationary point, then the stationary point is a local minimizer (maximizer).*

This is a classical theorem and we’ve already given the basic argument to justify it. You can find its formal “proof” (math lingo for a careful argument) in more advanced optimization texts (for instance, see [3, Theorem 2.1.1]). The optimality test in Theorem 1 is particularly nice when used in tandem with Newton’s optimization method, where the second-derivative matrix is already computed at every current guess.

Computational Problem 3.2. *Modify Newton’s optimization method you used in Implementation 1 by including a test for local optimality at the end. (Note: If your software computes eigenvalues automatically, use this feature.) Are the final guesses corresponding to (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3) local minimizers, local maximizers, or undetermined by the test for optimality?*

3.2.1 But I Learned a Different Second-Derivative Test in Multivariable Calculus

From your multivariable calculus class, you might recall a second-derivative test for optimizers of functions of two variables: If, for a stationary point (\bar{x}, \bar{y}) , the inequality holds that

$$f_{xx}(\bar{x}, \bar{y})f_{yy}(\bar{x}, \bar{y}) > f_{xy}(\bar{x}, \bar{y})^2, \quad (3.17)$$

then (\bar{x}, \bar{y}) is a local minimizer when $f_{xx}(\bar{x}, \bar{y}) > 0$ and a local maximizer when $f_{xx}(\bar{x}, \bar{y}) < 0$.

You might be wondering what in the world this has to do with the test just developed. It turns out that these two apparently different tests are one and the same. In the test just developed, the eigenvalues of the second-derivative matrix were needed. According to the Aside on Definite Matrices and Eigenvalues, these are simply the solutions ϵ to the quadratic equation

$$(a - \epsilon)(d - \epsilon) = b^2,$$

where $a = f_{xx}(\bar{x}, \bar{y})$, $d = f_{yy}(\bar{x}, \bar{y})$, and $b = f_{xy}(\bar{x}, \bar{y})$. Expanding this and solving this quadratic equation for ϵ gives

$$\epsilon = \frac{a + d \pm \sqrt{(a + d)^2 - 4(ad - b^2)}}{2}. \quad (3.18)$$

After expanding and then simplifying the term under the square root, it is apparent that this term is definitely greater than or equal to zero (hence there are real solutions for ϵ). On the other hand, according to the expression (3.18), the solutions ϵ come in same-sign pairs only when the term $(ad - b^2)$ is greater than zero, and the same-sign of the pair is determined by the sign of a .

Problem 3.4. *Complete the following:*

- (a) *Verify that the term under the square root in (3.18) is greater than or equal to zero in this case.*
- (b) *Verify that the solutions ϵ to (3.18) come in same-sign pairs only when $(ad - b^2) > 0$ and that the same-sign of the pair is determined by the sign of a .*

By plugging in the values $a = f_{xx}(\bar{x}, \bar{y})$, $d = f_{yy}(\bar{x}, \bar{y})$, and $b = f_{xy}(\bar{x}, \bar{y})$ for the second-derivative matrix, it follows that the test using (3.17) from the typical multivariable calculus class is just another version of Theorem 1. However, the version developed here in Theorem 1 has the great advantage of being easily generalized to functions of more than two variables. This is important for practical optimization since many optimization models use more than two variables.

3.2.2 Details for Functions of More Than Two Variables

Functions of more than two variables are common in practice. The three-variable monthly payment function (2.4) is one such example from this book, and integral discretizations using more than four t -values typically generate functions of more than two variables.

Problem 3.5. *By setting the variables*

$$\begin{aligned} x &= x(1/4), \\ y &= x(1/2), \\ z &= x(3/4), \end{aligned}$$

give the objective function $f(x, y, z)$ corresponding to the discretization using t -values 0, 1/4, 1/2, 3/4, and 1 of Newton's aerodynamics problem: Find the surface-generating function $x(t)$ (passing through (0,0) and (1,1)) minimizing the integral

$$\int_0^1 \frac{x(t)x'(t)^3}{1+x'(t)^2} dt.$$

Evidently, we would like to be able to apply Theorems 1 and 2 and Newton's optimization method to functions of more than two variables. Our ability to do so rests only on the computation of multivariable versions of the relevant mathematical objects, for instance, the second-derivative matrix. The form of this matrix for any multivariable function can easily be deduced by considering a function $f(x, y, z)$ of three variables, whose second-derivative matrix is given by

$$\nabla^2 f(x, y, z) = \begin{bmatrix} f_{xx}(x, y, z) & f_{xy}(x, y, z) & f_{xz}(x, y, z) \\ f_{yx}(x, y, z) & f_{yy}(x, y, z) & f_{yz}(x, y, z) \\ f_{zx}(x, y, z) & f_{zy}(x, y, z) & f_{zz}(x, y, z) \end{bmatrix}.$$

Graphical Optimization

A particularly critical (and misguided) reader might dismiss the analyses outlined in the preceding sections of this book as being unnecessary as long as one has access to quality graphics software. However, it is relatively rarely the case that the peaks and valleys of a function of two variables can be deduced by analyzing its graph alone.

Computational Problem 3.3. *Complete the following:*

- (a) *Use their graphs to try to find minimizers of (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3).*
- (b) *Compare your results to those you obtained in Implementation 1 and discuss the differences between these two methods of finding minimizers.*

The success of graphical optimization of any nontrivial function of two variables is entirely dependent on which piece of the graph's surface is examined. However, without prior knowledge, it is very difficult to be certain which piece is the right one. Moreover, even when the right piece is known, the function can have different scales of change in different directions, which can make it difficult to distinguish the true features of the graph. For functions of more than two variables, the case for graphical optimization is even weaker since there aren't particularly good ways of displaying their graphs (which live in more than three dimensions). This leaves nongraphical methods as the only practical means for optimizing functions of more than two variables, so it is especially reassuring that the optimization methods outlined in the preceding sections work for these functions.

3.2.3 More on Stopping

In Section 3.1.1, it was shown that instead of finding actual stationary points, methods typically terminate at a *nearly stationary* point whose gradient is nearly zero. The hope is that this nearly stationary point is close enough to an actual stationary point, but unfortunately this is not always the case. Of course, if the location of the target stationary point were known, nearness to it could be used as a stopping criterion for the method, but knowing the location of the target would make the method to find it unnecessary. However, even without knowing the location of the target, we know that it is a stationary point, and so we know that the gradient at the target is zero. What we need is a way to deduce something we can't know—whether our current guess is close to the target—from something we can compute—how close the gradient at the current guess is to the gradient at the target.

Reverse Continuity for One-Variable Functions

We wish to be able to conclude nearness to a stationary point from information about how nearly stationary a guess is. This means a kind of continuity property for the gradient is needed, where closer outputs ensure closer inputs. Since the usual continuity has closer inputs ensuring closer outputs, the desired property is *reverse continuity* for the gradient.

A mapping G is **reverse continuous** at \bar{x} if, eventually, outputs closer to $G(\bar{x})$ ensure inputs closer to \bar{x} . In this case, if we observe the outputs $G(x)$ approaching $G(\bar{x})$, then we know that the inputs x must be approaching \bar{x} .

Problem 3.6. Complete the following:

- (a) Graph the function $f(x) = (10^{-10})x^2/2$, making sure to use a scale that shows all of the features of the graph. Now graph its corresponding derivative function $f'(x) = (10^{-10})x$. How far from the lone stationary point at 0 can a point be while having its derivative less than or equal to 10^{-6} ?
- (b) Explain where the derivative function shown in Figure 3.3 is not reverse continuous (even though it is continuous at every point).
- (c) Explain why the derivative function $f'(x) = (10^{-10})x$ is reverse continuous everywhere.

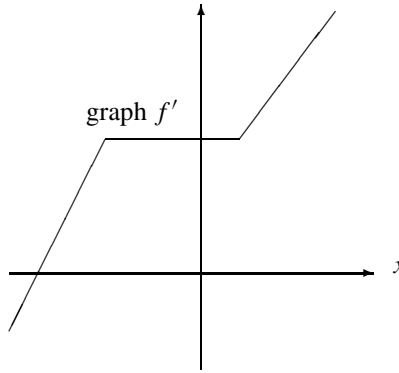


Figure 3.3. Not reverse continuous everywhere

It is apparent from Problem 3.6 that a function whose derivative is reverse continuous everywhere can still have nearly stationary points that aren't very close to a true stationary point. For some applications, this might not pose a serious problem; for instance, there is little danger in stopping at a nearly stationary point for the function $f(x) = (10^{-10})x^2/2$ from Problem 3.6 if that function represents a quantity to be minimized within some relatively large tolerance. Even though a nearly stationary point can be relatively far away from the true stationary point in this case, the values of the function at each point are close enough that the nearly stationary point might be an acceptable (near) minimizer for the application. On the other hand, if the same function represents a quantity to be minimized within a smaller tolerance, the value of the function at the nearly stationary point might be unacceptably large when compared to the true minimum. To cover both situations, we need something more than simple reverse continuity of the derivative.

In addition to knowing that closer derivative outputs ensure closer inputs, we need to be able to deduce an acceptable *scale* of closeness of the inputs (which cannot be measured directly) from the easily measured closeness of the derivative outputs. For a one-variable function $f(x)$, the closeness of inputs x to a target \bar{x} is measured by $|x - \bar{x}|$, and the corresponding closeness of derivative outputs is $|f'(x) - f'(\bar{x})|$. So we want a relationship like

$$L|x - \bar{x}| \leq |f'(x) - f'(\bar{x})| \quad (3.19)$$

for a fixed constant $L > 0$.

Problem 3.7. Complete the following:

- (a) Explain why the relationship (3.19) guarantees that f' is reverse continuous at the point \bar{x} .
- (b) Suppose the relationship (3.19) holds with $L = 1$. If the derivative at the current guess agrees with zero up to 6 decimal places, how close is the guess guaranteed to be to the stationary point \bar{x} ?
- (c) What if instead $L = 1000$?
- (d) What values of L work for the derivative function $f'(x) = (10^{-10})x$ from Problem 3.6?
- (e) In general, then, is a relatively large or a relatively small L more desirable?

To see how the relationship (3.19) manifests itself graphically, divide both sides by $|x - \bar{x}|$. The result is

$$L \leq \frac{|f'(x) - f'(\bar{x})|}{|x - \bar{x}|}. \quad (3.20)$$

The numerator here is the change in the output of f' and the denominator is the change in the input, so the right-side fraction is the slope of the “secant” line through the base point $(\bar{x}, f'(\bar{x}))$ on the graph of f' and any other point $(x, f'(x))$ on the graph of f' . It follows that the constant L in the relationship (3.19) is simply a lower bound on the magnitudes of the slopes of these secant lines.

Any such lower bound L will be called a **scaling factor** for the reverse continuity of f' at \bar{x} .

The graphical implications of any such scaling factor can be illustrated by constructing cross-hairs centered at $(\bar{x}, f'(\bar{x}))$ out of a pair of crossing lines with opposite-sign slopes of the same magnitude L , as in Figure 3.4. Since the graph of f' avoids the shaded triangular regions on either side of the cross-hairs, any secant line through $(\bar{x}, f'(\bar{x}))$ will also avoid

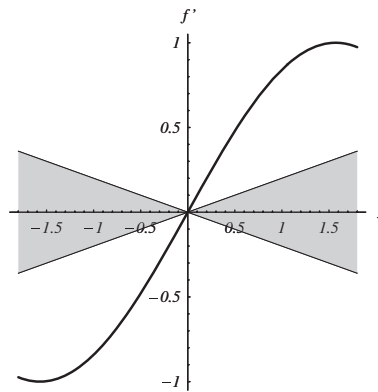


Figure 3.4. Graphical implication of a scaling factor

these shaded regions. It follows that the magnitude of the slope of any such secant line will be at least as great as the value of L used to construct these cross-hairs.

It has already been seen in Problem 3.7 that the larger the scaling factor the better, and in any given situation, it is desirable to know what the largest possible scaling factor is that works. For practical purposes, it is necessary only to consider points x near \bar{x} and satisfying the relationship (3.19), since the relationship (3.19) is used only near the end of the method to decide whether or not to stop. And at that stage, it is hoped that the current guess is within relatively close range of the target \bar{x} . Graphically, focusing on points near \bar{x} is achieved by zooming in on the graph of f' near $(\bar{x}, f'(\bar{x}))$, and if zooming in like this is allowed, the scaling factor can often be increased.

Exercise 3.2.2. *Complete the following:*

- (a) *The cross-hairs in Figure 3.4 are centered on the graph of f' at the point $(\bar{x}, f'(\bar{x})) = (0, 0)$. Indicate the points \bar{x} on the x -axis in Figure 3.4 for which cross-hairs with the same slope as shown, but now centered on the graph of f' at the new point $(\bar{x}, f'(\bar{x}))$, are too steep to correspond to a scaling factor for f' there.*
- (b) *Indicate the additional points \bar{x} on the x -axis in Figure 3.4 for which cross-hairs with the same slope as shown, but now centered on the graph of f' at the new point $(\bar{x}, f'(\bar{x}))$, correspond to a scaling factor for f' at some level of zooming in.*
- (c) *There are exactly two points on the graph in Figure 3.4 where no matter how small a scaling factor $L > 0$ and no matter what level of zooming is used, the corresponding cross-hairs will not work. Which are these and what value do they produce in the second derivative f'' ?*

You should recall from calculus that secant lines approach the tangent line while zooming in on the graph of any function. Since here the zooming in is done on the graph of f' , the secant lines approach the tangent line to f' at the point \bar{x} . The slope of the tangent line to f' at \bar{x} is given by the derivative of f' at \bar{x} , which is the second derivative of f at \bar{x} , $f''(\bar{x})$. This means that the right-side difference quotient in (3.20) approaches $|f''(\bar{x})|$ as the point x approaches \bar{x} , which results in the relationship $L \leq |f''(\bar{x})|$. Therefore, whatever the largest scaling factor is, it cannot exceed $|f''(\bar{x})|$, and, moreover it is clear from the discussion that any scaling factor $L > 0$ less than $|f''(\bar{x})|$ works for some level of zooming in. Therefore, as long as $f''(\bar{x}) \neq 0$, there is some scaling factor $L > 0$ which can be used to deduce the closeness of the current guess to the stationary point (assuming the current guess is within some small enough range of the stationary point).

Reverse Continuity for Multivariable Functions

To extend all of this to functions of vector-variables \mathbf{x} , the magnitudes of vectors (indicated by double bars $\|\cdot\|$) are used instead of the absolute values in the relationship (3.19) defining the scaling factor $L > 0$:

$$L \|\mathbf{x} - \bar{\mathbf{x}}\| \leq \|\nabla f(\mathbf{x}) - \nabla f(\bar{\mathbf{x}})\|. \quad (3.21)$$

Recall that the **magnitude** of a vector is the square root of the sum of the components squared. For instance, the magnitude of a vector with two components is

$$\left\| \begin{bmatrix} x \\ y \end{bmatrix} \right\| = \sqrt{x^2 + y^2}.$$

Note that the magnitude of a scalar is just its absolute value.

The classical inverse mapping theorem applied to the gradient mapping ∇f gives the following upper bound on the sizes of L that can work in (3.21).

Theorem 2 (reverse continuity of the gradient). *For any number $L > 0$ that is less than the magnitude of the lowest-magnitude eigenvalue of the second-derivative matrix $\nabla^2 f(\bar{x})$, the gradient is reverse continuous with scaling factor L for x close enough to \bar{x} .*

Notice that the lowest-magnitude eigenvalue of the second-derivative matrix at \bar{x} is the multivariable analogue of the term $|f''(\bar{x})|$ in the single-variable case. For a stationary point \bar{x} , Theorem 2 gives the kind of result sought to make conclusions about nearly stationary points being close enough to a true stationary point.

Exercise 3.2.3. *Using Theorem 2 and the bound (3.21), describe how close nearly stationary points are to a stationary point.*

Practical Application of Theorem 2

It is a nice coincidence that a scaling factor $L > 0$ is guaranteed by essentially the same second-derivative test developed in Theorem 1 for finding optimizers from stationary points. In practice, this means that it is possible to test for optimality and a scaling factor with the same computation. Ideally, the lowest-magnitude eigenvalue of the second-derivative matrix at a stationary point \bar{x} would be known. However, since it's impossible to know the target stationary point beforehand, Theorem 2 cannot be applied directly. Instead, Theorem 2 can be applied with the current guess \mathbf{x} in place of \bar{x} to get a “current guess” scaling factor $L > 0$ from the eigenvalues of the second-derivative matrix at the current guess \mathbf{x} . Then, for instance, the method might be stopped if the current guess had a small enough gradient and the scaling factor corresponding to the current guess was above an acceptable lower positive threshold.

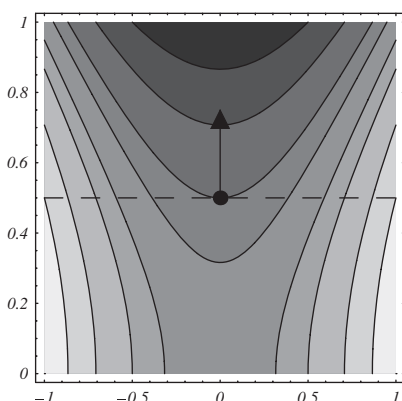
Computational Problem 3.4. *Modify Newton’s optimization method from Computational Problem 3.2 by including a scaling factor test along with a test for local optimality. (Note: If your software computes eigenvalues automatically, use this feature.) Explain how confident you are that your final guesses are close to stationary points in the case of (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3).*

3.3 Conclusions from Chapter 3

In this chapter, we introduced the *basic optimization method* and studied an important incarnation of it called *Newton’s optimization method*. We learned that the eigenvalues of the *second-derivative matrix* can tell us what kind of stationary point Newton’s optimization method finds:

- Positive eigenvalues mean the stationary point is a local minimizer.
- Negative eigenvalues mean the stationary point is a local maximizer.

We also learned that a nearly stationary point can be very far away from a true stationary point, unless the gradient mapping is reverse continuous with a large enough scaling factor. This is important because Newton's optimization method will typically stop at a nearly stationary point, so there needs to be a large enough scaling factor for us to confidently accept the result of the method. We learned that a large enough scaling factor can be guaranteed if all of the eigenvalues of the second-derivative matrix have large enough magnitude.



Chapter 4

Some Practical Modifications

So far Newton's optimization method has seemed like a pretty important method for optimization; however, in practice it is rarely used without some important modifications. Why then have you been wading through page after page about Newton's optimization method? Because much of what is really done in practice can be understood as some type of modification of Newton's optimization method in order to address one or more of the following problems (listed in order of general significance):

- Newton's optimization method finds only stationary points and not necessarily the optimizer you're looking for.
- Newton's optimization method can fail to work.
- Newton's optimization method can be relatively expensive (in time and storage) to carry out.

4.1 Finding Optimizers

Newton's optimization method obviously finds only stationary points since that's all it was designed to do. As seen in Chapter 3, a stationary point can be checked after it is found to see whether it is a minimizer or maximizer by testing the definiteness of the second-derivative matrix, but Newton's optimization method itself doesn't a priori favor one or the other of these outcomes. Interestingly, the same definiteness test motivates many of the modifications to Newton's optimization method which are made to improve the chances that an optimizer is found instead of merely a stationary point. The basic idea in all of these so-called *quasi-Newton methods* is to replace at each step the second-derivative matrix in the Taylor quadratic with some appropriate positive-definite matrix (or negative-definite in the case of maximization).

Newton's Update

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} + \nabla^2 f(x_{\text{old}}, y_{\text{old}})^{-1} [-\nabla f(x_{\text{old}}, y_{\text{old}})].$$

Quasi-Newton Update

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} + P^{-1} [-\nabla f(x_{\text{old}}, y_{\text{old}})], \quad P \text{ positive-definite.}$$

If the second-derivative matrix is already positive-definite at a step, then no replacement is necessary. The most basic quasi-Newton method uses the *identity matrix* in place of the second-derivative matrix.

Aside: The Identity Matrix

The identity matrix I always has ones down its diagonal from upper left to lower right and zeroes everywhere else. The two-dimensional version looks like

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The identity matrix is its own inverse, and multiplication against any vector by the (appropriate dimension) identity matrix returns the vector again $I\mathbf{x} = \mathbf{x}$.

Exercise 4.1.1. Complete the following:

- Verify that the identity matrix I is positive-definite.
- What simple replacement matrix can be used if a maximizer is sought instead of a minimizer?

The update formula when the identity is used in place of the second-derivative matrix is simply

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} - \nabla f(x_{\text{old}}, y_{\text{old}}),$$

so the upshot of this quasi-Newton method is that a new guess comes from stepping along the full length of the negative gradient from the current guess. Recall from calculus that the negative gradient points in the direction of steepest descent relative to the base point, which is why this quasi-Newton method is typically called the “steepest descent method.”

Steepest Descent Method

1. **Initialize** the current guess $\begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix}$.

2. **Update:**

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} - \nabla f(x_{\text{old}}, y_{\text{old}}).$$

3. **Stop or repeat.**

Since the negative gradient defines the direction of steepest descent relative to the base point, any direction that makes an acute angle with the negative gradient is a descent direction. Likewise any direction making an obtuse angle with the negative gradient is an ascent direction. See Figure 4.1, where the negative gradient vector at a base point is

pictured, along with the perpendicular dotted line separating descent directions at the base point from ascent directions for the function whose contours are shown with lower levels shaded darker. It appears that stepping along a direction within 90 degrees of the negative gradient is a good idea if it is desirable to drive the function value down (which it is for minimization), and stepping along the negative gradient direction itself provides the fastest immediate decrease.

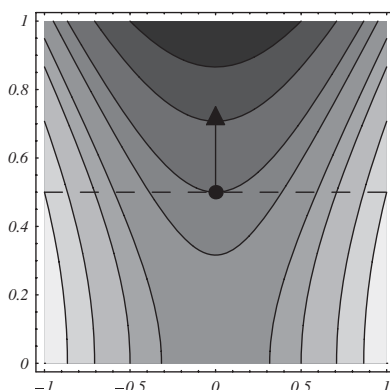


Figure 4.1. *The negative gradient determines uphill and downhill directions*

Exercise 4.1.2. *Complete the following:*

- (a) *Explain which directions in Figure 4.1 correspond to no instantaneous change in the function from the base point pictured.*
- (b) *Recall that one formula for computing the dot product of any two vectors \mathbf{v} and \mathbf{w} is*

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \|\mathbf{w}\| \cos(\theta),$$

where θ is the smallest angle between the two vectors. Explain how to use the dot product to determine whether or not a vector \mathbf{d} is a descent direction.

The only trouble with taking a full-step along any descent direction is that it can overshoot and end up ascending instead, since only instantaneous descent is guaranteed. So, maybe a full-step won't work, but it is certain that there is at least some (perhaps very small) range of steps along a descent direction causing reduction of the function.

Thus, the update formula is usually modified by multiplying the **step-direction** \mathbf{d} by some appropriate **step-factor** α .

For instance, a full-step corresponds to a step-factor of $\alpha = 1$, and a half-step corresponds to a step-factor of $\alpha = 1/2$. Note that the step-direction \mathbf{d} is actually a vector of length $\|\mathbf{d}\|$, so a full-step along the step-direction \mathbf{d} actually moves a distance $\|\mathbf{d}\|$.

If the negative gradient $\mathbf{d} = -\nabla f(x_{\text{old}}, y_{\text{old}})$ is being used as the step-direction, the update is

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} + \alpha \begin{bmatrix} -\nabla f(x_{\text{old}}, y_{\text{old}}) \end{bmatrix}. \quad (4.1)$$

This corresponds to replacing the second-derivative matrix in (3.9) with the (positive-definite) matrix $P = \frac{1}{\alpha}I$ (whose inverse matrix is $P^{-1} = \alpha I$).

One of the simplest methods for choosing an appropriate step-factor α is to start with $\alpha = 1$ (a full-step) and then successively halve the step-factor until descent is achieved.

This is called **backtracking** for obvious reasons.

Another way to determine an appropriate step-factor starts at the other end with $\alpha = 0$ and continues to increase α until just before *ascent* occurs.

In general, methods for choosing an appropriate step-factor are called **line-searches** since the appropriate length step is sought along the line determined by the fixed step-direction \mathbf{d} .

When the step-factor α is increased from 0 until ascent occurs, it is a local minimizer for the α -optimization problem

$$\min_{\alpha} f\left((x_{\text{old}}, y_{\text{old}}) + \alpha \mathbf{d}\right). \quad (4.2)$$

Since exactly the best step-factor in the step-direction \mathbf{d} is chosen, this kind of line-search is called **exact**.

As $\alpha = 0$ is one of the options considered in the α -optimization problem, the minimizing step-factor is guaranteed to generate at least as low a value as the function at the current guess.

Problem 4.1. Consider the following two functions:

$$f_1(x, y) = x^2 + y^2 \text{ and } f_2(x, y) = x^2 + \frac{y^2}{2}.$$

(a) Use the contour plots in Figure 4.2 to illustrate one step of the steepest descent method (4.1) applied to each function from the current guess $(x_{\text{old}}, y_{\text{old}}) = (1, 1)$, using each of following three line-search alternatives:

- No line-search, so $\alpha = 1$.
- The backtracking line-search.
- The exact line-search.

(b) What is going to happen in each case as more steps are taken? What is significant about the function f_1 that causes the behavior you observe?

(c) Now consider a general function f . It is a fact that consecutive steps of the steepest descent method with an exact line-search are perpendicular.

- (i) Verify this fact conceptually by explaining the relationship between the line of search and the contour of f at the new guess.
- (ii) Prove this fact analytically by applying the two-variable chain rule to the derivative with respect to α of the function being minimized in the α -optimization problem (4.2).

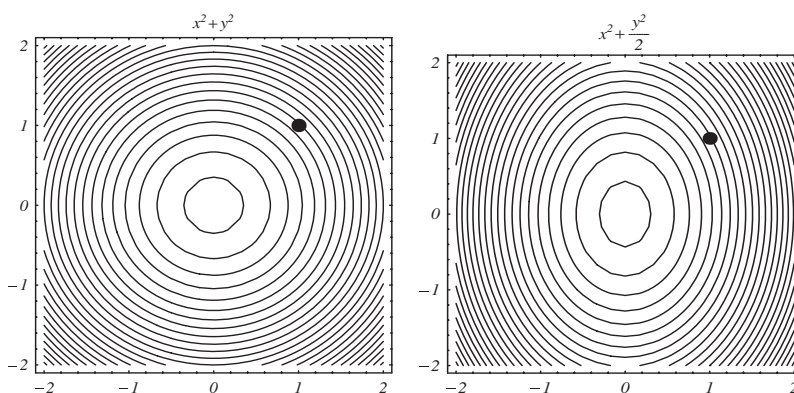


Figure 4.2. Contour plots of $x^2 + y^2$ and $x^2 + \frac{y^2}{2}$

It is no accident that the positive-definite matrices discussed so far result in descent for the quasi-Newton method, since any positive-definite replacement matrix P in the update formula creates a descent step-direction.

Exercise 4.1.3. By computing a dot product, show that the vector

$$\mathbf{d} = P^{-1} [-\nabla f(x_{\text{old}}, y_{\text{old}})]$$

is a descent direction from the current guess as long as P is a positive-definite matrix.

