

Contents

Preface	xvii
Preface to the Second Edition	xxi
1 Introduction	1
Mathematical Formulation	2
Example: A Transportation Problem	4
Continuous versus Discrete Optimization	5
Constrained and Unconstrained Optimization	6
Global and Local Optimization	6
Stochastic and Deterministic Optimization	7
Convexity	7
Optimization Algorithms	8
Notes and References	9
2 Fundamentals of Unconstrained Optimization	10
2.1 What Is a Solution?	12

	Recognizing a Local Minimum	14
	Nonsmooth Problems	17
2.2	Overview of Algorithms	18
	Two Strategies: Line Search and Trust Region	19
	Search Directions for Line Search Methods	20
	Models for Trust-Region Methods	25
	Scaling	26
	Exercises	27
3	Line Search Methods	30
3.1	Step Length	31
	The Wolfe Conditions	33
	The Goldstein Conditions	36
	Sufficient Decrease and Backtracking	37
3.2	Convergence of Line Search Methods	37
3.3	Rate of Convergence	41
	Convergence Rate of Steepest Descent	42
	Newton's Method	44
	Quasi-Newton Methods	46
3.4	Newton's Method with Hessian Modification	48
	Eigenvalue Modification	49
	Adding a Multiple of the Identity	51
	Modified Cholesky Factorization	52
	Modified Symmetric Indefinite Factorization	54
3.5	Step-Length Selection Algorithms	56
	Interpolation	57
	Initial Step Length	59
	A Line Search Algorithm for the Wolfe Conditions	60
	Notes and References	62
	Exercises	63
4	Trust-Region Methods	66
	Outline of the Trust-Region Approach	68
4.1	Algorithms Based on the Cauchy Point	71
	The Cauchy Point	71
	Improving on the Cauchy Point	73
	The Dogleg Method	73
	Two-Dimensional Subspace Minimization	76
4.2	Global Convergence	77
	Reduction Obtained by the Cauchy Point	77
	Convergence to Stationary Points	79
4.3	Iterative Solution of the Subproblem	83

	The Hard Case	87
	Proof of Theorem 4.1	89
	Convergence of Algorithms Based on Nearly Exact Solutions	91
4.4	Local Convergence of Trust-Region Newton Methods	92
4.5	Other Enhancements	95
	Scaling	95
	Trust Regions in Other Norms	97
	Notes and References	98
	Exercises	98
5	Conjugate Gradient Methods	101
5.1	The Linear Conjugate Gradient Method	102
	Conjugate Direction Methods	102
	Basic Properties of the Conjugate Gradient Method	107
	A Practical Form of the Conjugate Gradient Method	111
	Rate of Convergence	112
	Preconditioning	118
	Practical Preconditioners	120
5.2	Nonlinear Conjugate Gradient Methods	121
	The Fletcher–Reeves Method	121
	The Polak–Ribière Method and Variants	122
	Quadratic Termination and Restarts	124
	Behavior of the Fletcher–Reeves Method	125
	Global Convergence	127
	Numerical Performance	131
	Notes and References	132
	Exercises	133
6	Quasi-Newton Methods	135
6.1	The BFGS Method	136
	Properties of the BFGS Method	141
	Implementation	142
6.2	The SR1 Method	144
	Properties of SR1 Updating	147
6.3	The Broyden Class	149
6.4	Convergence Analysis	153
	Global Convergence of the BFGS Method	153
	Superlinear Convergence of the BFGS Method	156
	Convergence Analysis of the SR1 Method	160
	Notes and References	161
	Exercises	162

7	Large-Scale Unconstrained Optimization	164
7.1	Inexact Newton Methods	165
	Local Convergence of Inexact Newton Methods	166
	Line Search Newton–CG Method	168
	Trust-Region Newton–CG Method	170
	Preconditioning the Trust-Region Newton–CG Method	174
	Trust-Region Newton–Lanczos Method	175
7.2	Limited-Memory Quasi-Newton Methods	176
	Limited-Memory BFGS	177
	Relationship with Conjugate Gradient Methods	180
	General Limited-Memory Updating	181
	Compact Representation of BFGS Updating	181
	Unrolling the Update	184
7.3	Sparse Quasi-Newton Updates	185
7.4	Algorithms for Partially Separable Functions	186
7.5	Perspectives and Software	189
	Notes and References	190
	Exercises	191
8	Calculating Derivatives	193
8.1	Finite-Difference Derivative Approximations	194
	Approximating the Gradient	195
	Approximating a Sparse Jacobian	197
	Approximating the Hessian	201
	Approximating a Sparse Hessian	202
8.2	Automatic Differentiation	204
	An Example	205
	The Forward Mode	206
	The Reverse Mode	207
	Vector Functions and Partial Separability	210
	Calculating Jacobians of Vector Functions	212
	Calculating Hessians: Forward Mode	213
	Calculating Hessians: Reverse Mode	215
	Current Limitations	216
	Notes and References	217
	Exercises	217
9	Derivative-Free Optimization	220
9.1	Finite Differences and Noise	221
9.2	Model-Based Methods	223
	Interpolation and Polynomial Bases	226
	Updating the Interpolation Set	227

	A Method Based on Minimum-Change Updating	228
9.3	Coordinate and Pattern-Search Methods	229
	Coordinate Search Method	230
	Pattern-Search Methods	231
9.4	A Conjugate-Direction Method	234
9.5	Nelder–Mead Method	238
9.6	Implicit Filtering	240
	Notes and References	242
	Exercises	242
10	Least-Squares Problems	245
10.1	Background	247
10.2	Linear Least-Squares Problems	250
10.3	Algorithms for Nonlinear Least-Squares Problems	254
	The Gauss–Newton Method	254
	Convergence of the Gauss–Newton Method	255
	The Levenberg–Marquardt Method	258
	Implementation of the Levenberg–Marquardt Method	259
	Convergence of the Levenberg–Marquardt Method	261
	Methods for Large-Residual Problems	262
10.4	Orthogonal Distance Regression	265
	Notes and References	267
	Exercises	269
11	Nonlinear Equations	270
11.1	Local Algorithms	274
	Newton’s Method for Nonlinear Equations	274
	Inexact Newton Methods	277
	Broyden’s Method	279
	Tensor Methods	283
11.2	Practical Methods	285
	Merit Functions	285
	Line Search Methods	287
	Trust-Region Methods	290
11.3	Continuation/Homotopy Methods	296
	Motivation	296
	Practical Continuation Methods	297
	Notes and References	302
	Exercises	302
12	Theory of Constrained Optimization	304
	Local and Global Solutions	305

	Smoothness	306
12.1	Examples	307
	A Single Equality Constraint	308
	A Single Inequality Constraint	310
	Two Inequality Constraints	313
12.2	Tangent Cone and Constraint Qualifications	315
12.3	First-Order Optimality Conditions	320
12.4	First-Order Optimality Conditions: Proof	323
	Relating the Tangent Cone and the First-Order Feasible Direction Set	323
	A Fundamental Necessary Condition	325
	Farkas' Lemma	326
	Proof of Theorem 12.1	329
12.5	Second-Order Conditions	330
	Second-Order Conditions and Projected Hessians	337
12.6	Other Constraint Qualifications	338
12.7	A Geometric Viewpoint	340
12.8	Lagrange Multipliers and Sensitivity	341
12.9	Duality	343
	Notes and References	349
	Exercises	351
13	Linear Programming: The Simplex Method	355
	Linear Programming	356
13.1	Optimality and Duality	358
	Optimality Conditions	358
	The Dual Problem	359
13.2	Geometry of the Feasible Set	362
	Bases and Basic Feasible Points	362
	Vertices of the Feasible Polytope	365
13.3	The Simplex Method	366
	Outline	366
	A Single Step of the Method	370
13.4	Linear Algebra in the Simplex Method	372
13.5	Other Important Details	375
	Pricing and Selection of the Entering Index	375
	Starting the Simplex Method	378
	Degenerate Steps and Cycling	381
13.6	The Dual Simplex Method	382
13.7	Presolving	385
13.8	Where Does the Simplex Method Fit?	388
	Notes and References	389
	Exercises	389

14 Linear Programming: Interior-Point Methods	392
14.1 Primal-Dual Methods	393
Outline	393
The Central Path	397
Central Path Neighborhoods and Path-Following Methods	399
14.2 Practical Primal-Dual Algorithms	407
Corrector and Centering Steps	407
Step Lengths	409
Starting Point	410
A Practical Algorithm	411
Solving the Linear Systems	411
14.3 Other Primal-Dual Algorithms and Extensions	413
Other Path-Following Methods	413
Potential-Reduction Methods	414
Extensions	415
14.4 Perspectives and Software	416
Notes and References	417
Exercises	418
15 Fundamentals of Algorithms for Nonlinear Constrained Optimization	421
15.1 Categorizing Optimization Algorithms	422
15.2 The Combinatorial Difficulty of Inequality-Constrained Problems	424
15.3 Elimination of Variables	426
Simple Elimination using Linear Constraints	428
General Reduction Strategies for Linear Constraints	431
Effect of Inequality Constraints	434
15.4 Merit Functions and Filters	435
Merit Functions	435
Filters	437
15.5 The Maratos Effect	440
15.6 Second-Order Correction and Nonmonotone Techniques	443
Nonmonotone (Watchdog) Strategy	444
Notes and References	446
Exercises	446
16 Quadratic Programming	448
16.1 Equality-Constrained Quadratic Programs	451
Properties of Equality-Constrained QPs	451
16.2 Direct Solution of the KKT System	454
Factoring the Full KKT System	454
Schur-Complement Method	455
Null-Space Method	457

16.3	Iterative Solution of the KKT System	459
	CG Applied to the Reduced System	459
	The Projected CG Method	461
16.4	Inequality-Constrained Problems	463
	Optimality Conditions for Inequality-Constrained Problems	464
	Degeneracy	465
16.5	Active-Set Methods for Convex QPs	467
	Specification of the Active-Set Method for Convex QP	472
	Further Remarks on the Active-Set Method	476
	Finite Termination of Active-Set Algorithm on Strictly Convex QPs	477
	Updating Factorizations	478
16.6	Interior-Point Methods	480
	Solving the Primal-Dual System	482
	Step Length Selection	483
	A Practical Primal-Dual Method	484
16.7	The Gradient Projection Method	485
	Cauchy Point Computation	486
	Subspace Minimization	488
16.8	Perspectives and Software	490
	Notes and References	492
	Exercises	492
17	Penalty and Augmented Lagrangian Methods	497
17.1	The Quadratic Penalty Method	498
	Motivation	498
	Algorithmic Framework	501
	Convergence of the Quadratic Penalty Method	502
	Ill Conditioning and Reformulations	505
17.2	Nonsmooth Penalty Functions	507
	A Practical ℓ_1 Penalty Method	511
	A General Class of Nonsmooth Penalty Methods	513
17.3	Augmented Lagrangian Method: Equality Constraints	514
	Motivation and Algorithmic Framework	514
	Properties of the Augmented Lagrangian	517
17.4	Practical Augmented Lagrangian Methods	519
	Bound-Constrained Formulation	519
	Linearly Constrained Formulation	522
	Unconstrained Formulation	523
17.5	Perspectives and Software	525
	Notes and References	526
	Exercises	527

18 Sequential Quadratic Programming	529
18.1 Local SQP Method	530
SQP Framework	531
Inequality Constraints	532
18.2 Preview of Practical SQP Methods	533
IQP and EQP	533
Enforcing Convergence	534
18.3 Algorithmic Development	535
Handling Inconsistent Linearizations	535
Full Quasi-Newton Approximations	536
Reduced-Hessian Quasi-Newton Approximations	538
Merit Functions	540
Second-Order Correction	543
18.4 A Practical Line Search SQP Method	545
18.5 Trust-Region SQP Methods	546
A Relaxation Method for Equality-Constrained Optimization	547
SL_1QP (Sequential ℓ_1 Quadratic Programming)	549
Sequential Linear-Quadratic Programming (SLQP)	551
A Technique for Updating the Penalty Parameter	553
18.6 Nonlinear Gradient Projection	554
18.7 Convergence Analysis	556
Rate of Convergence	557
18.8 Perspectives and Software	560
Notes and References	561
Exercises	561
19 Interior-Point Methods for Nonlinear Programming	563
19.1 Two Interpretations	564
19.2 A Basic Interior-Point Algorithm	566
19.3 Algorithmic Development	569
Primal vs. Primal-Dual System	570
Solving the Primal-Dual System	570
Updating the Barrier Parameter	572
Handling Nonconvexity and Singularity	573
Step Acceptance: Merit Functions and Filters	575
Quasi-Newton Approximations	575
Feasible Interior-Point Methods	576
19.4 A Line Search Interior-Point Method	577
19.5 A Trust-Region Interior-Point Method	578
An Algorithm for Solving the Barrier Problem	578
Step Computation	580
Lagrange Multipliers Estimates and Step Acceptance	581

	Description of a Trust-Region Interior-Point Method	582
19.6	The Primal Log-Barrier Method	583
19.7	Global Convergence Properties	587
	Failure of the Line Search Approach	587
	Modified Line Search Methods	589
	Global Convergence of the Trust-Region Approach	589
19.8	Superlinear Convergence	591
19.9	Perspectives and Software	592
	Notes and References	593
	Exercises	594
A	Background Material	598
A.1	Elements of Linear Algebra	598
	Vectors and Matrices	598
	Norms	600
	Subspaces	602
	Eigenvalues, Eigenvectors, and the Singular-Value Decomposition	603
	Determinant and Trace	605
	Matrix Factorizations: Cholesky, LU, QR	606
	Symmetric Indefinite Factorization	610
	Sherman–Morrison–Woodbury Formula	612
	Interlacing Eigenvalue Theorem	613
	Error Analysis and Floating-Point Arithmetic	613
	Conditioning and Stability	616
A.2	Elements of Analysis, Geometry, Topology	617
	Sequences	617
	Rates of Convergence	619
	Topology of the Euclidean Space \mathbb{R}^n	620
	Convex Sets in \mathbb{R}^n	621
	Continuity and Limits	623
	Derivatives	625
	Directional Derivatives	628
	Mean Value Theorem	629
	Implicit Function Theorem	630
	Order Notation	631
	Root-Finding for Scalar Equations	633
B	A Regularization Procedure	635
	References	637
	Index	653

algorithms (Chapter 15) and the discussion of primal barrier methods moved to the new interior-point chapter. There is much new material in this part, including a treatment of nonlinear programming duality, an expanded discussion of algorithms for inequality constrained quadratic programming, a discussion of dual simplex and presolving in linear programming, a summary of practical issues in the implementation of interior-point linear programming algorithms, a description of conjugate-gradient methods for quadratic programming, and a discussion of filter methods and nonsmooth penalty methods in nonlinear programming algorithms.

In many chapters we have added a Perspectives and Software section near the end, to place the preceding discussion in context and discuss the state of the art in software. The appendix has been rearranged with some additional topics added, so that it can be used in a more stand-alone fashion to cover some of the mathematical background required for the rest of the book. The exercises have been revised in most chapters. After these many additions, deletions, and changes, the second edition is only slightly longer than the first, reflecting our belief that careful selection of the material to include and exclude is an important responsibility for authors of books of this type.

A manual containing solutions for selected problems will be available to bona fide instructors through the publisher. A list of typos will be maintained on the book's web site, which is accessible from the web pages of both authors.

We acknowledge with gratitude the comments and suggestions of many readers of the first edition, who sent corrections to many errors and provided valuable perspectives on the material, which led often to substantial changes. We mention in particular Frank Curtis, Michael Ferris, Andreas Griewank, Jacek Gondzio, Sven Leyffer, Philip Loewen, Rembert Reemtsen, and David Stewart.

Our special thanks goes to Michael Overton, who taught from a draft of the second edition and sent many detailed and excellent suggestions. We also thank colleagues who read various chapters of the new edition carefully during development, including Richard Byrd, Nick Gould, Paul Hovland, Gabo López-Calva, Long Hei, Katya Scheinberg, Andreas Wächter, and Richard Waltz. We thank Jill Wright for improving some of the figures and for the new cover graphic.

We mentioned in the original preface several areas of optimization that are not covered in this book. During the past six years, this list has only grown longer, as the field has continued to expand in new directions. In this regard, the following areas are particularly noteworthy: optimization problems with complementarity constraints, second-order cone and semidefinite programming, simulation-based optimization, robust optimization, and mixed-integer nonlinear programming. All these areas have seen theoretical and algorithmic advances in recent years, and in many cases developments are being driven by new classes of applications. Although this book does not cover any of these areas directly, it provides a foundation from which they can be studied.

CHAPTER *1*

Introduction

People optimize. Investors seek to create portfolios that avoid excessive risk while achieving a high rate of return. Manufacturers aim for maximum efficiency in the design and operation of their production processes. Engineers adjust parameters to optimize the performance of their designs.

Nature optimizes. Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time.

Optimization is an important tool in decision science and in the analysis of physical systems. To make use of this tool, we must first identify some *objective*, a quantitative measure of the performance of the system under study. This objective could be profit, time, potential energy, or any quantity or combination of quantities that can be represented by a single number. The objective depends on certain characteristics of the system, called *variables* or *unknowns*. Our goal is to find values of the variables that optimize the objective. Often the variables are restricted, or *constrained*, in some way. For instance, quantities such as electron density in a molecule and the interest rate on a loan cannot be negative.

The process of identifying objective, variables, and constraints for a given problem is known as *modeling*. Construction of an appropriate model is the first step—sometimes the most important step—in the optimization process. If the model is too simplistic, it will not give useful insights into the practical problem. If it is too complex, it may be too difficult to solve.

Once the model has been formulated, an optimization algorithm can be used to find its solution, usually with the help of a computer. There is no universal optimization algorithm but rather a collection of algorithms, each of which is tailored to a particular type of optimization problem. The responsibility of choosing the algorithm that is appropriate for a specific application often falls on the user. This choice is an important one, as it may determine whether the problem is solved rapidly or slowly and, indeed, whether the solution is found at all.

After an optimization algorithm has been applied to the model, we must be able to recognize whether it has succeeded in its task of finding a solution. In many cases, there are elegant mathematical expressions known as *optimality conditions* for checking that the current set of variables is indeed the solution of the problem. If the optimality conditions are not satisfied, they may give useful information on how the current estimate of the solution can be improved. The model may be improved by applying techniques such as *sensitivity analysis*, which reveals the sensitivity of the solution to changes in the model and data. Interpretation of the solution in terms of the application may also suggest ways in which the model can be refined or improved (or corrected). If any changes are made to the model, the optimization problem is solved anew, and the process repeats.

MATHEMATICAL FORMULATION

Mathematically speaking, optimization is the minimization or maximization of a function subject to constraints on its variables. We use the following notation:

- x is the vector of *variables*, also called *unknowns* or *parameters*;
- f is the *objective function*, a (scalar) function of x that we want to maximize or minimize;
- c_i are *constraint* functions, which are scalar functions of x that define certain equations and inequalities that the unknown vector x must satisfy.

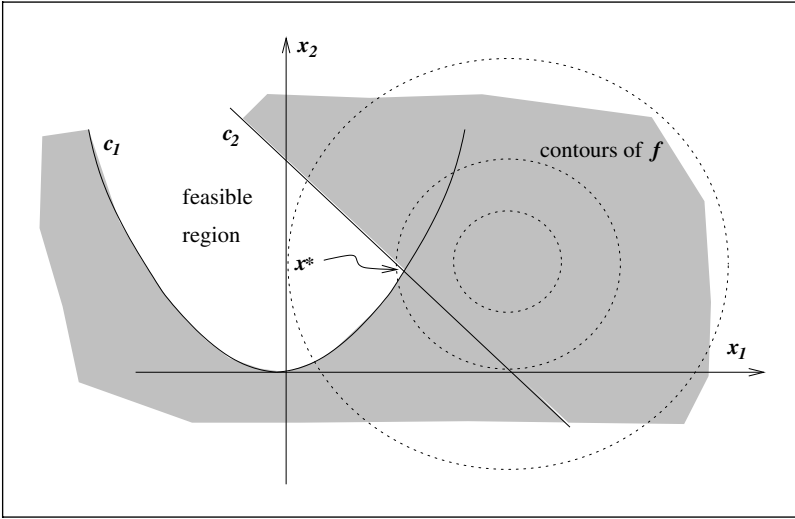


Figure 1.1 Geometrical representation of the problem (1.2).

Using this notation, the optimization problem can be written as follows:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, & i \in \mathcal{E}, \\ c_i(x) &\geq 0, & i \in \mathcal{I}. \end{aligned} \quad (1.1)$$

Here \mathcal{I} and \mathcal{E} are sets of indices for equality and inequality constraints, respectively.

As a simple example, consider the problem

$$\min (x_1 - 2)^2 + (x_2 - 1)^2 \quad \text{subject to} \quad \begin{aligned} x_1^2 - x_2 &\leq 0, \\ x_1 + x_2 &\leq 2. \end{aligned} \quad (1.2)$$

We can write this problem in the form (1.1) by defining

$$\begin{aligned} f(x) &= (x_1 - 2)^2 + (x_2 - 1)^2, & x &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \\ c(x) &= \begin{bmatrix} c_1(x) \\ c_2(x) \end{bmatrix} = \begin{bmatrix} -x_1^2 + x_2 \\ -x_1 - x_2 + 2 \end{bmatrix}, & \mathcal{I} &= \{1, 2\}, \quad \mathcal{E} = \emptyset. \end{aligned}$$

Figure 1.1 shows the contours of the objective function, that is, the set of points for which $f(x)$ has a constant value. It also illustrates the *feasible region*, which is the set of points satisfying all the constraints (the area between the two constraint boundaries), and the point

x^* , which is the solution of the problem. Note that the “infeasible side” of the inequality constraints is shaded.

The example above illustrates, too, that transformations are often necessary to express an optimization problem in the particular form (1.1). Often it is more natural or convenient to label the unknowns with two or three subscripts, or to refer to different variables by completely different names, so that relabeling is necessary to pose the problem in the form (1.1). Another common difference is that we are required to *maximize* rather than minimize f , but we can accommodate this change easily by *minimizing* $-f$ in the formulation (1.1). Good modeling systems perform the conversion to standardized formulations such as (1.1) transparently to the user.

EXAMPLE: A TRANSPORTATION PROBLEM

We begin with a much simplified example of a problem that might arise in manufacturing and transportation. A chemical company has 2 factories F_1 and F_2 and a dozen retail outlets R_1, R_2, \dots, R_{12} . Each factory F_i can produce a_i tons of a certain chemical product each week; a_i is called the *capacity* of the plant. Each retail outlet R_j has a known weekly *demand* of b_j tons of the product. The cost of shipping one ton of the product from factory F_i to retail outlet R_j is c_{ij} .

The problem is to determine how much of the product to ship from each factory to each outlet so as to satisfy all the requirements and minimize cost. The variables of the problem are x_{ij} , $i = 1, 2$, $j = 1, \dots, 12$, where x_{ij} is the number of tons of the product shipped from factory F_i to retail outlet R_j ; see Figure 1.2. We can write the problem as

$$\min \sum_{ij} c_{ij} x_{ij} \tag{1.3a}$$

$$\text{subject to } \sum_{j=1}^{12} x_{ij} \leq a_i, \quad i = 1, 2, \tag{1.3b}$$

$$\sum_{i=1}^2 x_{ij} \geq b_j, \quad j = 1, \dots, 12, \tag{1.3c}$$

$$x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, \dots, 12. \tag{1.3d}$$

This type of problem is known as a *linear programming* problem, since the objective function and the constraints are all linear functions. In a more practical model, we would also include costs associated with manufacturing and storing the product. There may be volume discounts in practice for shipping the product; for example the cost (1.3a) could be represented by $\sum_{ij} c_{ij} \sqrt{\delta + x_{ij}}$, where $\delta > 0$ is a small subscription fee. In this case, the problem is a *nonlinear program* because the objective function is nonlinear.

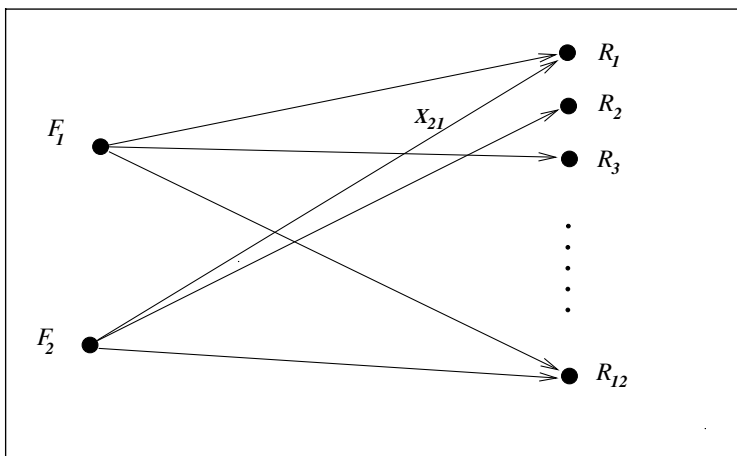


Figure 1.2 A transportation problem.

CONTINUOUS VERSUS DISCRETE OPTIMIZATION

In some optimization problems the variables make sense only if they take on integer values. For example, a variable x_i could represent the number of power plants of type i that should be constructed by an electricity provider during the next 5 years, or it could indicate whether or not a particular factory should be located in a particular city. The mathematical formulation of such problems includes integrality constraints, which have the form $x_i \in \mathbf{Z}$, where \mathbf{Z} is the set of integers, or binary constraints, which have the form $x_i \in \{0, 1\}$, in addition to algebraic constraints like those appearing in (1.1). Problems of this type are called *integer programming* problems. If some of the variables in the problem are *not* restricted to be integer or binary variables, they are sometimes called *mixed integer programming* problems, or MIPs for short.