There are many other possible choices of positive-definite replacement matrices P , one simple example of which is the sum matrix

$$P = \nabla^2 f(x_{\text{old}}, y_{\text{old}}) + \beta I \quad (4.3)$$

for a large enough multiple $\beta \geq 0$ of the identity matrix. As long as β is larger than the magnitude of the most negative eigenvalue of the second-derivative matrix, the sum matrix P will be positive-definite. This follows because the eigenvalues of the sum matrix P in this case are the same as β added to the eigenvalues of the second-derivative matrix. One nice feature of these quasi-Newton methods is that as a minimizer is approached, the second-derivative matrix itself generally becomes positive-definite, in which case $\beta = 0$ can be used and the replacement matrix is the same as the second-derivative matrix.

Implementation 2. Implement the steepest descent method with an exact line-search for minimizing (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3), and use a contour plot to chart the progress in each case.

Note that even the descent guaranteed by a quasi-Newton method with a line-search doesn't guarantee that an optimizer is found, since it is possible to descend toward a saddle point.

4.2 Improving Success

Another problem with Newton's optimization method is that it occasionally fails altogether. One source of failure has already been mentioned, namely, when the second-derivative

matrix is not invertible so the update can't be computed. This problem is fixed by any of the quasi-Newton methods discussed in the previous section, since those all involved positive-definite replacement matrices which are automatically invertible.

A more subtle version of the noninvertibility problem occurs when the matrix in the update formula is theoretically invertible (as is any definite matrix), but its inverse has an eigenvalue with very large magnitude. This happens in particular when an eigenvalue of the replacement matrix P is very small since the eigenvalues have an inverse relationship. The following computational problem illustrates the kind of thing that can go wrong in this situation.

Computational Problem 4.1. *Consider the one-variable function*

$$f(x) = \log(1 + e^{2x}) - x.$$

- (a) *Find the first and the second derivatives, and use them to verify that there is a minimizer at $x = 0$.*
- (b) *Apply Newton's optimization method to minimize this function starting with guess $x_{\text{old}} = 1$.*
- (c) *Do the same thing but starting with guess $x_{\text{old}} = 1.1$.*
- (d) *Verify that the second derivative is always positive (this is the one-variable analogue of a positive-definite second-derivative matrix) but that it gets very small at the ends of the x -axis.*

In Computational Problem 4.1, you should notice that when starting at 1, Newton's optimization method works, but when starting at 1.1, the method fails as the subsequent guesses tend toward the ends of the x -axis. To avoid this kind of trouble, a positive-definite replacement matrix P can be chosen whose eigenvalues have large enough magnitude so that the eigenvalues of P^{-1} aren't too large in magnitude. On the other hand, too large a magnitude eigenvalue for the replacement matrix P corresponds to a nearly zero eigenvalue for P^{-1} . If this happens, then even if the negative gradient is not small, multiplying it by P^{-1} can return a nearly zero vector, in which case the update won't really go anywhere. Accordingly, the magnitude of the eigenvalues of any positive-definite replacement matrix P used should not be too large.

Another source of failure occurs (very rarely) when Newton's optimization method gets stuck in a perpetual cycle where the same guess points are revisited again and again even though they aren't stationary points. Cycling is a problem too for the quasi-Newton methods without a line-search (as seen in Problem 4.1) but can be fixed by using a line-search.

Exercise 4.2.1. *Complete the following:*

- (a) *Using the formula (4.3) for the replacement matrix P , explain how you would arrange to have the magnitude of its eigenvalues larger than 1.*
- (b) *Explain why a quasi-Newton method with a line-search won't cycle.*

4.3 Reducing Expense

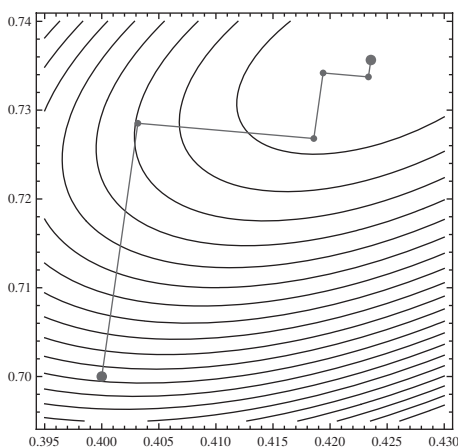
Another practical drawback of Newton's optimization method is that the second-derivative matrix is relatively time-consuming to compute and invert as well as being relatively space-consuming to store at each step. These issues are particularly significant for large-scale optimization problems. For instance, if there are 1000 variables, then the second-derivative matrix contains about half a million entries (1000 times 1000 minus double-counting for the mixed partial second derivatives) that have to be computed and stored for use at each step. Because of this issue, for large-scale problems, quasi-Newton methods are typically used which somehow avoid the computation and storage of the second-derivative matrix. The steepest descent method is one example of such a quasi-Newton method.

Exercise 4.3.1. *Explain how the computation, inversion, and storage of a replacement matrix at each step is completely unnecessary for the application of the steepest descent method.*

4.4 Conclusions from Chapter 4

In this chapter, we learned about some alternatives to Newton's optimization method that address one or more of its shortcomings. These alternatives can be thought of as modified versions of Newton's optimization method where replacements are used for the second-derivative matrix. The *steepest descent* method was studied as a prototypical modified method.

We introduced the terminology of *step-directions*, *step-factors*, and *line-searches* and learned that a replacement matrix with positive eigenvalues guarantees descent from one step to the next for some step-factor. We also saw that the eigenvalues of the replacement matrix should not be too small or too large for the modified method to be successful.



Chapter 5

How Methods Are Ranked

So, what makes one optimization method better than another? The answer often depends on the situation, as some methods are better than others in some cases but worse in other cases. One measuring stick for a method's value is the time it takes to produce an acceptable result. Since it's not possible beforehand to know what degree of accuracy every practitioner of a method will need, it is generally required that the updated guesses become arbitrarily close to the target.

This kind of arbitrarily close approach is called **convergence**, and in its absence the method is said to **diverge**.

Several obvious examples of divergence have already been seen in Problem 4.1 and Computational Problem 4.1, where either the guesses wander off aimlessly into oblivion or cycle between nonstationary points. It has also been shown already that a line-search tends to fix this kind of problem. The rest of this chapter will explore how to compare and rank different methods that are convergent.

The most practical ranking of convergent methods relies on measuring how much time they take to converge. The amount of time a method takes to converge can be broken down into two parts: (1) the amount of time needed for each step, and (2) the number of steps taken. The relatively large amount of time per step needed for Newton's optimization method to compute and store elements of the second-derivative matrix and modifications aimed at reducing this time per step were discussed in Chapter 4. So apparently Newton's optimization method is a relatively bad method in terms of its time per step, but it turns out that it is a relatively good method in terms of the number of steps taken. Measuring the number of steps taken is actually trickier than it sounds, since optimization methods usually take an infinite number of steps to find an exact solution. In practice, of course, a method is not run forever but instead is stopped when an acceptable level of accuracy has been achieved for the current guess even if it isn't an exact solution. Since arbitrary closeness is desired, however, arbitrarily strict levels of accuracy need to be accommodated. This means that somehow a way is needed to rank the different infinite number of steps taken by different converging methods to reach an exact solution. This is done by coming up with a measure of the *order of convergence* for an optimization method.

In general, measuring the order of convergence of an optimization method relies on a little magic; the exact solution is presumed known, and the closeness of subsequent guesses to it is compared. Since the presumed exact solution is the star of this show, it gets a superscript *, and vector notation will be used to accommodate any number of variables. Thus the exact solution is denoted \mathbf{x}^* and the two subsequent guesses at it are \mathbf{x}_{old} and \mathbf{x}_{new} . The closeness of each guess to the target \mathbf{x}^* is measured simply by computing the corresponding errors:

$$\begin{aligned}\text{error}_{\text{old}} &= \|\mathbf{x}_{\text{old}} - \mathbf{x}^*\|, \\ \text{error}_{\text{new}} &= \|\mathbf{x}_{\text{new}} - \mathbf{x}^*\|,\end{aligned}$$

which are just the magnitudes of the appropriate difference vectors. If the method is converging, a relationship would be expected like

$$\text{error}_{\text{new}} = \delta(\text{error}_{\text{old}}) \quad (5.1)$$

for some positive fraction $\delta < 1$ since that would indicate that the new guess is closer to the exact solution than the old guess.

Exercise 5.0.1. *Complete the following:*

- (a) *Explain why the same relationship (5.1) but with $\delta \geq 1$ would be symptomatic of divergence.*
- (b) *Now with $\delta < 1$ some positive fraction, explain why a smaller δ indicates faster convergence.*

It is helpful to rewrite the relationship (5.1) in fractional terms as

$$\frac{\text{error}_{\text{new}}}{\text{error}_{\text{old}}} = \delta$$

and then to notice that such a relationship would need to persist as old guesses are repeatedly exchanged for new guesses via whatever update rule the method uses. Now, it's not important if the first few guesses do not exhibit this kind of behavior, but it is hoped that at least the later guesses do. Thus, since the (possibly different) constants δ associated with each such relationship need to be positive fractions less than one, we want all of the later ones to be bounded above by a fixed constant δ that is a positive fraction less than one. This is the basis for the first notion of order of convergence.

The method has a **linear order of convergence** if

$$\limsup \frac{\text{error}_{\text{new}}}{\text{error}_{\text{old}}} \leq \delta < 1, \quad (5.2)$$

where the “limit superior” \limsup is the smallest eventual upper bound on the quotient of errors, after repeated updating.

If the error quotients happen to converge, the \limsup is the same as the limit to which they converge. The term “linear” is used to indicate that the new error is a linear function of the old error as in (5.1). The special case when the bounding constant δ is zero is called a **superlinear order of convergence** because that beats any possible linear rate with

fixed constant δ (recall from Exercise 5.0.1 that the smaller the constant δ , the faster the convergence).

So, what does this look like in practice? Well, suppose a sequence of guesses has been generated like

$$1.1, 1.05, 1.024, 1.012, 1.005, 1.002, 1.001, 1.0005, \dots,$$

which are presumed to be approaching the exact solution $x^* = 1$. The associated sequence of errors is thus

$$.1, .05, .024, .012, .005, .002, .001, .0005, \dots$$

To deduce whether or not this shows a linear order of convergence, the sequence of fractions needs to be considered whose numerators come from each element in the sequence and whose denominators are the element immediately preceding the one used for the numerator. These fractions are

$$\frac{.05}{.1}, \frac{.024}{.05}, \frac{.012}{.024}, \frac{.005}{.012}, \frac{.002}{.005}, \frac{.001}{.002}, \frac{.0005}{.001}, \dots,$$

which work out to be

$$.5, .48, .5, .416\bar{6}, .4, .5, .5, \dots$$

Based on this evidence, it looks like the original sequence of guesses is converging at a linear rate to the solution $x^* = 1$ with constant $\delta = .5$. Notice that the practical implication of this is that the error at each step is reduced by roughly one half.

Exercise 5.0.2. *If the constant $\delta = .1$, what might the list of guesses look like?*

If the method is converging, it would also be expected that the two errors would be relatively small, certainly less than one. That means that it would be even better if the guesses showed a relationship like

$$\text{error}_{\text{new}} = \delta (\text{error}_{\text{old}})^2, \quad (5.3)$$

which indicates even better improvement with the new guess than does (5.1) for the same constant δ .

Accordingly, there is an even faster order of convergence than linear, called a **quadratic order of convergence**, if

$$\limsup \frac{\text{error}_{\text{new}}}{(\text{error}_{\text{old}})^2} \leq \delta < \infty. \quad (5.4)$$

Notice that there is no stipulation here that δ necessarily be a fraction, as was the case for the linear order of convergence. Of course, a smaller constant δ still indicates faster convergence, but there is relatively fast convergence now even with a δ greater than 1.

Problem 5.1. *Complete the following:*

- (a) *If relationship (5.3) is viewed as a linear relationship instead, what would be the “constant” multiplying the old error on the right side? Use this to explain why having a relationship such as (5.3) persist in the limit, regardless of the size of δ , is at least as good as a superlinear order of convergence.*

- (b) *Generate a sequence of guesses that appears to approach $x^* = 1$ and that exhibits a quadratic order of convergence with constant $\delta = 4$.*

In theory, the power on $\text{error}_{\text{old}}$ could be increased to define higher orders of convergence, but quadratic is the highest order of convergence achieved by the methods in this book. Moreover, orders of convergence are not necessarily relevant until the method is near the target, since it is only in the limit that the order is guaranteed to be observed. When solving real-world optimization problems, it is not uncommon to terminate the process well before the limiting behavior is realized, in which case a method with a linear order of convergence might actually outperform a method with a quadratic order of convergence.

5.1 Ranking Newton's Optimization Method

As was mentioned at the beginning of this chapter, the shortcomings of Newton's optimization method in terms of time spent per step are mollified somewhat by its good performance in terms of the number of steps it generally takes to achieve an acceptable result. The good performance alluded to is that Newton's optimization method typically exhibits a quadratic order of convergence when it converges. To see how this kind of fact is established, consider the case of the one-variable function $f(x)$ with presumed optimizer x^* .

Even though we don't know the location of the presumed optimizer x^* , we do know the value of its derivative (i.e., $f'(x^*) = 0$ since x^* is definitely a stationary point). We also know from the definition (5.4) that to establish a quadratic order of convergence, we need to compare $(\text{error}_{\text{old}})^2$ to $\text{error}_{\text{new}}$. This suggests that we should explore an equation based on f' in which $(\text{error}_{\text{old}})^2$ and $\text{error}_{\text{new}}$ can be made to appear. The remainder form of the Taylor quadratic approximation to the derivative function $f'(x)$ at the guess x_{old} is one such equation:

$$f'(x) = f'(x_{\text{old}}) + f''(x_{\text{old}})(x - x_{\text{old}}) + \frac{f'''(\xi)}{2}(x - x_{\text{old}})^2, \quad (5.5)$$

where ξ is some point between x and x_{old} . (Note that to use this Taylor quadratic approximation, we are assuming f''' is a continuous function.) Since x^* is a stationary point with $f'(x^*) = 0$, x^* can be plugged in for x in (5.5) to get

$$0 = f'(x_{\text{old}}) + f''(x_{\text{old}})(x^* - x_{\text{old}}) + \frac{f'''(\xi)}{2}(\text{error}_{\text{old}})^2,$$

where the term $(\text{error}_{\text{old}})^2$ replaces $(x^* - x_{\text{old}})^2$ since squaring the negative old error is the same as squaring the old error.

Moving the first two terms to the left side and dividing by $f''(x_{\text{old}})$ (assuming this term is not zero) yields

$$x_{\text{old}} - \frac{f'(x_{\text{old}})}{f''(x_{\text{old}})} - x^* = \frac{f'''(\xi)}{2f''(x_{\text{old}})}(\text{error}_{\text{old}})^2,$$

which is the same as

$$x_{\text{new}} - x^* = \frac{f'''(\xi)}{2f''(x_{\text{old}})}(\text{error}_{\text{old}})^2$$

according to the update in Newton's optimization method. Using the notation for the new error and dividing both sides by the old error squared gives

$$\frac{\text{error}_{\text{new}}}{(\text{error}_{\text{old}})^2} = \left| \frac{f'''(\xi)}{2f''(x_{\text{old}})} \right|. \quad (5.6)$$

Since the point ξ is between x^* and the guess x_{old} , as long as the method is converging, we know that in the limit there is no difference between the points ξ , x_{old} , and x^* . This means that (5.6) implies that the limit superior of the error fraction satisfies

$$\limsup \frac{\text{error}_{\text{new}}}{(\text{error}_{\text{old}})^2} = \left| \frac{f'''(x^*)}{2f''(x^*)} \right|,$$

which shows a quadratic order of convergence with constant

$$\delta = \left| \frac{f'''(x^*)}{2f''(x^*)} \right|$$

as long as the denominator is not zero.

The argument for functions of more than one variable is essentially similar but complicated somewhat by the fact that there isn't a simple multivariable analogue to the third derivative. In the multivariable case, the constant δ is given by

$$\delta = K \text{ (the magnitude of the greatest-magnitude eigenvalue of } \nabla^2 f(\mathbf{x}^*)^{-1} \text{)}$$

for some constant $K > 0$ that serves the role of the third derivative term in the single-variable case. In this case too, then, the quadratic order of convergence depends on the invertibility of the second-derivative matrix, and a larger greatest-magnitude eigenvalue of the inverse corresponds to a larger constant δ and slower convergence.

Computational Problem 5.1. *Track the error fraction on the left side of (5.6) for Newton's optimization method you used in Implementation 1 on (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3). You should observe a quadratic order of convergence in each case. What are the corresponding constants δ ?*

In summary, the heavy per-step time consumption of Newton's optimization method is somewhat balanced by its relatively high order of convergence. However, the per-step time consumption is a certain cost for any practitioner on any machine. In contrast, the high order of convergence is certain to occur only at the infinite tail-end of a sequence of guesses and may not be exhibited at all for the level of accuracy sought in a particular case. On the other hand, a method with a high order of convergence offers the potential for relatively large gains at each extra step.

5.2 Ranking the Quasi-Newton Methods

In Chapter 4, we saw that the most significant problem with Newton's optimization method is that it does not discriminate among maximizers, minimizers, and other stationary points. In that same chapter, a variety of modifications were discussed that generally involved some replacement for the second-derivative matrix in case it is not positive-definite. These

modifications ensured that better guesses were obtained at every step, but how do they rate in terms of speed? All of the modified methods consumed no more time per step than Newton's optimization method, so they are no worse on that count (and, for example, the steepest descent method is much better). However, the quadratic order of convergence enjoyed by Newton's optimization method is better than the rates generally achieved by the quasi-Newton methods. In particular, the steepest descent method generally exhibits only a linear order of convergence.

Computational Problem 5.2. *Modify the steepest descent method with an exact line-search from Implementation 2 by computing the errors at each step. Show that there is a linear order of convergence for (i) the two-variable discretization function (3.4) and (ii) the monthly payment function (2.3), and give the constant fractions δ in each case.*

One fact about the steepest descent method with an exact line-search that we analyzed in Problem 4.1 was that consecutive steps zigzag in perpendicular directions. If the graph of the function being minimized looks like a chute near the minimizer as in Figure 5.1, then this perpendicular zigzagging may correspond to a linear order of convergence with constant δ close to one, which is very slow. In the following Aside, we explore how a chute-like graph is signaled by the eigenvalues of the second-derivative matrix at the minimizer having different scales.

Aside: Eigenvalues and Convexity/Concavity

Since the graph of a two-variable function is a three-dimensional surface, it exhibits "concavity" in three dimensions as well. To understand how concavity works in three dimensions, it is useful to think of slicing the graph with any plane perpendicular to the x - y plane. The image of the graph that intersects such a plane is a two-dimensional curve whose concavity can be characterized as in calculus. *Convex* is the official term used for what you may have learned as "concave up" in your calculus course, and simply *concave* is the official term for "concave down."

Thus, a two-dimensional curve is **convex** near some base point if the slopes are nondecreasing (moving from left to right along the horizontal axis), and it is **concave** if the slopes are nonincreasing. Both of these

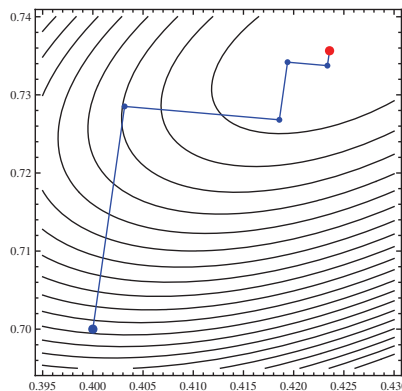


Figure 5.1. *Zigzagging steepest descent method*

concepts can be strengthened by demanding strict decrease or increase in the slopes, which is indicated by the terminology **strictly convex** or **strictly concave**.

Exercise 5.2.1. *What kind of two-dimensional curves are simultaneously convex and concave?*

Since the second derivative of a function f of one variable measures the rate at which the slopes change, we can use f'' to test whether the graph of f is convex or concave. In particular, when $f''(\bar{x})$ is positive (negative) we know that the graph of f is strictly convex (concave) near \bar{x} (this is the basis for the second-derivative test from calculus for functions of one variable). Something that may not have been covered in your calculus class is that the magnitude of the second derivative can be used to compare different degrees of convexity or concavity. This is because the second derivative measures the rate of change of the slopes along the graph, and a higher rate of change of slope means a higher degree of convexity or concavity. To see what different degrees of convexity might look like, we consider the two simple functions $f_1(x) := x^2$ and $f_2(x) := x^4$ which are graphed in Figure 5.2. For input values of x larger than about $1/2$, we can see that the graph of x^2 is relatively less convex than the graph of x^4 in the sense that the steepening of the slopes in the first case is less severe than in the latter. It is easy to compute the second derivatives in this case to be $f_1''(x) = 2$ and $f_2''(x) = 12x^2$, from which we can deduce that $f_1''(x) < f_2''(x)$ as long as $x > \sqrt{1/6} = 0.408248$, which is consistent with our eyeball guess from the graphs of about $1/2$.

The graph of a two-variable function is convex (concave) if every two-dimensional slice by a plane perpendicular to the x - y plane produces a curve that is convex (concave). A pair of positive (negative) eigenvalues for the second-derivative matrix $\nabla^2 f(\bar{x}, \bar{y})$ indicates that near (\bar{x}, \bar{y}) the graph of f is strictly convex (strictly concave). On the other hand, mixed-sign eigenvalues indicate that some slices produce strictly convex curves, and some produce strictly concave curves, which means that the graph of f is saddle-like near (\bar{x}, \bar{y}) .

Just as the relative magnitude of f'' gives the degree of convexity or concavity in the one-variable case, the relative sizes of the eigenvalues of $\nabla^2 f(\bar{x}, \bar{y})$ give the

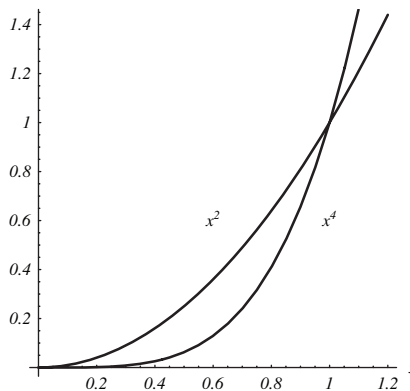


Figure 5.2. *Graphs of x^2 and x^4*

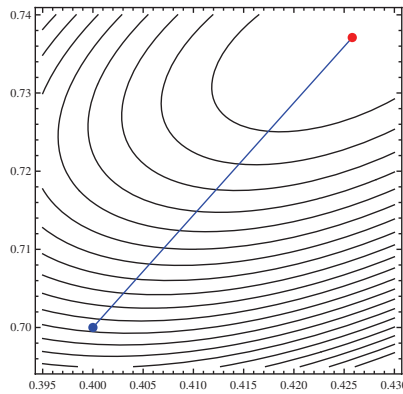


Figure 5.3. *No zigzagging in Newton's optimization method*

relative degree of the convexity or concavity of different slices. Thus, a pair of positive eigenvalues having different scales indicates that some slices have relatively small convexity and some have relatively large convexity. This is just the kind of behavior exhibited by a chute.

The larger the discrepancy in scale of the eigenvalues, the closer the linear order of convergence constant δ may be to one, and the worse the linear order of convergence. On the other hand, a smaller discrepancy in scale of the eigenvalues corresponds to a constant δ closer to 0 and a better linear order of convergence.

Computational Problem 5.3. *Revisit your results from Computational Problem 5.2, and compute the eigenvalues of the second-derivative matrix at the minimizer in each case. Explore the relationship between the discrepancy in scales of the eigenvalues and the linear order of convergence constant δ .*

Newton's optimization method, on the other hand, does not typically exhibit the zigzagging problem, as is illustrated in Figure 5.3, where the same initial point is used as in Figure 5.1. In terms of orders of convergence, then, the steepest descent method is quite slow, with a linear order of convergence, sometimes with a constant δ of nearly one, whereas Newton's optimization method is quite fast, with a quadratic order of convergence. This same kind of disparity is common for all quasi-Newton methods relative to Newton's optimization method, which is the price to pay for the guaranteed descent and simpler computations typical of the quasi-Newton methods. In practical terms, this means that Newton's optimization method is often a better choice than a quasi-Newton method when the current guess is near the minimizer, since that's when the theoretical orders of convergence are most likely to actually be observed. Moreover, when the current guess is near the minimizer, the second-derivative matrix is often already positive-definite, in which case it doesn't need to be replaced to ensure descent.

On the other hand, when the current guess is not necessarily near the minimizer, a quasi-Newton method is often better since it guarantees descent whereas Newton's optimization method does not. Because of this, a good hybrid scheme is to start with a quasi-Newton method until the gaps between subsequent guesses are small, which likely indi-

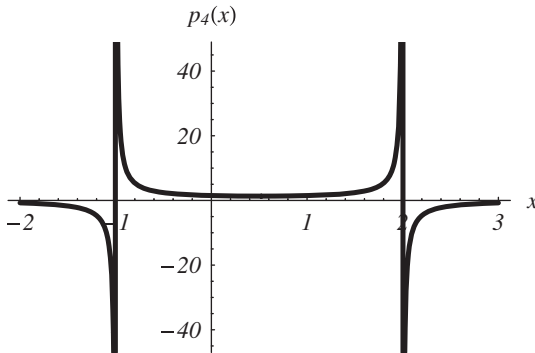
cates nearness to a minimizer. Then switch to Newton's optimization method to quickly refine the solution. Another alternative is to use the quasi-Newton method only when the second-derivative matrix is not definite.

5.3 Conclusions from Chapter 5

In this chapter, we introduced three *orders of convergence* with which to compare different methods:

- Linear: Slowest rate.
- Superlinear: Faster rate.
- Quadratic: Fastest rate.

We showed that Newton's optimization method enjoys a quadratic order of convergence and saw that the steepest descent method has a linear order of convergence. We also considered the ramifications of a linear rate constant near one and explored connections between the eigenvalues of the second-derivative matrix and the size of the linear rate constant.



Chapter 6

Constraints

Nearly every facet of the field of derivative-based optimization could be covered by this book so far if only the real world did not present one small difficulty: Most real optimization problems are relatively rarely expressible as the minimization or maximization of a simple function of any number of variables. This isn't merely the result of our inability to find the functions governing the real processes to be optimized, since functions can always be used that approximately model the real thing. The difficulty is actually caused by certain restrictions or *constraints* on the variables.

When constraints are present, the function being optimized is usually called the **objective function** to distinguish it from any functions that may be used to define the constraints.

In the monthly payment example with variable interest rate, the interest rate z was assumed to depend on the term x and rebate y according to the formula

$$z = \frac{10^9}{(108 - x)^2(30 - y)^5}.$$

This formula can be viewed simply as an equation constraint on the variables and can be expressed equivalently as $g(x, y, z) = 0$ for the *constraint function*

$$g(x, y, z) = z - \frac{10^9}{(108 - x)^2(30 - y)^5}.$$

The way we dealt with this constraint previously is one way to deal with equation constraints in general: The equation is used to reduce the variables in the objective function (in this case by replacing z with its equivalent expression in terms of the variables x and y). The result was the monthly payment function (2.3) that depended on only two variables and had the equation constraint built in.

Inequality constraints might also appear in the monthly payment example since, for instance, both the term x and the rebate amount y are obviously greater than or equal to zero, and the rebate amount y cannot exceed 30 (the price of the automobile). When the monthly payment function (2.3) was minimized in the preceding sections, these inequality

constraints were not enforced by the solution method, and the stationary point just coincidentally happened to satisfy $0 \leq x$ and $0 \leq y \leq 30$. It is also likely that the term x is bounded above as well as below $0 \leq x \leq 108$ since no lending agency has unlimited patience. Beyond commonsense constraints like these, there are many other kinds of constraints that might be present in a given optimization problem, and these might have profound effects on the solutions obtained. To see another example of how constraints arise in applications, we look at a different version of the aerodynamic problems we studied in Chapter 2.

Example: Newton's Aerodynamics Problem in Rare Media

Recall from Chapter 2 that Newton found the resistance of a surface of revolution defined by the function $x(t)$ to be a constant multiple of the integral (2.5). What we didn't mention in Chapter 2 is that Newton found that, as long as the surface-generating function satisfies the constraint $x'(t) \geq 0$, the resistance in a "rare" medium was equal to a constant multiple of a different integral (cf. [7]):

$$\int_0^1 \frac{t}{1 + x'(t)^2} dt. \quad (6.1)$$

(Though Newton didn't know it at the time, his notion of a rare medium was a good description of the very high altitudes of our atmosphere. Once derided because it relied on an absurd physical law, Newton's work on rare media has been validated in the past half-century as high-altitude travel has become common.) Recall too from Chapter 2 that we used discretization to convert an integral optimization problem with a continuum of variables (in the form of the surface-generating functions $x(t)$) to an optimization problem with a finite number of variables.

Problem 6.1. Complete the following:

- (a) Which of the surface-generating functions $x(t)$ whose graphs are pictured in Figure 2.4 satisfy the constraint $x'(t) \geq 0$?
- (b) What does the constraint $x'(t) \geq 0$ say in general about a surface-generating function?
- (c) Give the single-variable function that results from a discretization of the integral (6.1) using t -values 0, $1/2$, and 1.
- (d) How does the constraint $x'(t) \geq 0$ on the surface-generating function translate into constraints on the single variable in this discretization?

There are two main approaches for solving general constrained optimization problems, but both essentially boil down to the same thing: Remove the explicit constraints by building them into the objective function, and then use the methods outlined in Chapters 3 and 4. One main approach is the *reduction method*, which generalizes the tactic for optimizing the monthly payment function (2.4) under the equation constraint (2.2) by using the constraints to reduce the variables in the objective function. The other basic approach penalizes violation of the constraints by adding new terms to the objective function.

6.1 Penalty Methods

Penalty methods incorporate constraints directly in the optimization by penalizing their violation. To see how this works, we consider the following single-variable example.

Example: A Simple Constrained Problem

Suppose for some objective function $f(x)$ we want to solve the constrained optimization problem

$$\min f(x) \text{ over all real variables } x \text{ in the interval } [-1, 2].$$

Depending on the form of the objective function, there can be many ways to solve this problem. One way that essentially always works is to instead solve the *penalty problem*

$$\min f(x) + p(x) \text{ over all real variables } x, \quad (6.2)$$

where $p(x)$ is some *penalty function* with relatively large values at points outside the interval. The ideal penalty function in this case would be

$$p(x) = \begin{cases} 0 & \text{when } -1 \leq x \leq 2, \\ \infty & \text{otherwise,} \end{cases} \quad (6.3)$$

because then the penalty problem (6.2) automatically disregards variables x outside the interval $[-1, 2]$. Note that this kind of function is quite different from those used in calculus. In particular, you can't graph a function with an output of infinity in the usual way since the vertical axis can't be extended that far (at least not on paper). On the other hand, the ideal penalty function (6.3) assigns outputs to inputs just like any other function as long as you allow infinity as an output.

Exercise 6.1.1. Consider the objective function $f(x) = (x+2)^2$ and the ideal penalty function $p(x)$ given by (6.3).

- Explain why the minimization of $f(x)$ over the interval $[-1, 2]$ is equivalent to the penalty problem (6.2).
- Sketch the graph of the sum function $f(x) + p(x)$ and identify two points of discontinuity.

Unfortunately, an ideal penalty function like (6.3) is rarely practical because it is not continuous, much less differentiable, which dooms any method for solving the penalty problem (6.2) that relies on derivatives (e.g., Newton's optimization method and its modifications). The practical compromise usually made is to use a better behaved (i.e., differentiable) penalty function for which the penalty problem (6.2) is close enough to the original constrained optimization problem. Different choices of penalty functions have different features and yield different results, which is where this approach gets really interesting.

6.1.1 A Simple Differentiable Penalty Function

One easy way to manufacture well-behaved penalty functions involves first converting any equation constraints $h(\mathbf{x}) = 0$ into a pair of inequalities:

$$h(\mathbf{x}) = 0 \iff h(\mathbf{x}) \geq 0 \text{ and } h(\mathbf{x}) \leq 0.$$

In this way, the constraints (whether equation or inequality to begin with) can be reformulated entirely as inequality constraints of the form $g(\mathbf{x}) \geq 0$.

Exercise 6.1.2. Explain how you can convert any inequality like $h(\mathbf{x}) \leq 0$ into an inequality of the form $g(\mathbf{x}) \geq 0$ with the right choice of g .

Once all the constraints are of the form $g(\mathbf{x}) \geq 0$, a simple penalty function can be constructed by adding terms like $-\lambda g(\mathbf{x})$ with positive constants λ for each constraint function. Because a point \mathbf{x} at which a constraint is violated satisfies $g(\mathbf{x}) < 0$ and because $\lambda > 0$, the penalty term $-\lambda g(\mathbf{x})$ is positive for such points. It follows that these penalty terms truly penalize constraint violation in the minimization.

For instance, the interval $[-1, 2]$ in our simple constrained problem corresponds to two inequalities $x \geq -1$ and $x \leq 2$. These two inequalities translate into a pair of constraints of the form $g(x) \geq 0$ with the two constraint functions $g_1(x) = x + 1$ and $g_2(x) = 2 - x$, which results in the simple penalty function

$$p(x) = -\lambda_1(x + 1) - \lambda_2(2 - x). \quad (6.4)$$

The trouble with a penalty function like this is that it not only penalizes constraint violation but also rewards constraint satisfaction and does so with increasing rewards the farther from the boundary a point is. For instance, the first term $-\lambda_1(x + 1)$ in the penalty function (6.4) tends to negative infinity as x grows larger. An unfortunate consequence of rewarding constraint satisfaction in this way is that the penalized minimization can inappropriately favor points that have very little connection to the minimizers of the original problem.

Problem 6.2. Complete the following:

- (a) Using constants obeying $\lambda_2 > 1/2 + \lambda_1$ add the penalty function (6.4) to the function $f(x) = -x/2$ and try to find a penalized minimizer.
- (b) What happens if you construct the penalty function instead with constants obeying $\lambda_2 < 1/2 + \lambda_1$?
- (c) For what choices of constants will there be penalized minimizers?
- (d) What is the minimizer of $f(x) = -x/2$ over the interval $[-1, 2]$?
- (e) Now using $\lambda_2 = 1$, add the penalty function (6.4) to the objective function $f(x) = (x + 2)^2$. What penalized minimizers are possible considering all the choices of $\lambda_1 > 0$?
- (f) What is the minimizer of $f(x) = (x + 2)^2$ over the interval $[-1, 2]$?

6.1.2 Improved Penalty Functions

We have seen that the main problem with simple penalty functions like (6.4) is that they not only penalize constraint violation but also reward constraint satisfaction. One way to avoid this problem is to instead add a min-type term like

$$-\min\{g(\mathbf{x}), 0\}$$

for each inequality constraint $g(\mathbf{x}) \geq 0$. As in the case of the simple penalty terms, such a min-type term adds a positive penalty value of $-g(\mathbf{x})$ to the minimization when the corresponding constraint is violated. However, now nothing is added when the constraint $g(\mathbf{x}) \geq 0$ is satisfied, so artificial rewards are not created for constraint satisfaction.

Exercise 6.1.3. What analogous penalty function could you use if the original problem was one of maximization instead of minimization?

The min-type penalty function associated with the interval $[-1, 2]$ is

$$p_1(x) = -\min\{(x+1), 0\} - \min\{(2-x), 0\}, \quad (6.5)$$

whose graph is pictured in Figure 6.1. It is clear from the sharp corners of its graph that, even though it is continuous everywhere, the min-type penalty function is not differentiable at the endpoints of the interval. So, just as in the case of the ideal penalty function (6.3), we cannot apply methods that rely on derivatives to solve the penalty problem (6.2).

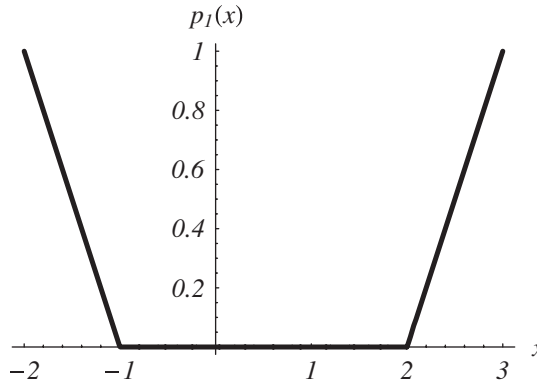


Figure 6.1. Min-type penalty function $p_1(x)$ for $[-1, 2]$

Differentiable Penalty Functions

It would seem then that our modification to avoid artificial rewards gets us right back to where we started with differentiability trouble. However, the sharp corners on the graph of the min-type penalty function (6.5) can be easily smoothed out by squaring each min-type term. The resulting penalty function is

$$p_2(x) = -\min\{(x+1), 0\}^2 - \min\{(2-x), 0\}^2, \quad (6.6)$$

whose graph appears in Figure 6.2. You should recall that Newton's optimization method relied not only on first derivatives but on (continuous) second derivatives as well, so we really want a penalty function that has these properties. Unfortunately, the smoothed penalty function $p_2(x)$ (6.6) is not such a penalty function.

Exercise 6.1.4. Using Figure 6.3, explain where $p_2(x)$ is not twice differentiable.

Twice-Differentiable Penalty Functions

One easy way to get (continuous) second derivatives in the penalty function is to cube rather than square each min-type term in (6.5). The result is the penalty function

$$p_3(x) = -\min\{(x+1), 0\}^3 - \min\{(2-x), 0\}^3, \quad (6.7)$$

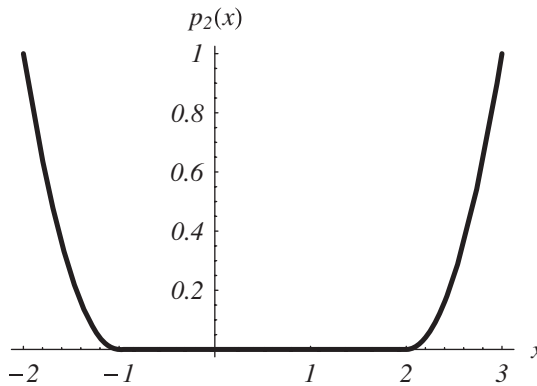


Figure 6.2. Smoothed penalty function $p_2(x)$ for $[-1, 2]$

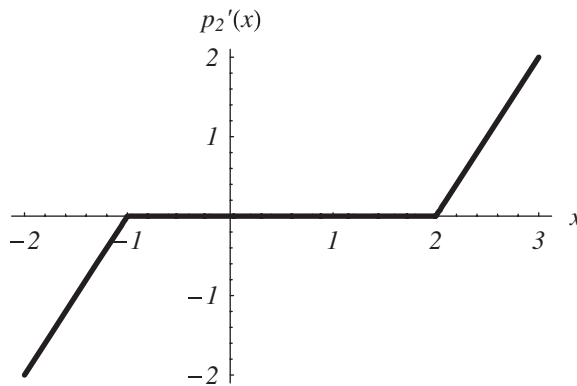


Figure 6.3. Derivative of the penalty function $p_2(x)$ for $[-1, 2]$

whose graph appears alongside the graph of its derivative in Figure 6.4. In summary, then, for any constraint of the form $g(\mathbf{x}) \geq 0$, the penalty term

$$-\min\{g(\mathbf{x}), 0\}^3$$

possesses all the desirable properties we have identified so far: It both penalizes constraint violation without rewarding constraint satisfaction and is amenable to solution methods like Newton's optimization method.

Unfortunately, there is at least one desirable property that this penalty construction does not possess, which we explore by again examining the simple minimization problem from Exercise 6.1.1.

Exercise 6.1.5. Consider the minimization problem

$$\min (x+2)^2 \text{ over } x \text{ contained in the interval } [-1, 2] \quad (6.8)$$

and the penalty function $p_3(x)$ (6.7) corresponding to the constraint interval $[-1, 2]$:

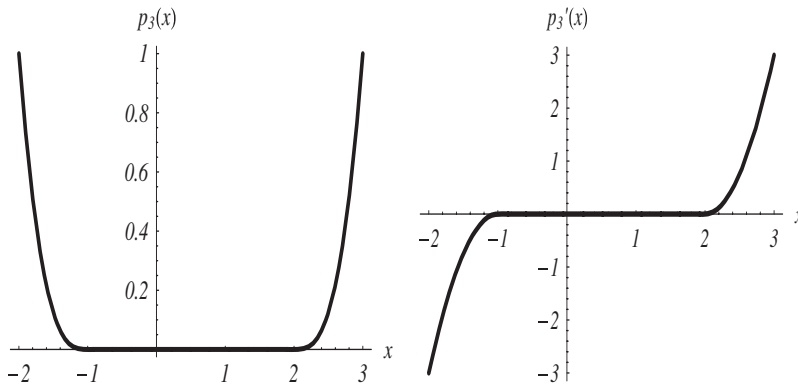


Figure 6.4. Smoothed penalty function $p_3(x)$ for $[-1, 2]$ and its derivative

- (a) Use Figure 6.5 to estimate the penalized minimizer.
- (b) Does the penalized minimizer solve the original minimization problem (6.8)?

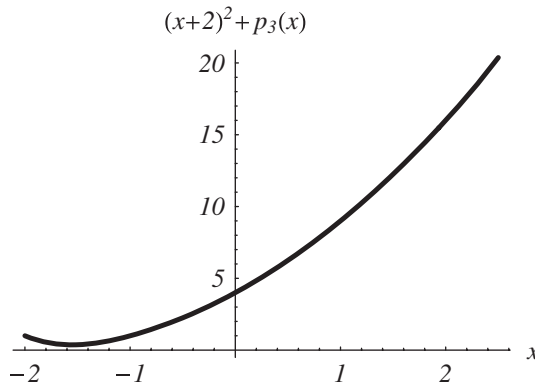


Figure 6.5. Graph of $(x+2)^2 + p_3(x)$

Evidently, a penalized minimizer can fail to obey the original constraints, which is actually a problem typical of many penalty methods. To improve this situation, the penalty terms could be amplified by some positive factor,

$$-\lambda \min\{g(\mathbf{x}), 0\}^3,$$

as in the case of the simple penalty function (6.4). By doing this, the penalized minimizers can generally be driven closer to the constraint region, but often there is no factor λ for which they actually obey the constraints. Notice that the ideal penalty function can essentially be reproduced in this construction by driving λ to infinity, so for larger values of λ the (theoretically well-behaved) penalty terms tend to inherit the bad behavior exhibited by the ideal penalty function. Thus, the desire to drive the factor λ up in order to move the penalized minimizer toward the constraint region conflicts with the desire to keep the

functions well behaved so that the optimization methods can solve them accurately. This conflict is accommodated in practice by using very large values of any factors λ only when the method appears to be near a solution.

Computational Problem 6.1. *Again consider the discretization using t -values 0, 1/2, and 1 of Newton's aerodynamics problem in rare media from Problem 6.1. Recall that the constraint $x'(t) \geq 0$ in this case translates into a simple interval constraint on the single variable of the discretization problem.*

- (a) *Using general factors $\lambda_1 > 0$ and $\lambda_2 > 0$, create an appropriate twice-differentiable penalty function p_3 for the interval in this case.*
- (b) *Explore the effect on the penalized minimizers of different choices of factors until you can answer the following questions:*
 - (i) *Is there any combination of factors for which the penalized minimizer solves the original discretization problem?*
 - (ii) *Do changes in each of the factors λ_1 and λ_2 have similar or very different effects on the location of the corresponding penalized minimizers?*
 - (iii) *For which values of factors λ_1 and λ_2 does the penalty function approach the ideal penalty function for the interval constraint in this case?*
 - (iv) *Does this approach happen graphically from the interior or the exterior of the interval constraint?*

6.1.3 Barrier Functions

Sometimes it isn't such a big deal if the penalized minimizers violate the original constraints. In the case of the monthly payment example, it might be acceptable if the term limit $x \leq 108$ isn't strictly enforced. On the other hand, some constraints are truly inviolable, like the lower term limit $x \geq 0$ (since negative terms make no practical sense). In situations when the constraint is inviolable, a penalty function needs to be used for which the penalized minimizers are guaranteed to obey the constraints. One way to enforce an inviolable constraint $g(\mathbf{x}) \geq 0$ is to add its inverse

$$1/g(\mathbf{x})$$

as a penalty term. For example, the constraint interval $[-1, 2]$ is enforced by the penalty function

$$p_4(x) = \frac{1}{x+1} + \frac{1}{2-x}, \quad (6.9)$$

whose graph is pictured in Figure 6.6. Another popular choice is to instead subtract the logarithmic function

$$-\log(g(\mathbf{x})),$$

which in the case of the constraint interval $[-1, 2]$ yields

$$p_5(x) = -\log(x+1) - \log(2-x), \quad (6.10)$$

whose graph is pictured in Figure 6.7.

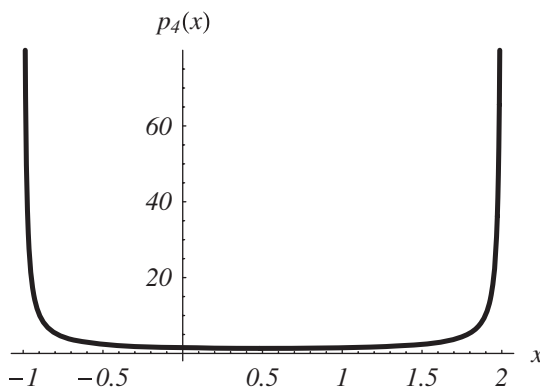


Figure 6.6. Inverse barrier function $p_4(x)$

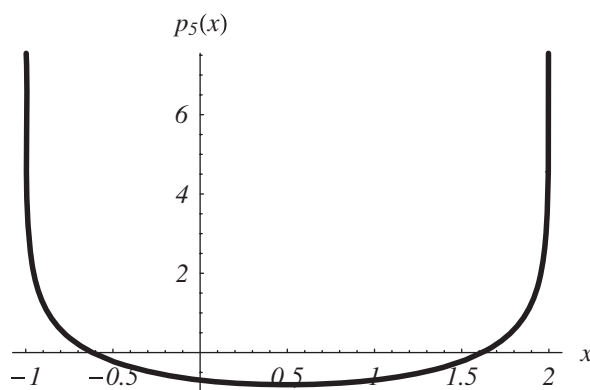


Figure 6.7. Logarithmic barrier function $p_5(x)$

Penalty functions of this type are often called **barrier functions** since these functions are infinite-valued on the boundary of the constraint region.

Thus, the effect of a barrier function on a penalized minimization is similar to that of the ideal penalty function (6.3), with infinite penalties keeping the penalized minimizers in the constraint region. In the case of the ideal penalty function, though, boundary points are not off-limits as penalized minimizers, which is not true for these barrier functions. Therefore, barrier functions cannot be used with equality constraints, since every point in the constraint region is a “boundary” point.

Exercise 6.1.6. Complete the following:

- Give the penalty function resulting from the sum of the two inverse barrier functions associated with the pair of inequalities $x \leq 0$ and $x \geq 0$.
- Explain how this penalty function fails to penalize violation of the equation constraint $x = 0$.

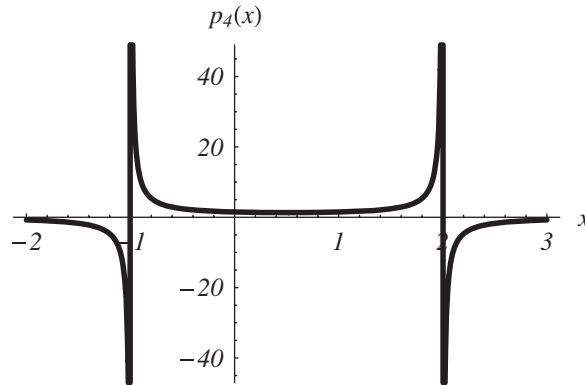


Figure 6.8. Inverse barrier function $p_4(x)$, wide view

The effects of the terms in barrier functions can be amplified by a constant parameter $\lambda > 0$ in the same manner as with other penalty functions. Thus the constraint $g(\mathbf{x}) \geq 0$ can be replaced with $\lambda/g(\mathbf{x})$ or $-\lambda \log(g(\mathbf{x}))$.

Computational Problem 6.2. Complete the following:

- (a) Use factors $\lambda_1 > 0$ and $\lambda_2 > 0$ to appropriately modify the inverse barrier function p_4 (6.9).
- (b) Do the same with the logarithmic barrier function p_5 (6.10).
- (c) Explore the effect of different choices of factors until you can answer the following questions for each of p_4 and p_5 :
 - (i) For which values of factors λ_1 and λ_2 do the penalty functions approach the ideal penalty function (6.3)?
 - (ii) Does this approach happen graphically from the interior or the exterior of the interval $[-1, 2]$?
- (d) Compare your results here for barrier functions to your results for penalty functions from Computational Problem 6.1.

6.1.4 Impassable Barriers?

Unfortunately, the barrier functions discussed so far don't necessarily create an impassable barrier. To see the problem, consider Figure 6.8, which shows a slightly wider view than that in Figure 6.6 of the graph of the inverse barrier function associated with the constraint interval $[-1, 2]$. When approaching the boundary from the interior of the interval $[-1, 2]$, the barrier function indeed assigns an increasing penalty for minimization by adding higher and higher positive values. On the other hand, there is incentive for minimization lurking on the other side of the boundary since the barrier function takes on negative values. So points outside the constraint region are attractive to any minimization method that can

skip past the steep penalty at the boundary, which entirely defeats the purpose of the barrier function. A similar problem exists for the logarithmic barrier function, except there the function is not defined for values outside the constraint region. In this case, any minimization method that skips past the boundary will immediately fail since the penalized objective function is not defined there.

Evidently, then, barrier functions don't actually create the barriers they are supposed to as long as the subsequent minimization methods can skip past the boundary of the constraints. Unfortunately, such skipping cannot be automatically ruled out in general for any of the minimization methods presented in Chapters 3 and 4. We illustrate this by considering the (inverse barrier) penalty problem

$$\min -10x^2 + \frac{1}{x} + \frac{1}{1-x} \quad (6.11)$$

corresponding to the constrained optimization problem

$$\min -10x^2 \text{ over } x \text{ in the interval } [0, 1].$$

Figure 6.9 shows the graph of the penalized objective function from (6.11) plotted in solid, and the dashed plot is of the Taylor quadratic at a guess in the interior of the interval $[0, 1]$. Clearly, if the new guess is taken as the minimizer of the Taylor quadratic, then the method will skip past the boundary in the next step. All of the quasi-Newton methods discussed so far exhibit similar skipping problems, and fixes typically depend on the method being used.

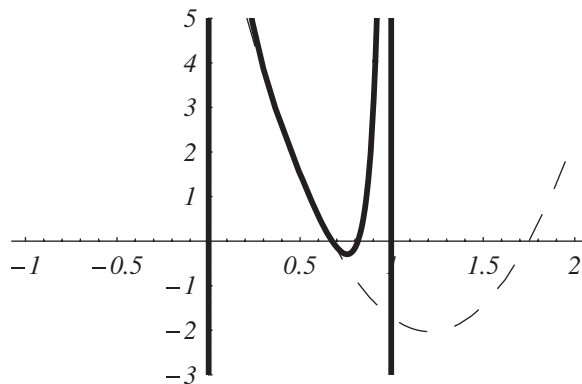


Figure 6.9. Taylor quadratic for barrier function minimization

Exercise 6.1.7. Explain why the steepest descent method with a backtracking line-search applied to the same penalty problem (6.11) could terminate with a new guess beyond the boundary of the constraint region (notice that in this case there are only two step-directions to choose from since there is only one variable).

One general approach is to carefully pick a step-factor for each new guess that keeps it from extending beyond the boundary of the constraints. This usually means using some scheme other than the basic line-searches already discussed. For Newton's optimization

method applied to barrier functions, instead of moving all the way to the stationary point of the Taylor quadratic, some step-factor in that direction could be taken. One such “damped” Newton’s optimization method picks a step-factor according to a formula that depends on the second-derivative matrix at the current guess in such a way that the new guess is guaranteed to obey the constraints. In general, as long as the initial guess obeys the constraints and subsequent step-factors are chosen carefully, the result of using barrier functions is to have genuine barriers at the boundaries of the constraints.

6.2 The Reduction Method

6.2.1 Converting Inequalities to Equations

In your first calculus course, you learned how to cope with simple inequality constraints when optimizing a function of one variable. The typical inequality constraints in that case restricted the single variable to some interval, and the strategy was pretty simple: Candidates for local optimizers are the function’s stationary points within the interval as well as the boundaries (i.e., endpoints) of the interval. No matter how many variables are being used, this fact still holds, so one strategy for solving inequality-constrained optimization problems is simply to first find all the stationary points obeying the constraints and then to analyze the boundary points. Finding stationary points obeying the inequality constraints is relatively easy no matter how many variables are being used: The methods in Chapters 3 and 4 can be applied to the objective function, with the addition of a final step to check if the solution obtained actually obeys the constraints. However, analyzing the boundary points when there is more than one variable tends to be more complicated since the boundaries of the inequality constraints tend to be more complicated. In particular, there are typically infinitely many boundary points as opposed to two in the case of a simple interval constraint on one variable.

Exercise 6.2.1. *Complete the following:*

(a) *On different copies of the x - y plane, sketch each of the following constraint regions:*

Constraint Region 1: $0 \leq x \leq 108$ and $0 \leq y \leq 30$;

Constraint Region 2: $0 \leq x$, $0 \leq y$, and $x + y \leq 30$;

Constraint Region 3: $0 \leq x$, $0 \leq y$, and $x y + e^{\frac{1}{xy}} \leq 30$.

(b) *Identify the boundaries of each of these constraint regions both graphically and by their defining equations.*

Instead of directly testing the optimality of each point on the boundary, the focus is put on points which are locally optimal relative to that piece. In this way, separate optimization problems are generated for each piece of the boundary. To see how this works, consider minimizing a generic function of two variables $f(x, y)$ over the nonnegative variables $x \geq 0$ and $y \geq 0$. The pairs (x, y) obeying both of these constraints define the upper right quadrant of the x - y plane. The separate minimization problems are generated by ignoring one of the constraints (say $y \geq 0$) and optimizing over the boundary of the other (in this case $x = 0$). The optimization problem for this piece of the boundary is thus

$$\min f(x, y) \text{ over pairs } (x, y) \text{ obeying } x = 0. \quad (6.12)$$

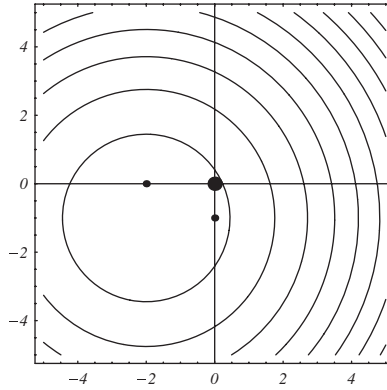


Figure 6.10. Contour plot of $(x + 2)^2 + (y + 1)^2$ with subproblem solutions

Then, the other boundary produces the optimization problem

$$\min f(x, y) \text{ over pairs } (x, y) \text{ obeying } y = 0. \quad (6.13)$$

It is important to collect all the local minimizers since global minimizers to these subproblems might not satisfy all the constraints of the original problem. Notice that each of these problems could have local minimizers that do not obey the original constraints since they completely ignore one inequality while using the boundary of the other.