Integer programming problems are a type of *discrete optimization* problem. Generally, discrete optimization problems may contain not only integers and binary variables, but also more abstract variable objects such as permutations of an ordered set. The defining feature of a discrete optimization problem is that the unknown x is drawn from a finite (but often very large) set. By contrast, the feasible set for *continuous optimization* problems—the class of problems studied in this book—is usually uncountably infinite, as when the components of x are allowed to be real numbers. Continuous optimization problems are normally easier to solve because the smoothness of the functions makes it possible to use objective and constraint information at a particular point x to deduce information about the function's behavior at all points close to x . In discrete problems, by contrast, the behavior of the objective and constraints may change significantly as we move from one feasible point to another, even if the two points are “close” by some measure. The feasible sets for discrete optimization problems can be thought of as exhibiting an extreme form of nonconvexity, as a convex combination of two feasible points is in general not feasible.

Discrete optimization problems are not addressed directly in this book; we refer the reader to the texts by Papadimitriou and Steiglitz [235], Nemhauser and Wolsey [224], Cook et al. [77], and Wolsey [312] for comprehensive treatments of this subject. We note, however, that continuous optimization techniques often play an important role in solving discrete optimization problems. For instance, the branch-and-bound method for integer linear programming problems requires the repeated solution of linear programming “relaxations,” in which some of the integer variables are fixed at integer values, while for other integer variables the integrality constraints are temporarily ignored. These subproblems are usually solved by the simplex method, which is discussed in Chapter 13 of this book.

CONSTRAINED AND UNCONSTRAINED OPTIMIZATION

Problems with the general form (1.1) can be classified according to the nature of the objective function and constraints (linear, nonlinear, convex), the number of variables (large or small), the smoothness of the functions (differentiable or nondifferentiable), and so on. An important distinction is between problems that have constraints on the variables and those that do not. This book is divided into two parts according to this classification.

Unconstrained optimization problems, for which we have $\mathcal{E} = \mathcal{I} = \emptyset$ in (1.1), arise directly in many practical applications. Even for some problems with natural constraints on the variables, it may be safe to disregard them as they do not affect on the solution and do not interfere with algorithms. Unconstrained problems arise also as reformulations of constrained optimization problems, in which the constraints are replaced by penalization terms added to objective function that have the effect of discouraging constraint violations.

Constrained optimization problems arise from models in which constraints play an essential role, for example in imposing budgetary constraints in an economic problem or shape constraints in a design problem. These constraints may be simple bounds such as $0 \leq x_1 \leq 100$, more general linear constraints such as $\sum_i x_i \leq 1$, or nonlinear inequalities that represent complex relationships among the variables.

When the objective function and all the constraints are linear functions of x , the problem is a *linear programming* problem. Problems of this type are probably the most widely formulated and solved of all optimization problems, particularly in management, financial, and economic applications. *Nonlinear programming* problems, in which at least some of the constraints or the objective are nonlinear functions, tend to arise naturally in the physical sciences and engineering, and are becoming more widely used in management and economic sciences as well.

GLOBAL AND LOCAL OPTIMIZATION

Many algorithms for nonlinear optimization problems seek only a local solution, a point at which the objective function is smaller than at all other feasible nearby points. They do not always find the *global solution*, which is the point with lowest function value among *all* feasible points. Global solutions are needed in some applications, but for many problems they

are difficult to recognize and even more difficult to locate. For *convex programming* problems, and more particularly for linear programs, local solutions are also global solutions. General nonlinear problems, both constrained and unconstrained, may possess local solutions that are not global solutions.

In this book we treat global optimization only in passing and focus instead on the computation and characterization of local solutions. We note, however, that many successful global optimization algorithms require the solution of many local optimization problems, to which the algorithms described in this book can be applied.

Research papers on global optimization can be found in Floudas and Pardalos [109] and in the *Journal of Global Optimization*.

STOCHASTIC AND DETERMINISTIC OPTIMIZATION

In some optimization problems, the model cannot be fully specified because it depends on quantities that are unknown at the time of formulation. This characteristic is shared by many economic and financial planning models, which may depend for example on future interest rates, future demands for a product, or future commodity prices, but uncertainty can arise naturally in almost any type of application.

Rather than just use a “best guess” for the uncertain quantities, modelers may obtain more useful solutions by incorporating additional knowledge about these quantities into the model. For example, they may know a number of possible scenarios for the uncertain demand, along with estimates of the probabilities of each scenario. *Stochastic optimization* algorithms use these quantifications of the uncertainty to produce solutions that optimize the *expected* performance of the model.

Related paradigms for dealing with uncertain data in the model include *chance-constrained optimization*, in which we ensure that the variables x satisfy the given constraints to some specified probability, and *robust optimization*, in which certain constraints are required to hold for all possible values of the uncertain data.

We do not consider stochastic optimization problems further in this book, focusing instead on *deterministic optimization* problems, in which the model is completely known. Many algorithms for stochastic optimization do, however, proceed by formulating one or more deterministic subproblems, each of which can be solved by the techniques outlined here.

Stochastic and robust optimization have seen a great deal of recent research activity. For further information on stochastic optimization, consult the books of Birge and Louveaux [22] and Kall and Wallace [174]. Robust optimization is discussed in Ben-Tal and Nemirovski [15].

CONVEXITY

The concept of convexity is fundamental in optimization. Many practical problems possess this property, which generally makes them easier to solve both in theory and practice.

The term “convex” can be applied both to sets and to functions. A set $S \in \mathbb{R}^n$ is a *convex set* if the straight line segment connecting any two points in S lies entirely inside S . Formally, for any two points $x \in S$ and $y \in S$, we have $\alpha x + (1 - \alpha)y \in S$ for all $\alpha \in [0, 1]$. The function f is a *convex function* if its domain S is a convex set and if for any two points x and y in S , the following property is satisfied:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \quad \text{for all } \alpha \in [0, 1]. \quad (1.4)$$

Simple instances of convex sets include the unit ball $\{y \in \mathbb{R}^n \mid \|y\|_2 \leq 1\}$; and any polyhedron, which is a set defined by linear equalities and inequalities, that is,

$$\{x \in \mathbb{R}^n \mid Ax = b, \ Cx \leq d\},$$

where A and C are matrices of appropriate dimension, and b and d are vectors. Simple instances of convex functions include the linear function $f(x) = c^T x + \alpha$, for any constant vector $c \in \mathbb{R}^n$ and scalar α ; and the convex quadratic function $f(x) = x^T H x$, where H is a symmetric positive semidefinite matrix.

We say that f is *strictly convex* if the inequality in (1.4) is strict whenever $x \neq y$ and α is in the open interval $(0, 1)$. A function f is said to be *concave* if $-f$ is convex.

If the objective function in the optimization problem (1.1) and the feasible region are both convex, then any local solution of the problem is in fact a global solution.

The term *convex programming* is used to describe a special case of the general constrained optimization problem (1.1) in which

- the objective function is convex,
- the equality constraint functions $c_i(\cdot)$, $i \in \mathcal{E}$, are linear, and
- the inequality constraint functions $c_i(\cdot)$, $i \in \mathcal{I}$, are concave.

OPTIMIZATION ALGORITHMS

Optimization algorithms are iterative. They begin with an initial guess of the variable x and generate a sequence of improved estimates (called “iterates”) until they terminate, hopefully at a solution. The strategy used to move from one iterate to the next distinguishes one algorithm from another. Most strategies make use of the values of the objective function f , the constraint functions c_i , and possibly the first and second derivatives of these functions. Some algorithms accumulate information gathered at previous iterations, while others use only local information obtained at the current point. Regardless of these specifics (which will receive plenty of attention in the rest of the book), good algorithms should possess the following properties:

- **Robustness.** They should perform well on a wide variety of problems in their class, for all reasonable values of the starting point.

- Efficiency. They should not require excessive computer time or storage.
- Accuracy. They should be able to identify a solution with precision, without being overly sensitive to errors in the data or to the arithmetic rounding errors that occur when the algorithm is implemented on a computer.

These goals may conflict. For example, a rapidly convergent method for a large unconstrained nonlinear problem may require too much computer storage. On the other hand, a robust method may also be the slowest. Tradeoffs between convergence rate and storage requirements, and between robustness and speed, and so on, are central issues in numerical optimization. They receive careful consideration in this book.

The mathematical theory of optimization is used both to characterize optimal points and to provide the basis for most algorithms. It is not possible to have a good understanding of numerical optimization without a firm grasp of the supporting theory. Accordingly, this book gives a solid (though not comprehensive) treatment of optimality conditions, as well as convergence analysis that reveals the strengths and weaknesses of some of the most important algorithms.

NOTES AND REFERENCES

Optimization traces its roots to the calculus of variations and the work of Euler and Lagrange. The development of linear programming in the 1940s broadened the field and stimulated much of the progress in modern optimization theory and practice during the past 60 years.

Optimization is often called *mathematical programming*, a somewhat confusing term coined in the 1940s, before the word “programming” became inextricably linked with computer software. The original meaning of this word (and the intended one in this context) was more inclusive, with connotations of algorithm design and analysis.

Modeling will not be treated extensively in the book. It is an essential subject in its own right, as it makes the connection between optimization algorithms and software on the one hand, and applications on the other hand. Information about modeling techniques for various application areas can be found in Dantzig [86], Ahuja, Magnanti, and Orlin [1], Fourer, Gay, and Kernighan [112], Winston [308], and Rardin [262].

CHAPTER 2

Fundamentals of Unconstrained Optimization

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$\min_x f(x), \tag{2.1}$$

where $x \in \mathbb{R}^n$ is a real vector with $n \geq 1$ components and $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a smooth function.

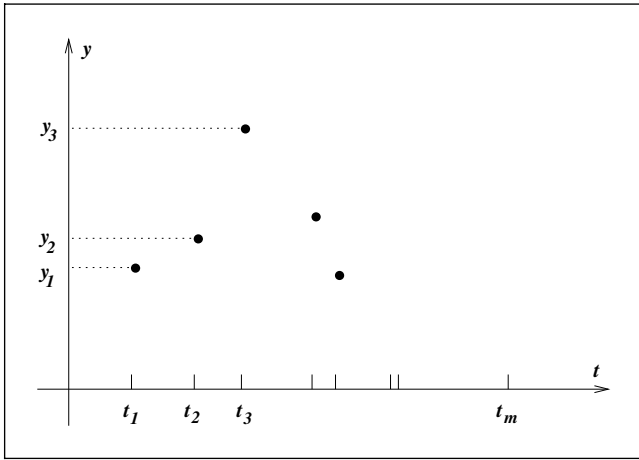


Figure 2.1 Least squares data fitting problem.

Usually, we lack a global perspective on the function f . All we know are the values of f and maybe some of its derivatives at a set of points x_0, x_1, x_2, \dots . Fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about f does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily.

EXAMPLE 2.1

Suppose that we are trying to find a curve that fits some experimental data. Figure 2.1 plots measurements y_1, y_2, \dots, y_m of a signal taken at times t_1, t_2, \dots, t_m . From the data and our knowledge of the application, we deduce that the signal has exponential and oscillatory behavior of certain types, and we choose to model it by the function

$$\phi(t; x) = x_1 + x_2 e^{-(x_3 - t)^2 / x_4} + x_5 \cos(x_6 t).$$

The real numbers x_i , $i = 1, 2, \dots, 6$, are the parameters of the model; we would like to choose them to make the model values $\phi(t_j; x)$ fit the observed data y_j as closely as possible. To state our objective as an optimization problem, we group the parameters x_i into a vector of unknowns $x = (x_1, x_2, \dots, x_6)^T$, and define the residuals

$$r_j(x) = y_j - \phi(t_j; x), \quad j = 1, 2, \dots, m, \quad (2.2)$$

which measure the discrepancy between the model and the observed data. Our estimate of

x will be obtained by solving the problem

$$\min_{x \in \mathbb{R}^6} f(x) = r_1^2(x) + r_2^2(x) + \cdots + r_m^2(x). \quad (2.3)$$

This is a *nonlinear least-squares problem*, a special case of unconstrained optimization. It illustrates that some objective functions can be expensive to evaluate even when the number of variables is small. Here we have $n = 6$, but if the number of measurements m is large (10^5 , say), evaluation of $f(x)$ for a given parameter vector x is a significant computation. □

Suppose that for the data given in Figure 2.1 the optimal solution of (2.3) is approximately $x^* = (1.1, 0.01, 1.2, 1.5, 2.0, 1.5)$ and the corresponding function value is $f(x^*) = 0.34$. Because the optimal objective is nonzero, there must be discrepancies between the observed measurements y_j and the model predictions $\phi(t_j, x^*)$ for some (usually most) values of j —the model has not reproduced all the data points exactly. How, then, can we verify that x^* is indeed a minimizer of f ? To answer this question, we need to define the term “solution” and explain how to recognize solutions. Only then can we discuss algorithms for unconstrained optimization problems.

2.1 WHAT IS A SOLUTION?

Generally, we would be happiest if we found a *global minimizer* of f , a point where the function attains its least value. A formal definition is

A point x^* is a *global minimizer* if $f(x^*) \leq f(x)$ for all x ,

where x ranges over all of \mathbb{R}^n (or at least over the domain of interest to the modeler). The global minimizer can be difficult to find, because our knowledge of f is usually only local. Since our algorithm does not visit many points (we hope!), we usually do not have a good picture of the overall shape of f , and we can never be sure that the function does not take a sharp dip in some region that has not been sampled by the algorithm. Most algorithms are able to find only a *local minimizer*, which is a point that achieves the smallest value of f in its neighborhood. Formally, we say:

A point x^* is a *local minimizer* if there is a neighborhood \mathcal{N} of x^* such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$.

(Recall that a neighborhood of x^* is simply an open set that contains x^* .) A point that satisfies this definition is sometimes called a *weak local minimizer*. This terminology distinguishes

it from a strict local minimizer, which is the outright winner in its neighborhood. Formally,

A point x^* is a *strict local minimizer* (also called a *strong local minimizer*) if there is a neighborhood \mathcal{N} of x^* such that $f(x^*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x^*$.

For the constant function $f(x) = 2$, every point x is a weak local minimizer, while the function $f(x) = (x - 2)^4$ has a strict local minimizer at $x = 2$.

A slightly more exotic type of local minimizer is defined as follows.

A point x^* is an *isolated local minimizer* if there is a neighborhood \mathcal{N} of x^* such that x^* is the only local minimizer in \mathcal{N} .

Some strict local minimizers are not isolated, as illustrated by the function

$$f(x) = x^4 \cos(1/x) + 2x^4, \quad f(0) = 0,$$

which is twice continuously differentiable and has a strict local minimizer at $x^* = 0$. However, there are strict local minimizers at many nearby points x_j , and we can label these points so that $x_j \rightarrow 0$ as $j \rightarrow \infty$.

While strict local minimizers are not always isolated, it is true that all isolated local minimizers are strict.

Figure 2.2 illustrates a function with many local minimizers. It is usually difficult to find the global minimizer for such functions, because algorithms tend to be “trapped” at local minimizers. This example is by no means pathological. In optimization problems associated with the determination of molecular conformation, the potential function to be minimized may have millions of local minima.

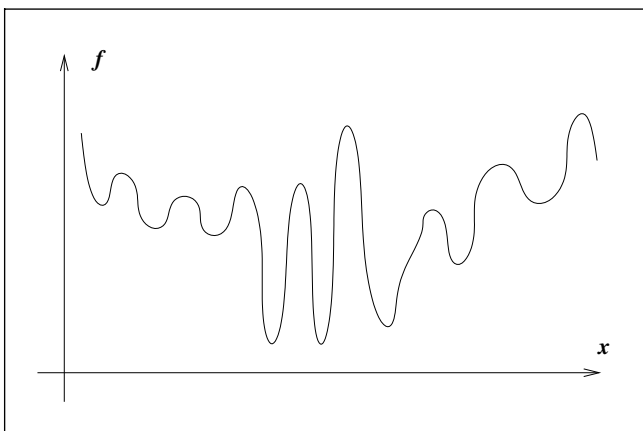


Figure 2.2 A difficult case for global minimization.

Sometimes we have additional “global” knowledge about f that may help in identifying global minima. An important special case is that of convex functions, for which every local minimizer is also a global minimizer.

RECOGNIZING A LOCAL MINIMUM

From the definitions given above, it might seem that the only way to find out whether a point x^* is a local minimum is to examine all the points in its immediate vicinity, to make sure that none of them has a smaller function value. When the function f is *smooth*, however, there are more efficient and practical ways to identify local minima. In particular, if f is twice continuously differentiable, we may be able to tell that x^* is a local minimizer (and possibly a strict local minimizer) by examining just the gradient $\nabla f(x^*)$ and the Hessian $\nabla^2 f(x^*)$.

The mathematical tool used to study minimizers of smooth functions is Taylor’s theorem. Because this theorem is central to our analysis throughout the book, we state it now. Its proof can be found in any calculus textbook.

Theorem 2.1 (Taylor’s Theorem).

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable and that $p \in \mathbb{R}^n$. Then we have that

$$f(x + p) = f(x) + \nabla f(x + tp)^T p, \quad (2.4)$$

for some $t \in (0, 1)$. Moreover, if f is twice continuously differentiable, we have that

$$\nabla f(x + p) = \nabla f(x) + \int_0^1 \nabla^2 f(x + tp) p \, dt, \quad (2.5)$$

and that

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p, \quad (2.6)$$

for some $t \in (0, 1)$.

Necessary conditions for optimality are derived by assuming that x^* is a local minimizer and then proving facts about $\nabla f(x^*)$ and $\nabla^2 f(x^*)$.

Theorem 2.2 (First-Order Necessary Conditions).

If x^ is a local minimizer and f is continuously differentiable in an open neighborhood of x^* , then $\nabla f(x^*) = 0$.*

PROOF. Suppose for contradiction that $\nabla f(x^*) \neq 0$. Define the vector $p = -\nabla f(x^*)$ and note that $p^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$. Because ∇f is continuous near x^* , there is a scalar $T > 0$ such that

$$p^T \nabla f(x^* + tp) < 0, \quad \text{for all } t \in [0, T].$$

For any $\bar{t} \in (0, T]$, we have by Taylor's theorem that

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^* + tp), \quad \text{for some } t \in (0, \bar{t}).$$

Therefore, $f(x^* + \bar{t}p) < f(x^*)$ for all $\bar{t} \in (0, T]$. We have found a direction leading away from x^* along which f decreases, so x^* is not a local minimizer, and we have a contradiction. \square

We call x^* a *stationary point* if $\nabla f(x^*) = 0$. According to Theorem 2.2, any local minimizer must be a stationary point.

For the next result we recall that a matrix B is positive definite if $p^T B p > 0$ for all $p \neq 0$, and positive semidefinite if $p^T B p \geq 0$ for all p (see the Appendix).

Theorem 2.3 (Second-Order Necessary Conditions).

If x^ is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open neighborhood of x^* , then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.*

PROOF. We know from Theorem 2.2 that $\nabla f(x^*) = 0$. For contradiction, assume that $\nabla^2 f(x^*)$ is not positive semidefinite. Then we can choose a vector p such that $p^T \nabla^2 f(x^*) p < 0$, and because $\nabla^2 f$ is continuous near x^* , there is a scalar $T > 0$ such that $p^T \nabla^2 f(x^* + tp) p < 0$ for all $t \in [0, T]$.

By doing a Taylor series expansion around x^* , we have for all $\bar{t} \in (0, T]$ and some $t \in (0, \bar{t})$ that

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^*) + \frac{1}{2}\bar{t}^2 p^T \nabla^2 f(x^* + tp) p < f(x^*).$$

As in Theorem 2.2, we have found a direction from x^* along which f is decreasing, and so again, x^* is not a local minimizer. \square

We now describe *sufficient conditions*, which are conditions on the derivatives of f at the point x^* that guarantee that x^* is a local minimizer.

Theorem 2.4 (Second-Order Sufficient Conditions).

Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f .

PROOF. Because the Hessian is continuous and positive definite at x^* , we can choose a radius $r > 0$ so that $\nabla^2 f(x)$ remains positive definite for all x in the open ball $\mathcal{D} = \{z \mid \|z - x^*\| < r\}$. Taking any nonzero vector p with $\|p\| < r$, we have $x^* + p \in \mathcal{D}$ and so

$$\begin{aligned} f(x^* + p) &= f(x^*) + p^T \nabla f(x^*) + \frac{1}{2} p^T \nabla^2 f(z) p \\ &= f(x^*) + \frac{1}{2} p^T \nabla^2 f(z) p, \end{aligned}$$

where $z = x^* + tp$ for some $t \in (0, 1)$. Since $z \in \mathcal{D}$, we have $p^T \nabla^2 f(z) p > 0$, and therefore $f(x^* + p) > f(x^*)$, giving the result. \square

Note that the second-order sufficient conditions of Theorem 2.4 guarantee something stronger than the necessary conditions discussed earlier; namely, that the minimizer is a *strict* local minimizer. Note too that the second-order sufficient conditions are not necessary: A point x^* may be a strict local minimizer, and yet may fail to satisfy the sufficient conditions. A simple example is given by the function $f(x) = x^4$, for which the point $x^* = 0$ is a strict local minimizer at which the Hessian matrix vanishes (and is therefore not positive definite).

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem 2.5.

When f is convex, any local minimizer x^* is a global minimizer of f . If in addition f is differentiable, then any stationary point x^* is a global minimizer of f .

PROOF. Suppose that x^* is a local but not a global minimizer. Then we can find a point $z \in \mathbb{R}^n$ with $f(z) < f(x^*)$. Consider the line segment that joins x^* to z , that is,

$$x = \lambda z + (1 - \lambda)x^*, \quad \text{for some } \lambda \in (0, 1]. \quad (2.7)$$

By the convexity property for f , we have

$$f(x) \leq \lambda f(z) + (1 - \lambda)f(x^*) < f(x^*). \quad (2.8)$$

Any neighborhood \mathcal{N} of x^* contains a piece of the line segment (2.7), so there will always be points $x \in \mathcal{N}$ at which (2.8) is satisfied. Hence, x^* is not a local minimizer.

For the second part of the theorem, suppose that x^* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x^*)^T(z - x^*) &= \frac{d}{d\lambda} f(x^* + \lambda(z - x^*))|_{\lambda=0} \quad (\text{see the Appendix}) \\ &= \lim_{\lambda \downarrow 0} \frac{f(x^* + \lambda(z - x^*)) - f(x^*)}{\lambda} \\ &\leq \lim_{\lambda \downarrow 0} \frac{\lambda f(z) + (1 - \lambda)f(x^*) - f(x^*)}{\lambda} \\ &= f(z) - f(x^*) < 0.\end{aligned}$$

Therefore, $\nabla f(x^*) \neq 0$, and so x^* is not a stationary point. \square

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

NONSMOOTH PROBLEMS

This book focuses on smooth functions, by which we generally mean functions whose second derivatives exist and are continuous. We note, however, that there are interesting problems in which the functions involved may be nonsmooth and even discontinuous. It is not possible in general to identify a minimizer of a general discontinuous function. If, however, the function consists of a few smooth pieces, with discontinuities between the pieces, it may be possible to find the minimizer by minimizing each smooth piece individually.

If the function is continuous everywhere but nondifferentiable at certain points, as in Figure 2.3, we can identify a solution by examining the *subgradient* or *generalized*

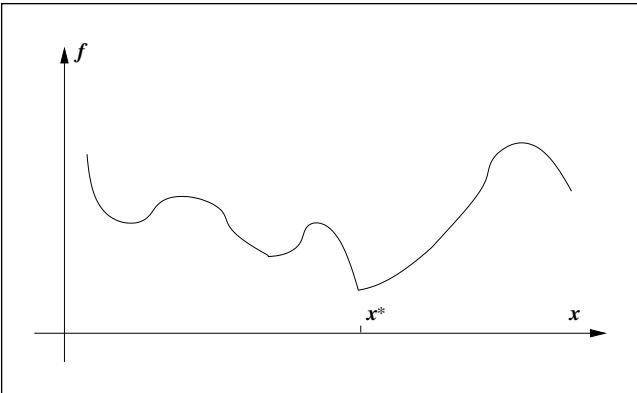


Figure 2.3 Nonsmooth function with minimum at a kink.

gradient, which are generalizations of the concept of gradient to the nonsmooth case. Nonsmooth optimization is beyond the scope of this book; we refer instead to Hiriart-Urruty and Lemaréchal [170] for an extensive discussion of theory. Here, we mention only that the minimization of a function such as the one illustrated in Figure 2.3 (which contains a jump discontinuity in the first derivative $f'(x)$ at the minimum) is difficult because the behavior of f is not predictable near the point of nonsmoothness. That is, we cannot be sure that information about f obtained at one point can be used to infer anything about f at neighboring points, because points of nondifferentiability may intervene. However, minimization of certain special nondifferentiable functions, such as

$$f(x) = \|r(x)\|_1, \quad f(x) = \|r(x)\|_\infty \quad (2.9)$$

(where $r(x)$ is a vector function), can be reformulated as smooth constrained optimization problems; see Exercise 12.5 in Chapter 12 and (17.31). The functions (2.9) are useful in data fitting, where $r(x)$ is the residual vector whose components are defined in (2.2).

2.2 OVERVIEW OF ALGORITHMS

The last forty years have seen the development of a powerful collection of algorithms for unconstrained optimization of smooth functions. We now give a broad description of their main properties, and we describe them in more detail in Chapters 3, 4, 5, 6, and 7. All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by x_0 . The user with knowledge about the application and the data set may be in a good position to choose x_0 to be a reasonable estimate of the solution. Otherwise, the starting point must be chosen by the algorithm, either by a systematic approach or in some arbitrary manner.