For example, consider the objective function given by

$$f(x, y) = (x + 2)^2 + (y + 1)^2,$$

whose contour plot is pictured in Figure 6.10. In this case, the only local minimizer associated with the constrained problem (6.12) is the global minimizer $(x, y) = (0, -1)$ and the only local minimizer associated with (6.13) is the global minimizer $(x, y) = (-2, 0)$. Neither of these obeys the constraints in the original problem:

$$\min (x + 2)^2 + (y + 1)^2 \text{ over pairs } (x, y) \text{ obeying } x \geq 0 \text{ and } y \geq 0, \quad (6.14)$$

whose solution is evidently $(x, y) = (0, 0)$. To make up for the fact that the problems (6.12) and (6.13) each include only one piece of the boundary, an additional equation constrained optimization problem needs to be considered at the intersection of these two pieces of the boundary:

$$\min f(x, y) \text{ over pairs } (x, y) \text{ obeying } x = 0 \text{ and } y = 0. \quad (6.15)$$

Essentially, the equation version of each inequality covers the interior of the corresponding piece of the boundary, and the intersection covers the boundaries of the boundary pieces. In our example (6.14), the final boundary-intersection problem (6.15) identifies the solution $(x, y) = (0, 0)$ which the problems (6.12) and (6.13) did not.

In general, the same procedure is followed: Consider all the combinations of equations corresponding to each component of the boundary of the constraint region. Then separately find the local optimizers for each of the subproblems generated by a different combination of equation constraints, and test these local optimizers to see if they obey the

original constraints. In problems with inequality *and* equation constraints originally, the original equation constraints are always included in each subproblem since every boundary point of the constraint region must satisfy these equations.

Exercise 6.2.2. *Complete the following:*

- (a) *Write the seven different equation constrained problems generated by the boundary in the case of the minimization of a generic function $f(x, y, z)$ under the constraints of nonnegativity of all three variables.*
- (b) *Describe the sets of points in three-dimensional space corresponding to each of the seven combinations of equations.*

6.2.2 Reducing Variables

From the preceding discussion, it is evident that to solve any optimization problem with inequality constraints, we need only consider subproblems with equation constraints (as well as the unconstrained optimization of the objective function). Because of this, attention will be restricted for the remainder of this section to optimization under equation constraints. Recall that in the process of optimizing the monthly payment function (2.4) under the equation constraint (2.2), this equation was used to directly replace the z 's in the monthly payment function, thus eliminating the explicit constraint and reducing the number of variables. To see how this might be imitated in situations with more equation constraints, we first restrict our attention to equation constraints defined by linear functions.

Reduction for Linear Equation Constraints

Consider the pair of linear equations

$$\begin{aligned}x + y + z &= 30, \\x - y + 2z &= 10.\end{aligned}\tag{6.16}$$

Exercise 6.2.3. *Complete the following:*

- (a) *Solve the first equation in the pair (6.16) for x in terms of the independent variables y and z .*
- (b) *Now solve the pair of equations for x and y in terms of the independent variable z only.*
- (c) *How many independent variables would you typically expect for a system of m equations involving n variables?*

It turns out that the variables in a system of equations can always be reduced like this, as long as all of the equations in the system are consistent (for instance, the equations $x + y + z = 30$ and $x + y + z = 40$ are not consistent). Notice that if the system does contain inconsistent equations, then there is no solution which simultaneously obeys all of the equations, so an optimization problem subject to these equation constraints would have no solution.

Since any solution to the pair of equations (6.16) can be represented as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 20 - 3z/2 \\ 10 + z/2 \\ z \end{bmatrix}$$

for the single independent variable z , to optimize any three-variable function $f(x, y, z)$ subject to these two equation constraints amounts to optimizing the *reduced objective function*

$$rf(z) = f(20 - 3z/2, 10 + z/2, z).$$

The notation rf and the label **reduced objective function** remind us that this is the reduced-variable version of the objective function f .

So, just as in the case of the single constraint (2.2) in the minimization of the monthly payment function, multiple equation constraints may be used to reduce the number of variables by directly incorporating the constraints into the optimization. Optimization subject to *linear* equation constraints can thus be done via the unconstrained optimization of reduced objective functions (using Newton's optimization method and its modifications).

Reduction Method

1. **Construct the subproblems** from each possible combination of equation constraints generated by the boundary of the constraint region.
2. **Reduce variables** in each subproblem and define the corresponding reduced objective functions.
3. **Find the local optimizers** of each reduced objective function (e.g., via Newton's optimization method).
4. **Compare** the local optimizers for each reduced objective function and choose the best among them that satisfies the original constraints.

Implementation 3. *Use the reduction method to solve the inequality constrained optimization problem resulting from a discretization using t values 0, 1/4, 1/2, 3/4, and 1 of Newton's aerodynamics problem in rare media: Find the surface-generating function $x(t)$ (passing through (0,0) and (1,1)) satisfying $x'(t) \geq 0$ and minimizing the integral (6.1)*

$$\int_0^1 \frac{t}{1 + x'(t)^2} dt.$$

What If the Equations Aren't Linear?

Not all equation constraints are linear, and dealing with nonlinear equations can be pretty tricky. One example of this was seen in Problem 2.2, where the exact solution to a pair of nonlinear equations was unsuccessfully sought for the stationary points of the monthly payment function (2.3). In a really lucky situation, the nonlinear equations might come in a useful form to begin with (as in the case of the monthly payment constraint (2.2)), but even when this is the case, direct substitution can lead to incorrect solutions.

Problem 6.3. *Complete the following:*

- (a) *Solve the optimization problem*

$$\min x^2 + y^2 \text{ over pairs } (x, y) \text{ obeying } y^2 = x - 1$$

by directly substituting y^2 from the constraint into the objective function and minimizing the resulting function of one variable.

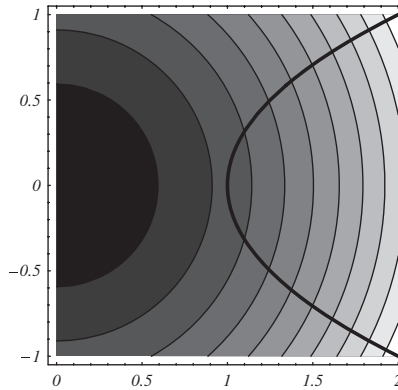


Figure 6.11. Contour plot of $x^2 + y^2$ and graph of curve $y^2 = x - 1$

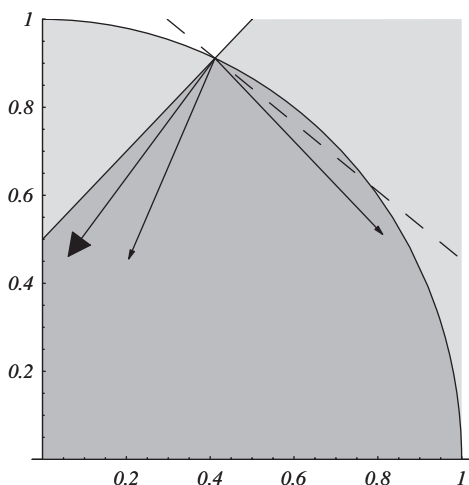
- (b) Now use Figure 6.11 to solve the same problem graphically.
- (c) Compare your solution via direct substitution with your graphical solution and explain the differences.

Evidently, direct substitution is not reliable even when it is possible. One popular alternative to direct substitution is to use derivatives to approximate any nonlinear equations by linear equations, which can then be reduced via the method in the preceding section (while also updating the linear approximations at each iteration). In this case, the objective function is often modified by adding a penalty for violating the original nonlinear equations. Another popular and related approach to solving optimization problems with nonlinear constraints will be covered in Section 7.3.2.

6.3 Conclusions from Chapter 6

In this chapter, we learned about two of the main strategies for incorporating constraints with the objective function, so that constrained optimization problems can be solved via methods designed for unconstrained problems:

- Penalties are added for constraint violation. For this case, we examined the important qualities of penalty functions including their ability to act as a barrier keeping iterates inside the constraint region.
- The number of variables is reduced by solving (linear) constraining equations. When inequality constraints are present, we saw that the equation-only subproblems corresponding to the components of the boundary of the constraint region need to be considered separately.



Chapter 7

More Practical Modifications

In Chapter 6, several methods for constrained optimization were presented and some of their potential difficulties and consequent modifications were discussed. In the current chapter, this discussion will be expanded, with particular focus on a few new practical issues.

7.1 Reduction Modifications

As seen in Subsection 6.2.1, the reduction method for a problem with inequality constraints involves solving a variety of equation-constrained subproblems that corresponded to different pieces of the boundary of the constraint region. In Exercise 6.2.2, seven different subproblems were seen to be spawned by a trio of inequalities in the original problem. The number of subproblems becomes dramatically larger as the number of original inequalities increases even moderately. For instance, 10 original inequalities spawn 1023 subproblems, and 20 original inequalities spawn over one million. You don't even want to know how many subproblems are spawned by 50 original inequalities (okay, it's 1,125,899,906,842,623). To put that enormous figure into context, if each separate subproblem could be solved in one second, it would take nearly 36 million years to solve all of them. Moreover, 50 or more original inequalities is by no means unreasonable for a complicated real-world problem, but exhaustively solving the corresponding number of subproblems is likely to be impractical.

If the constraints are defined by linear functions to begin with, then often only some of the full array of subproblems need to be solved without compromising the ultimate solution, as long as the subproblems are chosen carefully. One popular method for doing this is called the *working-set method*.

7.1.1 The Working-Set Method: Hitting the Boundary

The working-set method begins with any one of these subproblems and with a point obeying not only the equation constraints defining this subproblem but also all of the inequalities defining the full constraint region.

The set of equation constraints in the subproblem is called the **working-set** since it is the set on which the work will be done.

The basic idea behind the working-set method is to begin to solve the current subproblem via reduction and a quasi-Newton method with an exact line-search. However, now the exact line-search is terminated prematurely if the boundary of the full constraint region is hit before the line-search would normally terminate. This is illustrated in Figure 7.1, where the inequality constraint $x - y \geq 0$ defines the piece of the boundary that is hit by an exact line-search begun at the base of the arrow and normally terminating at the arrowhead. It is clear from Figure 7.1 that in order to avoid violating the constraint $x - y \geq 0$, the line-search should prematurely terminate at the boundary of the constraint region.

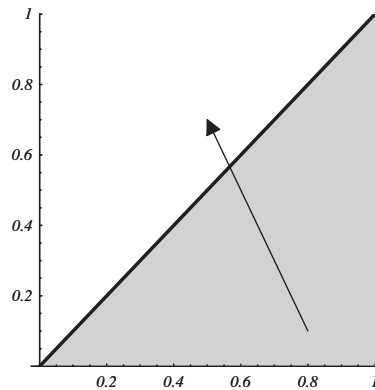


Figure 7.1. Line-search from the constraint region $x - y \geq 0$

Exercise 7.1.1. For the case illustrated in Figure 7.1, where the exact line-search would normally terminate at the arrowhead, explain why it is better to wait until the boundary of the constraint region is hit rather than terminating somewhere in the interior of the region.

Notice that for the constraint region pictured in Figure 7.1, the boundary is the set of points obeying $x - y = 0$. This illustrates a general property that is the basis for the working-set method: Hitting the boundary of a constraint region defined by inequalities corresponds to (at least) one of the inequalities being satisfied as an equation. Because of this, whenever the exact line-search is prematurely terminated at the boundary of the full constraint region, the corresponding equation is added to the working-set. In case the boundary is hit where more than one corresponding equation is satisfied, more than one of these equations can be added to the working-set (cycling can sometimes occur if this isn't done carefully). Then the process is repeated for the new subproblem. If the exact line-search normally terminates before it hits the boundary, the working-set stays the same and the next guess is generated from the quasi-Newton method as usual.

The Steps of the Working-Set Method:

1. Pick any subproblem and any point simultaneously obeying the equations defining the subproblem and the inequalities defining the full constraint region.

2. Apply one step of a quasi-Newton method to the reduced version of the subproblem but where the exact line-search is terminated prematurely if the boundary of the full constraint region is hit.
3. If the line-search is prematurely terminated, add to the working-set the equation (or some number of multiple equations) corresponding to the part of the boundary which was hit, and repeat step 2 for the new subproblem. If the line-search is not prematurely terminated, repeat step 2 for the current subproblem.

Since there are only a finite number of original constraints to consider, this process will not proceed indefinitely but will end at a solution to some subproblem when no descent step-direction is found that obeys the equation constraints. To illustrate how this all works, we'll carefully consider the following example.

Example: The Working-Set Method in Action

We consider the simple optimization problem

$$\begin{array}{ll} \min 2x + y \text{ over } (x, y) \text{ obeying} & \begin{array}{l} x + y \geq 1, \\ x - y \geq -2, \\ -x \geq -2 \end{array} \end{array} \quad (7.1)$$

and use steepest descent as our quasi-Newton method.

1. We need to pick some subproblem among the different possible combinations, for example

$$\min f(x, y) = 2x + y \text{ over } (x, y) \text{ obeying } -x = -2,$$

and some initial guess, say $(x_{\text{old}}, y_{\text{old}}) = (2, 2)$, that satisfies the equation defining this subproblem as well as all the original inequalities.

2. To solve this subproblem via reduction, we can solve the equation constraint for $x = 2$ with only the one independent variable y , in which case the reduced objective function is just $rf(y) = f(2, y) = 4 + y$. Now we minimize $rf(y)$ over all possible values of y by applying steepest descent from the initial guess $y_{\text{old}} = 2$ with an exact line-search.

- The gradient of $rf(y)$ is just its derivative, which is 1, so steepest descent with an exact line-search from the initial guess $y_{\text{old}} = 2$ solves the problem

$$\min rf(2 - (1)\alpha) = 6 - \alpha \text{ over step-factor factors } \alpha.$$

Obviously, this minimization favors the largest possible α , but violation of the unrepresented constraints from the original problem might occur if too large a step-factor factor α is chosen.

- To check if there is any such violation, we note that the points we are considering for this subproblem are of the form $(x, y) = (2, 2 - \alpha)$. The constraint $x - y \geq -2$ is evidently satisfied by the pair $(x, y) = (2, 2 - \alpha)$ as long as $\alpha \geq -2$, and the constraint $x + y \geq 1$ is satisfied by the pair $(x, y) = (2, 2 - \alpha)$ as long as $\alpha \leq 3$.

3. Since in this case the exact line-search favors the largest possible step-factor factor, $\alpha = 3$ is the best choice that does not violate the unrepresented constraints. It follows that the pair $(x_{\text{new}}, y_{\text{new}}) = (2, -1)$ is the next guess and the equation version of the second constraint is added to the working-set of constraints.

Thus, the new subproblem is

$$\min 2x + y \text{ over } (x, y) \text{ obeying } \begin{array}{rcl} -x & = & -2, \\ x + y & = & 1. \end{array} \quad (7.2)$$

But there is only one point $(x, y) = (2, -1)$ obeying both of these equation constraints, which means that this subproblem has solution $(2, -1)$.

The steps of the working-set method in this case are illustrated in Figure 7.2, where the initial guess at $(2, 2)$ is represented by a smaller dot than the final guess at $(2, -1)$ and the arrow indicates the step-direction of steepest descent for the first step. The working-set method in this case moved along the rightmost boundary from the first guess to the final guess in the direction of the arrow.

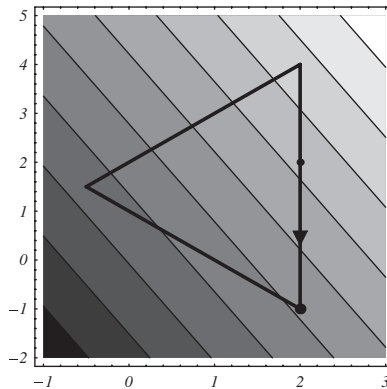


Figure 7.2. *Working-set method*

The main reason the working-set method needs linear constraints to begin with is that otherwise a line from the current guess will not necessarily move to a place where the current subproblem's (equation) constraints are still satisfied. Essentially, nonlinear equation constraints cannot be obeyed along a line.

Exercise 7.1.2. *Use a picture to show that a nonlinear equation constraint on two variables is satisfied very rarely (if at all) along a line through one point obeying the constraint.*

In general, the working-set method finds a solution to some subproblem. It is important to know if this is actually a solution to the original problem and, if it isn't, how to make progress toward a solution to the original problem. The key to this next step comes from a topic that you may have already heard about in your multivariable calculus class and which is relevant even in more general situations when the constraints are nonlinear.

7.1.2 Lagrange Multipliers

In this section, we'll consider the basic optimization problem

$$\min f(x, y) \text{ over } (x, y) \text{ obeying } g(x, y) = 0. \quad (7.3)$$

The idea behind Lagrange multipliers comes from the simple concept illustrated in Figure 7.3. The bold circle represents the set of points (x, y) obeying the constraint $g(x, y) = 0$, and the level curves correspond to the function f being optimized. The first thing to notice is that the optimizers of f relative to the constraint occur at one of the four points indicated on the circle, and the second thing to notice is that at these points the circle is tangent to the level curves of f .

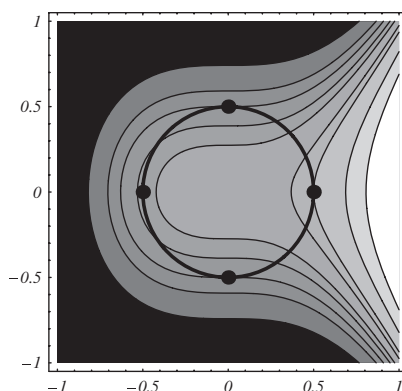


Figure 7.3. *Constrained optimization*

Exercise 7.1.3. Explain why a local constrained optimizer for f occurs only where the constraint curve is parallel to a level curve for f by describing what happens if the constraint curve is not parallel to a level curve for f .

One basic multiplier rule is simply an algebraic expression of this phenomenon. Recall that the gradient of any function at any input point is perpendicular to the function's level curve through that point, since the gradient points in the direction of greatest immediate increase for the function and the level curves are tangent to the direction of no immediate change in the function. Thus, at the four potential optimal points indicated in Figure 7.4, the gradient of f is perpendicular to the circle since the circle is tangent to the level curve through those points. Moreover, since the circle represents the 0-level curve for the constraint function g , the gradient of g at any point on the circle is perpendicular to the circle as in Figure 7.5.

At any of the four potential optimizers, then, the gradients of f and g are parallel since these vectors are both perpendicular to the graphical representation of the constraint there. This is illustrated in Figure 7.6, which is just an overlay of Figures 7.4 and 7.5. Recall that for any two parallel vectors, one can be written as a constant multiple of the other. So the gradient of f being parallel to the gradient of g can be expressed as the vector equation

$$\nabla f(x, y) = \lambda \nabla g(x, y), \quad (7.4)$$

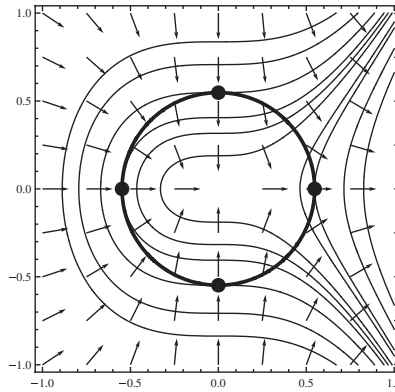


Figure 7.4. Gradient vectors for the objective function f

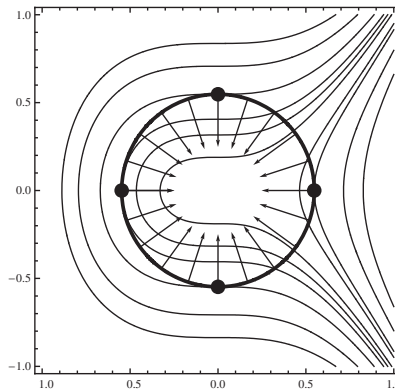


Figure 7.5. Gradient vectors for the constraint function g

where the number λ is the constant *multiplier* that makes the gradient of g look exactly like the gradient of f .

The vector equation (7.4) is the famed **Lagrange multiplier rule**, and it encodes two equations involving the partial derivatives of the two functions f and g (its analogue in the case of n variables encodes n equations).

Problem 7.1. Complete the following:

- Give three different constraint functions $g(x, y)$ whose 0-level curves are the same circle of radius 0.5 pictured in Figures 7.3, 7.4, and 7.5.
- For each of the four potential optimizers pictured in Figure 7.6, say what the sign of the corresponding Lagrange multiplier λ should be.
- For general functions f and g , write the two equations encoded by the vector equation (7.4). There should be three unknown variables in your two equations.

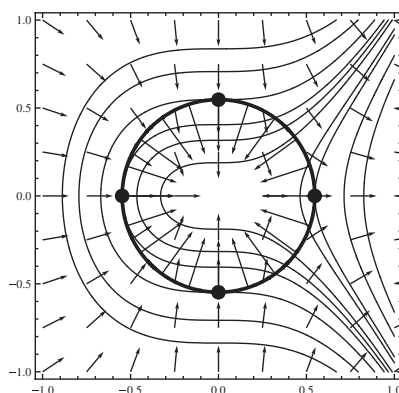


Figure 7.6. Gradient vectors for both f and g

- (d) Write a third equation you know to hold in this case due to the original problem statement (7.3).

If we use the constraint function $g(x, y) = (x^2 + y^2 - (0.5)^2)^2$ to represent the circle in Figure 7.3, we notice that something strange happens. In this case, the gradient of g is the zero-vector at all points on the circle (check for yourself). This means that the multiplier rule (7.4) is satisfied only when the gradient of the objective function f is also the zero-vector (in which case the point is an unconstrained stationary point). Unfortunately, there are lots of examples of objective functions (pick just about any linear function) with constrained local optimizers on this circle that aren't unconstrained stationary points. Moreover, these constrained local optimizers are identified by the multiplier rule (7.4) if a different constraint function g is used to represent the circle (for example, $g(x, y) = x^2 + y^2 - (0.5)^2$ works). The lesson is that it can matter how you represent the constraint. Problematic representations can be ruled out by confirming that the constraint function satisfies something called a *constraint qualification*, which is a topic for a more advanced text.

7.1.3 Multipliers for Inequalities

Now we refine our investigation of the geometry of constrained optimization in order to answer the questions that immediately preceded the discussion of Lagrange multipliers. Remember, the issue there involved inequality constraints of the form $g(x, y) \geq 0$, but so far in this section, only equation constraints of the form $g(x, y) = 0$ have been considered. Notice that if the circle in Figures 7.3, 7.4, 7.5, and 7.6 is a representation of the equation $g(x, y) = 0$, then it also represents the boundary of the constraint region $g(x, y) \geq 0$. Of course, there are essentially two regions bounded by the circle: everything outside the circle and everything inside the circle. The key to which of these regions represents the constraint region $g(x, y) \geq 0$ is the direction of the gradient of g along the circle. In Figure 7.5, the gradients point toward the inside of the circle. This means that the constraint function g is increasing from the circle toward its interior and decreasing from the circle toward the region surrounding it. Since the points on the circle have $g(x, y) = 0$ exactly, the points for

which $g(x, y) > 0$ must in this case be the interior points of the circle. Thus the constraint region $g(x, y) \geq 0$ is represented in this case by all the points on or inside the circle.

Exercise 7.1.4. *Complete the following:*

- (a) *For each of the three different functions $g(x, y)$ that you came up with in Problem 7.1, explain whether the interior or the exterior of the circle corresponds to the inequality $g(x, y) \geq 0$.*
- (b) *If none of your choices corresponds to the interior, come up with a new function $g(x, y)$ that does.*

What we have seen for the circle works in general: For any inequality constraint of the form $g(x, y) \geq 0$, the gradients of g along the boundary $g(x, y) = 0$ point to the interior of the constraint region. This is because these gradients point in the direction of greatest increase for g , and in order to increase away from points (x, y) where $g(x, y) = 0$, the values of $g(x, y)$ must become strictly greater than zero.

Exercise 7.1.5. *Complete the following:*

- (a) *To find local minimizers of some function $f(x, y)$ subject to the constraint $g(x, y) \geq 0$, explain why you should look at the boundary points where the gradient of f points toward the interior of the constraint region.*
- (b) *Explain where you should look to find local constrained maximizers.*

For minimization under inequality constraints $g(x, y) \geq 0$, then, the same multiplier rule (7.4) as before is used, but now restricting the multipliers λ to being nonnegative, since that indicates that the gradients of f and the constraint function g point in the same direction: toward the interior of the constraint region. If the multiplier λ were instead negative at a boundary point, then the function f could be decreased in value from there, without violating the constraint, by moving into the constraint region. This interpretation is the key to answering the questions that motivated this discussion of multipliers. Before finally getting to those questions, however, an extension is needed from what has been learned about single constraints to the case when there are more.

7.1.4 Multipliers for Each Constraint

The case of more than one constraint can be understood with much of the same intuition as the case of only one constraint. The only thing that changes is that now interactions between different constraints need to be considered, while still following the rule that the gradient of f at its local constrained minimizers must point in a direction that guarantees no decrease in the value of f for points nearby in the constraint region. Figure 7.7 illustrates the gradients of two constraint functions at a point of intersection of the two constraints. The interior defined by each separate constraint is shaded, and the more darkly shaded overlap is the actual constraint region where both constraints are satisfied. The only point on the boundary of the actual constraint region that needs to be considered is the intersection point where there are drawn two gradient vectors (one for each constraint function). All other boundary points involve only one constraint as equality, so the earlier discussion applies. Recall that the basic rule for constrained minimization of f is that the gradient of f points in a

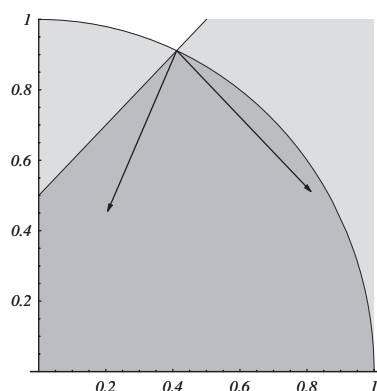


Figure 7.7. Gradient vectors at the intersection of two constraints

direction that guarantees no immediate decrease in the value of f for points nearby in the constraint region. Figure 7.8 illustrates a large-arrowed gradient vector for f that does not lie between the two constraint gradients (note that the dotted line indicates the separation between descent and ascent directions for f).

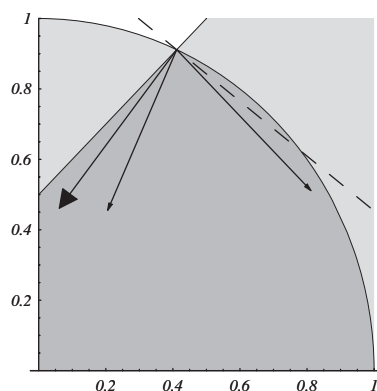


Figure 7.8. Gradient at a nonminimizer

Exercise 7.1.6. Complete the following:

- Draw on Figure 7.8 a path from the intersection point into the constraint region along which there is an immediate decrease in the value of the function f whose gradient is pictured with the large-arrowed vector.
- Explain why this means that for this function f the intersection point cannot be a local minimizer relative to this constraint region.