Beginning at x_0 , optimization algorithms generate a sequence of iterates $\{x_k\}_{k=0}^\infty$ that terminate when either no more progress can be made or when it seems that a solution point has been approximated with sufficient accuracy. In deciding how to move from one iterate x_k to the next, the algorithms use information about the function f at x_k , and possibly also information from earlier iterates x_0, x_1, \dots, x_{k-1} . They use this information to find a new iterate x_{k+1} with a lower function value than x_k . (There exist *nonmonotone* algorithms that do not insist on a decrease in f at every step, but even these algorithms require f to be decreased after some prescribed number m of iterations, that is, $f(x_k) < f(x_{k-m})$.)

There are two fundamental strategies for moving from the current point x_k to a new iterate x_{k+1} . Most of the algorithms described in this book follow one of these approaches.

TWO STRATEGIES: LINE SEARCH AND TRUST REGION

In the *line search* strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along p_k can be found by approximately solving the following one-dimensional minimization problem to find a step length α :

$$\min_{\alpha > 0} f(x_k + \alpha p_k). \quad (2.10)$$

By solving (2.10) exactly, we would derive the maximum benefit from the direction p_k , but an exact minimization may be expensive and is usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of (2.10). At the new point, a new search direction and step length are computed, and the process is repeated.

In the second algorithmic strategy, known as *trust region*, the information gathered about f is used to construct a *model function* m_k whose behavior near the current point x_k is similar to that of the actual objective function f . Because the model m_k may not be a good approximation of f when x is far from x_k , we restrict the search for a minimizer of m_k to some region around x_k . In other words, we find the candidate step p by approximately solving the following subproblem:

$$\min_p m_k(x_k + p), \quad \text{where } x_k + p \text{ lies inside the trust region.} \quad (2.11)$$

If the candidate solution does not produce a sufficient decrease in f , we conclude that the trust region is too large, and we shrink it and re-solve (2.11). Usually, the trust region is a ball defined by $\|p\|_2 \leq \Delta$, where the scalar $\Delta > 0$ is called the trust-region radius. Elliptical and box-shaped trust regions may also be used.

The model m_k in (2.11) is usually defined to be a quadratic function of the form

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p, \quad (2.12)$$

where f_k , ∇f_k , and B_k are a scalar, vector, and matrix, respectively. As the notation indicates, f_k and ∇f_k are chosen to be the function and gradient values at the point x_k , so that m_k and f are in agreement to first order at the current iterate x_k . The matrix B_k is either the Hessian $\nabla^2 f_k$ or some approximation to it.

Suppose that the objective function is given by $f(x) = 10(x_2 - x_1^2)^2 + (1 - x_1)^2$. At the point $x_k = (0, 1)$ its gradient and Hessian are

$$\nabla f_k = \begin{bmatrix} -2 \\ 20 \end{bmatrix}, \quad \nabla^2 f_k = \begin{bmatrix} -38 & 0 \\ 0 & 20 \end{bmatrix}.$$

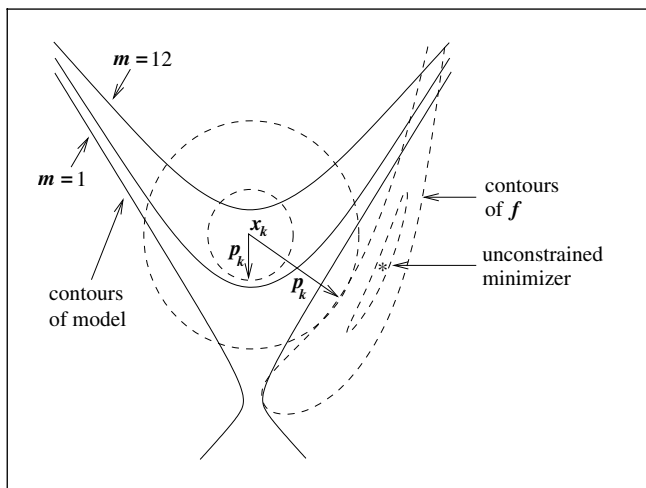


Figure 2.4 Two possible trust regions (circles) and their corresponding steps p_k . The solid lines are contours of the model function m_k .

The contour lines of the quadratic model (2.12) with $B_k = \nabla^2 f_k$ are depicted in Figure 2.4, which also illustrates the contours of the objective function f and the trust region. We have indicated contour lines where the model m_k has values 1 and 12. Note from Figure 2.4 that each time we decrease the size of the trust region after failure of a candidate iterate, the step from x_k to the new candidate will be shorter, and it usually points in a different direction from the previous candidate. The trust-region strategy differs in this respect from line search, which stays with a single search direction.

In a sense, the line search and trust-region approaches differ in the order in which they choose the *direction* and *distance* of the move to the next iterate. Line search starts by fixing the direction p_k and then identifying an appropriate distance, namely the step length α_k . In trust region, we first choose a maximum distance—the trust-region radius Δ_k —and then seek a direction and step that attain the best improvement possible subject to this distance constraint. If this step proves to be unsatisfactory, we reduce the distance measure Δ_k and try again.

The line search approach is discussed in more detail in Chapter 3. Chapter 4 discusses the trust-region strategy, including techniques for choosing and adjusting the size of the region and for computing approximate solutions to the trust-region problems (2.11). We now preview two major issues: choice of the search direction p_k in line search methods, and choice of the Hessian B_k in trust-region methods. These issues are closely related, as we now observe.

SEARCH DIRECTIONS FOR LINE SEARCH METHODS

The steepest descent direction $-\nabla f_k$ is the most obvious choice for search direction for a line search method. It is intuitive; among all the directions we could move from x_k ,

it is the one along which f decreases most rapidly. To verify this claim, we appeal again to Taylor's theorem (Theorem 2.1), which tells us that for any search direction p and step-length parameter α , we have

$$f(x_k + \alpha p) = f(x_k) + \alpha p^T \nabla f_k + \frac{1}{2} \alpha^2 p^T \nabla^2 f(x_k + t p) p, \quad \text{for some } t \in (0, \alpha)$$

(see (2.6)). The rate of change in f along the direction p at x_k is simply the coefficient of α , namely, $p^T \nabla f_k$. Hence, the unit direction p of most rapid decrease is the solution to the problem

$$\min_p p^T \nabla f_k, \quad \text{subject to } \|p\| = 1. \quad (2.13)$$

Since $p^T \nabla f_k = \|p\| \|\nabla f_k\| \cos \theta = \|\nabla f_k\| \cos \theta$, where θ is the angle between p and ∇f_k , it is easy to see that the minimizer is attained when $\cos \theta = -1$ and

$$p = -\nabla f_k / \|\nabla f_k\|,$$

as claimed. As we illustrate in Figure 2.5, this direction is orthogonal to the contours of the function.

The *steepest descent method* is a line search method that moves along $p_k = -\nabla f_k$ at every step. It can choose the step length α_k in a variety of ways, as we discuss in Chapter 3. One advantage of the steepest descent direction is that it requires calculation of the gradient ∇f_k but not of second derivatives. However, it can be excruciatingly slow on difficult problems.

Line search methods may use search directions other than the steepest descent direction. In general, any *descent* direction—one that makes an angle of strictly less than $\pi/2$ radians with $-\nabla f_k$ —is guaranteed to produce a decrease in f , provided that the step length

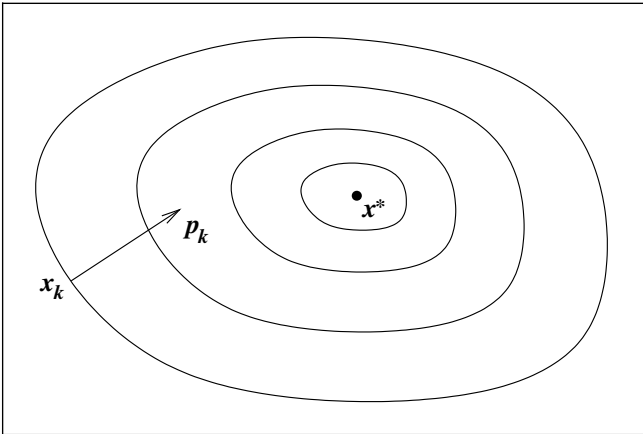


Figure 2.5 Steepest descent direction for a function of two variables.

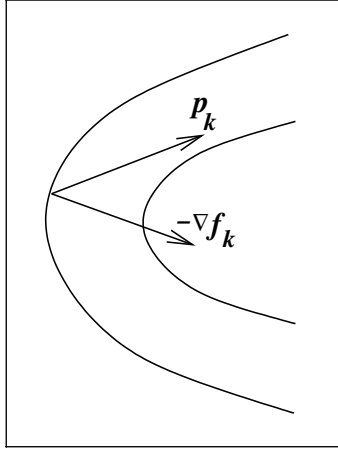


Figure 2.6
A downhill direction p_k .

is sufficiently small (see Figure 2.6). We can verify this claim by using Taylor's theorem. From (2.6), we have that

$$f(x_k + \epsilon p_k) = f(x_k) + \epsilon p_k^T \nabla f_k + O(\epsilon^2).$$

When p_k is a downhill direction, the angle θ_k between p_k and ∇f_k has $\cos \theta_k < 0$, so that

$$p_k^T \nabla f_k = \|p_k\| \|\nabla f_k\| \cos \theta_k < 0.$$

It follows that $f(x_k + \epsilon p_k) < f(x_k)$ for all positive but sufficiently small values of ϵ .

Another important search direction—perhaps the most important one of all—is the *Newton direction*. This direction is derived from the second-order Taylor series approximation to $f(x_k + p)$, which is

$$f(x_k + p) \approx f_k + p^T \nabla f_k + \frac{1}{2} p^T \nabla^2 f_k p \stackrel{\text{def}}{=} m_k(p). \quad (2.14)$$

Assuming for the moment that $\nabla^2 f_k$ is positive definite, we obtain the Newton direction by finding the vector p that minimizes $m_k(p)$. By simply setting the derivative of $m_k(p)$ to zero, we obtain the following explicit formula:

$$p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k. \quad (2.15)$$

The Newton direction is reliable when the difference between the true function $f(x_k + p)$ and its quadratic model $m_k(p)$ is not too large. By comparing (2.14) with (2.6), we see that the only difference between these functions is that the matrix $\nabla^2 f(x_k + tp)$ in the third term of the expansion has been replaced by $\nabla^2 f_k$. If $\nabla^2 f$ is sufficiently smooth, this difference introduces a perturbation of only $O(\|p\|^3)$ into the expansion, so that when $\|p\|$ is small, the approximation $f(x_k + p) \approx m_k(p)$ is quite accurate.

The Newton direction can be used in a line search method when $\nabla^2 f_k$ is positive definite, for in this case we have

$$\nabla f_k^T p_k^N = -p_k^{NT} \nabla^2 f_k p_k^N \leq -\sigma_k \|p_k^N\|^2$$

for some $\sigma_k > 0$. Unless the gradient ∇f_k (and therefore the step p_k^N) is zero, we have that $\nabla f_k^T p_k^N < 0$, so the Newton direction is a descent direction.

Unlike the steepest descent direction, there is a “natural” step length of 1 associated with the Newton direction. Most line search implementations of Newton’s method use the unit step $\alpha = 1$ where possible and adjust α only when it does not produce a satisfactory reduction in the value of f .

When $\nabla^2 f_k$ is not positive definite, the Newton direction may not even be defined, since $(\nabla^2 f_k)^{-1}$ may not exist. Even when it is defined, it may not satisfy the descent property $\nabla f_k^T p_k^N < 0$, in which case it is unsuitable as a search direction. In these situations, line search methods modify the definition of p_k to make it satisfy the descent condition while retaining the benefit of the second-order information contained in $\nabla^2 f_k$. We describe these modifications in Chapter 3.

Methods that use the Newton direction have a fast rate of local convergence, typically quadratic. After a neighborhood of the solution is reached, convergence to high accuracy often occurs in just a few iterations. The main drawback of the Newton direction is the need for the Hessian $\nabla^2 f(x)$. Explicit computation of this matrix of second derivatives can sometimes be a cumbersome, error-prone, and expensive process. Finite-difference and automatic differentiation techniques described in Chapter 8 may be useful in avoiding the need to calculate second derivatives by hand.

Quasi-Newton search directions provide an attractive alternative to Newton’s method in that they do not require computation of the Hessian and yet still attain a superlinear rate of convergence. In place of the true Hessian $\nabla^2 f_k$, they use an approximation B_k , which is updated after each step to take account of the additional knowledge gained during the step. The updates make use of the fact that changes in the gradient g provide information about the second derivative of f along the search direction. By using the expression (2.5) from our statement of Taylor’s theorem, we have by adding and subtracting the term $\nabla^2 f(x)p$ that

$$\nabla f(x + p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 [\nabla^2 f(x + tp) - \nabla^2 f(x)] p \, dt.$$

Because $\nabla f(\cdot)$ is continuous, the size of the final integral term is $o(\|p\|)$. By setting $x = x_k$ and $p = x_{k+1} - x_k$, we obtain

$$\nabla f_{k+1} = \nabla f_k + \nabla^2 f_k(x_{k+1} - x_k) + o(\|x_{k+1} - x_k\|).$$

When x_k and x_{k+1} lie in a region near the solution x^* , within which $\nabla^2 f$ is positive definite, the final term in this expansion is eventually dominated by the $\nabla^2 f_k(x_{k+1} - x_k)$ term, and

we can write

$$\nabla^2 f_k(x_{k+1} - x_k) \approx \nabla f_{k+1} - \nabla f_k. \quad (2.16)$$

We choose the new Hessian approximation B_{k+1} so that it mimics the property (2.16) of the true Hessian, that is, we require it to satisfy the following condition, known as the *secant equation*:

$$B_{k+1}s_k = y_k, \quad (2.17)$$

where

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f_{k+1} - \nabla f_k.$$

Typically, we impose additional conditions on B_{k+1} , such as symmetry (motivated by symmetry of the exact Hessian), and a requirement that the difference between successive approximations B_k and B_{k+1} have low rank.

Two of the most popular formulae for updating the Hessian approximation B_k are the *symmetric-rank-one* (SR1) formula, defined by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}, \quad (2.18)$$

and the *BFGS formula*, named after its inventors, Broyden, Fletcher, Goldfarb, and Shanno, which is defined by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \quad (2.19)$$

Note that the difference between the matrices B_k and B_{k+1} is a rank-one matrix in the case of (2.18) and a rank-two matrix in the case of (2.19). Both updates satisfy the secant equation and both maintain symmetry. One can show that BFGS update (2.19) generates positive definite approximations whenever the initial approximation B_0 is positive definite and $s_k^T y_k > 0$. We discuss these issues further in Chapter 6.

The quasi-Newton search direction is obtained by using B_k in place of the exact Hessian in the formula (2.15), that is,

$$p_k = -B_k^{-1} \nabla f_k. \quad (2.20)$$

Some practical implementations of quasi-Newton methods avoid the need to factorize B_k at each iteration by updating the *inverse* of B_k , instead of B_k itself. In fact, the equivalent

formula for (2.18) and (2.19), applied to the inverse approximation $H_k \stackrel{\text{def}}{=} B_k^{-1}$, is

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k}. \quad (2.21)$$

Calculation of p_k can then be performed by using the formula $p_k = -H_k \nabla f_k$. This matrix–vector multiplication is simpler than the factorization/back-substitution procedure that is needed to implement the formula (2.20).

Two variants of quasi-Newton methods designed to solve large problems—partially separable and limited-memory updating—are described in Chapter 7.

The last class of search directions we preview here is that generated by *nonlinear conjugate gradient methods*. They have the form

$$p_k = -\nabla f(x_k) + \beta_k p_{k-1},$$

where β_k is a scalar that ensures that p_k and p_{k-1} are *conjugate*—an important concept in the minimization of quadratic functions that will be defined in Chapter 5. Conjugate gradient methods were originally designed to solve systems of linear equations $Ax = b$, where the coefficient matrix A is symmetric and positive definite. The problem of solving this linear system is equivalent to the problem of minimizing the convex quadratic function defined by

$$\phi(x) = \frac{1}{2} x^T A x - b^T x,$$

so it was natural to investigate extensions of these algorithms to more general types of unconstrained minimization problems. In general, nonlinear conjugate gradient directions are much more effective than the steepest descent direction and are almost as simple to compute. These methods do not attain the fast convergence rates of Newton or quasi-Newton methods, but they have the advantage of not requiring storage of matrices. An extensive discussion of nonlinear conjugate gradient methods is given in Chapter 5.

All of the search directions discussed so far can be used directly in a line search framework. They give rise to the steepest descent, Newton, quasi-Newton, and conjugate gradient line search methods. All except conjugate gradients have an analogue in the trust-region framework, as we now discuss.

MODELS FOR TRUST-REGION METHODS

If we set $B_k = 0$ in (2.12) and define the trust region using the Euclidean norm, the trust-region subproblem (2.11) becomes

$$\min_p f_k + p^T \nabla f_k \quad \text{subject to } \|p\|_2 \leq \Delta_k.$$

We can write the solution to this problem in closed form as

$$p_k = -\frac{\Delta_k \nabla f_k}{\|\nabla f_k\|}.$$

This is simply a steepest descent step in which the step length is determined by the trust-region radius; the trust-region and line search approaches are essentially the same in this case.

A more interesting trust-region algorithm is obtained by choosing B_k to be the exact Hessian $\nabla^2 f_k$ in the quadratic model (2.12). Because of the trust-region restriction $\|p\|_2 \leq \Delta_k$, the subproblem (2.11) is guaranteed to have a solution even when $\nabla^2 f_k$ is not positive definite p_k , as we see in Figure 2.4. The trust-region Newton method has proved to be highly effective in practice, as we discuss in Chapter 7.

If the matrix B_k in the quadratic model function m_k of (2.12) is defined by means of a quasi-Newton approximation, we obtain a trust-region quasi-Newton method.

SCALING

The performance of an algorithm may depend crucially on how the problem is formulated. One important issue in problem formulation is *scaling*. In unconstrained optimization, a problem is said to be *poorly scaled* if changes to x in a certain direction produce much larger variations in the value of f than do changes to x in another direction. A simple example is provided by the function $f(x) = 10^9 x_1^2 + x_2^2$, which is very sensitive to small changes in x_1 but not so sensitive to perturbations in x_2 .

Poorly scaled functions arise, for example, in simulations of physical and chemical systems where different processes are taking place at very different rates. To be more specific, consider a chemical system in which four reactions occur. Associated with each reaction is a *rate constant* that describes the speed at which the reaction takes place. The optimization problem is to find values for these rate constants by observing the concentrations of each chemical in the system at different times. The four constants differ greatly in magnitude, since the reactions take place at vastly different speeds. Suppose we have the following rough estimates for the final values of the constants, each correct to within, say, an order of magnitude:

$$x_1 \approx 10^{-10}, \quad x_2 \approx x_3 \approx 1, \quad x_4 \approx 10^5.$$

Before solving this problem we could introduce a new variable z defined by

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10^{-10} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 10^5 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix},$$

and then define and solve the optimization problem in terms of the new variable z . The

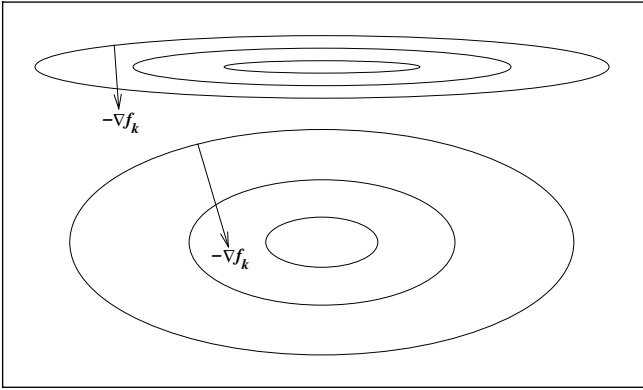


Figure 2.7 Poorly scaled and well scaled problems, and performance of the steepest descent direction.

optimal values of z will be within about an order of magnitude of 1, making the solution more balanced. This kind of scaling of the variables is known as *diagonal scaling*.

Scaling is performed (sometimes unintentionally) when the units used to represent variables are changed. During the modeling process, we may decide to change the units of some variables, say from meters to millimeters. If we do, the range of those variables and their size relative to the other variables will both change.

Some optimization algorithms, such as steepest descent, are sensitive to poor scaling, while others, such as Newton's method, are unaffected by it. Figure 2.7 shows the contours of two convex nearly quadratic functions, the first of which is poorly scaled, while the second is well scaled. For the poorly scaled problem, the one with highly elongated contours, the steepest descent direction does not yield much reduction in the function, while for the well-scaled problem it performs much better. In both cases, Newton's method will produce a much better step, since the second-order quadratic model (m_k in (2.14)) happens to be a good approximation of f .

Algorithms that are not sensitive to scaling are preferable, because they can handle poor problem formulations in a more robust fashion. In designing complete algorithms, we try to incorporate *scale invariance* into all aspects of the algorithm, including the line search or trust-region strategies and convergence tests. Generally speaking, it is easier to preserve scale invariance for line search algorithms than for trust-region algorithms.




EXERCISES





2.1 Compute the gradient $\nabla f(x)$ and Hessian $\nabla^2 f(x)$ of the Rosenbrock function


$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \quad (2.22)$$

Show that $x^* = (1, 1)^T$ is the only local minimizer of this function, and that the Hessian matrix at that point is positive definite.

 **2.2** Show that the function $f(x) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$ has only one stationary point, and that it is neither a maximum or minimum, but a saddle point. Sketch the contour lines of f .

 **2.3** Let a be a given n -vector, and A be a given $n \times n$ symmetric matrix. Compute the gradient and Hessian of $f_1(x) = a^T x$ and $f_2(x) = x^T A x$.

 **2.4** Write the second-order Taylor expansion (2.6) for the function $\cos(1/x)$ around a nonzero point x , and the third-order Taylor expansion of $\cos(x)$ around any point x . Evaluate the second expansion for the specific case of $x = 1$.


 **2.5** Consider the function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ defined by $f(x) = \|x\|^2$. Show that the sequence of iterates $\{x_k\}$ defined by


$$x_k = \left(1 + \frac{1}{2^k}\right) \begin{bmatrix} \cos k \\ \sin k \end{bmatrix}$$

satisfies $f(x_{k+1}) < f(x_k)$ for $k = 0, 1, 2, \dots$. Show that every point on the unit circle $\{x \mid \|x\|^2 = 1\}$ is a limit point for $\{x_k\}$. Hint: Every value $\theta \in [0, 2\pi]$ is a limit point of the subsequence $\{\xi_k\}$ defined by

$$\xi_k = k(\bmod 2\pi) = k - 2\pi \left\lfloor \frac{k}{2\pi} \right\rfloor,$$


where the operator $\lfloor \cdot \rfloor$ denotes rounding down to the next integer.


 **2.6** Prove that all isolated local minimizers are strict. (Hint: Take an isolated local minimizer x^* and a neighborhood \mathcal{N} . Show that for any $x \in \mathcal{N}$, $x \neq x^*$ we must have $f(x) > f(x^*)$.)


 **2.7** Suppose that $f(x) = x^T Q x$, where Q is an $n \times n$ symmetric positive semidefinite matrix. Show using the definition (1.4) that $f(x)$ is convex on the domain \mathbb{R}^n . Hint: It may be convenient to prove the following equivalent inequality:

$$f(y + \alpha(x - y)) - \alpha f(x) - (1 - \alpha)f(y) \leq 0,$$

for all $\alpha \in [0, 1]$ and all $x, y \in \mathbb{R}^n$.


 **2.8** Suppose that f is a convex function. Show that the set of global minimizers of f is a convex set.


 **2.9** Consider the function $f(x_1, x_2) = (x_1 + x_2^2)^2$. At the point $x^T = (1, 0)$ we consider the search direction $p^T = (-1, 1)$. Show that p is a descent direction and find all minimizers of the problem (2.10).


 **2.10** Suppose that $\tilde{f}(z) = f(x)$, where $x = Sz + s$ for some $S \in \mathbb{R}^{n \times n}$ and $s \in \mathbb{R}^n$. Show that


$$\nabla \tilde{f}(z) = S^T \nabla f(x), \quad \nabla^2 \tilde{f}(z) = S^T \nabla^2 f(x) S.$$


(Hint: Use the chain rule to express $d\tilde{f}/dz_j$ in terms of df/dx_i and dx_i/dz_j for all $i, j = 1, 2, \dots, n$.)


 **2.11** Show that the symmetric rank-one update (2.18) and the BFGS update (2.19) are scale-invariant if the initial Hessian approximations B_0 are chosen appropriately. That is, using the notation of the previous exercise, show that if these methods are applied to $f(x)$ starting from $x_0 = Sz_0 + s$ with initial Hessian B_0 , and to $\tilde{f}(z)$ starting from z_0 with initial Hessian $S^T B_0 S$, then all iterates are related by $x_k = Sz_k + s$. (Assume for simplicity that the methods take unit step lengths.)

 **2.12** Suppose that a function f of two variables is poorly scaled at the solution x^* . Write two Taylor expansions of f around x^* —one along each coordinate direction—and use them to show that the Hessian $\nabla^2 f(x^*)$ is ill-conditioned.

 **2.13** (For this and the following three questions, refer to the material on “Rates of Convergence” in Section A.2 of the Appendix.) Show that the sequence $x_k = 1/k$ is not Q-linearly convergent, though it does converge to zero. (This is called *sublinear convergence*.)

 **2.14** Show that the sequence $x_k = 1 + (0.5)^{2^k}$ is Q-quadratically convergent to 1.

 **2.15** Does the sequence $x_k = 1/k!$ converge Q-superlinearly? Q-quadratically?

 **2.16** Consider the sequence $\{x_k\}$ defined by

$$x_k = \begin{cases} \left(\frac{1}{4}\right)^{2^k}, & k \text{ even,} \\ (x_{k-1})/k, & k \text{ odd.} \end{cases}$$

Is this sequence Q-superlinearly convergent? Q-quadratically convergent? R-quadratically convergent?

CHAPTER 3

Line Search Methods

Each iteration of a line search method computes a search direction p_k and then decides how far to move along that direction. The iteration is given by

$$x_{k+1} = x_k + \alpha_k p_k, \tag{3.1}$$

where the positive scalar α_k is called the *step length*. The success of a line search method depends on effective choices of both the direction p_k and the step length α_k .

Most line search algorithms require p_k to be a *descent direction*—one for which $p_k^T \nabla f_k < 0$ —because this property guarantees that the function f can be reduced along

this direction, as discussed in the previous chapter. Moreover, the search direction often has the form

$$p_k = -B_k^{-1} \nabla f_k, \quad (3.2)$$

where B_k is a symmetric and nonsingular matrix. In the steepest descent method, B_k is simply the identity matrix I , while in Newton's method, B_k is the exact Hessian $\nabla^2 f(x_k)$. In quasi-Newton methods, B_k is an approximation to the Hessian that is updated at every iteration by means of a low-rank formula. When p_k is defined by (3.2) and B_k is positive definite, we have

$$p_k^T \nabla f_k = -\nabla f_k^T B_k^{-1} \nabla f_k < 0,$$

and therefore p_k is a descent direction.

In this chapter, we discuss how to choose α_k and p_k to promote convergence from remote starting points. We also study the rate of convergence of steepest descent, quasi-Newton, and Newton methods. Since the pure Newton iteration is not guaranteed to produce descent directions when the current iterate is not close to a solution, we discuss modifications in Section 3.4 that allow it to start from any initial point.

We now give careful consideration to the choice of the step-length parameter α_k .

3.1 STEP LENGTH

In computing the step length α_k , we face a tradeoff. We would like to choose α_k to give a substantial reduction of f , but at the same time we do not want to spend too much time making the choice. The ideal choice would be the global minimizer of the univariate function $\phi(\cdot)$ defined by

$$\phi(\alpha) = f(x_k + \alpha p_k), \quad \alpha > 0, \quad (3.3)$$

but in general, it is too expensive to identify this value (see Figure 3.1). To find even a local minimizer of ϕ to moderate precision generally requires too many evaluations of the objective function f and possibly the gradient ∇f . More practical strategies perform an *inexact* line search to identify a step length that achieves adequate reductions in f at minimal cost.

Typical line search algorithms try out a sequence of candidate values for α , stopping to accept one of these values when certain conditions are satisfied. The line search is done in two stages: A bracketing phase finds an interval containing desirable step lengths, and a bisection or interpolation phase computes a good step length within this interval. Sophisticated line search algorithms can be quite complicated, so we defer a full description until Section 3.5.

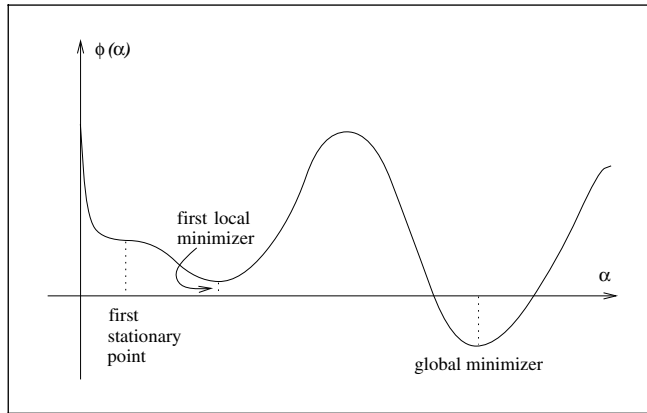


Figure 3.1 The ideal step length is the global minimizer.

We now discuss various termination conditions for line search algorithms and show that effective step lengths need not lie near minimizers of the univariate function $\phi(\alpha)$ defined in (3.3).

A simple condition we could impose on α_k is to require a reduction in f , that is, $f(x_k + \alpha_k p_k) < f(x_k)$. That this requirement is not enough to produce convergence to x^* is illustrated in Figure 3.2, for which the minimum function value is $f^* = -1$, but a sequence of iterates $\{x_k\}$ for which $f(x_k) = 5/k$, $k = 0, 1, \dots$ yields a decrease at each iteration but has a limiting function value of zero. The insufficient reduction in f at each step causes it to fail to converge to the minimizer of this convex function. To avoid this behavior we need to enforce a *sufficient decrease* condition, a concept we discuss next.

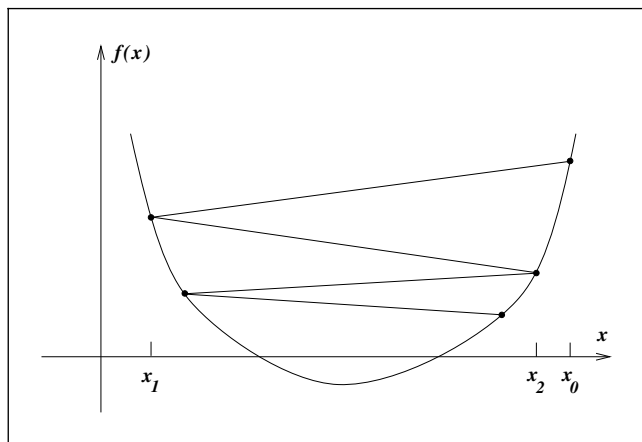


Figure 3.2 Insufficient reduction in f .

THE WOLFE CONDITIONS

A popular inexact line search condition stipulates that α_k should first of all give *sufficient decrease* in the objective function f , as measured by the following inequality:

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k, \quad (3.4)$$

for some constant $c_1 \in (0, 1)$. In other words, the reduction in f should be proportional to both the step length α_k and the directional derivative $\nabla f_k^T p_k$. Inequality (3.4) is sometimes called the *Armijo condition*.

The sufficient decrease condition is illustrated in Figure 3.3. The right-hand-side of (3.4), which is a linear function, can be denoted by $l(\alpha)$. The function $l(\cdot)$ has negative slope $c_1 \nabla f_k^T p_k$, but because $c_1 \in (0, 1)$, it lies above the graph of ϕ for small positive values of α . The sufficient decrease condition states that α is acceptable only if $\phi(\alpha) \leq l(\alpha)$. The intervals on which this condition is satisfied are shown in Figure 3.3. In practice, c_1 is chosen to be quite small, say $c_1 = 10^{-4}$.

The sufficient decrease condition is not enough by itself to ensure that the algorithm makes reasonable progress because, as we see from Figure 3.3, it is satisfied for all sufficiently small values of α . To rule out unacceptably short steps we introduce a second requirement, called the *curvature condition*, which requires α_k to satisfy

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k, \quad (3.5)$$

for some constant $c_2 \in (c_1, 1)$, where c_1 is the constant from (3.4). Note that the left-hand-side is simply the derivative $\phi'(\alpha_k)$, so the curvature condition ensures that the slope of ϕ at α_k is greater than c_2 times the initial slope $\phi'(0)$. This makes sense because if the slope $\phi'(\alpha)$

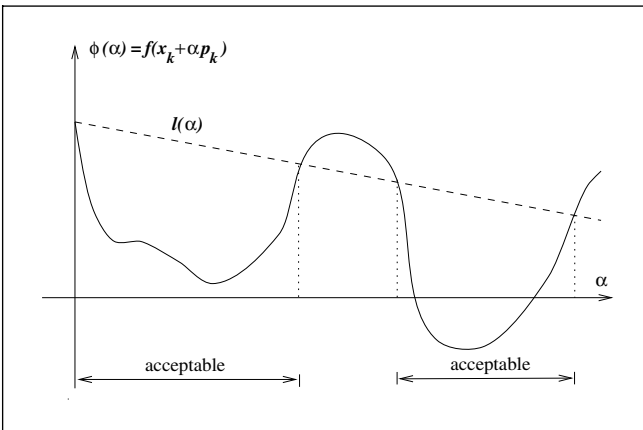


Figure 3.3 Sufficient decrease condition.

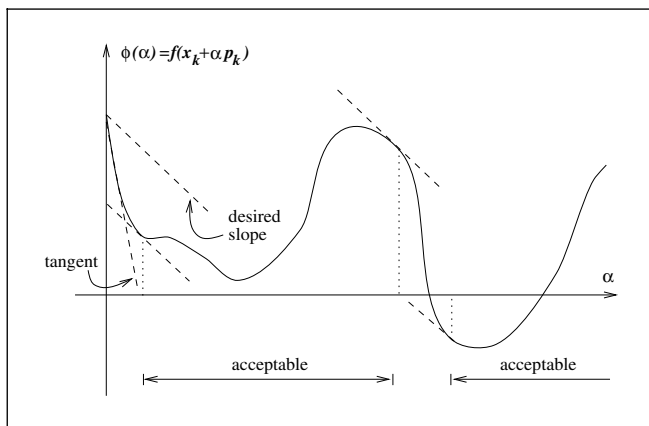


Figure 3.4 The curvature condition.

is strongly negative, we have an indication that we can reduce f significantly by moving further along the chosen direction.

On the other hand, if $\phi'(\alpha_k)$ is only slightly negative or even positive, it is a sign that we cannot expect much more decrease in f in this direction, so it makes sense to terminate the line search. The curvature condition is illustrated in Figure 3.4. Typical values of c_2 are 0.9 when the search direction p_k is chosen by a Newton or quasi-Newton method, and 0.1 when p_k is obtained from a nonlinear conjugate gradient method.

The sufficient decrease and curvature conditions are known collectively as the *Wolfe conditions*. We illustrate them in Figure 3.5 and restate them here for future reference:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, \quad (3.6a)$$

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k, \quad (3.6b)$$

with $0 < c_1 < c_2 < 1$.

A step length may satisfy the Wolfe conditions without being particularly close to a minimizer of ϕ , as we show in Figure 3.5. We can, however, modify the curvature condition to force α_k to lie in at least a broad neighborhood of a local minimizer or stationary point of ϕ . The *strong Wolfe conditions* require α_k to satisfy

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, \quad (3.7a)$$

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq c_2 |\nabla f_k^T p_k|, \quad (3.7b)$$

with $0 < c_1 < c_2 < 1$. The only difference with the Wolfe conditions is that we no longer allow the derivative $\phi'(\alpha_k)$ to be too positive. Hence, we exclude points that are far from stationary points of ϕ .

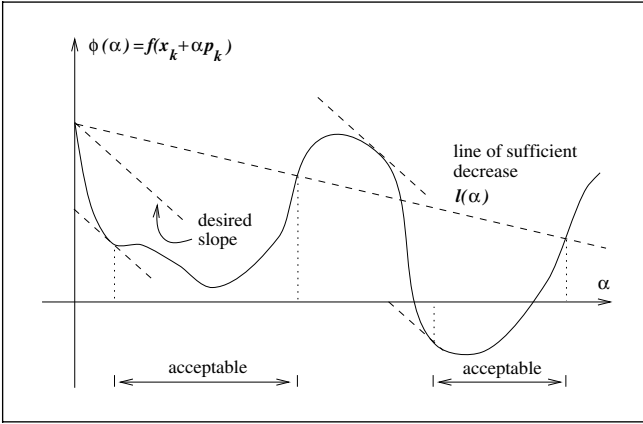


Figure 3.5 Step lengths satisfying the Wolfe conditions.

It is not difficult to prove that there exist step lengths that satisfy the Wolfe conditions for every function f that is smooth and bounded below.

Lemma 3.1.

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable. Let p_k be a descent direction at x_k , and assume that f is bounded below along the ray $\{x_k + \alpha p_k \mid \alpha > 0\}$. Then if $0 < c_1 < c_2 < 1$, there exist intervals of step lengths satisfying the Wolfe conditions (3.6) and the strong Wolfe conditions (3.7).

PROOF. Note that $\phi(\alpha) = f(x_k + \alpha p_k)$ is bounded below for all $\alpha > 0$. Since $0 < c_1 < 1$, the line $l(\alpha) = f(x_k) + \alpha c_1 \nabla f_k^T p_k$ is unbounded below and must therefore intersect the graph of ϕ at least once. Let $\alpha' > 0$ be the smallest intersecting value of α , that is,

$$f(x_k + \alpha' p_k) = f(x_k) + \alpha' c_1 \nabla f_k^T p_k. \quad (3.8)$$

The sufficient decrease condition (3.6a) clearly holds for all step lengths less than α' .

By the mean value theorem (see (A.55)), there exists $\alpha'' \in (0, \alpha')$ such that

$$f(x_k + \alpha' p_k) - f(x_k) = \alpha' \nabla f(x_k + \alpha'' p_k)^T p_k. \quad (3.9)$$

By combining (3.8) and (3.9), we obtain

$$\nabla f(x_k + \alpha'' p_k)^T p_k = c_1 \nabla f_k^T p_k > c_2 \nabla f_k^T p_k, \quad (3.10)$$

since $c_1 < c_2$ and $\nabla f_k^T p_k < 0$. Therefore, α'' satisfies the Wolfe conditions (3.6), and the inequalities hold strictly in both (3.6a) and (3.6b). Hence, by our smoothness assumption on f , there is an interval around α'' for which the Wolfe conditions hold. Moreover, since

the term in the left-hand side of (3.10) is negative, the strong Wolfe conditions (3.7) hold in the same interval. \square

The Wolfe conditions are scale-invariant in a broad sense: Multiplying the objective function by a constant or making an affine change of variables does not alter them. They can be used in most line search methods, and are particularly important in the implementation of quasi-Newton methods, as we see in Chapter 6.

THE GOLDSTEIN CONDITIONS

Like the Wolfe conditions, the *Goldstein conditions* ensure that the step length α achieves sufficient decrease but is not too short. The Goldstein conditions can also be stated as a pair of inequalities, in the following way:

$$f(x_k) + (1 - c)\alpha_k \nabla f_k^T p_k \leq f(x_k + \alpha_k p_k) \leq f(x_k) + c\alpha_k \nabla f_k^T p_k, \quad (3.11)$$

with $0 < c < 1/2$. The second inequality is the sufficient decrease condition (3.4), whereas the first inequality is introduced to control the step length from below; see Figure 3.6

A disadvantage of the Goldstein conditions vis-à-vis the Wolfe conditions is that the first inequality in (3.11) may exclude all minimizers of ϕ . However, the Goldstein and Wolfe conditions have much in common, and their convergence theories are quite similar. The Goldstein conditions are often used in Newton-type methods but are not well suited for quasi-Newton methods that maintain a positive definite Hessian approximation.

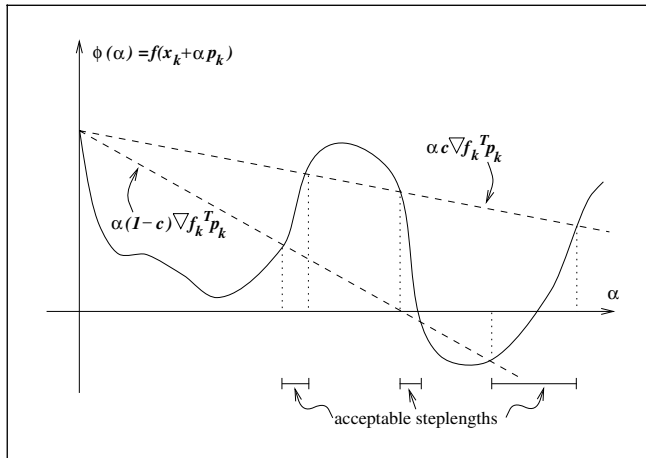


Figure 3.6 The Goldstein conditions.

SUFFICIENT DECREASE AND BACKTRACKING

We have mentioned that the sufficient decrease condition (3.6a) alone is not sufficient to ensure that the algorithm makes reasonable progress along the given search direction. However, if the line search algorithm chooses its candidate step lengths appropriately, by using a so-called *backtracking* approach, we can dispense with the extra condition (3.6b) and use just the sufficient decrease condition to terminate the line search procedure. In its most basic form, backtracking proceeds as follows.

Algorithm 3.1 (Backtracking Line Search).

Choose $\bar{\alpha} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$; Set $\alpha \leftarrow \bar{\alpha}$;
repeat until $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
 $\alpha \leftarrow \rho\alpha$;
end (repeat)
 Terminate with $\alpha_k = \alpha$.

In this procedure, the initial step length $\bar{\alpha}$ is chosen to be 1 in Newton and quasi-Newton methods, but can have different values in other algorithms such as steepest descent or conjugate gradient. An acceptable step length will be found after a finite number of trials, because α_k will eventually become small enough that the sufficient decrease condition holds (see Figure 3.3). In practice, the contraction factor ρ is often allowed to vary at each iteration of the line search. For example, it can be chosen by safeguarded interpolation, as we describe later. We need ensure only that at each iteration we have $\rho \in [\rho_{lo}, \rho_{hi}]$, for some fixed constants $0 < \rho_{lo} < \rho_{hi} < 1$.

The backtracking approach ensures either that the selected step length α_k is some fixed value (the initial choice $\bar{\alpha}$), or else that it is short enough to satisfy the sufficient decrease condition but not *too* short. The latter claim holds because the accepted value α_k is within a factor ρ of the previous trial value, α_k/ρ , which was rejected for violating the sufficient decrease condition, that is, for being too long.

This simple and popular strategy for terminating a line search is well suited for Newton methods but is less appropriate for quasi-Newton and conjugate gradient methods.

3.2 CONVERGENCE OF LINE SEARCH METHODS

To obtain global convergence, we must not only have well chosen step lengths but also well chosen search directions p_k . We discuss requirements on the search direction in this section, focusing on one key property: the angle θ_k between p_k and the steepest descent direction $-\nabla f_k$, defined by

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}. \quad (3.12)$$

The following theorem, due to Zoutendijk, has far-reaching consequences. It quantifies the effect of properly chosen step lengths α_k , and shows, for example, that the steepest descent method is globally convergent. For other algorithms, it describes how far p_k can deviate from the steepest descent direction and still produce a globally convergent iteration. Various line search termination conditions can be used to establish this result, but for concreteness we will consider only the Wolfe conditions (3.6). Though Zoutendijk's result appears at first to be technical and obscure, its power will soon become evident.

Theorem 3.2.

Consider any iteration of the form (3.1), where p_k is a descent direction and α_k satisfies the Wolfe conditions (3.6). Suppose that f is bounded below in \mathbb{R}^n and that f is continuously differentiable in an open set \mathcal{N} containing the level set $\mathcal{L} \stackrel{\text{def}}{=} \{x : f(x) \leq f(x_0)\}$, where x_0 is the starting point of the iteration. Assume also that the gradient ∇f is Lipschitz continuous on \mathcal{N} , that is, there exists a constant $L > 0$ such that

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathcal{N}. \quad (3.13)$$

Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty. \quad (3.14)$$

PROOF. From (3.6b) and (3.1) we have that

$$(\nabla f_{k+1} - \nabla f_k)^T p_k \geq (c_2 - 1) \nabla f_k^T p_k,$$

while the Lipschitz condition (3.13) implies that

$$(\nabla f_{k+1} - \nabla f_k)^T p_k \leq \alpha_k L \|p_k\|^2.$$

By combining these two relations, we obtain

$$\alpha_k \geq \frac{c_2 - 1}{L} \frac{\nabla f_k^T p_k}{\|p_k\|^2}.$$

By substituting this inequality into the first Wolfe condition (3.6a), we obtain

$$f_{k+1} \leq f_k - c_1 \frac{1 - c_2}{L} \frac{(\nabla f_k^T p_k)^2}{\|p_k\|^2}.$$

From the definition (3.12), we can write this relation as

$$f_{k+1} \leq f_k - c \cos^2 \theta_k \|\nabla f_k\|^2,$$

where $c = c_1(1 - c_2)/L$. By summing this expression over all indices less than or equal to k , we obtain

$$f_{k+1} \leq f_0 - c \sum_{j=0}^k \cos^2 \theta_j \|\nabla f_j\|^2. \quad (3.15)$$

Since f is bounded below, we have that $f_0 - f_{k+1}$ is less than some positive constant, for all k . Hence, by taking limits in (3.15), we obtain

$$\sum_{k=0}^{\infty} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty,$$

which concludes the proof. \square

Similar results to this theorem hold when the Goldstein conditions (3.11) or strong Wolfe conditions (3.7) are used in place of the Wolfe conditions. For all these strategies, the step length selection implies inequality (3.14), which we call the *Zoutendijk condition*.

Note that the assumptions of Theorem 3.2 are not too restrictive. If the function f were not bounded below, the optimization problem would not be well defined. The smoothness assumption—Lipschitz continuity of the gradient—is implied by many of the smoothness conditions that are used in local convergence theorems (see Chapters 6 and 7) and are often satisfied in practice.

The Zoutendijk condition (3.14) implies that

$$\cos^2 \theta_k \|\nabla f_k\|^2 \rightarrow 0. \quad (3.16)$$

This limit can be used in turn to derive global convergence results for line search algorithms.

If our method for choosing the search direction p_k in the iteration (3.1) ensures that the angle θ_k defined by (3.12) is bounded away from 90° , there is a positive constant δ such that

$$\cos \theta_k \geq \delta > 0, \quad \text{for all } k. \quad (3.17)$$

It follows immediately from (3.16) that

$$\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0. \quad (3.18)$$

In other words, we can be sure that the gradient norms $\|\nabla f_k\|$ converge to zero, provided that the search directions are never too close to orthogonality with the gradient. In particular, the method of steepest descent (for which the search direction p_k is parallel to the negative

gradient) produces a gradient sequence that converges to zero, provided that it uses a line search satisfying the Wolfe or Goldstein conditions.

We use the term *globally convergent* to refer to algorithms for which the property (3.18) is satisfied, but note that this term is sometimes used in other contexts to mean different things. For line search methods of the general form (3.1), the limit (3.18) is the strongest global convergence result that can be obtained: We cannot guarantee that the method converges to a minimizer, but only that it is attracted by stationary points. Only by making additional requirements on the search direction p_k —by introducing negative curvature information from the Hessian $\nabla^2 f(x_k)$, for example—can we strengthen these results to include convergence to a local minimum. See the Notes and References at the end of this chapter for further discussion of this point.

Consider now the Newton-like method (3.1), (3.2) and assume that the matrices B_k are positive definite with a uniformly bounded condition number. That is, there is a constant M such that

$$\|B_k\| \|B_k^{-1}\| \leq M, \quad \text{for all } k.$$

It is easy to show from the definition (3.12) that

$$\cos \theta_k \geq 1/M \tag{3.19}$$

(see Exercise 3.5). By combining this bound with (3.16) we find that

$$\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0. \tag{3.20}$$

Therefore, we have shown that Newton and quasi-Newton methods are globally convergent if the matrices B_k have a bounded condition number and are positive definite (which is needed to ensure that p_k is a descent direction), and if the step lengths satisfy the Wolfe conditions.

For some algorithms, such as conjugate gradient methods, we will be able to prove the limit (3.18), but only the weaker result

$$\liminf_{k \rightarrow \infty} \|\nabla f_k\| = 0. \tag{3.21}$$

In other words, just a subsequence of the gradient norms $\|\nabla f_{k_j}\|$ converges to zero, rather than the whole sequence (see Appendix A). This result, too, can be proved by using Zoutendijk's condition (3.14), but instead of a constructive proof, we outline a proof by contradiction. Suppose that (3.21) does not hold, so that the gradients remain bounded away from zero, that is, there exists $\gamma > 0$ such that

$$\|\nabla f_k\| \geq \gamma, \quad \text{for all } k \text{ sufficiently large.} \tag{3.22}$$

Then from (3.16) we conclude that

$$\cos \theta_k \rightarrow 0, \quad (3.23)$$

that is, the entire sequence $\{\cos \theta_k\}$ converges to 0. To establish (3.21), therefore, it is enough to show that a subsequence $\{\cos \theta_{k_j}\}$ is bounded away from zero. We will use this strategy in Chapter 5 to study the convergence of nonlinear conjugate gradient methods.

By applying this proof technique, we can prove global convergence in the sense of (3.20) or (3.21) for a general class of algorithms. Consider *any* algorithm for which (i) every iteration produces a decrease in the objective function, and (ii) every m th iteration is a steepest descent step, with step length chosen to satisfy the Wolfe or Goldstein conditions. Then, since $\cos \theta_k = 1$ for the steepest descent steps, the result (3.21) holds. Of course, we would design the algorithm so that it does something “better” than steepest descent at the other $m - 1$ iterates. The occasional steepest descent steps may not make much progress, but they at least guarantee overall global convergence.

Note that throughout this section we have used only the fact that Zoutendijk’s condition implies the limit (3.16). In later chapters we will make use of the bounded sum condition (3.14), which forces the sequence $\{\cos^2 \theta_k \|\nabla f_k\|^2\}$ to converge to zero at a sufficiently rapid rate.