The lesson of Figure 7.8 is that for the intersection point to be a local minimizer for f , the dotted ascent-descent separation line needs to have the constraint region entirely on the

same side as the gradient vector. Since the dotted separating line is always perpendicular to the gradient, this can happen only if the gradient of f lies between (or on top of either of) the two gradient vectors associated with the constraint functions at the intersection point. Since all of the vectors between any two fixed vectors \mathbf{v}_1 and \mathbf{v}_2 can be expressed as the vector sum $\lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2$ for some nonnegative numbers λ_1 and λ_2 , the multiplier rule for two inequality constraints $g_1(x, y) \geq 0$ and $g_2(x, y) \geq 0$ can be expressed algebraically as

$$\nabla f(x, y) = \lambda_1 \nabla g_1(x, y) + \lambda_2 \nabla g_2(x, y), \quad (7.5)$$

where the multipliers λ_1 and λ_2 are nonnegative numbers. Notice that the old rule for one constraint is covered by this one with either λ_1 or λ_2 equal to zero.

Exercise 7.1.7. *Explain which choice of multipliers in (7.5) corresponds to a stationary point of f in the interior of the constraint region.*

Evidently, the multiplier rule (7.5) covers all the possible local minimizers with different choices of multipliers, and to extend this rule to cover more constraints, we just add corresponding terms for each additional constraint. Thus, the general multiplier rule for minimizing f over the vector variable \mathbf{x} obeying m constraints of the form $g_i(\mathbf{x}) \geq 0$ is

$$\begin{aligned} \nabla f(\mathbf{x}) &= \lambda_1 \nabla g_1(\mathbf{x}) + \lambda_2 \nabla g_2(\mathbf{x}) + \cdots + \lambda_m \nabla g_m(\mathbf{x}), \\ \lambda_i &\geq 0 \text{ for all components } i, \\ \lambda_i g_i(\mathbf{x}) &= 0 \text{ for all components } i, \end{aligned} \quad (7.6)$$

where the final set of equations ensures that, for each constraint, either the multiplier or the constraint function at \mathbf{x} equals zero. This means that the only constraint functions contributing to the gradient sum in (7.6) are the ones activated as equations at \mathbf{x} . As in the case of a single equation constraint, this multiplier rule works only under a constraint qualification which filters out degenerate cases.

Exercise 7.1.8. *Complete the following:*

- (a) *Give the general multiplier rule for maximization analogous to (7.6).*
- (b) *Consider the optimization problem*

$$\min f(x, y) \text{ over } (x, y) \text{ obeying } g(x, y) = 0$$

and explain how the general multiplier rule (7.6) in this case reproduces the classical Lagrange multiplier rule

$$\nabla f(x, y) = \lambda \nabla g(x, y),$$

where λ has unrestricted sign. (Hint: This problem has an equation constraint which needs to be reformulated as a pair of inequalities to fit the format of (7.6).)

It is important to note that though local constrained minimizers of f satisfy the general multiplier rule (7.6), this rule might be satisfied by other points as well.

Problem 7.2. *Again consider the constrained optimization problem represented in Figure 7.9, and assume that the set of points obeying $g(x, y) \geq 0$ corresponds to the disc whose boundary is the circle $g(x, y) = 0$ shown:*

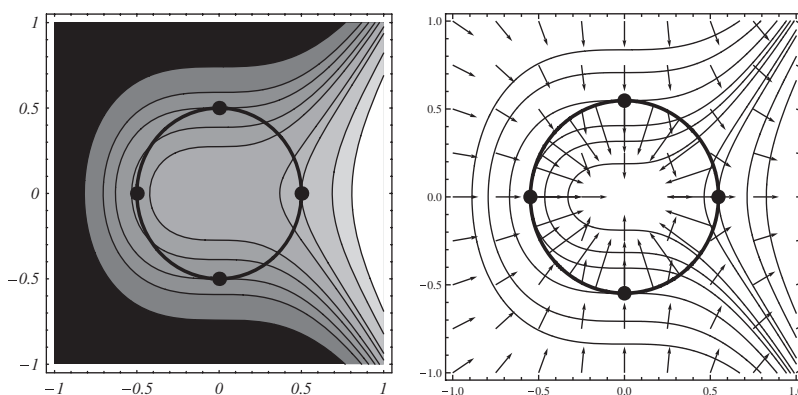


Figure 7.9. Contours and gradients of f and g

- (a) Confirm that the point $(-0.5, 0)$ is a local maximizer of f over the points (x, y) obeying $g(x, y) = 0$.
- (b) Confirm that the point $(-0.5, 0)$ satisfies the general multiplier rule (7.6) applied to the minimization of f over the points (x, y) obeying $g(x, y) \geq 0$ in this case.
- (c) Explain why the point $(-0.5, 0)$ is neither a local minimizer nor a local maximizer of f over the points (x, y) obeying $g(x, y) \geq 0$.

Problem 7.2 illustrates how a point can satisfy the general multiplier rule (7.6) without being a local (inequality-constrained) minimizer. As can be seen in Figure 7.10, the constrained function in this case exhibits saddle-like behavior at the point in question. This observation should not be too surprising since the general multiplier rule (7.6) includes the traditional stationary-point rule $\nabla f(\mathbf{x}) = 0$ for unconstrained optimization (just choose all $\lambda_i = 0$). Since the stationary-point rule identifies only stationary points, we cannot expect the general multiplier rule to do more.

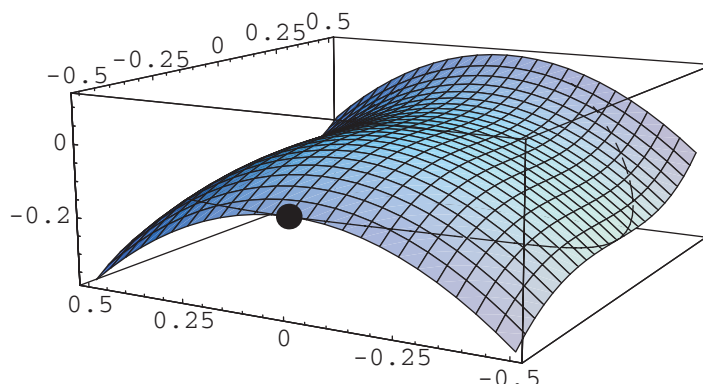


Figure 7.10. Plot of f with constraint boundary superimposed

7.1.5 Putting This All Together

The key to making use of the multiplier rule for minimization (7.6) is that the meaning of a negative multiplier is the same even when there are multiple constraints. This can be seen by observing Figure 7.8 a little more closely.

Exercise 7.1.9. *Complete the following:*

- (a) *The large-arrowed gradient vector pictured in Figure 7.8 can be written as a sum of multiples of the two constraint gradients pictured where exactly one of the constraint gradients has a positive multiplier. Which one is positive in this case, the one associated with the straight piece of the boundary or the one associated with the curved piece?*
- (b) *Now notice that the value of the function f in this case can be decreased by moving from the intersection point along either the straight piece or the curved piece of the boundary but not both. Which piece is it?*

Evidently, the presence of negative multipliers at a point signals that the objective function can be decreased from that point without violating the constraints. Moreover, the constraint boundary associated with the negative multiplier is the part of the boundary that is left behind as the objective function is decreased along the boundary. Thus, any equation constraint with a negative multiplier can safely be removed from a subproblem while progress is made toward finding an overall minimizer.

Another way to understand this is to recall that the constraint gradient $\nabla g(x, y)$ at a boundary point (x, y) points toward the interior of the constraint $g(x, y) \geq 0$. Thus, $\nabla g(x, y)$ points toward the set of pairs obeying $g(x, y) > 0$. If there is a negative multiplier associated with this constraint gradient, then the objective function can be decreased by moving into the constraint region where $g(x, y) > 0$ and away from the boundary $g(x, y) = 0$. In this way, the boundary of a constraint with a negative multiplier can be left behind while the minimization progresses. This gives us the rule we desired in Section 7.1.1 for confirming that a solution to a subproblem solves the original problem and for knowing what to do if it doesn't.

Multiplier Check:

1. Substitute the subproblem's solution \mathbf{x} into the multiplier rule for minimization (7.6), and solve for the multipliers λ_i without enforcing $\lambda_i \geq 0$.
2. If all the multipliers λ_i are nonnegative, then the solution to the subproblem is a candidate solution for the original problem since it satisfies all of the conditions in the multiplier rule for minimization (7.6).
3. If any of the multipliers λ_i are negative, pick one or more of these (often, the one with largest magnitude is chosen), drop the corresponding equation(s) from the working-set, and repeat the process outlined in Section 7.1.1 for solving the new subproblem.

Now the application of the working-set method to the simple example (7.1) that was started in Section 7.1.1 can be finished. Recall that there the solution to the subproblem (7.2) was found to be $(2, -1)$.

Problem 7.3. Use the multiplier check to finish solving the minimization problem (7.1) via the working-set method.

7.2 Special Case: Linear Programming

There have been many texts written about one special kind of optimization problem, called a *linear program*, where not only are the constraints linear but so is the objective function. The “linear” part of this terminology should be obvious, and the “program” part is an old-fashioned word for optimization problem. For instance, the simple example (7.1) is a linear program.

Problem 7.4. Complete the following:

(a) Using Figure 7.11, determine the minimizer of the linear program

$$\begin{array}{rcl} \min 2x + y & \text{over } (x, y) \text{ obeying} & \begin{array}{l} x + y \geq 1, \\ x - y \geq -2, \\ -x \geq -2. \end{array} \end{array}$$

- (b) Find the (minimizing) value of $f(x, y)$ at the solution.
- (c) Explain why the solution to any linear program cannot occur in the interior of the constraint region unless the objective function is a constant.
- (d) Give an example of a linear program with infinitely many solutions, without using a constant objective function. Justify your example with a picture.

The working-set method when applied to linear programs is usually referred to as the **simplex method**, and the linearity of the objective function leads to several simplifications.

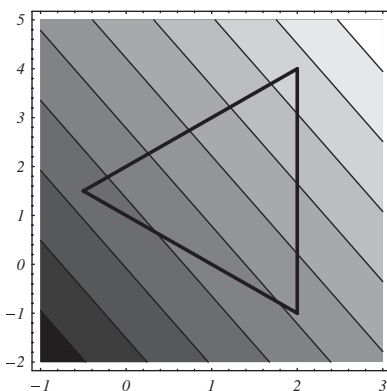


Figure 7.11. Graphical linear optimization

How the simplex method works on the linear program (7.1) has already been shown, and it works similarly on most other linear programs. In particular, it is typical for the exact line-search to be terminated prematurely at the boundary of the constraint region since

there are no peaks or valleys for a linear objective function and so there is no reason for the exact line-search to stop in the interior of the constraint region. There are other methods besides the simplex method for solving linear programs. In particular, many important *interior point methods* use barrier functions to reformulate the linear program as an unconstrained optimization problem and then apply Newton's optimization method or one of its modifications. These methods get their name from the fact that the guesses they generate stay in the interior of the constraint region, unlike the simplex method, where the guesses are on the boundary.

The role of multipliers in linear programs is particularly interesting since they can be viewed as variables for a "dual" linear program whose maximizing value is the same as the minimizing value for the original problem. This kind of relationship between two different problems also exists for more general optimization problems, but the connections are particularly strong and easy to recognize in the case of linear programs.

7.2.1 Dual Linear Programs

The basic ideas of duality will be illustrated by again considering the simple example (7.1). For now, pretend that the solution to this problem is not already known, and think about ways of getting quick estimates for the minimizing value of f without actually solving the problem.

The idea behind one quick estimation is to manipulate and combine the constraints in such a way that lower bounds on the value of f can be obtained. This typically involves adding different multiples of the constraints so that the left sides add up to the objective function and deducing that the corresponding right side is a lower bound on the value of f . For example, if 3 times the first constraint in

$$\begin{array}{rcl} \min 2x + y \text{ over } (x, y) \text{ obeying} & x + y & \geq 1, \\ & x - y & \geq -2, \\ & -x & \geq -2 \end{array}$$

is added to 2 times the second constraint to 3 times the third constraint, the result is the inequality $2x + y \geq -7$. Evidently, then, the minimizing value of the function $f(x, y) = 2x + y$ over the constraints in this linear program is no less than -7 . Notice that the multipliers were carefully chosen so that the left sides of the three multiplied constraints added up to the function $f(x, y) = 2x + y$. Then, since the multipliers were all nonnegative, none of the inequalities changed direction and the lower bound came from the sum of the corresponding right sides.

Problem 7.5. Complete the following:

- What lower bound can you deduce from the trio of multipliers $(2, 1, 1)$ instead of the trio $(3, 2, 3)$?
- Explain why this lower bound gives a better estimate for the minimizing value of f in this linear program.
- Do the same for the trio of multipliers $(3/2, 1/2, 0)$, and explain why there is no better estimate to be obtained.

Let's summarize what we've learned from this example so far.

Observations from the Quick Estimation Process:

1. The “variables” in this process are actually the multipliers $(\lambda_1, \lambda_2, \lambda_3)$ of the constraints that are being used to obtain the different lower bounds.
2. Since nonnegative multipliers are always used, the lower bounds are always simply the sum of the multipliers times the right sides of the original constraints.
3. The choice of multipliers $(\lambda_1, \lambda_2, \lambda_3)$ is constrained by the condition that the multiplied left sides of the original constraints must add up to the original function $f(x, y)$ (so that the lower bound on the original function can easily be deduced).

It is natural at this point to wonder how good an estimate of the minimizing value of f can typically be made in this manner. In the example explored so far, the trio of multipliers $(3/2, 1/2, 0)$ achieves the exact minimizing value of f , but does this always happen? We can gain some insight for this question by noticing what’s essentially going on here: The largest possible lower bound is found by adding appropriate multiples of the constraints to match the original function f with the resulting left side. For our simple example (7.1), this suggests a *dual linear program* of the form

$$\begin{aligned} \max \lambda_1 - 2\lambda_2 - 2\lambda_3 \text{ over } (\lambda_1, \lambda_2, \lambda_3) \text{ obeying } & \lambda_i \geq 0, \\ & \lambda_1 + \lambda_2 - \lambda_3 = 2, \\ & \lambda_1 - \lambda_2 = 1, \end{aligned} \quad (7.7)$$

where the objective function gives the lower bound on the objective function from the original problem (7.1).

Exercise 7.2.1. Justify the two equation constraints in the optimization problem (7.7).

To develop this in general, we again turn to matrix notation. Notice that the inequality constraints in the original problem

$$\min 2x + y \text{ over } (x, y) \text{ obeying } \begin{aligned} x + y &\geq 1, \\ x - y &\geq -2, \\ -x &\geq -2 \end{aligned}$$

can be written as

$$A \begin{bmatrix} x \\ y \end{bmatrix} \geq \begin{bmatrix} 1 \\ -2 \\ -2 \end{bmatrix}$$

in terms of the matrix

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 0 \end{bmatrix}. \quad (7.8)$$

Moreover, the equation constraints in the dual linear program (7.7) can be written as

$$A^T \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

in terms of the “transposed” version of the matrix A given by

$$A^T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & 0 \end{bmatrix}. \quad (7.9)$$

Aside: Transposing Matrices

The process of exchanging columns and rows in any matrix A is called **transposing** the matrix and is denoted by A^T . Thus, the transpose matrix A^T has as its first row the first column of the matrix A , and its second row is the second column of A , etc., as in the examples (7.8) and (7.9). For a single-column vector \mathbf{c} , the transpose \mathbf{c}^T is the same vector in row form.

A general linear program can be written in the form

$$\min \mathbf{c}^T \mathbf{x} \text{ over vectors } \mathbf{x} \text{ obeying } A\mathbf{x} \geq \mathbf{b}. \quad (7.10)$$

It is traditional to assume a constant term of zero (instead of some other constant) in the objective function since any other constant won't affect the minimization anyway.

Exercise 7.2.2. Explain how to use the solution of (7.10) to solve the same problem but with objective function $\mathbf{c}^T \mathbf{x} + a$ for some constant $a \neq 0$.

For any linear program in the form (7.10), the associated dual linear program is

$$\max \mathbf{b}^T \lambda \text{ over vectors } \lambda \text{ obeying } \lambda_i \geq 0 \text{ and } A^T \lambda = \mathbf{c}. \quad (7.11)$$

Problem 7.6. Complete the following:

- (a) Explain which of the three observations from the quick estimation process listed previously is behind the matrix constraint $A^T \lambda = \mathbf{c}$.
- (b) Now explain which of these three observations is behind the maximization of $\mathbf{b}^T \lambda$.
- (c) Explain why the maximizing value of the dual linear program is never greater than the minimizing value of the original linear program.
- (d) Give the matrix A and the vectors \mathbf{c}^T and \mathbf{b} corresponding to the general form (7.10) of the linear program

$$\begin{array}{rcl} \max -3x + 7y \text{ over } (x, y) \text{ obeying} & y & \geq 0, \\ & 2x - y & \leq 3, \\ & x + y & = 1. \end{array}$$

- (e) Give the dual linear program associated with this linear program.
- (f) In general, how is the number of corresponding dual (multiplier) variables related to an original linear program (7.10)?

In Problem 7.6, we saw that the maximizing value of the dual linear program is never greater than the minimizing value of the original linear program. It is an interesting fact that these values will actually be the same whenever the original linear program has a solution. This fact is the basis for some popular solution methods which at each step approximate both the solution to the original linear program and the solution to the dual linear program. The decision to stop and declare success is then based on whether the approximate minimizing value of the original linear program is close enough to the approximate maximizing value of the dual linear program.

7.2.2 Dual Variables as Multipliers

You might be wondering whether the multipliers discussed in this section have anything to do with the multipliers discussed in the context of the general multiplier rule for minimization (7.6), which we restate here for convenience:

$$\begin{aligned}\nabla f(\mathbf{x}) &= \lambda_1 \nabla g_1(\mathbf{x}) + \lambda_2 \nabla g_2(\mathbf{x}) + \cdots + \lambda_m \nabla g_m(\mathbf{x}), \\ \lambda_i &\geq 0 \text{ for all components } i, \\ \lambda_i g_i(\mathbf{x}) &= 0 \text{ for all components } i.\end{aligned}$$

Happily, it turns out that there is a very nice connection between these two incarnations of multipliers; namely, the vector of multipliers satisfying the general multiplier rule (7.6) applied to the original linear program is precisely the solution to the dual linear program. It is not difficult to verify that the gradient of any linear function $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$ is just $\nabla f(\mathbf{x}) = \mathbf{c}$ and, likewise, that the sum on the right side of the general multiplier rule for minimization (7.6) is the same as $A^T \lambda$ when the constraints are given by the matrix equation $A\mathbf{x} \geq \mathbf{b}$. This means that any vector of multipliers obeying the constraints of the dual linear program (7.11) also satisfies the first two elements of the general multiplier rule for minimization (7.6). It is the third element of the general multiplier rule for minimization (7.6) that actually singles out the maximizing multiplier solving the dual linear program.

Problem 7.7. *Again consider the following linear program:*

$$\begin{array}{rcl}\min 2x + y & \text{over } (x, y) \text{ obeying} & \begin{array}{l} x + y \geq 1, \\ x - y \geq -2, \\ -x \geq -2. \end{array}\end{array}$$

- (a) *Verify that each of the trios $(3, 2, 3)$, $(2, 1, 1)$, and $(3/2, 1/2, 0)$ satisfies the first two elements of the general multiplier rule for minimization (7.6):*

$$\begin{aligned}\nabla f(\mathbf{x}) &= \lambda_1 \nabla g_1(\mathbf{x}) + \lambda_2 \nabla g_2(\mathbf{x}) + \cdots + \lambda_m \nabla g_m(\mathbf{x}), \\ \lambda_i &\geq 0 \text{ for all components } i\end{aligned}$$

in this case.

- (b) *Now verify that among these three trios, only $(3/2, 1/2, 0)$ satisfies the third element of the general multiplier rule*

$$\lambda_i g_i(\mathbf{x}) = 0 \text{ for all components } i$$

in this case.

- (c) *What pair (x, y) makes the third element of the general multiplier rule work for $(3/2, 1/2, 0)$ in this case?*
- (d) *What is the significance of this pair (x, y) for the original linear program (7.1)?*

Evidently, we can solve the dual linear program (and find its corresponding multipliers) to solve the original linear program. Strong connections like this between the dual and original linear programs can be exploited for practical use in various situations, for example, when the dual linear program is more tractable.

7.2.3 Stopping for Linear Programs

We have already had some discussion in Section 3.1.1 about why optimization methods can't be expected to produce exact solutions. Moreover, Theorem 2 was developed in Section 3.2.3 to help decide when the current guess might be close enough to an exact solution. For linear programs, the issues of inexactness discussed in Section 3.1.1 are just as relevant as for any other optimization problems. However, Theorem 2 does not apply to linear programs, because any subproblem associated with a linear program is also a linear program, and so the corresponding reduced objective functions are always *linear* functions. Moreover, for a linear function, the second-derivative matrix is the trivial matrix of all zeroes, having only zero as its eigenvalue. In this situation, no scaling factor $L > 0$ can be deduced from Theorem 2 for the reverse continuity of the gradient.

Evidently, some other way is needed to decide when the current guess is close enough to the exact solution of a linear program. One alternative approach that is well suited to linear programs is to study how solutions to the linear programs change when the problem data changes. To illustrate the basic ideas, we again consider the simple example (7.1), this time focusing carefully on the effects of inexactness in the data defining the objective function. In Figure 7.12, the constraint region for this problem is displayed along with the contour of the objective function that passes through the minimizer at the left-most corner. First we will consider the effect of changes in the first component of the objective function. To do this, we consider the more general objective function $ax + y$ and see what varying the value of a away from 2 does to the minimizer. You should notice from Figure 7.12 that a variety of contour lines with slopes similar to the one pictured will produce the same minimizer at the left-most corner of the constraint region. Since the value of a in the objective function $ax + y$ is just the negative of the slope of the corresponding contour line, different values of a apparently produce the same minimizer. Thus, a whole range of values for a in the objective function $ax + y$ produce exactly the same minimizer as the original objective function $2x + y$, which means that the data needs to be accurate only within this range to be sure that the minimizer found is the right one. Notice, however, that if the safe range for a is exceeded, a dramatic change in the minimizer results.

Of course, in general it isn't just one component of the objective function that might cause trouble while the rest are exactly accurate, so the effect of simultaneous changes in

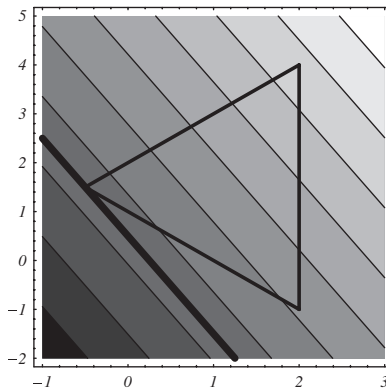


Figure 7.12. *Objective contour through the minimizer*

an objective function's components on the resulting minimizer needs to be understood. To examine this, the same constraints can be used but with the more general objective function $ax + by$. The effects of changes to the data defining the constraints can be analyzed in a similar fashion; however, in that case, small changes typically produce different, but hopefully nearby, minimizers.

Problem 7.8. *Consider the constraint region*

$$\begin{aligned}x + y &\geq 1, \\x - y &\geq -2, \\-x &\geq -2\end{aligned}$$

pictured in Figure 7.12:

- (a) *What is the smallest value of a in the objective function $ax + y$ that still produces a minimizer at the left-most corner of the constraint region?*
- (b) *What is the largest value of a that still produces the same minimizer?*
- (c) *If you were to change the value 2 to some other value in the original objective function, what would be the most precarious choice that still has the same minimizer? Explain.*
- (d) *Apply the multiplier rule for minimization (7.6) to the linear program with the objective function $ax + by$ (and with the same constraint region above), and give the resulting conditions.*
- (e) *Explain which values of a and b produce the same minimizer as the original objective function $2x + y$.*

For linear programs, then, we see that solutions do not change at all with small changes in most values of data. This means that in most cases when our guesses are not changing from one step of a method to the next, we can be confident that the current guess is close enough to the exact solution of a linear program. Of course, it also true that for a few special data values, small changes may cause dramatic changes in solutions. In practice, we try to avoid these special values when modeling the optimization problem.

7.3 Sequential Quadratic Programming

In Section 7.1, we saw that the standard reduction method is practical only when there are relatively few constraints. Moreover, it was seen that modifications to the reduction method involving careful choices of subproblems can work when there are many *linear* constraints. Fortunately, there is a more general method that works well even when there are many nonlinear constraints. To fit this final piece in the puzzle, a famous construction will be used for succinctly stating the general multiplier rule (7.6).

7.3.1 The Lagrangian Function

Recall that one of the simplest penalty functions discussed in Section 6.1 involved replacing inequalities of the form $g(x, y) \geq 0$ with a term like $-\lambda g(x, y)$ for some parameter $\lambda \geq 0$. It is not a coincidence that the label for the parameter is the same as the label used for multipliers in Section 7.1, as the multipliers can be thought of as certain appropriate penalty

parameters. The idea here is to construct a *Lagrangian function* $L(\mathbf{x}, \lambda)$ associated with the optimization of $f(\mathbf{x})$ subject to m inequality constraints of the form $g_i(\mathbf{x}) \geq 0$ by adding the simple penalty term $-\lambda_i g_i(x, y)$ for each constraint:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda_1 g_1(\mathbf{x}) - \lambda_2 g_2(\mathbf{x}) - \cdots - \lambda_m g_m(\mathbf{x}), \quad (7.12)$$

where this function depends both on the vector \mathbf{x} of variables and on the vector λ of multipliers. The same type of Lagrangian function can be used when there are equation constraints in the original problem, and it can also be used to compactly state the general multiplier rule (7.6).

Problem 7.9. *Complete the following:*

- (a) *By converting the equation constraint into a pair of inequalities, show that the Lagrangian function for the problem*

$$\min f(x, y) \text{ over pairs } (x, y) \text{ obeying } g(x, y) = 0$$

can be regarded simply as $L(x, y, \lambda) = f(x, y) - \lambda g(x, y)$ with a single multiplier λ .

- (b) *Show that the first equation in the general multiplier rule (7.6)*

$$\nabla f(\mathbf{x}) = \lambda_1 \nabla g_1(\mathbf{x}) + \lambda_2 \nabla g_2(\mathbf{x}) + \cdots + \lambda_m \nabla g_m(\mathbf{x})$$

is the same as the vector equation $L_{\mathbf{x}}(\mathbf{x}, \lambda) = \mathbf{0}$, where $L_{\mathbf{x}}$ denotes the vector of partial derivatives of L with respect to each of the component variables in \mathbf{x} .

Notice that according to Problem 7.9, solutions to the original optimization problem can be found among the \mathbf{x} -stationary points of the Lagrangian function. This fact motivates a popular method for solving constrained optimization problems which we now explore.

7.3.2 Prelude to the SQP Method

We know from Problem 7.9 that solutions to the original optimization problem can be found among the \mathbf{x} -stationary points of the Lagrangian function, and we know from Chapters 3 and 4 how to find stationary points. This seems to be exactly like what has already been advocated in the case of general penalty functions; however, there is a subtlety here to the role of the penalty parameters that needs special attention.