3.3 RATE OF CONVERGENCE

It would seem that designing optimization algorithms with good convergence properties is easy, since all we need to ensure is that the search direction p_k does not tend to become orthogonal to the gradient ∇f_k , or that steepest descent steps are taken regularly. We could simply compute $\cos \theta_k$ at every iteration and turn p_k toward the steepest descent direction if $\cos \theta_k$ is smaller than some preselected constant $\delta > 0$. Angle tests of this type ensure global convergence, but they are undesirable for two reasons. First, they may impede a fast rate of convergence, because for problems with an ill-conditioned Hessian, it may be necessary to produce search directions that are almost orthogonal to the gradient, and an inappropriate choice of the parameter δ may cause such steps to be rejected. Second, angle tests destroy the invariance properties of quasi-Newton methods.

Algorithmic strategies that achieve rapid convergence can sometimes conflict with the requirements of global convergence, and vice versa. For example, the steepest descent method is the quintessential globally convergent algorithm, but it is quite slow in practice, as we shall see below. On the other hand, the pure Newton iteration converges rapidly when started close enough to a solution, but its steps may not even be descent directions away from the solution. The challenge is to design algorithms that incorporate both properties: good global convergence guarantees and a rapid rate of convergence.

We begin our study of convergence rates of line search methods by considering the most basic approach of all: the steepest descent method.

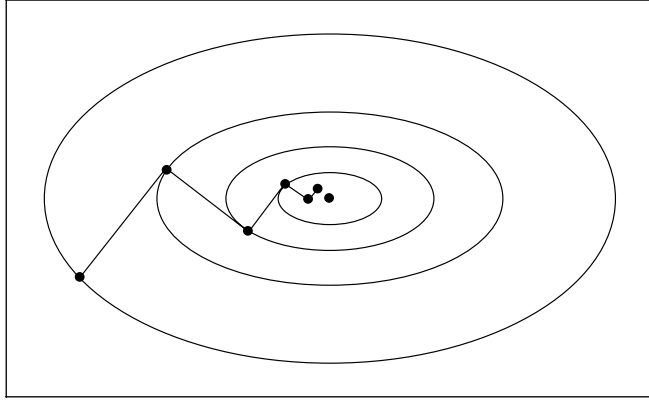


Figure 3.7 Steepest descent steps.

CONVERGENCE RATE OF STEEPEST DESCENT

We can learn much about the steepest descent method by considering the ideal case, in which the objective function is quadratic and the line searches are exact. Let us suppose that

$$f(x) = \frac{1}{2}x^T Qx - b^T x, \quad (3.24)$$

where Q is symmetric and positive definite. The gradient is given by $\nabla f(x) = Qx - b$ and the minimizer x^* is the unique solution of the linear system $Qx = b$.

It is easy to compute the step length α_k that minimizes $f(x_k - \alpha \nabla f_k)$. By differentiating the function

$$f(x_k - \alpha \nabla f_k) = \frac{1}{2}(x_k - \alpha \nabla f_k)^T Q(x_k - \alpha \nabla f_k) - b^T (x_k - \alpha \nabla f_k)$$

with respect to α , and setting the derivative to zero, we obtain

$$\alpha_k = \frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T Q \nabla f_k}. \quad (3.25)$$

If we use this exact minimizer α_k , the steepest descent iteration for (3.24) is given by

$$x_{k+1} = x_k - \left(\frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T Q \nabla f_k} \right) \nabla f_k. \quad (3.26)$$

Since $\nabla f_k = Qx_k - b$, this equation yields a closed-form expression for x_{k+1} in terms of x_k . In Figure 3.7 we plot a typical sequence of iterates generated by the steepest descent method on a two-dimensional quadratic objective function. The contours of f are ellipsoids whose

axes lie along the orthogonal eigenvectors of Q . Note that the iterates zigzag toward the solution.

To quantify the rate of convergence we introduce the weighted norm $\|x\|_Q^2 = x^T Q x$. By using the relation $Qx^* = b$, we can show that

$$\frac{1}{2} \|x - x^*\|_Q^2 = f(x) - f(x^*), \quad (3.27)$$

so this norm measures the difference between the current objective value and the optimal value. By using the equality (3.26) and noting that $\nabla f_k = Q(x_k - x^*)$, we can derive the equality

$$\|x_{k+1} - x^*\|_Q^2 = \left\{ 1 - \frac{(\nabla f_k^T \nabla f_k)^2}{(\nabla f_k^T Q \nabla f_k)(\nabla f_k^T Q^{-1} \nabla f_k)} \right\} \|x_k - x^*\|_Q^2 \quad (3.28)$$

(see Exercise 3.7). This expression describes the exact decrease in f at each iteration, but since the term inside the brackets is difficult to interpret, it is more useful to bound it in terms of the condition number of the problem.

Theorem 3.3.

When the steepest descent method with exact line searches (3.26) is applied to the strongly convex quadratic function (3.24), the error norm (3.27) satisfies

$$\|x_{k+1} - x^*\|_Q^2 \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1} \right)^2 \|x_k - x^*\|_Q^2, \quad (3.29)$$

where $0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ are the eigenvalues of Q .

The proof of this result is given by Luenberger [195]. The inequalities (3.29) and (3.27) show that the function values f_k converge to the minimum f_* at a linear rate. As a special case of this result, we see that convergence is achieved in one iteration if all the eigenvalues are equal. In this case, Q is a multiple of the identity matrix, so the contours in Figure 3.7 are circles and the steepest descent direction always points at the solution. In general, as the condition number $\kappa(Q) = \lambda_n/\lambda_1$ increases, the contours of the quadratic become more elongated, the zigzagging in Figure 3.7 becomes more pronounced, and (3.29) implies that the convergence degrades. Even though (3.29) is a worst-case bound, it gives an accurate indication of the behavior of the algorithm when $n > 2$.

The rate-of-convergence behavior of the steepest descent method is essentially the same on general nonlinear objective functions. In the following result we assume that the step length is the global minimizer along the search direction.

Theorem 3.4.

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable, and that the iterates generated by the steepest-descent method with exact line searches converge to a point x^ at*

which the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Let r be any scalar satisfying

$$r \in \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}, 1 \right),$$

where $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f(x^*)$. Then for all k sufficiently large, we have

$$f(x_{k+1}) - f(x^*) \leq r^2 [f(x_k) - f(x^*)].$$

In general, we cannot expect the rate of convergence to improve if an inexact line search is used. Therefore, Theorem 3.4 shows that the steepest descent method can have an unacceptably slow rate of convergence, even when the Hessian is reasonably well conditioned. For example, if $\kappa(Q) = 800$, $f(x_1) = 1$, and $f(x^*) = 0$, Theorem 3.4 suggests that the function value will still be about 0.08 after one thousand iterations of the steepest descent method with exact line search.

NEWTON'S METHOD

We now consider the Newton iteration, for which the search is given by

$$p_k^N = -\nabla^2 f_k^{-1} \nabla f_k. \quad (3.30)$$

Since the Hessian matrix $\nabla^2 f_k$ may not always be positive definite, p_k^N may not always be a descent direction, and many of the ideas discussed so far in this chapter no longer apply. In Section 3.4 and Chapter 4 we will describe two approaches for obtaining a globally convergent iteration based on the Newton step: a line search approach, in which the Hessian $\nabla^2 f_k$ is modified, if necessary, to make it positive definite and thereby yield descent, and a trust region approach, in which $\nabla^2 f_k$ is used to form a quadratic model that is minimized in a ball around the current iterate x_k .

Here we discuss just the local rate-of-convergence properties of Newton's method. We know that for all x in the vicinity of a solution point x^* such that $\nabla^2 f(x^*)$ is positive definite, the Hessian $\nabla^2 f(x)$ will also be positive definite. Newton's method will be well defined in this region and will converge quadratically, provided that the step lengths α_k are eventually always 1.

Theorem 3.5.

Suppose that f is twice differentiable and that the Hessian $\nabla^2 f(x)$ is Lipschitz continuous (see (A.42)) in a neighborhood of a solution x^* at which the sufficient conditions (Theorem 2.4) are satisfied. Consider the iteration $x_{k+1} = x_k + p_k$, where p_k is given by (3.30). Then

- (i) if the starting point x_0 is sufficiently close to x^* , the sequence of iterates converges to x^* ;
- (ii) the rate of convergence of $\{x_k\}$ is quadratic; and
- (iii) the sequence of gradient norms $\{\|\nabla f_k\|\}$ converges quadratically to zero.

PROOF. From the definition of the Newton step and the optimality condition $\nabla f_* = 0$ we have that

$$\begin{aligned} x_k + p_k^N - x^* &= x_k - x^* - \nabla^2 f_k^{-1} \nabla f_k \\ &= \nabla^2 f_k^{-1} [\nabla^2 f_k(x_k - x^*) - (\nabla f_k - \nabla f_*)]. \end{aligned} \quad (3.31)$$

Since Taylor's theorem (Theorem 2.1) tells us that

$$\nabla f_k - \nabla f_* = \int_0^1 \nabla^2 f(x_k + t(x^* - x_k))(x_k - x^*) dt,$$

we have

$$\begin{aligned} &\| \nabla^2 f(x_k)(x_k - x^*) - (\nabla f_k - \nabla f(x^*)) \| \\ &= \left\| \int_0^1 [\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))] (x_k - x^*) dt \right\| \\ &\leq \int_0^1 \| \nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k)) \| \|x_k - x^*\| dt \\ &\leq \|x_k - x^*\|^2 \int_0^1 L t dt = \frac{1}{2} L \|x_k - x^*\|^2, \end{aligned} \quad (3.32)$$

where L is the Lipschitz constant for $\nabla^2 f(x)$ for x near x^* . Since $\nabla^2 f(x^*)$ is nonsingular, there is a radius $r > 0$ such that $\|\nabla^2 f_k^{-1}\| \leq 2\|\nabla^2 f(x^*)^{-1}\|$ for all x_k with $\|x_k - x^*\| \leq r$. By substituting in (3.31) and (3.32), we obtain

$$\|x_k + p_k^N - x^*\| \leq L \|\nabla^2 f(x^*)^{-1}\| \|x_k - x^*\|^2 = \tilde{L} \|x_k - x^*\|^2, \quad (3.33)$$

where $\tilde{L} = L \|\nabla^2 f(x^*)^{-1}\|$. Choosing x_0 so that $\|x_0 - x^*\| \leq \min(r, 1/(2\tilde{L}))$, we can use this inequality inductively to deduce that the sequence converges to x^* , and the rate of convergence is quadratic.

By using the relations $x_{k+1} - x_k = p_k^N$ and $\nabla f_k + \nabla^2 f_k p_k^N = 0$, we obtain that

$$\begin{aligned} \|\nabla f(x_{k+1})\| &= \|\nabla f(x_{k+1}) - \nabla f_k - \nabla^2 f(x_k) p_k^N\| \\ &= \left\| \int_0^1 \nabla^2 f(x_k + t p_k^N)(x_{k+1} - x_k) dt - \nabla^2 f(x_k) p_k^N \right\| \\ &\leq \int_0^1 \| \nabla^2 f(x_k + t p_k^N) - \nabla^2 f(x_k) \| \|p_k^N\| dt \\ &\leq \frac{1}{2} L \|p_k^N\|^2 \\ &\leq \frac{1}{2} L \|\nabla^2 f(x_k)^{-1}\|^2 \|\nabla f_k\|^2 \\ &\leq 2L \|\nabla^2 f(x^*)^{-1}\|^2 \|\nabla f_k\|^2, \end{aligned}$$

proving that the gradient norms converge to zero quadratically. \square

As the iterates generated by Newton's method approach the solution, the Wolfe (or Goldstein) conditions will accept the step length $\alpha_k = 1$ for all large k . This observation follows from Theorem 3.6 below. Indeed, when the search direction is given by Newton's method, the limit (3.35) is satisfied—the ratio is zero for all k ! Implementations of Newton's method using these line search conditions, and in which the line search always tries the unit step length first, will set $\alpha_k = 1$ for all large k and attain a local quadratic rate of convergence.

QUASI-NEWTON METHODS

Suppose now that the search direction has the form

$$p_k = -B_k^{-1} \nabla f_k, \quad (3.34)$$

where the symmetric and positive definite matrix B_k is updated at every iteration by a quasi-Newton updating formula. We already encountered one quasi-Newton formula, the BFGS formula, in Chapter 2; others will be discussed in Chapter 6. We assume here that the step length α_k is computed by an inexact line search that satisfies the Wolfe or strong Wolfe conditions, with the same proviso mentioned above for Newton's method: The line search algorithm will always try the step length $\alpha = 1$ first, and will accept this value if it satisfies the Wolfe conditions. (We could enforce this condition by setting $\bar{\alpha} = 1$ in Algorithm 3.1, for example.) This implementation detail turns out to be crucial in obtaining a fast rate of convergence.

The following result shows that if the search direction of a quasi-Newton method approximates the Newton direction well enough, then the unit step length will satisfy the Wolfe conditions as the iterates converge to the solution. It also specifies a condition that the search direction must satisfy in order to give rise to a superlinearly convergent iteration. To bring out the full generality of this result, we state it first in terms of a general descent iteration, and then examine its consequences for quasi-Newton and Newton methods.

Theorem 3.6.

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable. Consider the iteration $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction and α_k satisfies the Wolfe conditions (3.6) with $c_1 \leq 1/2$. If the sequence $\{x_k\}$ converges to a point x^ such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, and if the search direction satisfies*

$$\lim_{k \rightarrow \infty} \frac{\|\nabla f_k + \nabla^2 f_k p_k\|}{\|p_k\|} = 0, \quad (3.35)$$

then

- (i) *the step length $\alpha_k = 1$ is admissible for all k greater than a certain index k_0 ; and*
- (ii) *if $\alpha_k = 1$ for all $k > k_0$, $\{x_k\}$ converges to x^* superlinearly.*

It is easy to see that if $c_1 > 1/2$, then the line search would exclude the minimizer of a quadratic, and unit step lengths may not be admissible.

If p_k is a quasi-Newton search direction of the form (3.34), then (3.35) is equivalent to

$$\lim_{k \rightarrow \infty} \frac{\|(B_k - \nabla^2 f(x^*))p_k\|}{\|p_k\|} = 0. \quad (3.36)$$

Hence, we have the surprising (and delightful) result that a superlinear convergence rate can be attained even if the sequence of quasi-Newton matrices B_k does not converge to $\nabla^2 f(x^*)$; it suffices that the B_k become increasingly accurate approximations to $\nabla^2 f(x^*)$ along the search directions p_k . Importantly, condition (3.36) is *both necessary and sufficient* for the superlinear convergence of quasi-Newton methods.

Theorem 3.7.

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable. Consider the iteration $x_{k+1} = x_k + p_k$ (that is, the step length α_k is uniformly 1) and that p_k is given by (3.34). Let us assume also that $\{x_k\}$ converges to a point x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then $\{x_k\}$ converges superlinearly if and only if (3.36) holds.

PROOF. We first show that (3.36) is equivalent to

$$p_k - p_k^N = o(\|p_k\|), \quad (3.37)$$

where $p_k^N = -\nabla^2 f_k^{-1} \nabla f_k$ is the Newton step. Assuming that (3.36) holds, we have that

$$\begin{aligned} p_k - p_k^N &= \nabla^2 f_k^{-1} (\nabla^2 f_k p_k + \nabla f_k) \\ &= \nabla^2 f_k^{-1} (\nabla^2 f_k - B_k) p_k \\ &= O(\|(\nabla^2 f_k - B_k) p_k\|) \\ &= o(\|p_k\|), \end{aligned}$$

where we have used the fact that $\|\nabla^2 f_k^{-1}\|$ is bounded above for x_k sufficiently close to x^* , since the limiting Hessian $\nabla^2 f(x^*)$ is positive definite. The converse follows readily if we multiply both sides of (3.37) by $\nabla^2 f_k$ and recall (3.34).

By combining (3.33) and (3.37), we obtain that

$$\|x_k + p_k - x^*\| \leq \|x_k + p_k^N - x^*\| + \|p_k - p_k^N\| = O(\|x_k - x^*\|^2) + o(\|p_k\|).$$

A simple manipulation of this inequality reveals that $\|p_k\| = O(\|x_k - x^*\|)$, so we obtain

$$\|x_k + p_k - x^*\| \leq o(\|x_k - x^*\|),$$

giving the superlinear convergence result. \square

We will see in Chapter 6 that quasi-Newton methods normally satisfy condition (3.36) and are therefore superlinearly convergent.

3.4 NEWTON'S METHOD WITH HESSIAN MODIFICATION

Away from the solution, the Hessian matrix $\nabla^2 f(x)$ may not be positive definite, so the Newton direction p_k^N defined by

$$\nabla^2 f(x_k) p_k^N = -\nabla f(x_k) \quad (3.38)$$

(see (3.30)) may not be a descent direction. We now describe an approach to overcome this difficulty when a direct linear algebra technique, such as Gaussian elimination, is used to solve the Newton equations (3.38). This approach obtains the step p_k from a linear system identical to (3.38), except that the coefficient matrix is replaced with a positive definite approximation, formed before or during the solution process. The modified Hessian is obtained by adding either a positive diagonal matrix or a full matrix to the true Hessian $\nabla^2 f(x_k)$. A general description of this method follows.

Algorithm 3.2 (Line Search Newton with Modification).

Given initial point x_0 ;

for $k = 0, 1, 2, \dots$

Factorize the matrix $B_k = \nabla^2 f(x_k) + E_k$, where $E_k = 0$ if $\nabla^2 f(x_k)$ is sufficiently positive definite; otherwise, E_k is chosen to ensure that B_k is sufficiently positive definite;

Solve $B_k p_k = -\nabla f(x_k)$;

Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or Armijo backtracking conditions;

end

Some approaches do not compute E_k explicitly, but rather introduce extra steps and tests into standard factorization procedures, modifying these procedures “on the fly” so that the computed factors are the factors of a positive definite matrix. Strategies based on modifying a Cholesky factorization and on modifying a symmetric indefinite factorization of the Hessian are described in this section.

Algorithm 3.2 is a practical Newton method that can be applied from any starting point. We can establish fairly satisfactory global convergence results for it, provided that the strategy for choosing E_k (and hence B_k) satisfies the *bounded modified factorization* property. This property is that the matrices in the sequence $\{B_k\}$ have bounded condition number whenever the sequence of Hessians $\{\nabla^2 f(x_k)\}$ is bounded; that is,

$$\kappa(B_k) = \|B_k\| \|B_k^{-1}\| \leq C, \quad \text{some } C > 0 \text{ and all } k = 0, 1, 2, \dots \quad (3.39)$$

If this property holds, global convergence of the modified line search Newton method follows from the results of Section 3.2.

Theorem 3.8.

Let f be twice continuously differentiable on an open set \mathcal{D} , and assume that the starting point x_0 of Algorithm 3.2 is such that the level set $\mathcal{L} = \{x \in \mathcal{D} : f(x) \leq f(x_0)\}$ is compact. Then if the bounded modified factorization property holds, we have that

$$\lim_{k \rightarrow \infty} \nabla f(x_k) = 0.$$

For a proof this result see [215].

We now consider the convergence rate of Algorithm 3.2. Suppose that the sequence of iterates x_k converges to a point x^* where $\nabla^2 f(x^*)$ is sufficiently positive definite in the sense that the modification strategies described in the next section return the modification $E_k = 0$ for all sufficiently large k . By Theorem 3.6, we have that $\alpha_k = 1$ for all sufficiently large k , so that Algorithm 3.2 reduces to a pure Newton method, and the rate of convergence is quadratic.

For problems in which ∇f^* is close to singular, there is no guarantee that the modification E_k will eventually vanish, and the convergence rate may be only linear. Besides requiring the modified matrix B_k to be well conditioned (so that Theorem 3.8 holds), we would like the modification to be as small as possible, so that the second-order information in the Hessian is preserved as far as possible. Naturally, we would also like the modified factorization to be computable at moderate cost.

To set the stage for the matrix factorization techniques that will be used in Algorithm 3.2, we will begin by assuming that the eigenvalue decomposition of $\nabla^2 f(x_k)$ is available. This is not realistic for large-scale problems because this decomposition is generally too expensive to compute, but it will motivate several practical modification strategies.

EIGENVALUE MODIFICATION

Consider a problem in which, at the current iterate x_k , $\nabla f(x_k) = (1, -3, 2)^T$ and $\nabla^2 f(x_k) = \text{diag}(10, 3, -1)$, which is clearly indefinite. By the spectral decomposition theorem (see Appendix A) we can define $Q = I$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$, and write

$$\nabla^2 f(x_k) = Q \Lambda Q^T = \sum_{i=1}^n \lambda_i q_i q_i^T. \quad (3.40)$$

The pure Newton step—the solution of (3.38)—is $p_k^N = (-0.1, 1, 2)^T$, which is not a descent direction, since $\nabla f(x_k)^T p_k^N > 0$. One might suggest a modified strategy in which we replace $\nabla^2 f(x_k)$ by a positive definite approximation B_k , in which all negative eigenvalues in $\nabla^2 f(x_k)$ are replaced by a small positive number δ that is somewhat larger than machine precision \mathbf{u} ; say $\delta = \sqrt{\mathbf{u}}$. For a machine precision of 10^{-16} , the resulting matrix in

our example is

$$B_k = \sum_{i=1}^2 \lambda_i q_i q_i^T + \delta q_3 q_3^T = \text{diag}(10, 3, 10^{-8}), \quad (3.41)$$

which is numerically positive definite and whose curvature along the eigenvectors q_1 and q_2 has been preserved. Note, however, that the search direction based on this modified Hessian is

$$\begin{aligned} p_k &= -B_k^{-1} \nabla f_k = - \sum_{i=1}^2 \frac{1}{\lambda_i} q_i (q_i^T \nabla f_k) - \frac{1}{\delta} q_3 (q_3^T \nabla f(x_k)) \\ &\approx -(2 \times 10^8) q_3. \end{aligned} \quad (3.42)$$

For small δ , this step is nearly parallel to q_3 (with relatively small contributions from q_1 and q_2) and quite long. Although f decreases along the direction p_k , its extreme length violates the spirit of Newton's method, which relies on a quadratic approximation of the objective function that is valid in a neighborhood of the current iterate x_k . It is therefore not clear that this search direction is effective.

Various other modification strategies are possible. We could flip the signs of the negative eigenvalues in (3.40), which amounts to setting $\delta = 1$ in our example. We could set the last term in (3.42) to zero, so that the search direction has no components along the negative curvature directions. We could adapt the choice of δ to ensure that the length of the step is not excessive, a strategy that has the flavor of trust-region methods. As this discussion shows, there is a great deal of freedom in devising modification strategies, and there is currently no agreement on which strategy is best.

Setting the issue of the choice of δ aside for the moment, let us look more closely at the process of modifying a matrix so that it becomes positive definite. The modification (3.41) to the example matrix (3.40) can be shown to be optimal in the following sense. If A is a symmetric matrix with spectral decomposition $A = Q \Lambda Q^T$, then the correction matrix ΔA of *minimum Frobenius norm* that ensures that $\lambda_{\min}(A + \Delta A) \geq \delta$ is given by

$$\Delta A = Q \text{diag}(\tau_i) Q^T, \quad \text{with} \quad \tau_i = \begin{cases} 0, & \lambda_i \geq \delta, \\ \delta - \lambda_i, & \lambda_i < \delta. \end{cases} \quad (3.43)$$

Here, $\lambda_{\min}(A)$ denotes the smallest eigenvalue of A , and the Frobenius norm of a matrix is defined as $\|A\|_F^2 = \sum_{i,j=1}^n a_{ij}^2$ (see (A.9)). Note that ΔA is not diagonal in general, and that the modified matrix is given by

$$A + \Delta A = Q(\Lambda + \text{diag}(\tau_i))Q^T.$$

By using a different norm we can obtain a *diagonal* modification. Suppose again that A is a symmetric matrix with spectral decomposition $A = Q \Lambda Q^T$. A correction matrix

ΔA with minimum Euclidean norm that satisfies $\lambda_{\min}(A + \Delta A) \geq \delta$ is given by

$$\Delta A = \tau I, \quad \text{with} \quad \tau = \max(0, \delta - \lambda_{\min}(A)). \quad (3.44)$$

The modified matrix now has the form

$$A + \tau I, \quad (3.45)$$

which happens to have the same form as the matrix occurring in (unscaled) trust-region methods (see Chapter 4). All the eigenvalues of (3.45) have thus been shifted, and all are greater than δ .

These results suggest that both diagonal and nondiagonal modifications can be considered. Even though we have not answered the question of what constitutes a good modification, various practical diagonal and nondiagonal modifications have been proposed and implemented in software. They do not make use of the spectral decomposition of the Hessian, since it is generally too expensive to compute. Instead, they use Gaussian elimination, choosing the modifications indirectly and hoping that somehow they will produce good steps. Numerical experience indicates that the strategies described next often (but not always) produce good search directions.

ADDING A MULTIPLE OF THE IDENTITY

Perhaps the simplest idea is to find a scalar $\tau > 0$ such that $\nabla^2 f(x_k) + \tau I$ is sufficiently positive definite. From the previous discussion we know that τ must satisfy (3.44), but a good estimate of the smallest eigenvalue of the Hessian is normally not available. The following algorithm describes a method that tries successively larger values of τ . (Here, a_{ii} denotes a diagonal element of A .)