In the case of general penalty functions, the penalty parameters are just fixed numbers that affect the cost of violating the constraints. However, in the case of the Lagrangian function, the penalty parameters are genuine multipliers whose role is more like that of the variables themselves. Remember, not just any old multiplier will do in the general multiplier rule (7.6); an appropriate multiple of the gradient of the constraint function is needed to match the gradient of the objective function. This means that when finding the \mathbf{x} -stationary points of the Lagrangian function, the multipliers need to be determined alongside the actual solutions to the constrained optimization problem. In practice, the current guess multiplier vector λ_{old} is updated at each step as the current guess solution \mathbf{x}_{old} is updated.

Whatever method we choose from Chapters 3 and 4 to find an \mathbf{x} -stationary point for the Lagrangian function will determine exactly how to update the current guess solution \mathbf{x}_{old} . However, we will still need to determine how to update the current guess multiplier vector λ_{old} . To see how we might do this, it is helpful first to focus on the case when the constraints are all equations.

Equation Constraints Only

We consider the general problem of minimizing $f(\mathbf{x})$ over vectors \mathbf{x} obeying m equation constraints of the form $g_i(\mathbf{x}) = 0$. In this case, the general multiplier rule (7.6) works out as

$$\begin{aligned}\nabla f(\mathbf{x}) &= \lambda_1 \nabla g_1(\mathbf{x}) + \lambda_2 \nabla g_2(\mathbf{x}) + \cdots + \lambda_m \nabla g_m(\mathbf{x}), \\ g_i(\mathbf{x}) &= 0 \text{ for all components } i\end{aligned}\tag{7.13}$$

with no sign restriction on the multipliers λ_i . In Problem 7.9, we saw that the first (vector) equation in (7.13) is the same as the \mathbf{x} -stationary point equation for the Lagrangian function. We can capture the second series of equations in (7.13) in a similar way if we consider the stationary points of the Lagrangian function with respect to both λ and \mathbf{x} .

Problem 7.10. *Complete the following:*

- (a) *By writing each equation $g_i(\mathbf{x}) = 0$ as an appropriate pair of inequalities, show that the general multiplier rule (7.6) applied to the problem of minimizing $f(\mathbf{x})$ over vectors \mathbf{x} obeying m equation constraints of the form $g_i(\mathbf{x}) = 0$ works out as (7.13).*
- (b) *Show that the stationary point (vector) equation for the Lagrangian function*

$$\nabla L(\mathbf{x}, \lambda) = \mathbf{0}$$

is the same as the multiplier rule (7.13) for equation-constrained optimization.

There is a subtle difference here from what we saw in Problem 7.9, where the \mathbf{x} -stationary points of the Lagrangian function were shown to satisfy the first equation of the general multiplier rule (7.6). In that case, the multiplier λ was fixed while the corresponding Lagrangian function's \mathbf{x} -stationary points were found. The trouble is that the prechosen multiplier λ and its corresponding \mathbf{x} -stationary point do not necessarily satisfy the last condition in the general multiplier rule (7.6).

In contrast, Problem 7.10 says that the stationary pairs (\mathbf{x}, λ) of the Lagrangian function satisfy all of the conditions of the general multiplier rule applied to an equation-constrained optimization problem, so these pairs (\mathbf{x}, λ) contain solution candidates \mathbf{x} for the equation-constrained optimization problem along with their corresponding multipliers λ . One strategy for solving equation-constrained optimization problems is thus to apply the stationary-point finding methods from Chapters 3 and 4 to the Lagrangian function.

Computational Problem 7.1. *Apply Newton's optimization method to the Lagrangian in order to solve the problem*

$$\min x^2 - 2xy + 2y^2 \text{ over pairs } (x, y) \text{ obeying } y = (x - y)^2 + 1,$$

and give a full convergence analysis in this case.

The key feature of this approach is that the multipliers are updated alongside the variables as Newton's optimization method is applied to the Lagrangian, which is a function of both the multipliers λ and the variables \mathbf{x} .

Multiplier Updating

To examine exactly how the multipliers are being updated, we will consider the details of Newton's optimization method applied to the Lagrangian associated with a simple optimization problem with one equation constraint:

$$\min f(x) \text{ over points } x \text{ obeying } g(x) = 0.$$

The Lagrangian in this case is

$$L(x, \lambda) := f(x) - \lambda g(x),$$

so the analogue of the matrix equation (3.7) defining a step of Newton's optimization method applied to L in this case is

$$\overbrace{\begin{bmatrix} L_{xx}(x_{\text{old}}, \lambda_{\text{old}}) & L_{x\lambda}(x_{\text{old}}, \lambda_{\text{old}}) \\ L_{\lambda x}(x_{\text{old}}, \lambda_{\text{old}}) & L_{\lambda\lambda}(x_{\text{old}}, \lambda_{\text{old}}) \end{bmatrix}}^{\nabla^2 L(x_{\text{old}}, \lambda_{\text{old}})} \begin{bmatrix} x - x_{\text{old}} \\ \lambda - \lambda_{\text{old}} \end{bmatrix} = \overbrace{\begin{bmatrix} -L_x(x_{\text{old}}, \lambda_{\text{old}}) \\ -L_\lambda(x_{\text{old}}, \lambda_{\text{old}}) \end{bmatrix}}^{-\nabla L(x_{\text{old}}, \lambda_{\text{old}})}. \quad (7.14)$$

We can simplify some of the partial derivatives to get

$$\begin{bmatrix} L_{xx}(x_{\text{old}}, \lambda_{\text{old}}) & -g'(x_{\text{old}}) \\ -g'(x_{\text{old}}) & 0 \end{bmatrix} \begin{bmatrix} x - x_{\text{old}} \\ \lambda - \lambda_{\text{old}} \end{bmatrix} = \begin{bmatrix} -L_x(x_{\text{old}}, \lambda_{\text{old}}) \\ g(x_{\text{old}}) \end{bmatrix},$$

which works out to the following pair of equations:

$$\begin{aligned} L_x(x_{\text{old}}, \lambda_{\text{old}}) + L_{xx}(x_{\text{old}}, \lambda_{\text{old}})(x - x_{\text{old}}) &= \tilde{\lambda} g'(x_{\text{old}}), \\ g(x_{\text{old}}) + g'(x_{\text{old}})(x - x_{\text{old}}) &= 0, \end{aligned} \quad (7.15)$$

where $\tilde{\lambda} := (\lambda - \lambda_{\text{old}})$. These two equations are closely related to the following optimization problem:

$$\min TQ(x) \text{ over points } x \text{ obeying } g(x_{\text{old}}) + g'(x_{\text{old}})(x - x_{\text{old}}) = 0, \quad (7.16)$$

where

$$TQ(x) := L(x_{\text{old}}, \lambda_{\text{old}}) + L_x(x_{\text{old}}, \lambda_{\text{old}})(x - x_{\text{old}}) + \frac{1}{2}L_{xx}(x_{\text{old}}, \lambda_{\text{old}})(x - x_{\text{old}})^2$$

is the Taylor quadratic with respect to x at x_{old} of the Lagrangian function for the multiplier λ_{old} . Notice that this optimization problem has a quadratic objective function and a linear constraint function.

Any such problem (with any number of variables and linear equation or inequality constraints) is called a **quadratic program**.

To explore the connection between the quadratic program (7.16) and the pair of equations (7.15), it is helpful to notice that the derivative of the Taylor quadratic TQ in this case satisfies

$$TQ'(x) = L_x(x_{\text{old}}, \lambda_{\text{old}}) + L_{xx}(x_{\text{old}}, \lambda_{\text{old}})(x - x_{\text{old}}). \quad (7.17)$$

Problem 7.11. Complete the following:

- (a) Verify that the matrix equation (7.14) works out as the pair of regular equations (7.15).
- (b) Apply the general multiplier rule (7.13) for equation-constrained optimization to the particular quadratic program given in (7.16), and use the formula for $TQ'(x)$ in (7.17) to show that the two equations in (7.15) result.

Evidently, the candidate solutions and multipliers associated with the quadratic program (7.16) identify new guesses for a Newton's optimization step (7.14) applied to the Lagrangian. Moreover, the formula for $TQ'(x)$ in (7.17) and the first equation in (7.15) together yield the following formula for $\tilde{\lambda}$:

$$\tilde{\lambda} = \frac{TQ'(x)}{g'(x_{\text{old}})}. \quad (7.18)$$

Since the equations (7.15) define Newton's optimization method applied to the Lagrangian, their solution (x, λ) becomes the new guess $(x_{\text{new}}, \lambda_{\text{new}})$ when that method is applied. In particular, this means that we can replace x with x_{new} in the expression (7.18) for $\tilde{\lambda}$ to get

$$\tilde{\lambda} = \frac{TQ'(x_{\text{new}})}{g'(x_{\text{old}})},$$

which, after recalling the definition

$$\tilde{\lambda} := \lambda - \lambda_{\text{old}} = \lambda_{\text{new}} - \lambda_{\text{old}},$$

gives the following update formula for λ_{new} :

$$\lambda_{\text{new}} = \lambda_{\text{old}} + \frac{TQ'(x_{\text{new}})}{g'(x_{\text{old}})}.$$

Note that in nondegenerate cases when there are more variables, this formula becomes

$$\lambda_{\text{new}} = \lambda_{\text{old}} + \frac{TQ_{x_i}(\mathbf{x}_{\text{new}})}{g_{x_i}(\mathbf{x}_{\text{old}})},$$

where we get the same value of λ_{new} no matter which component x_i of the vector \mathbf{x} of variables we use for partial differentiation.

7.3.3 The SQP Method

In the preceding sections, we saw that solutions to equation-constrained minimization problems can be found among the stationary points of the corresponding Lagrangian function. We also saw that performing a single step of Newton's optimization method on the Lagrangian function is equivalent to finding solution candidates and multipliers associated with a related quadratic program. This pair of relationships forms the basis of the popular and successful *SQP method* for solving constrained minimization problems. The acronym "SQP" stands for *sequential quadratic programming*, as this process involves solving an appropriate sequence of quadratic programs. The SQP method works the same way for

general constrained optimization problems with a vector \mathbf{x} of variables and equation or inequality constraints; the next guess at a solution to the original problem is the minimizer of the quadratic program whose objective function $TQ(\mathbf{x})$ is the Taylor quadratic with respect to \mathbf{x} at the current guess solution \mathbf{x}_{old} of the Lagrangian function (with the current guess multiplier vector λ_{old}) and whose constraints are the linearized (at \mathbf{x}_{old}) versions of the original constraints. To linearize a constraint of the form $g(\mathbf{x}) = 0$ or $g(\mathbf{x}) \geq 0$ at \mathbf{x}_{old} , just replace $g(\mathbf{x})$ in the constraint with its linearization at \mathbf{x}_{old} :

$$g(\mathbf{x}) \Rightarrow g(\mathbf{x}_{\text{old}}) + \nabla g(\mathbf{x}_{\text{old}})^T (\mathbf{x} - \mathbf{x}_{\text{old}}).$$

Then the multiplier vector $\tilde{\lambda}$ corresponding—via the general multiplier rule (7.6)—to the minimizer of the quadratic program is added to the current guess multiplier vector λ_{old} for the original problem

$$\lambda_{\text{new}} = \lambda_{\text{old}} + \tilde{\lambda},$$

and the process is repeated.

SQP Method

1. **Construct the quadratic program** with the objective function being the Taylor quadratic with respect to \mathbf{x} at the current guess solution \mathbf{x}_{old} of the Lagrangian function (with the current guess multiplier vector λ_{old}), and with constraints being the linearized (at \mathbf{x}_{old}) original constraints.
2. **Minimize** the quadratic program and set its solution as \mathbf{x}_{new} .
3. **Update** the new guess multiplier vector

$$\lambda_{\text{new}} = \lambda_{\text{old}} + \tilde{\lambda},$$

where $\tilde{\lambda}$ is the multiplier vector corresponding to the minimizer of the quadratic program.

4. **Stop or repeat.**

The SQP method typically exhibits a quadratic order of convergence (remember this is really good), which is kind of surprising given that only linear approximations to the constraints are used at each step. The quadratic order of convergence is less surprising when you notice that L_{xx} contains second-order information about the original objective and the original constraints.

Implementation 4. *Implement the SQP method for the problem*

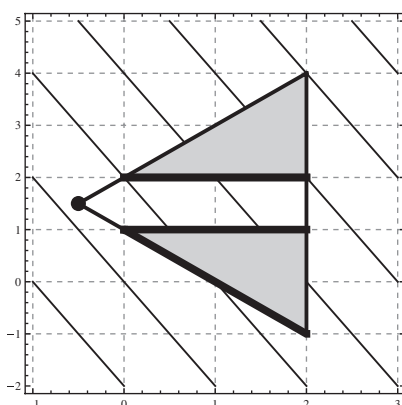
$$\min x^2 - 2xy + 2y^2 \text{ over pairs } (x, y) \text{ obeying } y \geq (x - y)^2 + 1.$$

Give a full convergence analysis and compare your results to those from Computational Problem 7.1.

7.4 Conclusions from Chapter 7

In this chapter, we introduced the *working-set method*, which is a modification of the reduction method from the preceding chapter where now only certain subproblems are chosen for consideration. This method is important since even a moderate number of constraints can generate an enormous number of subproblems. Lagrange multipliers from calculus were reviewed and used to determine which subproblems to choose in the working-set method.

We paid special attention to *linear programming*, which has a rich history of its own. In particular, we learned about dual linear programs and parameter sensitivity as a guide for stopping criteria. We also introduced the *SQP method*, which works well even with nonlinear constraints when the working-set method does not work. The *Lagrangian function* was introduced as a vehicle for understanding the SQP method.



Chapter 8

Integer Variables

For many optimization problems, it's perfectly fine to assume that the variables are real numbers. The monthly payment problem at the beginning of the book is one example where the variables (term, interest rate, and rebate amount) can reasonably include decimal places. On the other hand, there are many cases when real numbers just aren't appropriate. For instance, if a variable measures the number of employees at a small firm, then the only variables that can be reasonably considered are integers (and nonnegative ones at that).

Such optimization problems are usually called **integer programs**.

We can think of the restriction of a variable to integer values as a constraint for the problem, but this kind of constraint is really unlike anything discussed so far. For instance, think of the graphical representation of the simple requirement that the first of the two variables (x, y) be an integer. This is simply the set of vertical lines passing through the different integers along the x -axis.

Exercise 8.0.1. *What is the graphical representation of the requirement that both variables (x, y) be integers?*

The difficulty with these requirements is that they separate the variables into islands that are entirely isolated from one another. To get from one island under consideration to another, a solution method must jump over entire regions of the x - y plane. This would be fine if there was an optimization method that behaved like a helicopter, able to jump from island to island in an ocean. However, most of the methods discussed so far aren't designed to do that. Moreover, everything discussed so far has involved local optimizers, but this concept often doesn't make any sense for integer programs where each point obeying the constraints is an island unto itself.

Exercise 8.0.2. *Consider the integer program with any objective function of two variables (x, y) and the constraint that both the variables be integers. Explain why every point obeying the constraint is both a local minimizer and a local maximizer.*

We have previously seen an instance of something similar to this with the discretization problems covered earlier in this book. There, a variable function $x(t)$ with a continuum of values is replaced by a set of variables separated from one another along the t -axis. Now

we're considering a variable with a continuum of values and restricting it to some set of values that are separated from one another.

A variable separated in this way is called **discrete** to distinguish it from a *continuous variable* having a continuum of values.

Integer variables are a special kind of discrete variables where there is always a separation of length one between two neighboring values.

There is a fundamental mathematical difference between the continuous variables we have been dealing with so far in this book and the discrete (integer) variables we are now considering. Continuous variables are the basis for the functions seen in calculus, and the tools of calculus apply to them. Discrete variables like numbers of employees occur frequently in the real world, but the functions and tools from calculus don't apply directly to them. Nonetheless, many methods for solving integer programs essentially involve clever adaptations of the methods developed for continuous variables to the hostile environment of discrete variables. Soon, the "branching method" will be presented, which is the basis for many methods used to solve linear programs where some of the variables are integers (called *integer linear programs*). But first a naive approach to solving integer linear programs will be explored that sheds light on the issues we'll encounter.

8.1 Simple Rounding

The basic idea behind any integer-friendly adaptation of a continuous optimization method is to temporarily ignore the fact that some of the variables need to be integer-valued. If the solution obtained happens to have integers in the necessary variables, then the method happily concludes. Otherwise, we need to do something additional. For example, recall Figure 7.11, which we used to solve the problem of minimizing $f(x, y) = 2x + y$ over the constraint region given by

$$\begin{aligned}x + y &\geq 1, \\x - y &\geq -2, \\-x &\geq -2,\end{aligned}$$

and we found that the solution was $(-1/2, 3/2)$. If the variables needed to be integers in this case, we would have more work to do. However, if we instead wanted to *maximize* this same objective function over the same constraint set, it is clear from Figure 7.11 that the solution would be $(2, 4)$ with both variables integers.

Obviously, we cannot count on luck when solving integer programs, so what can we do? The most straightforward approach is to somehow move to the nearest acceptable variable, that is, to round unwanted noninteger parts of the solution to the nearest integer. Unfortunately, the most straightforward answer is not necessarily the best answer. For instance, take the solution $(-1/2, 3/2)$ to the minimization problem represented by Figure 7.11. To find the integer minimizer, this would need to be rounded to the "nearest" integer-pair. However, there are four equally close candidates: $(-1, 1)$, $(-1, 2)$, $(0, 1)$, and $(0, 2)$. All of these are clearly integer-pairs, but do they all obey the inequality constraints? The constraints can be checked to see that only the last two of these candidates obey the three inequalities. So which of these is the solution to the integer linear program? Plugging them into the linear function $f(x, y) = 2x + y$, it is clear that $(0, 1)$ has the lower value and

so is the minimizer for the integer linear program. Or at least it appears to be. This process is illustrated in Figure 8.1, where the levels of $f(x, y) = 2x + y$ are shown decreasing toward the lower-left corner, and the largest dot represents the winner among the four equally close integer-pairs.

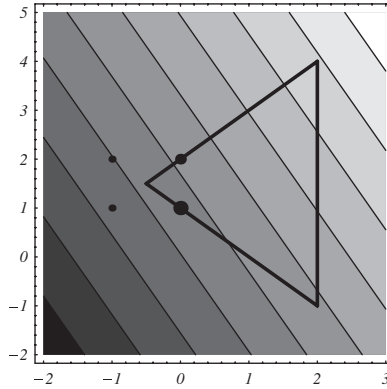


Figure 8.1. *Integer program solution*

8.1.1 Trouble with Simple Rounding

To see how simple rounding might fail, consider the minimization of the same objective function $f(x, y) = 2x + y$ as before, but now over the constraint region

$$\begin{aligned}\frac{3}{2}x + y &\geq 1, \\ -y &\geq -2, \\ -x &\geq -2,\end{aligned}$$

which is represented graphically in Figure 8.2. The solution to this problem is $(-2/3, 2)$, which can be seen in Figure 8.2 in the upper-left corner of the constraint region. Simple rounding gives only two integer-pair possibilities at $(-1, 2)$ and $(0, 2)$, and only the second of these is in the constraint region. Simple rounding would therefore say that the integer-variable version of this problem has solution $(0, 2)$ with optimal objective value 2. However, the objective function has an even lower value of 1 at the integer-pair $(0, 1)$. Moreover, the point $(0, 1)$ is also in the constraint region, as can be seen in Figure 8.3, where the smaller dots represent the two simple rounding solutions, and the larger dot is the true integer-variable program solution at $(0, 1)$.

Simple rounding can cause even worse trouble when the shape of the constraint region is such that the integer-variable solution is quite far away from the noninteger solution. One example of this involves the linear function $f(x, y) = -x - y$, whose contours are pictured in Figure 8.4 together with the constraint region defined by the two inequalities

$$\begin{aligned}-x + y &\geq 1/2, \\ 4x - 5y &\geq -13/2.\end{aligned}\tag{8.1}$$

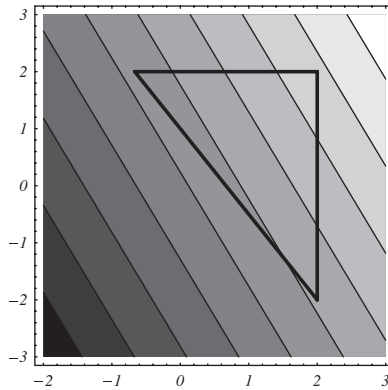


Figure 8.2. *Trouble with simple rounding*

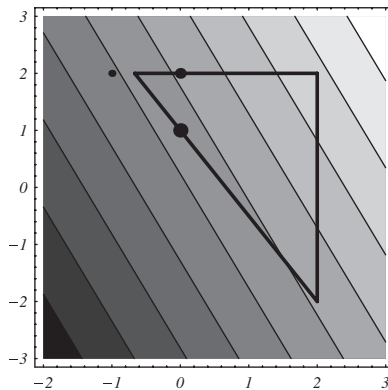


Figure 8.3. *Integer solution not from simple rounding*

Problem 8.1. *Complete the following:*

- Use Figure 8.4 to find the minimizer of the linear function $f(x, y) = -x - y$ over the integer-pairs (x, y) obeying the constraints (8.1).
- Compare what you get to the result from the rounding approach in this case.

Evidently, the rounding approach is not always successful. However, the basic idea of rounding is useful when applied in a different way.

8.2 The Branching Method

To see how the branching method works, again consider the integer linear program that demands integer variables in the linear program (7.1). The first step in the branching method is the same as the first step in the simple rounding approach already described: Ignore any integer requirements and solve the linear program as usual. As we have already seen, the

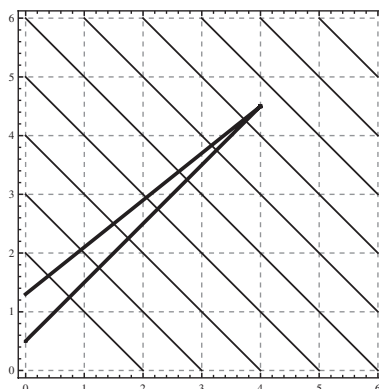


Figure 8.4. Contour plot of $-x - y$ and the constraint region (8.1)

minimizer of the linear program (7.1) is $(-1/2, 3/2)$. Now pick one of the noninteger elements of the solution that is supposed to be integer-valued and round that element both up and down to the nearest integers. In the example, both elements are supposed to be integers, so we can pick either; let's try the second element, which rounds up to 2 and down to 1. That's where rounding comes in, now for the branching that gives the method its name. Formulate two new linear programs (two "branches" of the original linear program) which are identical to the original linear program but with the additional constraints that $y \leq 1$ in one case and $y \geq 2$ in the other. (Because these two new constraints are simple bounds on the variable, this method is also sometimes referred to as the *branch-and-bound* method.) The branch linear programs for this example are

$$\begin{array}{ll} \min 2x + y \text{ over } (x, y) \text{ obeying} & \begin{array}{ll} x + y & \geq 1, \\ x - y & \geq -2, \\ -x & \geq -2, \\ y & \leq 1 \end{array} \end{array}$$

with solution $(0, 1)$ having a minimizing value of 1, and

$$\begin{array}{ll} \min 2x + y \text{ over } (x, y) \text{ obeying} & \begin{array}{ll} x + y & \geq 1, \\ x - y & \geq -2, \\ -x & \geq -2, \\ y & \geq 2 \end{array} \end{array}$$

with solution $(0, 2)$ having a minimizing value of 2. Graphically, the constraint region has been split into three parts (see Figure 8.5) with the solution to the original linear program (7.1) residing in the middle part, which is subsequently discarded from consideration.

Exercise 8.2.1. Explain why branching in this way is sure not to miss a potential solution to the integer linear program.

In this example, since the first branch has an integer solution and its minimizing value is lower than the minimizing value of the second branch, the solution $(0, 1)$ to the first branch also solves the original integer linear program. If the first branch instead happened to have a solution that was not integer-valued, it would be necessary to again branch on one of its noninteger elements.

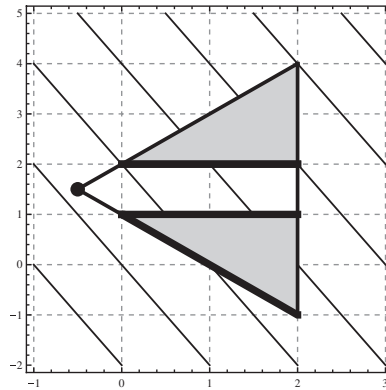


Figure 8.5. *Branching method*

Two Basic Principles of Branching:

1. When an appropriately integer-valued solution is reached at any branch, the minimizing value there is compared to the lowest minimizing value obtained so far for an appropriately integer-valued solution to any previously considered branch. If the new minimizing value is lower, then it replaces the one obtained so far.
2. If at any point in the branching a minimizing value is obtained (whether or not the solution is appropriately integer-valued) that is higher than the lowest one obtained so far for some appropriately integer-valued solution, rule out this branch for finding a solution to the original integer linear program.

The method proceeds until all possible branches have been either fully examined or ruled out, and the appropriately integer-valued solution having the lowest minimizing value at the end is the solution to the original integer linear program. If the original integer linear program has some integer variables and some noninteger variables, then it is necessary only to carry out the branching on the integer variables.

Special Case: 0-or-1 Variables

Another special kind of integer linear program arises when some variables are not merely restricted to being integers but need to be either 0 or 1. Such variables typically represent situations when some action is either taken or not; for instance, a factory is either built or not built. Branching can be used on these variables, too, where the branches are defined by including either the equation $x = 1$ or the equation $x = 0$ for a 0-or-1 variable x .

Problem 8.2. Complete the following:

- (a) Explain why solutions obtained farther along a branch can't have lower minimizing values, and use this to justify the second of the two basic principles of branching.

(b) Carry out the branching method on the integer linear program

$$\begin{array}{rcl} \min 2x + y & \text{over integers } (x, y) \text{ obeying} & \\ & & \frac{3}{2}x + y \geq 1, \\ & & -y \geq -2, \\ & & -x \geq -2 \end{array}$$

pictured in Figure 8.2.

(c) Carry out the branching method on the following 0-or-1 linear program:

$$\begin{array}{rcl} \min -x - y & \text{over } (x, y) \text{ obeying} & \\ & & -x + y \geq 1/2, \\ & & 4x - 5y \geq -13/2, \\ & & x = 0 \text{ or } 1 \end{array}$$

pictured in Figure 8.4.

8.2.1 Practical Modifications to Branching

When there are many integer variables, the branching method may generate more subproblems than can be solved in a reasonable amount of time. Most of the modifications made to the basic branching method involve clever ways of using special structures in the given problem to avoid having to solve too many subproblems. One particularly important special kind of structure occurs when all the variables are required to be either 0 or 1.