Algorithm 3.3 (Cholesky with Added Multiple of the Identity).

```

Choose  $\beta > 0$ ;
if  $\min_i a_{ii} > 0$ 
    set  $\tau_0 \leftarrow 0$ ;
else
     $\tau_0 = -\min(a_{ii}) + \beta$ ;
end (if)
for  $k = 0, 1, 2, \dots$ 
    Attempt to apply the Cholesky algorithm to obtain  $LL^T = A + \tau_k I$ ;
    if the factorization is completed successfully
        stop and return  $L$ ;
    else
         $\tau_{k+1} \leftarrow \max(2\tau_k, \beta)$ ;
    end (if)
end (for)
```

The choice of β is heuristic; a typical value is $\beta = 10^{-3}$. We could choose the first nonzero shift τ_0 to be proportional to be the final value of τ used in the latest Hessian modification; see also Algorithm B.1. The strategy implemented in Algorithm 3.3 is quite simple and may be preferable to the modified factorization techniques described next, but it suffers from one drawback. Every value of τ_k requires a new factorization of $A + \tau_k I$, and the algorithm can be quite expensive if several trial values are generated. Therefore it may be advantageous to increase τ more rapidly, say by a factor of 10 instead of 2 in the last **else** clause.

MODIFIED CHOLSKY FACTORIZATION

Another approach for modifying a Hessian matrix that is not positive definite is to perform a Cholesky factorization of $\nabla^2 f(x_k)$, but to increase the diagonal elements encountered during the factorization (where necessary) to ensure that they are sufficiently positive. This *modified Cholesky* approach is designed to accomplish two goals: It guarantees that the modified Cholesky factors exist and are bounded relative to the norm of the actual Hessian, and it does not modify the Hessian if it is sufficiently positive definite.

We begin our description of this approach by briefly reviewing the Cholesky factorization. Every symmetric positive definite matrix A can be written as

$$A = LDL^T, \quad (3.46)$$

where L is a lower triangular matrix with unit diagonal elements and D is a diagonal matrix with positive elements on the diagonal. By equating the elements in (3.46), column by column, it is easy to derive formulas for computing L and D .

□ EXAMPLE 3.1

Consider the case $n = 3$. The equation $A = LDL^T$ is given by

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \begin{bmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{bmatrix}.$$

(The notation indicates that A is symmetric.) By equating the elements of the first column, we have

$$\begin{aligned} a_{11} &= d_1, \\ a_{21} &= d_1 l_{21} \quad \Rightarrow \quad l_{21} = a_{21}/d_1, \\ a_{31} &= d_1 l_{31} \quad \Rightarrow \quad l_{31} = a_{31}/d_1. \end{aligned}$$

Proceeding with the next two columns, we obtain

$$\begin{aligned} a_{22} &= d_1 l_{21}^2 + d_2 &\Rightarrow d_2 &= a_{22} - d_1 l_{21}^2, \\ a_{32} &= d_1 l_{31} l_{21} + d_2 l_{32} &\Rightarrow l_{32} &= (a_{32} - d_1 l_{31} l_{21}) / d_2, \\ a_{33} &= d_1 l_{31}^2 + d_2 l_{32}^2 + d_3 &\Rightarrow d_3 &= a_{33} - d_1 l_{31}^2 - d_2 l_{32}^2. \end{aligned}$$

□

This procedure is generalized in the following algorithm.

Algorithm 3.4 (Cholesky Factorization, LDL^T Form).

```

for    $j = 1, 2, \dots, n$ 
     $c_{jj} \leftarrow a_{jj} - \sum_{s=1}^{j-1} d_s l_{js}^2$ ;
     $d_j \leftarrow c_{jj}$ ;
    for    $i = j + 1, \dots, n$ 
         $c_{ij} \leftarrow a_{ij} - \sum_{s=1}^{j-1} d_s l_{is} l_{js}$ ;
         $l_{ij} \leftarrow c_{ij} / d_j$ ;
    end
end

```

One can show (see, for example, Golub and Van Loan [136, Section 4.2.3]) that the diagonal elements d_{jj} are all positive whenever A is positive definite. The scalars c_{ij} have been introduced only to facilitate the description of the modified factorization discussed below. We should note that Algorithm 3.4 differs a little from the standard form of the Cholesky factorization, which produces a lower triangular matrix M such that

$$A = MM^T. \quad (3.47)$$

In fact, we can make the identification $M = LD^{1/2}$ to relate M to the factors L and D computed in Algorithm 3.4. The technique for computing M appears as Algorithm A.2 in Appendix A.

If A is indefinite, the factorization $A = LDL^T$ may not exist. Even if it does exist, Algorithm 3.4 is numerically unstable when applied to such matrices, in the sense that the elements of L and D can become arbitrarily large. It follows that a strategy of computing the LDL^T factorization and then modifying the diagonal after the fact to force its elements to be positive may break down, or may result in a matrix that is drastically different from A .

Instead, we can modify the matrix A during the course of the factorization in such a way that all elements in D are sufficiently positive, and so that the elements of D and L are not too large. To control the quality of the modification, we choose two positive parameters δ and β , and require that during the computation of the j th columns of L and D in Algorithm 3.4 (that is, for each j in the outer loop of the algorithm) the following

bounds be satisfied:

$$d_j \geq \delta, \quad |m_{ij}| \leq \beta, \quad i = j + 1, j + 2, \dots, n, \quad (3.48)$$

where $m_{ij} = l_{ij}\sqrt{d_j}$. To satisfy these bounds we only need to change one step in Algorithm 3.4: The formula for computing the diagonal element d_j in Algorithm 3.4 is replaced by

$$d_j = \max \left(|c_{jj}|, \left(\frac{\theta_j}{\beta} \right)^2, \delta \right), \quad \text{with } \theta_j = \max_{j < i \leq n} |c_{ij}|. \quad (3.49)$$

To verify that (3.48) holds, we note from Algorithm 3.4 that $c_{ij} = l_{ij}d_j$, and therefore

$$|m_{ij}| = |l_{ij}\sqrt{d_j}| = \frac{|c_{ij}|}{\sqrt{d_j}} \leq \frac{|c_{ij}|\beta}{\theta_j} \leq \beta, \quad \text{for all } i > j.$$

We note that θ_j can be computed prior to d_j because the elements c_{ij} in the second **for** loop of Algorithm 3.4 do not involve d_j . In fact, this is the reason for introducing the quantities c_{ij} into the algorithm.

These observations are the basis of the modified Cholesky algorithm described in detail in Gill, Murray, and Wright [130], which introduces symmetric interchanges of rows and columns to try to reduce the size of the modification. If P denotes the permutation matrix associated with the row and column interchanges, the algorithm produces the Cholesky factorization of the permuted, modified matrix $PAP^T + E$, that is,

$$PAP^T + E = LDL^T = MM^T, \quad (3.50)$$

where E is a nonnegative diagonal matrix that is zero if A is sufficiently positive definite. One can show (Moré and Sorensen [215]) that the matrices B_k obtained by this modified Cholesky algorithm to the exact Hessians $\nabla^2 f(x_k)$ have bounded condition numbers, that is, the bound (3.39) holds for some value of C .

MODIFIED SYMMETRIC INDEFINITE FACTORIZATION

Another strategy for modifying an indefinite Hessian is to use a procedure based on a symmetric indefinite factorization. Any symmetric matrix A , whether positive definite or not, can be written as

$$PAP^T = LBL^T, \quad (3.51)$$

where L is unit lower triangular, B is a block diagonal matrix with blocks of dimension 1 or 2, and P is a permutation matrix (see our discussion in Appendix A and also Golub and

Van Loan [136, Section 4.4]). We mentioned earlier that attempting to compute the LDL^T factorization of an indefinite matrix (where D is a *diagonal* matrix) is inadvisable because even if the factors L and D are well defined, they may contain entries that are larger than the original elements of A , thus amplifying rounding errors that arise during the computation. However, by using the block diagonal matrix B , which allows 2×2 blocks as well as 1×1 blocks on the diagonal, we can guarantee that the factorization (3.51) always exists and can be computed by a numerically stable process.

□ **EXAMPLE 3.2**

The matrix

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 2 & 2 \\ 2 & 2 & 3 & 3 \\ 3 & 2 & 3 & 4 \end{bmatrix}$$

can be written in the form (3.51) with $P = [e_1, e_4, e_3, e_2]$,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{9} & \frac{2}{3} & 1 & 0 \\ \frac{2}{9} & \frac{1}{3} & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & \frac{7}{9} & \frac{5}{9} \\ 0 & 0 & \frac{5}{9} & \frac{10}{9} \end{bmatrix}. \quad (3.52)$$

Note that both diagonal blocks in B are 2×2 . Several algorithms for computing symmetric indefinite factorizations are discussed in Section A.1 of Appendix A. □

The symmetric indefinite factorization allows us to determine the *inertia* of a matrix, that is, the number of positive, zero, and negative eigenvalues. One can show that the inertia of B equals the inertia of A . Moreover, the 2×2 blocks in B are always constructed to have one positive and one negative eigenvalue. Thus the number of positive eigenvalues in A equals the number of positive 1×1 blocks plus the number of 2×2 blocks.

As for the Cholesky factorization, an indefinite symmetric factorization algorithm can be modified to ensure that the modified factors are the factors of a positive definite matrix. The strategy is first to compute the factorization (3.51), as well as the spectral decomposition $B = Q\Lambda Q^T$, which is inexpensive to compute because B is block diagonal

(see Exercise 3.12). We then construct a modification matrix F such that

$$L(B + F)L^T$$

is sufficiently positive definite. Motivated by the modified spectral decomposition (3.43), we choose a parameter $\delta > 0$ and define F to be

$$F = Q \operatorname{diag}(\tau_i) Q^T, \quad \tau_i = \begin{cases} 0, & \lambda_i \geq \delta, \\ \delta - \lambda_i, & \lambda_i < \delta, \end{cases} \quad i = 1, 2, \dots, n, \quad (3.53)$$

where λ_i are the eigenvalues of B . The matrix F is thus the modification of minimum Frobenius norm that ensures that all eigenvalues of the modified matrix $B + F$ are no less than δ . This strategy therefore modifies the factorization (3.51) as follows:

$$P(A + E)P^T = L(B + F)L^T, \quad \text{where } E = P^T L F L^T P.$$

(Note that E will not be diagonal, in general.) Hence, in contrast to the modified Cholesky approach, this modification strategy changes the entire matrix A , not just its diagonal. The aim of strategy (3.53) is that the modified matrix satisfies $\lambda_{\min}(A + E) \approx \delta$ whenever the original matrix A has $\lambda_{\min}(A) < \delta$. It is not clear, however, whether it always comes close to attaining this goal.

3.5 STEP-LENGTH SELECTION ALGORITHMS

We now consider techniques for finding a minimum of the one-dimensional function

$$\phi(\alpha) = f(x_k + \alpha p_k), \quad (3.54)$$

or for simply finding a step length α_k satisfying one of the termination conditions described in Section 3.1. We assume that p_k is a descent direction—that is, $\phi'(0) < 0$ —so that our search can be confined to positive values of α .

If f is a convex quadratic, $f(x) = \frac{1}{2}x^T Qx - b^T x$, its one-dimensional minimizer along the ray $x_k + \alpha p_k$ can be computed analytically and is given by

$$\alpha_k = -\frac{\nabla f_k^T p_k}{p_k^T Q p_k}. \quad (3.55)$$

For general nonlinear functions, it is necessary to use an iterative procedure. The line search procedure deserves particular attention because it has a major impact on the robustness and efficiency of all nonlinear optimization methods.

Line search procedures can be classified according to the type of derivative information they use. Algorithms that use only function values can be inefficient since, to be theoretically sound, they need to continue iterating until the search for the minimizer is narrowed down to a small interval. In contrast, knowledge of gradient information allows us to determine whether a suitable step length has been located, as stipulated, for example, by the Wolfe conditions (3.6) or Goldstein conditions (3.11). Often, particularly when x_k is close to the solution, the very first choice of α satisfies these conditions, so the line search need not be invoked at all. In the rest of this section, we discuss only algorithms that make use of derivative information. More information on derivative-free procedures is given in the notes at the end of this chapter.

All line search procedures require an initial estimate α_0 and generate a sequence $\{\alpha_i\}$ that either terminates with a step length satisfying the conditions specified by the user (for example, the Wolfe conditions) or determines that such a step length does not exist. Typical procedures consist of two phases: a *bracketing phase* that finds an interval $[\bar{a}, \bar{b}]$ containing acceptable step lengths, and a *selection phase* that zooms in to locate the final step length. The selection phase usually reduces the bracketing interval during its search for the desired step length and interpolates some of the function and derivative information gathered on earlier steps to guess the location of the minimizer. We first discuss how to perform this interpolation.

In the following discussion we let α_k and α_{k-1} denote the step lengths used at iterations k and $k - 1$ of the optimization algorithm, respectively. On the other hand, we denote the trial step lengths generated during the line search by α_i and α_{i-1} and also α_j . We use α_0 to denote the initial guess.

INTERPOLATION

We begin by describing a line search procedure based on interpolation of known function and derivative values of the function ϕ . This procedure can be viewed as an enhancement of Algorithm 3.1. The aim is to find a value of α that satisfies the sufficient decrease condition (3.6a), without being “too small.” Accordingly, the procedures here generate a decreasing sequence of values α_i such that each value α_i is not too much smaller than its predecessor α_{i-1} .

Note that we can write the sufficient decrease condition in the notation of (3.54) as

$$\phi(\alpha_k) \leq \phi(0) + c_1 \alpha_k \phi'(0), \quad (3.56)$$

and that since the constant c_1 is usually chosen to be small in practice ($c_1 = 10^{-4}$, say), this condition asks for little more than descent in f . We design the procedure to be “efficient” in the sense that it computes the derivative $\nabla f(x)$ as few times as possible.

Suppose that the initial guess α_0 is given. If we have

$$\phi(\alpha_0) \leq \phi(0) + c_1 \alpha_0 \phi'(0),$$

this step length satisfies the condition, and we terminate the search. Otherwise, we know that the interval $[0, \alpha_0]$ contains acceptable step lengths (see Figure 3.3). We form a quadratic approximation $\phi_q(\alpha)$ to ϕ by interpolating the three pieces of information available— $\phi(0)$, $\phi'(0)$, and $\phi(\alpha_0)$ —to obtain

$$\phi_q(\alpha) = \left(\frac{\phi(\alpha_0) - \phi(0) - \alpha_0 \phi'(0)}{\alpha_0^2} \right) \alpha^2 + \phi'(0)\alpha + \phi(0). \quad (3.57)$$

(Note that this function is constructed so that it satisfies the interpolation conditions $\phi_q(0) = \phi(0)$, $\phi'_q(0) = \phi'(0)$, and $\phi_q(\alpha_0) = \phi(\alpha_0)$.) The new trial value α_1 is defined as the minimizer of this quadratic, that is, we obtain

$$\alpha_1 = -\frac{\phi'(0)\alpha_0^2}{2[\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0]}. \quad (3.58)$$

If the sufficient decrease condition (3.56) is satisfied at α_1 , we terminate the search. Otherwise, we construct a *cubic* function that interpolates the four pieces of information $\phi(0)$, $\phi'(0)$, $\phi(\alpha_0)$, and $\phi(\alpha_1)$, obtaining

$$\phi_c(\alpha) = a\alpha^3 + b\alpha^2 + \alpha\phi'(0) + \phi(0),$$

where

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\alpha_0^2 \alpha_1^2 (\alpha_1 - \alpha_0)} \begin{bmatrix} \alpha_0^2 & -\alpha_1^2 \\ -\alpha_0^3 & \alpha_1^3 \end{bmatrix} \begin{bmatrix} \phi(\alpha_1) - \phi(0) - \phi'(0)\alpha_1 \\ \phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0 \end{bmatrix}.$$

By differentiating $\phi_c(x)$, we see that the minimizer α_2 of ϕ_c lies in the interval $[0, \alpha_1]$ and is given by

$$\alpha_2 = \frac{-b + \sqrt{b^2 - 3a\phi'(0)}}{3a}.$$

If necessary, this process is repeated, using a cubic interpolant of $\phi(0)$, $\phi'(0)$ and the two most recent values of ϕ , until an α that satisfies (3.56) is located. If any α_i is either too close to its predecessor α_{i-1} or else too much smaller than α_{i-1} , we reset $\alpha_i = \alpha_{i-1}/2$. This safeguard procedure ensures that we make reasonable progress on each iteration and that the final α is not too small.

The strategy just described assumes that derivative values are significantly more expensive to compute than function values. It is often possible, however, to compute the directional derivative simultaneously with the function, at little additional cost; see Chapter 8. Accordingly, we can design an alternative strategy based on cubic interpolation of the values of ϕ and ϕ' at the two most recent values of α .

Cubic interpolation provides a good model for functions with significant changes of curvature. Suppose we have an interval $[\bar{a}, \bar{b}]$ known to contain desirable step lengths, and two previous step length estimates α_{i-1} and α_i in this interval. We use a cubic function to interpolate $\phi(\alpha_{i-1})$, $\phi'(\alpha_{i-1})$, $\phi(\alpha_i)$, and $\phi'(\alpha_i)$. (This cubic function always exists and is unique; see, for example, Bulirsch and Stoer [41, p. 52].) The minimizer of this cubic in $[\bar{a}, \bar{b}]$ is either at one of the endpoints or else in the interior, in which case it is given by

$$\alpha_{i+1} = \alpha_i - (\alpha_i - \alpha_{i-1}) \left[\frac{\phi'(\alpha_i) + d_2 - d_1}{\phi'(\alpha_i) - \phi'(\alpha_{i-1}) + 2d_2} \right], \quad (3.59)$$

with

$$\begin{aligned} d_1 &= \phi'(\alpha_{i-1}) + \phi'(\alpha_i) - 3 \frac{\phi(\alpha_{i-1}) - \phi(\alpha_i)}{\alpha_{i-1} - \alpha_i}, \\ d_2 &= \text{sign}(\alpha_i - \alpha_{i-1}) [d_1^2 - \phi'(\alpha_{i-1})\phi'(\alpha_i)]^{1/2}. \end{aligned}$$

The interpolation process can be repeated by discarding the data at one of the step lengths α_{i-1} or α_i and replacing it by $\phi(\alpha_{i+1})$ and $\phi'(\alpha_{i+1})$. The decision on which of α_{i-1} and α_i should be kept and which discarded depends on the specific conditions used to terminate the line search; we discuss this issue further below in the context of the Wolfe conditions. Cubic interpolation is a powerful strategy, since it usually produces a quadratic rate of convergence of the iteration (3.59) to the minimizing value of α .

INITIAL STEP LENGTH

For Newton and quasi-Newton methods, the step $\alpha_0 = 1$ should always be used as the initial trial step length. This choice ensures that unit step lengths are taken whenever they satisfy the termination conditions and allows the rapid rate-of-convergence properties of these methods to take effect.

For methods that do not produce well scaled search directions, such as the steepest descent and conjugate gradient methods, it is important to use current information about the problem and the algorithm to make the initial guess. A popular strategy is to assume that the first-order change in the function at iterate x_k will be the same as that obtained at the previous step. In other words, we choose the initial guess α_0 so that $\alpha_0 \nabla f_k^T p_k = \alpha_{k-1} \nabla f_{k-1}^T p_{k-1}$, that is,

$$\alpha_0 = \alpha_{k-1} \frac{\nabla f_{k-1}^T p_{k-1}}{\nabla f_k^T p_k}.$$

Another useful strategy is to interpolate a quadratic to the data $f(x_{k-1})$, $f(x_k)$, and $\nabla f_{k-1}^T p_{k-1}$ and to define α_0 to be its minimizer. This strategy yields

$$\alpha_0 = \frac{2(f_k - f_{k-1})}{\phi'(0)}. \quad (3.60)$$

It can be shown that if $x_k \rightarrow x^*$ superlinearly, then the ratio in this expression converges to 1. If we adjust the choice (3.60) by setting

$$\alpha_0 \leftarrow \min(1, 1.01\alpha_0),$$

we find that the unit step length $\alpha_0 = 1$ will eventually always be tried and accepted, and the superlinear convergence properties of Newton and quasi-Newton methods will be observed.

A LINE SEARCH ALGORITHM FOR THE WOLFE CONDITIONS

The Wolfe (or strong Wolfe) conditions are among the most widely applicable and useful termination conditions. We now describe in some detail a one-dimensional search procedure that is guaranteed to find a step length satisfying the *strong* Wolfe conditions (3.7) for any parameters c_1 and c_2 satisfying $0 < c_1 < c_2 < 1$. As before, we assume that p is a descent direction and that f is bounded below along the direction p .

The algorithm has two stages. This first stage begins with a trial estimate α_1 , and keeps increasing it until it finds either an acceptable step length or an interval that brackets the desired step lengths. In the latter case, the second stage is invoked by calling a function called **zoom** (Algorithm 3.6, below), which successively decreases the size of the interval until an acceptable step length is identified.

A formal specification of the line search algorithm follows. We refer to (3.7a) as the *sufficient decrease condition* and to (3.7b) as the *curvature condition*. The parameter α_{\max} is a user-supplied bound on the maximum step length allowed. The line search algorithm terminates with α_* set to a step length that satisfies the strong Wolfe conditions.

Algorithm 3.5 (Line Search Algorithm).

```

Set  $\alpha_0 \leftarrow 0$ , choose  $\alpha_{\max} > 0$  and  $\alpha_1 \in (0, \alpha_{\max})$ ;
 $i \leftarrow 1$ ;
repeat
    Evaluate  $\phi(\alpha_i)$ ;
    if  $\phi(\alpha_i) > \phi(0) + c_1\alpha_i\phi'(0)$  or  $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$  and  $i > 1]$ 
         $\alpha_* \leftarrow \text{zoom}(\alpha_{i-1}, \alpha_i)$  and stop;
    Evaluate  $\phi'(\alpha_i)$ ;
    if  $|\phi'(\alpha_i)| \leq -c_2\phi'(0)$ 
        set  $\alpha_* \leftarrow \alpha_i$  and stop;
    if  $\phi'(\alpha_i) \geq 0$ 
        set  $\alpha_* \leftarrow \text{zoom}(\alpha_i, \alpha_{i-1})$  and stop;
    Choose  $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$ ;
     $i \leftarrow i + 1$ ;
end (repeat)
```

Note that the sequence of trial step lengths $\{\alpha_i\}$ is monotonically increasing, but that the order of the arguments supplied to the **zoom** function may vary. The procedure uses the knowledge that the interval (α_{i-1}, α_i) contains step lengths satisfying the strong Wolfe conditions if one of the following three conditions is satisfied:

- (i) α_i violates the sufficient decrease condition;
- (ii) $\phi(\alpha_i) \geq \phi(\alpha_{i-1})$;
- (iii) $\phi'(\alpha_i) \geq 0$.

The last step of the algorithm performs extrapolation to find the next trial value α_{i+1} . To implement this step we can use approaches like the interpolation procedures above, or we can simply set α_{i+1} to some constant multiple of α_i . Whichever strategy we use, it is important that the successive steps increase quickly enough to reach the upper limit α_{\max} in a finite number of iterations.

We now specify the function **zoom**, which requires a little explanation. The order of its input arguments is such that each call has the form **zoom**(α_{lo}, α_{hi}), where

- (a) the interval bounded by α_{lo} and α_{hi} contains step lengths that satisfy the strong Wolfe conditions;
- (b) α_{lo} is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest function value; and
- (c) α_{hi} is chosen so that $\phi'(\alpha_{lo})(\alpha_{hi} - \alpha_{lo}) < 0$.

Each iteration of **zoom** generates an iterate α_j between α_{lo} and α_{hi} , and then replaces one of these endpoints by α_j in such a way that the properties (a), (b), and (c) continue to hold.

Algorithm 3.6 (zoom).

repeat

Interpolate (using quadratic, cubic, or bisection) to find
a trial step length α_j between α_{lo} and α_{hi} ;

Evaluate $\phi(\alpha_j)$;

if $\phi(\alpha_j) > \phi(0) + c_1\alpha_j\phi'(0)$ or $\phi(\alpha_j) \geq \phi(\alpha_{lo})$

$\alpha_{hi} \leftarrow \alpha_j$;

else

Evaluate $\phi'(\alpha_j)$;

if $|\phi'(\alpha_j)| \leq -c_2\phi'(0)$

Set $\alpha_* \leftarrow \alpha_j$ and **stop**;

if $\phi'(\alpha_j)(\alpha_{hi} - \alpha_{lo}) \geq 0$

$\alpha_{hi} \leftarrow \alpha_{lo}$;

$\alpha_{lo} \leftarrow \alpha_j$;

end (repeat)

If the new estimate α_j happens to satisfy the strong Wolfe conditions, then **zoom** has served its purpose of identifying such a point, so it terminates with $\alpha_* = \alpha_j$. Otherwise, if α_j satisfies the sufficient decrease condition and has a lower function value than x_{lo} , then we set $\alpha_{lo} \leftarrow \alpha_j$ to maintain condition (b). If this setting results in a violation of condition (c), we remedy the situation by setting α_{hi} to the old value of α_{lo} . Readers should sketch some graphs to see for themselves how **zoom** works!

As mentioned earlier, the interpolation step that determines α_j should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval. Practical line search algorithms also make use of the properties of the interpolating polynomials to make educated guesses of where the next step length should lie; see [39, 216]. A problem that can arise is that as the optimization algorithm approaches the solution, two consecutive function values $f(x_k)$ and $f(x_{k-1})$ may be indistinguishable in finite-precision arithmetic. Therefore, the line search must include a stopping test if it cannot attain a lower function value after a certain number (typically, ten) of trial step lengths. Some procedures also stop if the relative change in x is close to machine precision, or to some user-specified threshold.

A line search algorithm that incorporates all these features is difficult to code. We advocate the use of one of the several good software implementations available in the public domain. See Dennis and Schnabel [92], Lemaréchal [189], Fletcher [101], Moré and Thuente [216] (in particular), and Hager and Zhang [161].