Shortcuts for 0-or-1 Variables

To see how modifications are tailor-made in this case, consider a simple example of minimizing the function

$$2x_1 - 3x_2 + x_3 + 5x_4 - x_5$$

over some (as yet unspecified) linear inequality constraints under the requirement that the variables x_i take on only the values of 0 or 1. Suppose you are in the midst of the branching method applied to this problem, at a branch where the variables x_3 and x_4 are fixed at 0 and 1, respectively. Suppose also that the lowest minimizing value so far (from some previously visited branch) is 2. Instead of solving the current branch as a linear program, simply choose the minimizing assignment of 0's and 1's for the free variables (in this case $x_1 = 0$, $x_2 = 1$, $x_5 = 1$) which yields a minimizing value of 1. If this assignment obeys all of the constraints in the problem, then use it as the current best guess at the solution since its minimizing value is lower than the current lowest minimizing value. On the other hand, suppose that the current lowest minimizing value was -1 instead of 2. Then this branch can be ruled out immediately, whether or not the minimizing assignment satisfies all the constraints in the problem, since in this case its minimizing value would definitely be higher than the current lowest minimizing value.

Branches can be ruled out for violating constraints in a similar fashion. Suppose you are at the same branch as above, and that one of the (previously unspecified) constraints is

$$6x_3 - x_4 + x_5 \geq 2.$$

Since $x_3 = 0$ and $x_4 = 1$ are already fixed, this constraint reduces to $-1 + x_5 \geq 2$. Since the variable x_5 can be only 0 or 1, there is no hope of obeying this constraint, and this branch can be ruled out.

8.3 The Cutting Method

Recall from Figure 8.5 that the branching method essentially cuts up the original constraint region in such a way that a section of it (the middle one in Figure 8.5) containing no integers can be removed from consideration. There is another method for solving integer linear programs that does a different kind of cutting of the constraint region.

Instead of cutting up the region through the middle, the **cutting method** systematically cuts away the outer edges of the constraint region while retaining the grid of integer-pairs lying within the boundaries.

Again consider the case of the integer linear program that demands integer variables in the linear program (7.1). The constraint region in this case and the grid of integer-pairs are pictured in Figure 8.6. The cutting method begins in the same way as all the other methods discussed for integer linear programs: by solving the linear program without the requirement that the variables be integers. As before, this would give the point $(-1/2, 3/2)$, which is at the left-most corner of the triangular constraint region in Figure 8.6. Since this point is not an integer-pair, the cutting method would cut it off from the constraint region using the line between it and the grid of integer-pairs in the constraint region. The result in this case would be the vertical line through the integer-pairs at $(0, 1)$ and $(0, 2)$ since these are the first integer-pairs encountered along the two boundaries of the constraint region through the point $(-1/2, 3/2)$. The cut constraint region for this example is shown in Figure 8.7.

Exercise 8.3.1. Give the linear inequality corresponding to the cut described here and pictured in Figures 8.6 and 8.7.

In general, the new constraint corresponding to a cut is a linear inequality constraint, though not necessarily a simple bound on a variable as with the branching method. For instance, Figure 8.8 shows a slightly more complicated (though still linear) cut corresponding to the same linear program (7.1) but with constraint $x + y \geq 1.5$ replacing $x + y \geq 1$. In practice, there is a relatively straightforward way to use the solution of the current stage linear program to generate the new linear inequality constraint corresponding to any cut.

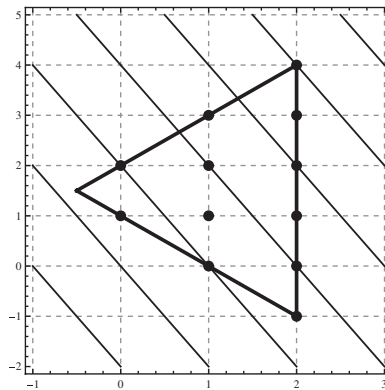


Figure 8.6. Integer-pairs obeying constraints

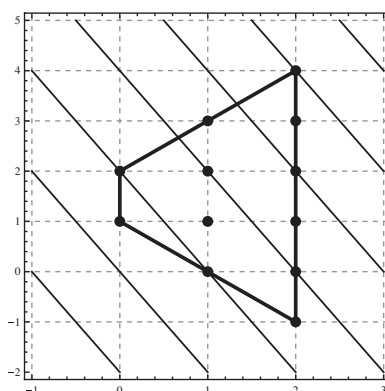


Figure 8.7. *Cut constraint region*

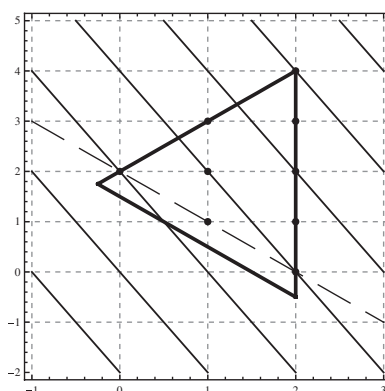


Figure 8.8. *Linear cut*

Notice that, instead of creating two new subproblems at each stage as in the branching method, the cutting method at each stage just adds one new linear inequality constraint to the original problem. (Since the new constraint is expressed graphically in the three-variable case as a plane, this method is also referred to as the *cutting-plane method*.) Since the cut constraint region at this stage has integer-pairs at all the corners, an integer-pair solution to the linear program over this region is guaranteed (of course, it's the same solution at $(0, 1)$ obtained before).

Exercise 8.3.2. *Explain why any linear program whose constraint region has integer-pairs at all the corners is guaranteed to have an integer-pair solution.*

If the solution to the linear program over the cut constraint region instead did not have an integer-pair solution at this stage, whatever solution it did have could be cut off from the grid of integers in the constraint region in the same manner.

8.4 Automatic Integers: Network Problems

Networks can be used to encode many interesting optimization problems, and there is a simple way to convert a network into a linear program. The networks we'll consider are made up of *nodes* (represented graphically by a point) and *arcs* (represented graphically as arrows) between the nodes. Each node may have arcs out of it to any node (including itself), and may also have supplies or demands associated with it, as long as the cumulative supply of all the nodes in the network equals the cumulative demand. Arcs typically have costs associated with their use, though any given arc could have cost equal to zero. The general network problem is to move all of the supplies through the network to satisfy all the demands while keeping the cumulative cost as low as possible.

A simple example of a network problem involves moving 10 tanks of oil from a single source well, possibly via three depots, to two destination cities needing 7 and 3 tanks of oil, respectively. Only certain connections between the source, the depots, and the cities are possible, as indicated in Figure 8.9. There are six nodes in this network (the source, three depots, and two cities), where the supply of 10 is next to the source and the demands of -7 and -3 are next to the two cities. There is an arc for each possible connection between the source well, depots, and cities, and the per-unit costs associated with each connection are given by the numbers next to each arc. Evidently, there are several different ways to use the network connections to move 10 tanks from the source well to the two cities, and the point is to find the least expensive one. This can be converted into a linear program by assigning variables x_i to the number of tanks moved through each arc. See the variables as indicated in Figure 8.10, where the corresponding variable appears on each arc instead of the per-unit cost associated with using the arc.

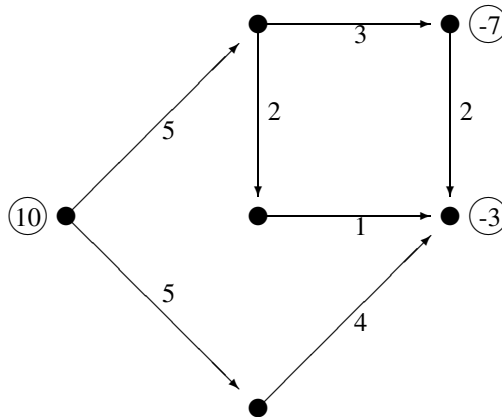


Figure 8.9. *An oil delivery network*

Using Figure 8.10, the constraints governing this problem can be formulated by realizing that at each node the amount in must equal the amount out (since the cumulative supply of 10 tanks at the source well is the same as the cumulative demand $7 + 3 = 10$ at the cities, and no tanks can be left along the way). At the source well, this means that $10 = x_1 + x_2$, and at the two cities that $x_4 = 7 + x_7$ and $x_5 + x_6 + x_7 = 3$.

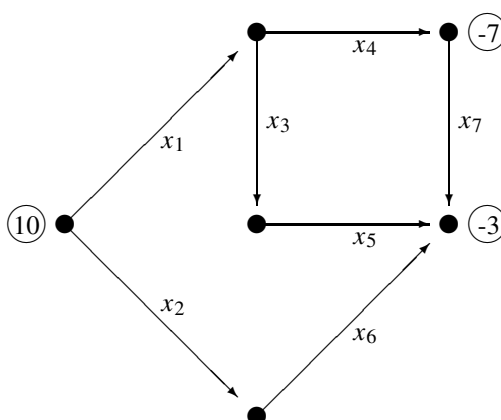


Figure 8.10. Variables for each arc

Problem 8.3. Complete the following:

- Formulate the in-equals-out equation constraints corresponding to each of the depots.
- Explain why each variable appears exactly twice in the set of six in-equals-out equation constraints.
- Explain why there are the seven inequality constraints $x_i \geq 0$ for $i = 1, \dots, 7$.
- Formulate the objective function to be minimized in this problem.

Computational Problem 8.1. Solve this oil delivery problem as a linear program.

You should automatically get integers for each variable in your solutions, which means that you won't have to use branching or cutting methods at all. This is always the case with the network problems having integer supplies and demands, which is why it's useful to reformulate and solve these network problems as linear programs.

Why Networks Have Integer Solutions, Part I: No Cycles

To explain why solvable network problems (with integer data) always have an integer-valued solution, we will need the concept of a *cycle*, which is a path in a network that connects all the nodes along the path when the directions along the arcs are disregarded. Consider the four-node network pictured in Figure 8.11 with one supply node, one demand node, and four arcs in a cycle connecting all four nodes. One cost minimizing assignment in this case is to send two units through every arc.

Exercise 8.4.1. Find a cost-minimizing assignment for the network in Figure 8.11 having at least one zero variable.

At least in this example, if there is a solution assignment with no zero variables in a cycle, then another solution assignment can be constructed without a cycle by assigning a zero to some of the variables. It turns out that this is true in general. To see why, we

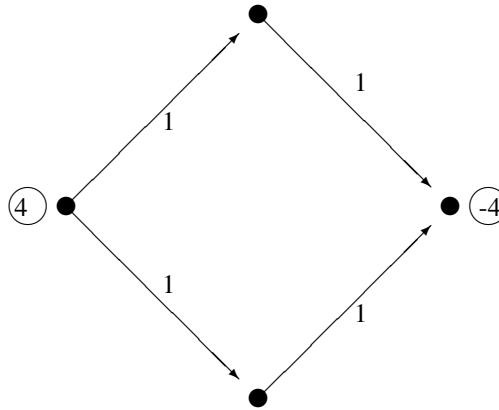


Figure 8.11. A cycling network

consider a generic network having a solution assignment with no zero variables in a cycle. We fix $\mu > 0$ equal to the smallest variable x_i among those corresponding to the arcs in the cycle. If we follow the cycle around in the direction of this “smallest-variable” arc, the arcs in the cycle are pointing either in the direction of our movement or in the opposite direction. We subtract μ from the smallest variable, as well as from each of the variables x_i associated with arcs pointing in the direction of our movement, and add μ to all the other variables associated with the cycle.

Problem 8.4. Complete the following:

- (a) Explain why the new assignment of the variables has at least one zero variable.
- (b) Explain why the new assignment is certain to satisfy the original constraints.
- (c) Explain why, no matter what the costs of the arcs in the cycle, the new assignment must have the same cost as the original solution assignment with the cycle. To do this you should do the following:
 - i. Explain why the cost of the new assignment differs from the original solution's cost by $c\mu$ for some constant c .
 - ii. Explain why the cost differs by $-c\mu$ from the original solution's cost if instead we use the opposite assignment, where μ is added to each of the variables associated with arcs in the direction of our movement and subtracted from all the other variables associated with the cycle.
 - iii. Explain why we can conclude that $c = 0$, so that the cost of the original solution assignment is the same as that of the two new assignments.

Evidently, any time there is a solution assignment with no zero variables in a cycle, there is another solution assignment with a zero variable for one of the arcs in the cycle. Since any arc with a zero associated with it can be removed from the network without changing the solution, this means we can assume that any solvable network problem has a solution assignment with all nonzero variables and no cycles.

Why Networks Have Integer Solutions, Part II: Single-Arc Nodes

Any assignment with all nonzero variables, with no cycles, and that satisfies all of the constraints must have at least one supply or demand node connected to the rest of the network by a single arc. To see why, we pick any node in the network and follow any of the arcs away from it (disregarding the direction of the arcs). We repeat this process at each subsequent node encountered. Since there are no cycles and a finite number of nodes, this process must end at some node, and this node must have only one arc connecting it to the network (namely the arriving arc). This last node must be either a supply or a demand node since the variable associated with its lone connecting arc needs to be balanced.

Now we pretend that we don't know the assignment and we try to regenerate it one node at a time. Knowing that there is a supply or demand node connected to the rest of the network by a single arc, we start there and solve its in-equals-out equation. Since there's only one arc, and since the supplies and demands are integers, the result of this will be to assign an integer to the variable corresponding to the lone arc. Now we replace the lone arc with a supply or demand equal to the integer of the just-assigned variable and remove the node from the network. Essentially, then, a subnetwork has been created by removing one supply or demand node from the network and creating a new supply or demand at one of the remaining nodes. The same procedure is then applied to the subnetwork and repeated until there are no more nodes to remove. Since at each step one of the variables is set equal to some integer, the result is an assignment of integers to all of the variables. Moreover, at each step there is only one possible assignment from the in-equals-out equation, so this all-integer assignment must be exactly the same assignment we started with.

Problem 8.5. *Complete the following:*

- (a) *Carry out the one-node-at-a-time assignment procedure to solve the oil delivery problem described by Figure 8.9.*
- (b) *How many different integer assignments are generated by this procedure besides the solution assignment?*

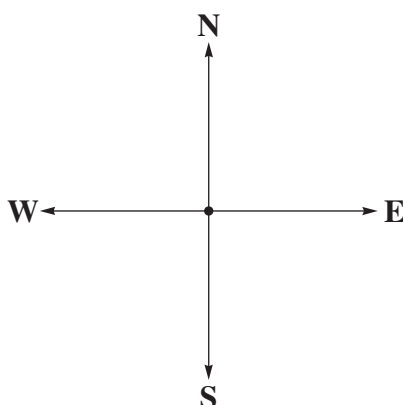
We conclude that any assignment with all nonzero variables, with no cycles, and that satisfies all of the constraints must be integer-valued. Since any solution assignment certainly satisfies all of the constraints, and since we've already seen that we can assume a solution assignment with all nonzero variables and no cycles, it follows that this solution assignment is integer-valued. Thus, an integer-valued solution is guaranteed for any solvable network problem with integer data.

8.5 Conclusions from Chapter 8

In this chapter, we studied the special methods needed for *integer programs* with integer variables. We saw why simple rounding is dangerous and introduced two alternative methods which both ignore any integer requirements until a solution is obtained:

- **Branching:** Solution variables which need to be integers but are not generate new subproblems with the rounded variables as constraints.
- **Cutting:** Solution variables which need to be integers but are not generate a new linear constraint which cuts the offending variable from the constraint region.

We also looked at the special case of 0-1 variables and learned why *network problems* with integer data are guaranteed to have integer solutions.



Chapter 9

Other Methods

The numerical methods presented in the preceding chapters of this text include the basics of derivative-based practical optimization. However, there are many other numerical methods for solving optimization problems. Some of these are specialized methods designed to solve particular forms of optimization problems, in the same way that the branching method is designed to solve integer programs. One famous example which we'll cover in depth now is called *dynamic programming*. Other methods can be explored through the Research Project described at the end of this chapter. The format for the Research Project is the same as the presentation here of dynamic programming, so readers can use the following presentation as a model for their Research Project.

9.1 Dynamic Programming

The term “dynamic programming” was coined by Richard Bellman while he was working on multistage decision processes at the RAND Corporation in the early 1950's (Bellman, 1984). The basic principle behind dynamic programming is to establish generic links between consecutive stages of a multistage decision process. Bellman was particularly interested in applying dynamic programming to solve integral optimization problems called “optimal control” problems (Bellman, 1984). In these problems, both the decision—called the “control”—and the state of the system to be controlled are functions of time.

9.1.1 Multistage Decision Processes

A multistage decision process is just what it sounds like: a process involving decisions made at multiple stages. Just as an optimization model is defined by its variables, constraints, and objective function, a multistage decision process is defined by its stages and decisions. At any stage, the possible *states* of the process include enough information about decisions made at preceding stages to inform a decision at the present stage. A multistage decision process *problem* associates costs (or rewards) with the possible decisions in the process.

Many optimization problems can be reformulated as multistage decision process problems. The key to this reformulation is to identify the stages, decisions, and states

Table 9.1. *Distances for college-visit problem*

	Town 1	Town 2	Town 3	Town 4
	Portland	Brunswick	Lewiston	Waterville
Portland	0	26	34	78
Brunswick	26	0	18	52
Lewiston	34	18	0	51
Waterville	78	52	51	0

of the associated multistage decision process. For example, consider the college-visit problem from Chapter 1. This problem involved visits to three towns in Maine on one extended trip starting from and returning to Portland. For convenience, we again give the table of distances between the towns in Table 9.1. This optimization problem naturally suggests the following stages:

- Stage 1 → Start in Portland
- Stage 2 → One town visited after arrival in Portland
- Stage 3 → Two towns visited after arrival in Portland
- Stage 4 → Three towns visited after arrival in Portland
- Stage 5 → Finish in Portland

since these are the stages at which a prospective student would find herself during this college visit process. The decision at any stage of this problem is simply the choice of which town to visit next. To make such a decision, the prospective student needs to know not only what town she is currently visiting, but what towns she has already visited, since she doesn't want to visit any college town more than once. The states of this process need to include all of this information. For instance, at Stage 3, the student will have visited two towns after Portland, so one possible state could be expressed as $(3, \{3, 4\})$, where the first component (3 in this case) denotes that the current town is Lewiston, and the second component pair ($\{3, 4\}$ in this case) indicates that the towns already visited after Portland are Lewiston and Waterville. Of course, we could express the same information without including the current town on the list of towns already visited—in which case the example state would be $(3, \{4\})$ —but the fuller form that we've chosen will prove to be more convenient. The “costs” of the decisions in this multistage decision process problem are the distances to travel between the towns.

Exercise 9.1.1. *Complete the following:*

- (a) *Express all the states at Stage 3 in the same form as the state $(3, \{3, 4\})$.*
- (b) *Express all the states at Stage 2.*
- (c) *Express all the states at Stage 5.*
- (d) *How many states are there at Stage 4?*

9.1.2 The Dynamic Programming Method

To use dynamic programming to solve a multistage decision process problem, we need to consider the “least remaining cost” (or greatest remaining reward) from each possible state at each stage. This is the least cost (or greatest reward) accrued by proceeding from that

state through the subsequent stages until the final stage of the process is reached. Of course, determining the least remaining cost at the first stage—and knowing how to realize it via a policy of decisions—is tantamount to solving the multistage decision process problem. Therefore, we will assume that we are dealing with problems for which we cannot easily determine the least remaining cost at the first stage. Dynamic programming attempts to solve such problems by instead determining the least remaining costs at *later* stages and then connecting these costs recursively through the preceding stages until the first stage is reached.

To see how dynamic programming works, we will again consider the college-visit problem (formulated as a multistage decision process problem). It is easy to see that the least remaining cost from the state $(3, \{2, 3, 4\})$ at Stage 4 in the college-visit problem is 34 since that is the distance from Lewiston to Portland. To keep track of our steps, it will be useful to use function notation f_4 to denote least remaining costs at Stage 4: So the least remaining cost from the state $(3, \{2, 3, 4\})$ at Stage 4 is denoted by $f_4(3, \{2, 3, 4\})$. Here the subscript “4” on the function f_4 indicates the stage, and the input $(3, \{2, 3, 4\})$ indicates the state. Our simple observation of the distance from Lewiston to Portland then tells us that the following equation holds:

$$f_4(3, \{2, 3, 4\}) = 34.$$

Exercise 9.1.2. Complete the following:

- (a) What is the value of $f_4(4, \{2, 3, 4\})$?
- (b) Explain why it does not make sense to evaluate $f_3(3, \{2, 3, 4\})$.
- (c) Use the table of distances in Table 9.1 justify the identity

$$f_3(4, \{2, 4\}) = 51 + f_4(3, \{2, 3, 4\}). \quad (9.1)$$

The identity (9.1) is an example of a *recursion* relationship between consecutive-stage least remaining costs. The term “recursion” refers to the fact that the least remaining cost at one stage can be found from the least remaining cost at the next stage, and it is precisely this kind of relationship that makes dynamic programming work. The idea is to generate recursion relationships like (9.1) between all the least remaining costs and then to evaluate later least remaining costs to recursively find the earlier ones. For instance, the identity (9.1) leads immediately to the evaluation

$$f_3(4, \{2, 4\}) = 51 + f_4(3, \{2, 3, 4\}) = 51 + 34 = 85$$

since we have already evaluated the least remaining cost $f_4(3, \{2, 3, 4\}) = 34$. The value $f_3(4, \{2, 4\}) = 85$ can then be used to deduce least remaining costs at Stage 2 from the analogous recursion relationships from Stage 2 to Stage 3.

Dynamic Programming

1. **Initialize** by solving for the least remaining costs at the final stages.
2. **Recursively** solve for the least remaining costs at the preceding stages.
3. **Choose** optimal decision policy to realize the least remaining cost from the start.

Problem 9.1. Complete the following:

- (a) Explain why recursion relationships between Stages 2 and 3 of the college-visit problem will use the minimization of two options involving least remaining costs at Stage 3, while recursion relationships between Stages 1 and 2 will use the minimization of three options involving least remaining costs at Stage 2.
- (b) Record all of the recursion relationships between least remaining costs at consecutive stages of the college visit-problem and apply dynamic programming (by hand) to solve the problem.

To encode the dynamic programming method on a computer, it is helpful to identify a *generic recursion relationship* between stages. For example, our detailed investigation suggests the following generic recursion relationship for the college-visit problem:

$$f_s(i, \text{Vtowns}) = \min_{j \notin \text{Vtowns}} \{c_{i,j} + f_{s+1}(j, \text{Vtowns} \cup \{j\})\}, \quad (9.2)$$

in terms of the distances $c_{i,j}$ between towns i and j . The following translations might help in deciphering the notation used in the generic recursion relationship (9.2):

- The subscript “ s ” on the least remaining cost f_s is a placeholder for any particular stage to be considered, and the subscript “ $s + 1$ ” corresponds the stage following that.
- “Vtowns” is any particular set of towns already visited at stage s (e.g., $\text{Vtowns} = \{2,3,4\}$ would be the only possibility at stage $s = 4$, and either $\text{Vtowns} = \{2,3\}$, $\text{Vtowns} = \{2,4\}$, or $\text{Vtowns} = \{3,4\}$ could be used at stage $s = 3$).
- Note that the town i must be in the set Vtowns for the expression $f_s(i, \text{Vtowns})$ on the left side to make sense.
- The notation $j \notin \text{Vtowns}$ found under the “min” symbolizes that the minimum is taken over all towns j not already visited.
- The notation $\text{Vtowns} \cup \{j\}$ refers to the new set of towns created when town j is added to the original set of towns Vtowns.

For instance, the recursion relationship (9.1) fits the generic recursion relationship with choices of $s = 3$, $i = 4$, and $\text{Vtowns} = \{2,4\}$ since the minimization in that case is taken over only one town (i.e., $j = 3$).

9.1.3 Comparison and Current Work

The main drawback to the dynamic programming method is the same kind of scaling issue that can afflict the “direct comparison” approach for solving the college-visit problem. For instance, recall that direct comparison led to 1,814,400 different round-trips to compare in a college-visit problem with only 10 college towns. Similar scaling issues are present in the branching method applied to college-visit problems: The most basic branching method involves a number of steps similar to that for direct comparison, though this improves when branches can be systematically ruled out (e.g., as in the 0-1 variables case). The cutting method generally involves fewer steps than direct comparison, but this improvement is balanced somewhat by the relatively complicated process of finding the equation for the new “cutting” constraint at each step.

In the dynamic programming method, the least remaining cost from each state needs to be computed via a recursion relationship, so the number of states determines the number of steps in the method. For instance, the college-visit problem with three college towns had 12 different intermediate states (not counting the states corresponding to the first and last stages). If the number of college towns is increased to 10, the number of intermediate states increases to 5120. This is significantly smaller than the nearly two million round-trips to be considered via direct comparison, but as the number of college towns increases even more, the number of states eventually gets impractically large. In general, the college-visit problem with n college towns to visit after Portland has $n(2^{n-1})$ intermediate states to consider, which is about 10.5 million when the number of college towns is $n = 20$. There are many important multistage optimization problems with much larger scales than this, and in such cases the dynamic programming method is computationally infeasible.

Roughly speaking, the dynamic programming method is similar to the cutting method in the sense that the number of steps is fewer than some other methods (e.g., direct comparison or unmodified branching for integer programs), but the effort at each step—establishing the recursion relationship for dynamic programming or finding the cutting constraint for the cutting method—is greater. This is similar to the way Newton's optimization method compares to its modifications.

Dynamic programming's most important recent application is to "stochastic" problems where there is fundamental uncertainty present (Denardo, 1996). In the usual multistage decision processes, we can determine the next state from the current state and the decision at that stage; hence, these are sometimes called "deterministic" multistage decision processes. In stochastic multistage decision processes, we instead work with probabilities governing the transitions from one state to another under each decision. For instance, the outcomes of many medical decisions are "known" beforehand only as probabilities based on historical data.

Another example from (Denardo, 1996) of a stochastic multistage decision process problem is to minimize the cost associated with pest control given the following information:

- Burning the entire infested area at any point achieves certain eradication. The cost of this option is \$60 million if done now, \$84 million if postponed one month, and \$120 million if postponed two months.
- Spraying now with poison costs \$3 million but has only a 30% chance of eradication after a month.
- Spraying both now and one month later costs an additional \$4 million but has another 30% chance of eradication after the second month.
- Spraying now and incubating sterilized males costs \$9 million. If the spraying fails, the sterilized males can be released one month later with a 50% chance of eradicating the entire population.

This series of decisions can be analyzed via dynamic programming to determine that the optimal strategy which guarantees eradication with least "expected" cost (of \$49 million) is to spray now as well as to incubate sterilized males for release if necessary and then to burn afterward if both of the first two eradication attempts fail (Denardo, 1996). The *expected* cost weighs the different potential costs by the probabilities with which they might be incurred, so the expected cost is not necessarily a cost that could actually be incurred. For instance, the optimal strategy in the pest eradication example will cost either

\$9 million (if either spraying works or spraying fails and release works) or \$129 million (if both spraying and release fail and burning is necessary to eradicate the pest). The expected cost of \$49 million is the weighted (by probability) sum of the two potential costs. The practical principle behind the concept of expected cost is that if the same optimal strategy were followed again and again in a number of identical situations, the initial approach of spraying or release would be successful often enough to balance the relatively few times when a final burning would be necessary.

In general, stochastic problems are more difficult to solve than deterministic ones; however, the dynamic programming method works well for both kinds of multistage decision process problems. To apply dynamic programming in the stochastic case, it helps to create a decision network with expected costs and connections based on the probabilities. With such a network, the dynamic programming method works just as it does in the deterministic case: by considering the latest stages first and solving backwards in time via recurrence relationships between the least remaining *expected* costs.

Current research on dynamic programming is being done in at least the following three categories:

- “Stochastic dynamic programming” where dynamic programming is applied to stochastic multistage decision process problems (e.g., (Cervellera, Wen, and Chen, 2007) and (Chang, 2006)).
- Special applications of dynamic programming (e.g., to single-source network problems (Fontes, Hadjiconstantinou, and Christofides, 2006) and to input estimation on time-variant systems (Nordström, 2006)).
- Modifications to the basic dynamic programming method (e.g., memory saving (Betzler, Niedermeier, and Uhlmann, 2006) and relaxations (Lincoln and Rantzer, 2006)).

Annotated Bibliography for Dynamic Programming

Bellman, Richard (1952): *On the theory of dynamic programming*, in Proc. Nat. Acad. Sci. U.S.A. 38, 716–719.

This paper by the founding author introduces the term “dynamic programming.”

Bellman, Richard (1984): *Eye of the Hurricane*, World Scientific Publishing Company, Singapore.

The autobiography of Richard Bellman, published in the year he died.

Betzler, Nadja; Niedermeier, Rolf; Uhlmann, Johannes (2006): *Tree decompositions of graphs: Saving memory in dynamic programming*, in Discrete Optim. 3, no. 3, 220–229.

This paper discusses how to save memory when applying dynamic programming to solve graph optimization problems via their tree decompositions.

Cervellera, Cristiano; Wen, Aihong; Chen, Victoria C. P. (2007): *Neural network and regression spline value function approximations for stochastic dynamic programming*, in *Comput. Oper. Res.* 34, no. 1, 70–90.

This paper uses artificial neural networks to model value function approximation in certain high-dimensional stochastic dynamic programming problems.

Chang, Hyeong Soo (2006): *Converging marriage in honey-bees optimization and application to stochastic dynamic programming*, in *J. Global Optim.* 35, no. 3, 423–441.

This paper adapts the “marriage in honey-bees” approach for solving combinatorial optimization problems to solve certain stochastic dynamic programming problems.

Denardo, Eric V. (1996): *Dynamic Programming*, in *Mathematical Programming for Industrial Engineers*, eds. Mordecai Avriel and Boaz Golany, Marcel Dekker, Inc., New York.

A contributed chapter in an edited compilation. This gives a detailed description of dynamic programming on which I relied heavily.

Fontes, Dalila B. M. M.; Hadjiconstantinou, Eleni; Christofides, Nicos (2006): *A dynamic programming approach for solving single-source uncapacitated concave minimum cost network flow problems*, in *European J. Oper. Res.* 174, no. 2, 1205–1219.

This paper uses dynamic programming to solve network problems with one source and no limits on how much can be moved through an arc and where the costs on each arc are described by a concave function.

Lincoln, Bo; Rantzer, Anders (2006): *Relaxing dynamic programming*, in *IEEE Trans. Automat. Control* 51, no. 8, 1249–1260.

This paper addresses the scaling issues associated with the dynamic programming method by allowing the method to terminate when the distance from optimality is within certain bounds (i.e., by “relaxing” the demand for optimality).

Nordström, Lars J. L. (2006): *A dynamic programming algorithm for input estimation on linear time-variant systems*, in *Comput. Methods Appl. Mech. Engrg.* 195, no. 44–47, 6407–6427.

This paper applies dynamic programming to estimate time-domain input for linear systems whose parameters depend on time.

Trick, Michael A. (1997): *A Tutorial on Dynamic Programming*, Carnegie Mellon University, Pittsburgh, PA.

This brief online tutorial from an academic at a well-established institution can be found at

<http://mat.gsia.cmu.edu/classes/dynamic/dynamic.html>

Several classic examples are given of multistage decision process problems, including a version of the traveling salesman problem which is similar to our college-visit problem.

9.2 What Else?

The methods described in this text represent the tip of the (inverted) iceberg that is the field of practical optimization. There are even other “icebergs” in this field with no direct relationship to the methods covered here. The conclusion to this text is therefore an invitation to explore more of this broad and vibrant field. The Research Project at the end of this chapter is the most obvious component of this invitation, but readers should also explore the Bibliography for some other resources for continuing study in this field. Before describing the details of the Research Project, we’ll outline a few methods on which the reader might want to focus. The methods are organized in five categories, listed roughly in order of “difficulty.”

9.2.1 Direct Search Methods

Some call these “derivative-free” methods since they typically optimize without resorting to derivative computations. Within this category are some geometry-based methods that a typical reader of this text should be able to understand without any additional background:

- Compass-search method
- Hooke and Jeeves pattern search method
- Nelder–Mead method

Even though these methods are not derivative-based, natural connections can be made to the derivative-based methods described previously in this text. A research project on one of these methods should investigate such a connection.

9.2.2 Stochastic Search Methods

These methods employ randomness in some way to generate guesses and might be interesting to readers with science backgrounds (including computer science) who have seen some probability.

- Random search
- Genetic algorithms
- Simulated annealing

A research project on one of these topics should explore how randomness distinguishes these methods from the methods presented in this text.

9.2.3 Linear Methods

These methods include some element of linearity, by solving either linear equations (the conjugate gradient method) or linear programs (the next two on the list). Readers who wish to explore the conjugate gradient method will benefit from having some background in linear algebra.

- Conjugate gradient method
- Khachiyan's ellipsoid method
- Karmarkar's projective algorithm

A research project on the conjugate gradient method should explain how systems of linear equations are related to optimization and should describe the extension of this method to nonlinear optimization. Projects on either of the latter two methods should make comparisons with the simplex method.

9.2.4 Primal-Dual Methods

As the title suggests, these utilize duality in some way. These are generally nonlinear methods with strong connections to linear programs.

- Path-following
- Mehrotra's predictor-corrector algorithm
- Potential-reduction methods

Some readers will find a research project on these methods to be too ambitious, but readers with the right background and interests should consider them. Projects on these methods should describe the connections to linear programs.

9.2.5 Nonlinear Methods

The following are a variety of other methods for solving nonlinear optimization problems and are generally part of the same "iceberg" as the methods presented in this text.

- Augmented Lagrangian method
- Trust-region methods
- Reduced Lagrangian method
- Truncated Newton method
- Polak–Ribière method
- BFGS (Broyden, Fletcher, Goldfarb, and Shanno) method
- Broyden's quasi-Newton method
- Fletcher–Reeves method
- Gauss–Newton method
- Homotopy methods
- Inexact Newton methods
- Levenberg–Marquardt method
- SR1 (symmetric-rank-1) method
- Tensor methods

Some of these methods may be too advanced for many readers, but readers who want a challenge should consider them. Research projects on these methods should explore the natural connections to the methods presented in this text.

9.3 A Research Project

Research one of the numerical optimization methods from the lists above, and write a textbook section modeled on Section 9.1. Your section should include four subsections as described below. The complete textbook section should smoothly integrate all the subsections with at least one Exercise (less complicated) and one Problem (more complicated), that are relevant, original, and within the ability of a typical reader of this text, without being trivial. Carefully worked solutions should be provided separately as part of the final project submission.

9.3.1 Annotated Bibliography

This should list all the particular resources used for this project, together with a brief description of the content and its use in each case. Any online citations should come through legitimate, verifiable, scholarly sources (e.g., scholarly search engines like MathSciNet). Be sure to include at least the following three kinds of sources:

- “Big-picture” sources that present your method in a broader context: These include survey papers and text sections and can generally be identified by their very general titles.
- The original paper(s) in which your method was introduced.
- Recent research papers which illustrate the current state of development of your method and/or its surrounding category. You do not have to fully understand the material in these sources, but you should have enough of an idea to write a synopsis of several sentences.

Remember to browse the abstracts, introductions, and bibliographies of your sources to find more sources.

9.3.2 Problem Setting

This should include a clear and complete description of the problem(s) that the method under study is designed to solve. At least one original example of such a problem should be described in detail.

9.3.3 Method Description

This should include a clear and complete description of the method, with the algorithm outline and an illustrating example.

9.3.4 Comparison and Current Work

Explain pros, cons, and how this method compares to other methods—including those covered in the text when appropriate. Include a brief review of the most recent literature on the method (e.g., via MathSciNet).

Appendix of Asides

The following are all of the Asides found throughout the text, compiled here for easy reference.

Quadratic Functions

For this discussion, the letter Q will be used to denote a generic quadratic function. Also, for simplicity, the focus will be on a function of two variables x and y since these are the simplest quadratic functions that exhibit all of the important properties of quadratic functions of any number of variables. Moreover, the properties of quadratic functions of more variables can be understood via obvious extensions of the results for two-variable quadratics. Any quadratic function of two variables x and y can be written as

$$Q(x, y) = \frac{a}{2}x^2 + bxy + \frac{c}{2}y^2 + dx + gy + h$$

for some constants a, b, c, d, g , and h . It's easy to find the stationary points of Q because it's easy to compute the gradient (vector) of Q :

$$\nabla Q(x, y) = \begin{bmatrix} ax + by + d \\ bx + cy + g \end{bmatrix}.$$

The stationary points of Q are then simply the solutions (x, y) to the two equations obtained by setting the $\nabla Q(x, y)$ equal to the zero-vector:

$$\begin{aligned} ax + by + d &= 0, \\ bx + cy + g &= 0. \end{aligned}$$

It is relatively easy to solve a pair of linear equations like this for values of x and y .

You should notice in Exercise 3.1.1 that either the trio of constants a, b , and c lead to equations that can be uniquely solved for all choices of d and g , or the trio of constants a, b , and c gives multiple solutions for certain values of d and g and no solutions for other values of d and g . In other words, the presence of degeneracy in the form of multiple or no solutions is dependent entirely on the trio of constants a, b , and c , whereas the constants d and g merely determine the type of degeneracy.

Matrix Notation and Multiplication

A matrix is a grouping of numbers into rows and columns. Vectors can be thought of as matrices with only one column, and matrices in general can be thought of as a group of column vectors. Matrices come in all different sizes, but for now the focus will be on matrices A with two rows and two columns:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

where a , b , c , and d are any numbers. For any such matrix A , its multiplication against a vector whose components are x and y is defined by

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}. \quad (\text{A.1})$$

Notice that the basic rule for generating each row of the result is to consecutively multiply the corresponding row elements of A against the column elements of the vector, adding the results as you go (this same rule applies for vectors having any number of components as long as A has that same number of columns). You might recognize this procedure as just a pair of dot products generated by each row (vector) of A against the vector whose components are x and y .

You can also multiply a vector against a matrix against another vector, which we call a *vector-matrix-vector multiplication*. For instance, the vector-matrix-vector multiplication

$$\begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix}$$

represents the dot product of the first vector $\begin{bmatrix} u & v \end{bmatrix}$ with the vector that results from the matrix multiplication (A.1) and thus is given by the formula

$$\begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix} = u(ax + by) + v(cx + dy).$$

Notice in particular that the result of such a vector-matrix-vector multiplication is a number.

Matrix Inversion

Solving an equation like $ax = b$ for the variable x is easy; simply multiply both sides by the inverse of the number a to get the solution $x = a^{-1}b$. What is needed now is a generalization of this to matrix equations $A\mathbf{x} = \mathbf{b}$, where \mathbf{x} and \mathbf{b} are vectors with two components and A is a matrix with two rows and two columns. The idea is to come up with another matrix A^{-1} , the inverse of A , which multiplies both sides of the matrix equation to give the solution $\mathbf{x} = A^{-1}\mathbf{b}$.

Not all matrices A have inverses (just as the number 0 has no inverse), and this happens precisely when the linear equations encoded by $A\mathbf{x} = \mathbf{b}$, for any right-side vector \mathbf{b} , have either no solution or infinitely many solutions. When the encoded linear equations can be solved uniquely for every \mathbf{b} , the matrix encoding the equations has an inverse, and vice versa. If a matrix of the form

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

has an inverse, it is given by

$$A^{-1} = \begin{bmatrix} \frac{d}{ad-bc} & \frac{-b}{ad-bc} \\ \frac{-c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix}. \quad (\text{A.2})$$

The lack of an inverse for A then corresponds exactly to the situation when $ad - bc = 0$, in which case the formula (A.2) does not produce a matrix with finite-valued entries.

Continuity

You can get some intuition for the fact that the sign of the vector-matrix-vector multiplication in (3.15) determines the sign of the analogous vector-matrix-vector multiplication at nearby points by considering any continuous function of one variable. Figure A.1 shows the graph of a particular continuous function with a particular base point \bar{x} , but the idea is the same for any continuous function and any base point. Notice in Figure A.1 that for a point like \bar{x} with positive value $c(\bar{x})$, the points x nearby \bar{x} on the x -axis have values that are also positive and that get closer to the value of $c(\bar{x})$ as x gets closer to \bar{x} . For general functions of any number of variables, the property of being continuous ensures that the difference between output values shrinks to nothing as the distance between the corresponding input points is shrunk to nothing. The vector-matrix-vector multiplication in (3.15) defines a continuous function of (\bar{x}, \bar{y}) as long as the partial derivatives which define the second-derivative matrix are continuous functions, which we are assuming for all the functions we're considering.

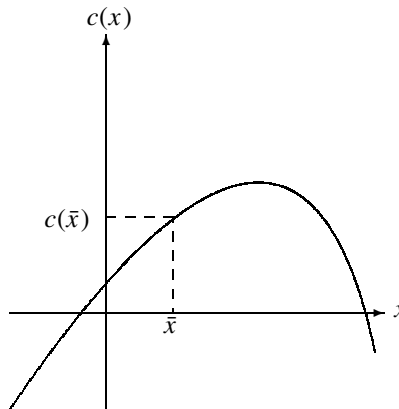


Figure A.1. A continuous function $c(x)$

Definite Matrices and Eigenvalues

There is terminology for the sign of a vector-matrix-vector multiplication like that in (3.15) or (3.14) being always positive or negative, and it actually applies to any matrix

$$A = \begin{bmatrix} a & b \\ b & d \end{bmatrix} \quad (\text{A.3})$$

which is *symmetric* above and below its diagonal.

A symmetric matrix A is called **positive-definite** if the vector-matrix-vector multiplication

$$\begin{bmatrix} x & y \end{bmatrix} A \begin{bmatrix} x \\ y \end{bmatrix}$$

definitely gives a positive result in all cases when $(x, y) \neq (0, 0)$, and A is called **negative-definite** if the same vector-matrix-vector multiplication definitely gives a negative result in all cases when $(x, y) \neq (0, 0)$.

There is also a practical way to verify the definiteness of a symmetric matrix (A.3) which doesn't involve actually testing the sign of the vector-matrix-vector multiplication over and over with every possible vector (of course, such direct verification is impossible). The idea is to instead find the *eigenvalues* of A and test their sign.

A matrix with two rows and two columns has at most two distinct **eigenvalues**, which are simply the scalars ϵ solving the matrix equation

$$A\mathbf{v} = \epsilon\mathbf{v}$$

for some special vector \mathbf{v} (called an **eigenvector** corresponding to that eigenvalue ϵ).

Notice that an eigenvalue ϵ is thus the number for which matrix multiplication of the matrix A against the eigenvector \mathbf{v} is identical to scalar multiplication of the vector \mathbf{v} by ϵ . It turns out that the eigenvalues of a symmetric matrix (A.3) can be found by solving for ϵ in the quadratic equation

$$(a - \epsilon)(d - \epsilon) = b^2.$$

Most mathematical software these days (e.g., *Mathematica* or MATLAB) has built-in eigenvalue finders so you don't have to do these computations by hand. In many cases, finding the eigenvalues of a matrix is relatively straightforward, so testing their signs is too. The relationship between eigenvalues and the definiteness of symmetric matrices is as follows:

- Both eigenvalues of A are positive if and only if A is positive-definite.
- Both eigenvalues of A are negative if and only if A is negative-definite.

When the eigenvalues of the matrix are reasonable to compute, this relationship gives a very practical test of definiteness (as opposed to direct verification, which is impossible). For very large scale problems (e.g., with tens of thousands of variables) where the computation of eigenvalues can become too expensive, other methods are used to test for positive-definiteness. A nice side effect of definiteness is that it ensures the existence of the inverse matrix (which shares the same definiteness property via Problem 3.3).

The Identity Matrix

The identity matrix I always has ones down its diagonal from upper left to lower right and zeroes everywhere else. The two-dimensional version looks like

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The identity matrix is its own inverse, and multiplication against any vector by the (appropriate dimension) identity matrix again returns the vector $I\mathbf{x} = \mathbf{x}$.

Eigenvalues and Convexity/Concavity

Since the graph of a two-variable function is a three-dimensional surface, it exhibits “concavity” in three dimensions as well. To understand how concavity works in three dimensions, it is useful to think of slicing the graph with any plane perpendicular to the x - y plane. The image of the graph that intersects such a plane is a two-dimensional curve whose concavity can be characterized as in calculus. *Convex* is the official term used for what you may have learned as “concave up” in your calculus course, and simply *concave* is the official term for “concave down.”

Thus, a two-dimensional curve is **convex** near some base point if the slopes are nondecreasing (moving from left to right along the horizontal axis), and it is **concave** if the slopes are nonincreasing. Both of these concepts can be strengthened by demanding strict decrease or increase in the slopes, which is indicated by the terminology **strictly convex** or **strictly concave**.

Since the second derivative of a function f of one variable measures the rate at which the slopes change, we can use f'' to test whether the graph of f is convex or concave. In particular, when $f''(\bar{x})$ is positive (negative) we know that the graph of f is strictly convex (concave) near \bar{x} (this is the basis for the second-derivative test from calculus for functions of one variable). Something that may not have been covered in your calculus class is that the magnitude of the second derivative can be used to compare different degrees of convexity or concavity. This is because the second derivative measures the rate of change of the slopes along the graph, and a higher rate of change of slope means a higher degree of convexity or concavity. To see what different degrees of convexity might look like, we consider the two simple functions $f_1(x) := x^2$ and $f_2(x) := x^4$, which are graphed in Figure A.2. For input values of x larger than about $1/2$, we can see that the graph of x^2 is relatively less convex than the graph of x^4 in the sense that the steepening of the slopes in the first case is less severe than in the latter. It is easy to compute the second derivatives in this case to be $f_1''(x) = 2$ and $f_2''(x) = 12x^2$, from which we can deduce that $f_1''(x) < f_2''(x)$ as long as $x > \sqrt{1/6} = 0.408248$, which is consistent with our eyeball guess from the graphs of about $1/2$.

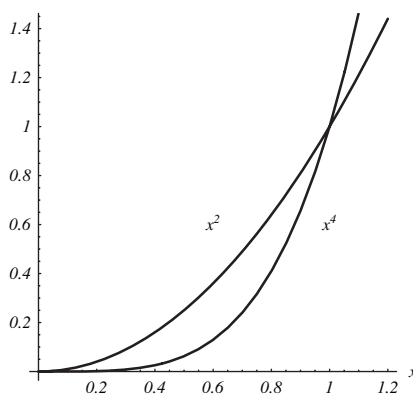


Figure A.2. Graphs of x^2 and x^4

The graph of a two-variable function is convex (concave) if every two-dimensional slice by a plane perpendicular to the x - y plane produces a curve that is convex (concave). A pair of positive (negative) eigenvalues for the second-derivative matrix $\nabla^2 f(\bar{x}, \bar{y})$ indicates that near (\bar{x}, \bar{y}) the graph of f is strictly convex (strictly concave). On the other hand, mixed-sign eigenvalues indicate that some slices produce strictly convex curves, and some produce strictly concave curves, which means that the graph of f is saddle-like near (\bar{x}, \bar{y}) .

Just as the relative magnitude of f'' gives the degree of convexity or concavity in the one-variable case, the relative sizes of the eigenvalues of $\nabla^2 f(\bar{x}, \bar{y})$ give the relative degree of the convexity or concavity of different slices. Thus, a pair of positive eigenvalues having different scales indicates that some slices have relatively small convexity and some have relatively large convexity. This is just the kind of behavior exhibited by a chute.

Transposing Matrices

The process of exchanging columns and rows in any matrix A is called **transposing** the matrix and is denoted by A^T . Thus, the transpose matrix A^T has as its first row the first column of the matrix A , and its second row is the second column of A , etc., as in the examples (7.8) and (7.9). For a single-column vector \mathbf{c} , the transpose \mathbf{c}^T is the same vector in row form.

Bibliography

- [1] Bhatti, M. Asghar (2000): *Practical Optimization Methods with Mathematica Applications*. TELOS, Springer-Verlag, New York.

This introductory textbook covers some of the same material as the present book but in a more traditional manner and with a greater emphasis on computer programming.

- [2] Bliss, Gilbert Ames (1925): *Calculus of Variations*. The University of Chicago Press, Chicago, IL.

The title of this text refers to a classical kind of integral optimization. This text presents theory as well as several interesting applications in this area.

- [3] Fletcher, R. (1987): *Practical Methods of Optimization* (second edition). John Wiley & Sons Ltd., New York.

This covers the same topics at a more advanced level than the present book. Even though it is not so old, it qualifies as a classic in this area.

- [4] Griva, I., Nash, Stephen G., and Sofer, Ariela (2009): *Linear and Nonlinear Programming, Second Edition*. SIAM, Philadelphia, PA.

This also covers many of the same topics but at a more advanced level than the present book, as well as many additional related topics. This is an excellent choice of text to use if you want to learn more about the subject.

- [5] Guéret, Christelle, Prins, Christian, and Sevaux, Marc (2000): *Programmation Linéaire*. Editions Eyrolles, Paris, France.

The revised translation by Susanne Heipcke and published by Dash Optimization Ltd. describes many applications of optimization, some of which inspired Modeling Problems and examples in Chapter 1.

- [6] Nocedal, Jorge, and Wright, Stephen J. (1999): *Numerical Optimization*. Springer-Verlag, New York.

This is another a good follow-up to the present book at an even more advanced level and covering additional related topics.

- [7] Tikhomirov, V. M. (translated by Abe Shenitzer, 1990): *Stories about Maxima and Minima*. The American Mathematical Society, Providence, RI.

This is an accessible book with some historical background to go along with interesting problems in finite-dimensional and integral optimization.

- [8] Winston, Wayne L. (1995): *Introduction to Mathematical Programming, Applications and Algorithms* (second edition). Wadsworth, Inc., Belmont, CA.

This text is a good example of a standard text focusing on linear programming. This text and its competitors are widely available and, in particular, are great resources for interesting examples and problems. Some of the Modeling Problems in Chapter 1 are adaptations of problems in this text.

- [9] Wright, Margaret H. (2004): *The interior-point revolution in optimization: History, recent developments, and lasting consequences*. In Bulletin of the American Mathematical Society, volume 42, number 1, pages 39–56.

This is a recent research survey paper which traces the dissolution of the traditional boundary between linear and nonlinear programming.

Index

- barrier function, 80–84
 - inverse, 81–83
 - logarithmic, 81–83
- branch-and-bound method, *see* branching method
- branching method, 116, 118, 120–123
- calculus of variations, 30
- constraint qualification, 95, 98
- convexity, 68
 - concave, 68
 - of functions, 68
 - of graphs, 69
 - strict, 69
 - via second derivatives, 69, 70
- cutting method, 122, 123, 125
- cutting-plane method, *see* cutting method
- cycle, 63
 - in networks, 125–127
 - in Newton’s optimization method, 60
 - in quasi-Newton methods, 60
 - in the working-set method, 90
- discrete variables, 116
- discretization, 29–33, 48, 74
 - linear spline interpolation, 31
 - Newton’s aerodynamics problem, 31, 32, 48
 - Newton’s aerodynamics problem in rare media, 74, 80, 87
 - two-variable problem, 38, 40–43, 46, 47, 49, 53, 59, 67, 68
- dual linear program, 102–105
- dynamic programming, 129, 130
 - definition of method, 131
 - least remaining cost, 130
 - recursion relationship, 131, 132
 - scaling issues with, 132
 - stochastic, 133
 - expected cost, 133
- Hessian, *see* second-derivative matrix
- integral optimization, 29, 30, 32, 33, 38, 74, 129, 145, 146
- interior point methods, 102
 - Karmarkar’s algorithm, xvi
- Lagrangian function, 107, 108
 - and the SQP method, 108, 112
 - multiplier rule, 109
 - Newton’s optimization method, 109–111
 - Taylor quadratic of, 110
- line-search, 58, 83
 - backtracking, 58, 83
 - exact, 58, 59, 68, 90–92, 101, 102
 - no cycles, 60, 63
- minimizer
 - global, 25, 26
 - local, 25, 26, 44, 45, 47, 58, 85, 96–98, 115
- monthly payment function, 25, 26, 48
 - and reduction, 29, 74, 86, 87
 - fixed interest rate, 25–27
 - variable interest rate, 28, 87
 - and reduction, 73
 - defining equation, 27

- graph, 49
- minimizing, 28, 43, 44, 47, 53, 59, 67, 68
- multipliers
 - as dual variables, 102–105
 - for one inequality, 95, 96, 98
 - for two inequalities, 96, 98
 - in general optimization rule, 98, 100, 105, 107–109, 112
 - equations only, 109
 - in SQP method, 108, 110, 112
 - in working-set method, 100, 101
 - Lagrange, 29, 93–95, 108
 - significance of sign, 96, 100
- Newton's aerodynamics problem, 29, 31, 32, 48
 - in rare media, 74, 80, 87
- Newton's optimization method, 40, 63, 67, 75, 77, 78
 - and optimality, 44, 47, 55
 - and the SQP method, 109
 - applying, 42, 43, 47, 53, 60, 67
 - damped, 84
 - definition of, 42
 - in hybrid method, 70, 71
 - in interior point methods, 102
 - in reduction, 87
 - multivariable, 48
 - Newton's method, 43
 - order of convergence, 63, 66–68, 70
 - practical problems, 55, 59–61
 - update formula, 41
- optimal control, 30, 129
- order of convergence, 63, 64, 66, 70
 - linear, 64–66, 68, 70
 - quadratic, 65–68, 70, 112
 - superlinear, 64, 65
- penalty function, 75, 82, 108
 - barrier, 80, 81
 - ideal, 75, 79–82
 - min-type, 76, 77
 - smoothed, 77–80
 - simple, 75, 76, 79, 107
- quadratic program, 110–112
- quasi-Newton methods, 90, 91
 - descent in, 59
 - in hybrid method, 70
 - order of convergence, 67, 68
 - practical problems, 60, 83
 - rationale for, 55
 - robustness of, 60
 - simplicity of, 61
 - steepest descent, 56
- reduction, 74, 84, 86
 - applying, 91
 - modifications to, 89
 - versus SQP method, 107
- reduction method
 - applying, 87
 - definition of, 87
- rounding, 116
 - applying, 118
 - in branching method, 118, 119
 - trouble with, 117, 118
- search direction, *see* step-direction
- second-derivative matrix, 38, 42, 55, 56, 58, 59, 67
 - for optimality test, 44–48
 - for scaling factor, 53
 - graphical implications, 68–70
 - in damped Newton's optimization method, 84
 - invertibility, 60, 67
 - multivariable, 48
 - size, 61, 63
 - trivial, 106
- sequential quadratic programming, *see* SQP method
- simplex method, xvi, 101
- SQP method, 108, 111, 112
 - definition of, 112
- steepest descent method
 - applying, 58, 59, 68, 91
 - order of convergence, 68, 70
 - perpendicular steps in, 58
 - practical problems, 83
 - simplicity of, 61

-
- step-direction, 42, 57–59
 - in working-set method, 91
 - one-variable, 83
 - steepest descent, 57, 92
 - step-factor, 57, 58, 83, 84, 91, 92
 - step-size, *see* step-factor
 - working-set method, 89, 90, 92
 - applying, 91, 92
 - linearity needed, 92
 - zigzagging, 68, 70