One may ask how much more expensive it is to require the strong Wolfe conditions instead of the regular Wolfe conditions. Our experience suggests that for a “loose” line search (with parameters such as $c_1 = 10^{-4}$ and $c_2 = 0.9$), both strategies require a similar amount of work. The strong Wolfe conditions have the advantage that by decreasing c_2 we can directly control the quality of the search, by forcing the accepted value of α to lie closer to a local minimum. This feature is important in steepest descent or nonlinear conjugate gradient methods, and therefore a step selection routine that enforces the strong Wolfe conditions has wide applicability.

NOTES AND REFERENCES

For an extensive discussion of line search termination conditions see Ortega and Rheinboldt [230]. Akaike [2] presents a probabilistic analysis of the steepest descent method with exact line searches on quadratic functions. He shows that when $n > 2$, the worst-case bound (3.29) can be expected to hold for most starting points. The case $n = 2$ can be studied in closed form; see Bazaraa, Sherali, and Shetty [14]. Theorem 3.6 is due to Dennis and Moré.

Some line search methods (see Goldfarb [132] and Moré and Sorensen [213]) compute a direction of negative curvature, whenever it exists, to prevent the iteration from converging to nonminimizing stationary points. A direction of negative curvature p_- is one that satisfies $p_-^T \nabla^2 f(x_k) p_- < 0$. These algorithms generate a search direction by combining p_- with the steepest descent direction $-\nabla f_k$, often performing a curvilinear backtracking line search.

It is difficult to determine the relative contributions of the steepest descent and negative curvature directions. Because of this fact, the approach fell out of favor after the introduction of trust-region methods.

For a more thorough treatment of the modified Cholesky factorization see Gill, Murray, and Wright [130] or Dennis and Schnabel [92]. A modified Cholesky factorization based on Gershgorin disk estimates is described in Schnabel and Eskow [276]. The modified indefinite factorization is from Cheng and Higham [58].

Another strategy for implementing a line search Newton method when the Hessian contains negative eigenvalues is to compute a direction of negative curvature and use it to define the search direction (see Moré and Sorensen [213] and Goldfarb [132]).

Derivative-free line search algorithms include golden section and Fibonacci search. They share some of the features with the line search method given in this chapter. They typically store three trial points that determine an interval containing a one-dimensional minimizer. Golden section and Fibonacci differ in the way in which the trial step lengths are generated; see, for example, [79, 39].

Our discussion of interpolation follows Dennis and Schnabel [92], and the algorithm for finding a step length satisfying the strong Wolfe conditions can be found in Fletcher [101].



EXERCISES



3.1 Program the steepest descent and Newton algorithms using the backtracking line search, Algorithm 3.1. Use them to minimize the Rosenbrock function (2.22). Set the initial step length $\alpha_0 = 1$ and print the step length used by each method at each iteration. First try the initial point $x_0 = (1.2, 1.2)^T$ and then the more difficult starting point $x_0 = (-1.2, 1)^T$.



3.2 Show that if $0 < c_2 < c_1 < 1$, there may be no step lengths that satisfy the Wolfe conditions.



3.3 Show that the one-dimensional minimizer of a strongly convex quadratic function is given by (3.55).




3.4 Show that the one-dimensional minimizer of a strongly convex quadratic function always satisfies the Goldstein conditions (3.11).



3.5 Prove that $\|Bx\| \geq \|x\|/\|B^{-1}\|$ for any nonsingular matrix B . Use this fact to establish (3.19).



3.6 Consider the steepest descent method with exact line searches applied to the convex quadratic function (3.24). Using the properties given in this chapter, show that if the initial point is such that $x_0 - x^*$ is parallel to an eigenvector of Q , then the steepest descent method will find the solution in one step.

 **3.7** Prove the result (3.28) by working through the following steps. First, use (3.26) to show that


$$\|x_k - x^*\|_Q^2 - \|x_{k+1} - x^*\|_Q^2 = 2\alpha_k \nabla f_k^T Q(x_k - x^*) - \alpha_k^2 \nabla f_k^T Q \nabla f_k,$$

where $\|\cdot\|_Q$ is defined by (3.27). Second, use the fact that $\nabla f_k = Q(x_k - x^*)$ to obtain

$$\|x_k - x^*\|_Q^2 - \|x_{k+1} - x^*\|_Q^2 = \frac{2(\nabla f_k^T \nabla f_k)^2}{(\nabla f_k^T Q \nabla f_k)} - \frac{(\nabla f_k^T \nabla f_k)^2}{(\nabla f_k^T Q \nabla f_k)}$$


and


$$\|x_k - x^*\|_Q^2 = \nabla f_k^T Q^{-1} \nabla f_k.$$


 **3.8** Let Q be a positive definite symmetric matrix. Prove that for any vector x , we have


$$\frac{(x^T x)^2}{(x^T Q x)(x^T Q^{-1} x)} \geq \frac{4\lambda_n \lambda_1}{(\lambda_n + \lambda_1)^2},$$


where λ_n and λ_1 are, respectively, the largest and smallest eigenvalues of Q . (This relation, which is known as the Kantorovich inequality, can be used to deduce (3.29) from (3.28).)

 **3.9** Program the BFGS algorithm using the line search algorithm described in this chapter that implements the strong Wolfe conditions. Have the code verify that $y_k^T s_k$ is always positive. Use it to minimize the Rosenbrock function using the starting points given in Exercise 3.1.

 **3.10** Compute the eigenvalues of the 2 diagonal blocks of (3.52) and verify that each block has a positive and a negative eigenvalue. Then compute the eigenvalues of A and verify that its inertia is the same as that of B .


 **3.11** Describe the effect that the modified Cholesky factorization (3.50) would have on the Hessian $\nabla^2 f(x_k) = \text{diag}(-2, 12, 4)$.


 **3.12** Consider a block diagonal matrix B with 1×1 and 2×2 blocks. Show that the eigenvalues and eigenvectors of B can be obtained by computing the spectral decomposition of each diagonal block separately.

 **3.13** Show that the quadratic function that interpolates $\phi(0)$, $\phi'(0)$, and $\phi(\alpha_0)$ is given by (3.57). Then, make use of the fact that the sufficient decrease condition (3.6a) is not satisfied at α_0 to show that this quadratic has positive curvature and that the minimizer satisfies

$$\alpha_1 < \frac{\alpha_0}{2(1 - c_1)}.$$

Since c_1 is chosen to be quite small in practice, this inequality indicates that α_1 cannot be much greater than $\frac{1}{2}$ (and may be smaller), which gives us an idea of the new step length.

 **3.14** If $\phi(\alpha_0)$ is large, (3.58) shows that α_1 can be quite small. Give an example of a function and a step length α_0 for which this situation arises. (Drastic changes to the estimate of the step length are not desirable, since they indicate that the current interpolant does not provide a good approximation to the function and that it should be modified before being trusted to produce a good step length estimate. In practice, one imposes a lower bound—typically, $\rho = 0.1$ —and defines the new step length as $\alpha_i = \max(\rho\alpha_{i-1}, \hat{\alpha}_i)$, where $\hat{\alpha}_i$ is the minimizer of the interpolant.)

 **3.15** Suppose that the sufficient decrease condition (3.6a) is not satisfied at the step lengths α_0 , and α_1 , and consider the cubic interpolating $\phi(0)$, $\phi'(0)$, $\phi(\alpha_0)$ and $\phi(\alpha_1)$. By drawing graphs illustrating the two situations that can arise, show that the minimizer of the cubic lies in $[0, \alpha_1]$. Then show that if $\phi(0) < \phi(\alpha_1)$, the minimizer is less than $\frac{2}{3}\alpha_1$.

CHAPTER 4

Trust-Region Methods

Line search methods and trust-region methods both generate steps with the help of a quadratic model of the objective function, but they use this model in different ways. Line search methods use it to generate a search direction, and then focus their efforts on finding a suitable step length α along this direction. Trust-region methods define a region around the current iterate within which they *trust* the model to be an adequate representation of the objective function, and then choose the step to be the approximate minimizer of the model in this region. In effect, they choose the direction and length of the step simultaneously. If a step is not acceptable, they reduce the size of the region and find a new

CHAPTER 6

Quasi-Newton Methods

In the mid 1950s, W.C. Davidon, a physicist working at Argonne National Laboratory, was using the coordinate descent method (see Section 9.3) to perform a long optimization calculation. At that time computers were not very stable, and to Davidon's frustration, the computer system would always crash before the calculation was finished. So Davidon decided to find a way of accelerating the iteration. The algorithm he developed—the first quasi-Newton algorithm—turned out to be one of the most creative ideas in nonlinear optimization. It was soon demonstrated by Fletcher and Powell that the new algorithm was much faster and more reliable than the other existing methods, and this dramatic

advance transformed nonlinear optimization overnight. During the following twenty years, numerous variants were proposed and hundreds of papers were devoted to their study. An interesting historical irony is that Davidon's paper [87] was not accepted for publication; it remained as a technical report for more than thirty years until it appeared in the first issue of the *SIAM Journal on Optimization* in 1991 [88].

Quasi-Newton methods, like steepest descent, require only the gradient of the objective function to be supplied at each iterate. By measuring the changes in gradients, they construct a model of the objective function that is good enough to produce superlinear convergence. The improvement over steepest descent is dramatic, especially on difficult problems. Moreover, since second derivatives are not required, quasi-Newton methods are sometimes more efficient than Newton's method. Today, optimization software libraries contain a variety of quasi-Newton algorithms for solving unconstrained, constrained, and large-scale optimization problems. In this chapter we discuss quasi-Newton methods for small and medium-sized problems, and in Chapter 7 we consider their extension to the large-scale setting.

The development of automatic differentiation techniques has made it possible to use Newton's method without requiring users to supply second derivatives; see Chapter 8. Still, automatic differentiation tools may not be applicable in many situations, and it may be much more costly to work with second derivatives in automatic differentiation software than with the gradient. For these reasons, quasi-Newton methods remain appealing.

6.1 THE BFGS METHOD

The most popular quasi-Newton algorithm is the BFGS method, named for its discoverers Broyden, Fletcher, Goldfarb, and Shanno. In this section we derive this algorithm (and its close relative, the DFP algorithm) and describe its theoretical properties and practical implementation.

We begin the derivation by forming the following quadratic model of the objective function at the current iterate x_k :

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p. \quad (6.1)$$

Here B_k is an $n \times n$ symmetric positive definite matrix that will be revised or *updated* at every iteration. Note that the function value and gradient of this model at $p = 0$ match f_k and ∇f_k , respectively. The minimizer p_k of this convex quadratic model, which we can write explicitly as

$$p_k = -B_k^{-1} \nabla f_k, \quad (6.2)$$

is used as the search direction, and the new iterate is

$$x_{k+1} = x_k + \alpha_k p_k, \quad (6.3)$$

where the step length α_k is chosen to satisfy the Wolfe conditions (3.6). This iteration is quite similar to the line search Newton method; the key difference is that the approximate Hessian B_k is used in place of the true Hessian.

Instead of computing B_k afresh at every iteration, Davidon proposed to update it in a simple manner to account for the curvature measured during the most recent step. Suppose that we have generated a new iterate x_{k+1} and wish to construct a new quadratic model, of the form

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p.$$

What requirements should we impose on B_{k+1} , based on the knowledge gained during the latest step? One reasonable requirement is that the gradient of m_{k+1} should match the gradient of the objective function f at the latest two iterates x_k and x_{k+1} . Since $\nabla m_{k+1}(0)$ is precisely ∇f_{k+1} , the second of these conditions is satisfied automatically. The first condition can be written mathematically as

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k = \nabla f_k.$$

By rearranging, we obtain

$$B_{k+1} \alpha_k p_k = \nabla f_{k+1} - \nabla f_k. \quad (6.4)$$

To simplify the notation it is useful to define the vectors

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \quad y_k = \nabla f_{k+1} - \nabla f_k, \quad (6.5)$$

so that (6.4) becomes

$$B_{k+1} s_k = y_k. \quad (6.6)$$

We refer to this formula as the *secant equation*.

Given the displacement s_k and the change of gradients y_k , the secant equation requires that the symmetric positive definite matrix B_{k+1} map s_k into y_k . This will be possible only if s_k and y_k satisfy the *curvature condition*

$$s_k^T y_k > 0, \quad (6.7)$$

as is easily seen by premultiplying (6.6) by s_k^T . When f is strongly convex, the inequality (6.7) will be satisfied for any two points x_k and x_{k+1} (see Exercise 6.1). However, this condition

will not always hold for nonconvex functions, and in this case we need to enforce (6.7) explicitly, by imposing restrictions on the line search procedure that chooses the step length α . In fact, the condition (6.7) is guaranteed to hold if we impose the Wolfe (3.6) or strong Wolfe conditions (3.7) on the line search. To verify this claim, we note from (6.5) and (3.6b) that $\nabla f_{k+1}^T s_k \geq c_2 \nabla f_k^T s_k$, and therefore

$$y_k^T s_k \geq (c_2 - 1) \alpha_k \nabla f_k^T p_k. \quad (6.8)$$

Since $c_2 < 1$ and since p_k is a descent direction, the term on the right is positive, and the curvature condition (6.7) holds.

When the curvature condition is satisfied, the secant equation (6.6) always has a solution B_{k+1} . In fact, it admits an infinite number of solutions, since the $n(n+1)/2$ degrees of freedom in a symmetric positive definite matrix exceed the n conditions imposed by the secant equation. The requirement of positive definiteness imposes n additional inequalities—all principal minors must be positive—but these conditions do not absorb the remaining degrees of freedom.

To determine B_{k+1} uniquely, we impose the additional condition that *among all symmetric matrices satisfying the secant equation, B_{k+1} is, in some sense, closest to the current matrix B_k* . In other words, we solve the problem

$$\min_B \|B - B_k\| \quad (6.9a)$$

$$\text{subject to } B = B^T, \quad B s_k = y_k, \quad (6.9b)$$

where s_k and y_k satisfy (6.7) and B_k is symmetric and positive definite. Different matrix norms can be used in (6.9a), and each norm gives rise to a different quasi-Newton method. A norm that allows easy solution of the minimization problem (6.9) and gives rise to a scale-invariant optimization method is the weighted Frobenius norm

$$\|A\|_W \equiv \|W^{1/2} A W^{1/2}\|_F, \quad (6.10)$$

where $\|\cdot\|_F$ is defined by $\|C\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n c_{ij}^2$. The weight matrix W can be chosen as *any* matrix satisfying the relation $W y_k = s_k$. For concreteness, the reader can assume that $W = \bar{G}_k^{-1}$ where \bar{G}_k is the *average Hessian* defined by

$$\bar{G}_k = \left[\int_0^1 \nabla^2 f(x_k + \tau \alpha_k p_k) d\tau \right]. \quad (6.11)$$

The property

$$y_k = \bar{G}_k \alpha_k p_k = \bar{G}_k s_k \quad (6.12)$$

follows from Taylor's theorem, Theorem 2.1. With this choice of weighting matrix W , the

norm (6.10) is non-dimensional, which is a desirable property, since we do not wish the solution of (6.9) to depend on the units of the problem.

With this weighting matrix and this norm, the unique solution of (6.9) is

$$(DFP) \quad B_{k+1} = (I - \rho_k y_k s_k^T) B_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T, \quad (6.13)$$

with

$$\rho_k = \frac{1}{y_k^T s_k}. \quad (6.14)$$

This formula is called the DFP updating formula, since it is the one originally proposed by Davidon in 1959, and subsequently studied, implemented, and popularized by Fletcher and Powell.

The inverse of B_k , which we denote by

$$H_k = B_k^{-1},$$

is useful in the implementation of the method, since it allows the search direction (6.2) to be calculated by means of a simple matrix–vector multiplication. Using the Sherman–Morrison–Woodbury formula (A.28), we can derive the following expression for the update of the inverse Hessian approximation H_k that corresponds to the DFP update of B_k in (6.13):

$$(DFP) \quad H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}. \quad (6.15)$$

Note that the last two terms in the right-hand-side of (6.15) are rank-one matrices, so that H_k undergoes a rank-two modification. It is easy to see that (6.13) is also a rank-two modification of B_k . This is the fundamental idea of quasi-Newton updating: Instead of recomputing the approximate Hessians (or inverse Hessians) from scratch at every iteration, we apply a simple modification that combines the most recently observed information about the objective function with the existing knowledge embedded in our current Hessian approximation.

The DFP updating formula is quite effective, but it was soon superseded by the BFGS formula, which is presently considered to be the most effective of all quasi-Newton updating formulae. BFGS updating can be derived by making a simple change in the argument that led to (6.13). Instead of imposing conditions on the Hessian approximations B_k , we impose similar conditions on their inverses H_k . The updated approximation H_{k+1} must be symmetric and positive definite, and must satisfy the secant equation (6.6), now written as

$$H_{k+1} y_k = s_k.$$

The condition of closeness to H_k is now specified by the following analogue of (6.9):

$$\min_H \|H - H_k\| \quad (6.16a)$$

$$\text{subject to } H = H^T, \quad H y_k = s_k. \quad (6.16b)$$

The norm is again the weighted Frobenius norm described above, where the weight matrix W is now any matrix satisfying $Ws_k = y_k$. (For concreteness, we assume again that W is given by the average Hessian \bar{G}_k defined in (6.11).) The unique solution H_{k+1} to (6.16) is given by

$$(\text{BFGS}) \quad H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad (6.17)$$

with ρ_k defined by (6.14).

Just one issue has to be resolved before we can define a complete BFGS algorithm: How should we choose the initial approximation H_0 ? Unfortunately, there is no magic formula that works well in all cases. We can use specific information about the problem, for instance by setting it to the inverse of an approximate Hessian calculated by finite differences at x_0 . Otherwise, we can simply set it to be the identity matrix, or a multiple of the identity matrix, where the multiple is chosen to reflect the scaling of the variables.

Algorithm 6.1 (BFGS Method).

Given starting point x_0 , convergence tolerance $\epsilon > 0$,

inverse Hessian approximation H_0 ;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

 Compute search direction

$$p_k = -H_k \nabla f_k; \quad (6.18)$$

 Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search procedure to satisfy the Wolfe conditions (3.6);

 Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} by means of (6.17);

$k \leftarrow k + 1$;

end (while)

Each iteration can be performed at a cost of $O(n^2)$ arithmetic operations (plus the cost of function and gradient evaluations); there are no $O(n^3)$ operations such as linear system solves or matrix–matrix operations. The algorithm is robust, and its rate of convergence is superlinear, which is fast enough for most practical purposes. Even though Newton’s method converges more rapidly (that is, quadratically), its cost per iteration usually is higher, because of its need for second derivatives and solution of a linear system.

We can derive a version of the BFGS algorithm that works with the Hessian approximation B_k rather than H_k . The update formula for B_k is obtained by simply applying the Sherman–Morrison–Woodbury formula (A.28) to (6.17) to obtain

$$(\text{BFGS}) \quad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \quad (6.19)$$

A naive implementation of this variant is not efficient for unconstrained minimization, because it requires the system $B_k p_k = -\nabla f_k$ to be solved for the step p_k , thereby increasing the cost of the step computation to $O(n^3)$. We discuss later, however, that less expensive implementations of this variant are possible by updating Cholesky factors of B_k .

PROPERTIES OF THE BFGS METHOD

It is usually easy to observe the superlinear rate of convergence of the BFGS method on practical problems. Below, we report the last few iterations of the steepest descent, BFGS, and an inexact Newton method on Rosenbrock's function (2.22). The table gives the value of $\|x_k - x^*\|$. The Wolfe conditions were imposed on the step length in all three methods. From the starting point $(-1.2, 1)$, the steepest descent method required 5264 iterations, whereas BFGS and Newton took only 34 and 21 iterations, respectively to reduce the gradient norm to 10^{-5} .

steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08

A few points in the derivation of the BFGS and DFP methods merit further discussion. Note that the minimization problem (6.16) that gives rise to the BFGS update formula does not explicitly require the updated Hessian approximation to be positive definite. It is easy to show, however, that H_{k+1} will be positive definite whenever H_k is positive definite, by using the following argument. First, note from (6.8) that $y_k^T s_k$ is positive, so that the updating formula (6.17), (6.14) is well-defined. For any nonzero vector z , we have

$$z^T H_{k+1} z = w^T H_k w + \rho_k (z^T s_k)^2 \geq 0,$$

where we have defined $w = z - \rho_k y_k (s_k^T z)$. The right hand side can be zero only if $s_k^T z = 0$, but in this case $w = z \neq 0$, which implies that the first term is greater than zero. Therefore, H_{k+1} is positive definite.

To make quasi-Newton updating formulae invariant to transformations in the variables (such as scaling transformations), it is necessary for the objectives (6.9a) and (6.16a) to be invariant under the same transformations. The choice of the weighting matrices W used to define the norms in (6.9a) and (6.16a) ensures that this condition holds. Many other choices of the weighting matrix W are possible, each one of them giving a different update formula. However, despite intensive searches, no formula has been found that is significantly more effective than BFGS.

The BFGS method has many interesting properties when applied to quadratic functions. We discuss these properties later in the more general context of the Broyden family of updating formulae, of which BFGS is a special case.

It is reasonable to ask whether there are situations in which the updating formula such as (6.17) can produce bad results. If at some iteration the matrix H_k becomes a poor approximation to the true inverse Hessian, is there any hope of correcting it? For example, when the inner product $y_k^T s_k$ is tiny (but positive), then it follows from (6.14), (6.17) that H_{k+1} contains very large elements. Is this behavior reasonable? A related question concerns the rounding errors that occur in finite-precision implementation of these methods. Can these errors grow to the point of erasing all useful information in the quasi-Newton approximate Hessian?

These questions have been studied analytically and experimentally, and it is now known that the BFGS formula has very effective self-correcting properties. If the matrix H_k incorrectly estimates the curvature in the objective function, and if this bad estimate slows down the iteration, then the Hessian approximation will tend to correct itself within a few steps. It is also known that the DFP method is less effective in correcting bad Hessian approximations; this property is believed to be the reason for its poorer practical performance. The self-correcting properties of BFGS hold only when an adequate line search is performed. In particular, the Wolfe line search conditions ensure that the gradients are sampled at points that allow the model (6.1) to capture appropriate curvature information.

It is interesting to note that the DFP and BFGS updating formulae are *duals* of each other, in the sense that one can be obtained from the other by the interchanges $s \leftrightarrow y$, $B \leftrightarrow H$. This symmetry is not surprising, given the manner in which we derived these methods above.

IMPLEMENTATION

A few details and enhancements need to be added to Algorithm 6.1 to produce an efficient implementation. The line search, which should satisfy either the Wolfe conditions (3.6) or the strong Wolfe conditions (3.7), should always try the step length $\alpha_k = 1$ first, because this step length will eventually always be accepted (under certain conditions), thereby producing superlinear convergence of the overall algorithm. Computational observations strongly suggest that it is more economical, in terms of function evaluations, to perform a fairly inaccurate line search. The values $c_1 = 10^{-4}$ and $c_2 = 0.9$ are commonly used in (3.6).

As mentioned earlier, the initial matrix H_0 often is set to some multiple βI of the identity, but there is no good general strategy for choosing the multiple β . If β is too large, so that the first step $p_0 = -\beta g_0$ is too long, many function evaluations may be required to find a suitable value for the step length α_0 . Some software asks the user to prescribe a value δ for the norm of the first step, and then set $H_0 = \delta \|g_0\|^{-1} I$ to achieve this norm.

A heuristic that is often quite effective is to scale the starting matrix *after* the first step has been computed but before the first BFGS update is performed. We change the

provisional value $H_0 = I$ by setting

$$H_0 \leftarrow \frac{y_k^T s_k}{y_k^T y_k} I, \quad (6.20)$$

before applying the update (6.14), (6.17) to obtain H_1 . This formula attempts to make the size of H_0 similar to that of $\nabla^2 f(x_0)^{-1}$, in the following sense. Assuming that the average Hessian defined in (6.11) is positive definite, there exists a square root $\bar{G}_k^{1/2}$ satisfying $\bar{G}_k = \bar{G}_k^{1/2} \bar{G}_k^{1/2}$ (see Exercise 6.6). Therefore, by defining $z_k = \bar{G}_k^{1/2} s_k$ and using the relation (6.12), we have

$$\frac{y_k^T s_k}{y_k^T y_k} = \frac{(\bar{G}_k^{1/2} s_k)^T \bar{G}_k^{1/2} s_k}{(\bar{G}_k^{1/2} s_k)^T \bar{G}_k \bar{G}_k^{1/2} s_k} = \frac{z_k^T z_k}{z_k^T \bar{G}_k z_k}. \quad (6.21)$$

The *reciprocal* of (6.21) is an approximation to one of the eigenvalues of \bar{G}_k , which in turn is close to an eigenvalue of $\nabla^2 f(x_k)$. Hence, the quotient (6.21) itself approximates an eigenvalue of $\nabla^2 f(x_k)^{-1}$. Other scaling factors can be used in (6.20), but the one presented here appears to be the most successful in practice.

In (6.19) we gave an update formula for a BFGS method that works with the Hessian approximation B_k instead of the the inverse Hessian approximation H_k . An efficient implementation of this approach does not store B_k explicitly, but rather the Cholesky factorization $L_k D_k L_k^T$ of this matrix. A formula that updates the factors L_k and D_k *directly* in $O(n^2)$ operations can be derived from (6.19). Since the linear system $B_k p_k = -\nabla f_k$ also can be solved in $O(n^2)$ operations (by performing triangular substitutions with L_k and L_k^T and a diagonal substitution with D_k), the total cost is quite similar to the variant described in Algorithm 6.1. A potential advantage of this alternative strategy is that it gives us the option of modifying diagonal elements in the D_k factor if they are not sufficiently large, to prevent instability when we divide by these elements during the calculation of p_k . However, computational experience suggests no real advantages for this variant, and we prefer the simpler strategy of Algorithm 6.1.

The performance of the BFGS method can degrade if the line search is not based on the Wolfe conditions. For example, some software implements an Armijo backtracking line search (see Section 3.1): The unit step length $\alpha_k = 1$ is tried first and is successively decreased until the sufficient decrease condition (3.6a) is satisfied. For this strategy, there is no guarantee that the curvature condition $y_k^T s_k > 0$ (6.7) will be satisfied by the chosen step, since a step length greater than 1 may be required to satisfy this condition. To cope with this shortcoming, some implementations simply *skip* the BFGS update by setting $H_{k+1} = H_k$ when $y_k^T s_k$ is negative or too close to zero. This approach is not recommended, because the updates may be skipped much too often to allow H_k to capture important curvature information for the objective function f . In Chapter 18 we discuss a *damped* BFGS update that is a more effective strategy for coping with the case where the curvature condition (6.7) is not satisfied.

6.2 THE SR1 METHOD

In the BFGS and DFP updating formulae, the updated matrix B_{k+1} (or H_{k+1}) differs from its predecessor B_k (or H_k) by a rank-2 matrix. In fact, as we now show, there is a simpler rank-1 update that maintains symmetry of the matrix and allows it to satisfy the secant equation. Unlike the rank-two update formulae, this *symmetric-rank-1*, or SR1, update does not guarantee that the updated matrix maintains positive definiteness. Good numerical results have been obtained with algorithms based on SR1, so we derive it here and investigate its properties.

The symmetric rank-1 update has the general form

$$B_{k+1} = B_k + \sigma v v^T,$$

where σ is either $+1$ or -1 , and σ and v are chosen so that B_{k+1} satisfies the secant equation (6.6), that is, $y_k = B_{k+1}s_k$. By substituting into this equation, we obtain

$$y_k = B_k s_k + [\sigma v^T s_k] v. \quad (6.22)$$

Since the term in brackets is a scalar, we deduce that v must be a multiple of $y_k - B_k s_k$, that is, $v = \delta(y_k - B_k s_k)$ for some scalar δ . By substituting this form of v into (6.22), we obtain

$$(y_k - B_k s_k) = \sigma \delta^2 [s_k^T (y_k - B_k s_k)] (y_k - B_k s_k), \quad (6.23)$$

and it is clear that this equation is satisfied if (and only if) we choose the parameters δ and σ to be

$$\sigma = \text{sign} [s_k^T (y_k - B_k s_k)], \quad \delta = \pm |s_k^T (y_k - B_k s_k)|^{-1/2}.$$

Hence, we have shown that the only symmetric rank-1 updating formula that satisfies the secant equation is given by

$$(SR1) \quad B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \quad (6.24)$$

By applying the Sherman–Morrison formula (A.27), we obtain the corresponding update formula for the inverse Hessian approximation H_k :

$$(SR1) \quad H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}. \quad (6.25)$$

This derivation is so simple that the SR1 formula has been rediscovered a number of times.

It is easy to see that even if B_k is positive definite, B_{k+1} may not have the same property. (The same is, of course, true of H_k .) This observation was considered a major drawback

CHAPTER 8

Calculating Derivatives

Most algorithms for nonlinear optimization and nonlinear equations require knowledge of derivatives. Sometimes the derivatives are easy to calculate by hand, and it is reasonable to expect the user to provide code to compute them. In other cases, the functions are too complicated, so we look for ways to calculate or approximate the derivatives automatically. A number of interesting approaches are available, of which the most important are probably the following.

Finite Differencing. This technique has its roots in Taylor's theorem (see Chapter 2). By observing the change in function values in response to small perturbations of the unknowns

near a given point x , we can estimate the response to *infinitesimal* perturbations, that is, the derivatives. For instance, the partial derivative of a smooth function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with respect to the i th variable x_i can be approximated by the central-difference formula

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon},$$

where ϵ is a small positive scalar and e_i is the i th unit vector, that is, the vector whose elements are all 0 except for a 1 in the i th position.

Automatic Differentiation. This technique takes the view that the computer code for evaluating the function can be broken down into a composition of elementary arithmetic operations, to which the chain rule (one of the basic rules of calculus) can be applied. Some software tools for automatic differentiation (such as ADIFOR [25]) produce new code that calculates both function and derivative values. Other tools (such as ADOL-C [154]) keep a record of the elementary computations that take place while the function evaluation code for a given point x is executing on the computer. This information is processed to produce the derivatives at the same point x .

Symbolic Differentiation. In this technique, the algebraic specification for the function f is manipulated by symbolic manipulation tools to produce new algebraic expressions for each component of the gradient. Commonly used symbolic manipulation tools can be found in the packages Mathematica [311], Maple [304], and Macsyma [197].

In this chapter we discuss the first two approaches: finite differencing and automatic differentiation.

The usefulness of derivatives is not restricted to *algorithms* for optimization. Modelers in areas such as design optimization and economics are often interested in performing post-optimal *sensitivity analysis*, in which they determine the sensitivity of the optimum to small perturbations in the parameter or constraint values. Derivatives are also important in other areas such as nonlinear differential equations and simulation.

8.1 FINITE-DIFFERENCE DERIVATIVE APPROXIMATIONS

Finite differencing is an approach to the calculation of approximate derivatives whose motivation (like that of so many algorithms in optimization) comes from Taylor's theorem. Many software packages perform automatic calculation of finite differences whenever the user is unable or unwilling to supply code to calculate exact derivatives. Although they yield only approximate values for the derivatives, the results are adequate in many situations.

By definition, derivatives are a measure of the sensitivity of the function to infinitesimal changes in the values of the variables. Our approach in this section is to make small, *finite* perturbations in the values of x and examine the resulting *differences* in the function values.

By taking ratios of the function difference to variable difference, we obtain approximations to the derivatives.

APPROXIMATING THE GRADIENT

An approximation to the gradient vector $\nabla f(x)$ can be obtained by evaluating the function f at $(n + 1)$ points and performing some elementary arithmetic. We describe this technique, along with a more accurate variant that requires additional function evaluations.

A popular formula for approximating the partial derivative $\partial f / \partial x_i$ at a given point x is the *forward-difference*, or *one-sided-difference*, approximation, defined as

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}. \quad (8.1)$$

The gradient can be built up by simply applying this formula for $i = 1, 2, \dots, n$. This process requires evaluation of f at the point x as well as the n perturbed points $x + \epsilon e_i$, $i = 1, 2, \dots, n$: a total of $(n + 1)$ points.

The basis for the formula (8.1) is Taylor's theorem, Theorem 2.1 in Chapter 2. When f is twice continuously differentiable, we have

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p, \quad \text{some } t \in (0, 1) \quad (8.2)$$

(see (2.6)). If we choose L to be a bound on the size of $\|\nabla^2 f(\cdot)\|$ in the region of interest, it follows directly from this formula that the last term in this expression is bounded by $(L/2)\|p\|^2$, so that

$$\|f(x + p) - f(x) - \nabla f(x)^T p\| \leq (L/2)\|p\|^2. \quad (8.3)$$

We now choose the vector p to be ϵe_i , so that it represents a small change in the value of a single component of x (the i th component). For this p , we have that $\nabla f(x)^T p = \nabla f(x)^T e_i = \partial f / \partial x_i$, so by rearranging (8.3), we conclude that

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon} + \delta_\epsilon, \quad \text{where } |\delta_\epsilon| \leq (L/2)\epsilon. \quad (8.4)$$

We derive the forward-difference formula (8.1) by simply ignoring the error term δ_ϵ in this expression, which becomes smaller and smaller as ϵ approaches zero.

An important issue in implementing the formula (8.1) is the choice of the parameter ϵ . The error expression (8.4) suggests that we should choose ϵ as small as possible. Unfortunately, this expression ignores the roundoff errors that are introduced when the function f is evaluated on a real computer, in floating-point arithmetic. From our discussion in the Appendix (see (A.30) and (A.31)), we know that the quantity \mathbf{u} known as *unit roundoff*

is crucial: It is a bound on the relative error that is introduced whenever an arithmetic operation is performed on two floating-point numbers. (\mathbf{u} is about 1.1×10^{-16} in double-precision IEEE floating-point arithmetic.) The effect of these errors on the final computed value of f depends on the way in which f is computed. It could come from an arithmetic formula, or from a differential equation solver, with or without refinement.

As a rough estimate, let us assume simply that the relative error in the computed f is bounded by \mathbf{u} , so that the computed values of $f(x)$ and $f(x + \epsilon e_i)$ are related to the exact values in the following way:

$$\begin{aligned} |\text{comp}(f(x)) - f(x)| &\leq \mathbf{u}L_f, \\ |\text{comp}(f(x + \epsilon e_i)) - f(x + \epsilon e_i)| &\leq \mathbf{u}L_f, \end{aligned}$$

where $\text{comp}(\cdot)$ denotes the computed value, and L_f is a bound on the value of $|f(\cdot)|$ in the region of interest. If we use these computed values of f in place of the exact values in (8.4) and (8.1), we obtain an error that is bounded by

$$(L/2)\epsilon + 2\mathbf{u}L_f/\epsilon. \quad (8.5)$$

Naturally, we would like to choose ϵ to make this error as small as possible; it is easy to see that the minimizing value is

$$\epsilon^2 = \frac{4L_f\mathbf{u}}{L}.$$

If we assume that the problem is well scaled, then the ratio L_f/L (the ratio of function values to second derivative values) does not exceed a modest size. We can conclude that the following choice of ϵ is fairly close to optimal:

$$\epsilon = \sqrt{\mathbf{u}}. \quad (8.6)$$

(In fact, this value is used in many of the optimization software packages that use finite differencing as an option for estimating derivatives.) For this value of ϵ , we have from (8.5) that the total error in the forward-difference approximation is fairly close to $\sqrt{\mathbf{u}}$.

A more accurate approximation to the derivative can be obtained by using the *central-difference* formula, defined as

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon}. \quad (8.7)$$

As we show below, this approximation is more accurate than the forward-difference approximation (8.1). It is also about twice as expensive, since we need to evaluate f at the points x and $x \pm \epsilon e_i$, $i = 1, 2, \dots, n$: a total of $2n + 1$ points.

The basis for the central difference approximation is again Taylor's theorem. When the second derivatives of f exist and are Lipschitz continuous, we have from (8.2) that

$$\begin{aligned} f(x + p) &= f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p \quad \text{for some } t \in (0, 1) \\ &= f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x) p + O(\|p\|^3). \end{aligned} \quad (8.8)$$

By setting $p = \epsilon e_i$ and $p = -\epsilon e_i$, respectively, we obtain

$$\begin{aligned} f(x + \epsilon e_i) &= f(x) + \epsilon \frac{\partial f}{\partial x_i} + \frac{1}{2} \epsilon^2 \frac{\partial^2 f}{\partial x_i^2} + O(\epsilon^3), \\ f(x - \epsilon e_i) &= f(x) - \epsilon \frac{\partial f}{\partial x_i} + \frac{1}{2} \epsilon^2 \frac{\partial^2 f}{\partial x_i^2} + O(\epsilon^3). \end{aligned}$$

(Note that the final error terms in these two expressions are generally not the same, but they are both bounded by some multiple of ϵ^3 .) By subtracting the second equation from the first and dividing by 2ϵ , we obtain the expression

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} + O(\epsilon^2).$$

We see from this expression that the error is $O(\epsilon^2)$, as compared to the $O(\epsilon)$ error in the forward-difference formula (8.1). However, when we take evaluation error in f into account, the accuracy that can be achieved in practice is less impressive; the same assumptions that were used to derive (8.6) lead to an optimal choice of ϵ of about $\mathbf{u}^{1/3}$ and an error of about $\mathbf{u}^{2/3}$. In some situations, the extra few digits of accuracy may improve the performance of the algorithm enough to make the extra expense worthwhile.

APPROXIMATING A SPARSE JACOBIAN

Consider now the case of a vector function $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$, such as the residual vector that we consider in Chapter 10 or the system of nonlinear equations from Chapter 11. The matrix $J(x)$ of first derivatives for this function is defined as follows:

$$J(x) = \left[\frac{\partial r_j}{\partial x_i} \right]_{\substack{j=1,2,\dots,m \\ i=1,2,\dots,n}} = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}, \quad (8.9)$$

where $r_j, j = 1, 2, \dots, m$ are the components of r . The techniques described in the previous

section can be used to evaluate the full Jacobian $J(x)$ one column at a time. When r is twice continuously differentiable, we can use Taylor's theorem to deduce that

$$\|r(x + p) - r(x) - J(x)p\| \leq (L/2)\|p\|^2, \quad (8.10)$$

where L is a Lipschitz constant for J in the region of interest. If we require an approximation to the Jacobian–vector product $J(x)p$ for a given vector p (as is the case with inexact Newton methods for nonlinear systems of equations; see Section 11.1), this expression immediately suggests choosing a small nonzero ϵ and setting

$$J(x)p \approx \frac{r(x + \epsilon p) - r(x)}{\epsilon}, \quad (8.11)$$

an approximation that is accurate to $O(\epsilon)$. A two-sided approximation can be derived from the formula (8.7).

If an approximation to the full Jacobian $J(x)$ is required, we can compute it a column at a time, analogously to (8.1), by setting set $p = \epsilon e_i$ in (8.10) to derive the following estimate of the i th column:

$$\frac{\partial r}{\partial x_i}(x) \approx \frac{r(x + \epsilon e_i) - r(x)}{\epsilon}. \quad (8.12)$$

A full Jacobian estimate can be obtained at a cost of $n + 1$ evaluations of the function r . When the Jacobian is sparse, however, we can often obtain the estimate at a much lower cost, sometimes just three or four evaluations of r . The key is to estimate a number of different columns of the Jacobian simultaneously, by judicious choices of the perturbation vector p in (8.10).

We illustrate the technique with a simple example. Consider the function $r : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined by

$$r(x) = \begin{bmatrix} 2(x_2^3 - x_1^2) \\ 3(x_2^3 - x_1^2) + 2(x_3^3 - x_2^2) \\ 3(x_3^3 - x_2^2) + 2(x_4^3 - x_3^2) \\ \vdots \\ 3(x_n^3 - x_{n-1}^2) \end{bmatrix}. \quad (8.13)$$

Each component of r depends on just two or three components of x , so that each row of the Jacobian contains only two or three nonzero elements. For the case of $n = 6$, the Jacobian

has the following structure:

$$\begin{bmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & \times & \times \end{bmatrix}, \quad (8.14)$$

where each cross represents a nonzero element, with zeros represented by a blank space.

Staying for the moment with the case $n = 6$, suppose that we wish to compute a finite-difference approximation to the Jacobian. (Of course, it is easy to calculate this particular Jacobian by hand, but there are complicated functions with similar structure for which hand calculation is more difficult.) A perturbation $p = \epsilon e_1$ to the first component of x will affect only the first and second components of r . The remaining components will be unchanged, so that the right-hand-side of formula (8.12) will correctly evaluate to zero in the components 3, 4, 5, 6. It is wasteful, however, to reevaluate these components of r when we know in advance that their values are not affected by the perturbation. Instead, we look for a way to modify the perturbation vector so that it does not have any further effect on components 1 and 2, but *does* produce a change in some of the components 3, 4, 5, 6, which we can then use as the basis of a finite-difference estimate for some *other* column of the Jacobian. It is not hard to see that the additional perturbation ϵe_4 has the desired property: It alters the 3rd, 4th, and 5th elements of r , but leaves the 1st and 2nd elements unchanged. The changes in r as a result of the perturbations ϵe_1 and ϵe_4 do not interfere with each other.

To express this discussion in mathematical terms, we set

$$p = \epsilon(e_1 + e_4),$$

and note that

$$r(x + p)_{1,2} = r(x + \epsilon(e_1 + e_4))_{1,2} = r(x + \epsilon e_1)_{1,2} \quad (8.15)$$

(where the notation $[\cdot]_{1,2}$ denotes the subvector consisting of the first and second elements), while

$$r(x + p)_{3,4,5} = r(x + \epsilon(e_1 + e_4))_{3,4,5} = r(x + \epsilon e_4)_{3,4,5}. \quad (8.16)$$

By substituting (8.15) into (8.10), we obtain

$$r(x + p)_{1,2} = r(x)_{1,2} + \epsilon[J(x)e_1]_{1,2} + O(\epsilon^2).$$

By rearranging this expression, we obtain the following difference formula for estimating the (1, 1) and (2, 1) elements of the Jacobian matrix:

$$\begin{bmatrix} \frac{\partial r_1}{\partial x_1}(x) \\ \frac{\partial r_2}{\partial x_1}(x) \end{bmatrix} = [J(x)e_1]_{1,2} \approx \frac{r(x+p)_{1,2} - r(x)_{1,2}}{\epsilon}. \quad (8.17)$$

A similar argument shows that the nonzero elements of the fourth column of the Jacobian can be estimated by substituting (8.16) into (8.10); we obtain

$$\begin{bmatrix} \frac{\partial r_4}{\partial x_3}(x) \\ \frac{\partial r_4}{\partial x_4}(x) \\ \frac{\partial r_4}{\partial x_5}(x) \end{bmatrix} = [J(x)e_4]_{3,4,5} \approx \frac{r(x+p)_{3,4,5} - r(x)_{3,4,5}}{\epsilon}. \quad (8.18)$$

To summarize: We have been able to estimate *two* columns of the Jacobian $J(x)$ by evaluating the function r at the single extra point $x + \epsilon(e_1 + e_4)$.

We can approximate the remainder of $J(x)$ in an economical manner as well. Columns 2 and 5 can be approximated by choosing $p = \epsilon(e_2 + e_5)$, while we can use $p = \epsilon(e_3 + e_6)$ to approximate columns 3 and 6. In total, we need 3 evaluations of the function r (after the initial evaluation at x) to estimate the entire Jacobian matrix.

In fact, for *any* choice of n in (8.13) (no matter how large), three extra evaluations of r are sufficient to approximate the entire Jacobian. The corresponding choices of perturbation vectors p are

$$p = \epsilon(e_1 + e_4 + e_7 + e_{10} + \cdots),$$

$$p = \epsilon(e_2 + e_5 + e_8 + e_{11} + \cdots),$$

$$p = \epsilon(e_3 + e_6 + e_9 + e_{12} + \cdots).$$

In the first of these vectors, the nonzero components are chosen so that no two of the columns 1, 4, 7, \dots have a nonzero element in the same row. The same property holds for the other two vectors and, in fact, points the way to the criterion that we can apply to general problems to decide on a valid set of perturbation vectors.

Algorithms for choosing the perturbation vectors can be expressed conveniently in the language of graphs and graph coloring. For any function $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$, we can construct a *column incidence graph* G with n nodes by drawing an arc between nodes i and k if there is some component of r that depends on both x_i and x_k . In other words, the i th and k th columns of the Jacobian $J(x)$ each have a nonzero element in some row j , for some $j = 1, 2, \dots, m$ and some value of x . (The intersection graph for the function defined in

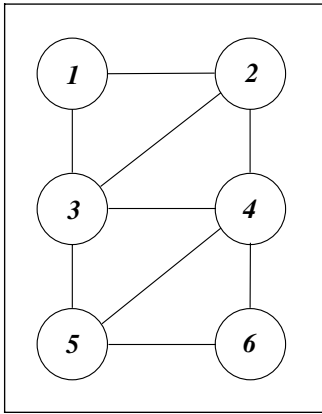


Figure 8.1
Column incidence graph for $r(x)$ defined in (8.13).

(8.13), with $n = 6$, is shown in Figure 8.1.) We now assign each node a “color” according to the following rule: Two nodes can have the same color if there is no arc that connects them. Finally, we choose one perturbation vector corresponding to each color: If nodes i_1, i_2, \dots, i_ℓ have the same color, the corresponding p is $\epsilon(e_{i_1} + e_{i_2} + \dots + e_{i_\ell})$.

Usually, there are many ways to assign colors to the n nodes in the graph in a way that satisfies the required condition. The simplest way is just to assign each node a different color, but since that scheme produces n perturbation vectors, it is usually not the most efficient approach. It is generally very difficult to find the coloring scheme that uses the fewest possible colors, but there are simple algorithms that do a good job of finding a near-optimal coloring at low cost. Curtis, Powell, and Reid [83] and Coleman and Moré [68] provide descriptions of some methods and performance comparisons. Newsam and Ramsdell [227] show that by considering a more general class of perturbation vectors p , it is possible to evaluate the full Jacobian using no more than n_z evaluations of r (in addition to the evaluation at the point x), where n_z is the maximum number of nonzeros in each row of $J(x)$.

For some functions r with well-studied structures (those that arise from discretizations of differential operators, or those that give rise to banded Jacobians, as in the example above), optimal coloring schemes are known. For the tridiagonal Jacobian of (8.14) and its associated graph in Figure 8.1, the scheme with three colors is optimal.

APPROXIMATING THE HESSIAN

In some situations, the user may be able to provide a routine to calculate the gradient $\nabla f(x)$ but not the Hessian $\nabla^2 f(x)$. We can obtain the Hessian by applying the techniques described above for the vector function r to the gradient ∇f . By using the graph coloring techniques discussed above, sparse Hessians often can be approximated in this manner by using considerably fewer than n perturbation vectors. This approach ignores symmetry of the Hessian, and will usually produce a nonsymmetric approximation. We can recover

symmetry by adding the approximation to its transpose and dividing the result by 2. Alternative differencing approaches that take symmetry of $\nabla^2 f(x)$ explicitly into account are discussed below.

Some important algorithms—most notably the Newton–CG methods described in Chapter 7—do not require knowledge of the full Hessian. Instead, each iteration requires us to supply the Hessian–vector product $\nabla^2 f(x)p$, for a given vector p . We can obtain an approximation to this matrix–vector product by appealing once again to Taylor’s theorem. When second derivatives of f exist and are Lipschitz continuous near x , we have

$$\nabla f(x + \epsilon p) = \nabla f(x) + \epsilon \nabla^2 f(x)p + O(\epsilon^2), \quad (8.19)$$

so that

$$\nabla^2 f(x)p \approx \frac{\nabla f(x + \epsilon p) - \nabla f(x)}{\epsilon} \quad (8.20)$$

(see also (7.10)). The approximation error is $O(\epsilon)$, and the cost of obtaining the approximation is a single gradient evaluation at the point $x + \epsilon p$. The formula (8.20) corresponds to the forward-difference approximation (8.1). A central-difference formula like (8.7) can be derived by evaluating $\nabla f(x - \epsilon p)$ as well.

For the case in which even gradients are not available, we can use Taylor’s theorem once again to derive formulae for approximating the Hessian that use only function values. The main tool is the formula (8.8): By substituting the vectors $p = \epsilon e_i$, $p = \epsilon e_j$, and $p = \epsilon(e_i + e_j)$ into this formula and combining the results appropriately, we obtain

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x + \epsilon e_i + \epsilon e_j) - f(x + \epsilon e_i) - f(x + \epsilon e_j) + f(x)}{\epsilon^2} + O(\epsilon). \quad (8.21)$$

If we wished to approximate every element of the Hessian with this formula, then we would need to evaluate f at $x + \epsilon(e_i + e_j)$ for all possible i and j (a total of $n(n + 1)/2$ points) as well as at the n points $x + \epsilon e_i$, $i = 1, 2, \dots, n$. If the Hessian is sparse, we can, of course, reduce this operation count by skipping the evaluation whenever we know the element $\partial^2 f / \partial x_i \partial x_j$ to be zero.

APPROXIMATING A SPARSE HESSIAN

We noted above that a Hessian approximation can be obtained by applying finite-difference Jacobian estimation techniques to the gradient ∇f , treated as a vector function. We now show how symmetry of the Hessian $\nabla^2 f$ can be used to reduce the number of perturbation vectors p needed to obtain a complete approximation, when the Hessian is sparse. The key observation is that, because of symmetry, any estimate of the element $[\nabla^2 f(x)]_{i,j} = \partial^2 f(x) / \partial x_i \partial x_j$ is also an estimate of its symmetric counterpart $[\nabla^2 f(x)]_{j,i}$.

We illustrate the point with the simple function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ defined by

$$f(x) = x_1 \sum_{i=1}^n i^2 x_i^2. \quad (8.22)$$

It is easy to show that the Hessian $\nabla^2 f$ has the “arrowhead” structure depicted below, for the case of $n = 6$:

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ \times & & \times & & & \\ \times & & & \times & & \\ \times & & & & \times & \\ \times & & & & & \times \end{bmatrix}. \quad (8.23)$$

If we were to construct the intersection graph for the function ∇f (analogous to Figure 8.1), we would find that every node is connected to every other node, for the simple reason that row 1 has a nonzero in every column. According to the rule for coloring the graph, then, we would have to assign a different color to every node, which implies that we would need to evaluate ∇f at the $n + 1$ points x and $x + \epsilon e_i$ for $i = 1, 2, \dots, n$.

We can construct a much more efficient scheme by taking the symmetry into account. Suppose we first use the perturbation vector $p = \epsilon e_1$ to estimate the first column of $\nabla^2 f(x)$. Because of symmetry, the same estimates apply to the first *row* of $\nabla^2 f$. From (8.23), we see that all that remains is to find the diagonal elements $\nabla^2 f(x)_{22}, \nabla^2 f(x)_{33}, \dots, \nabla^2 f(x)_{66}$. The intersection graph for these remaining elements is completely disconnected, so we can assign them all the same color and choose the corresponding perturbation vector to be

$$p = \epsilon(e_2 + e_3 + \dots + e_6) = \epsilon(0, 1, 1, 1, 1, 1)^T. \quad (8.24)$$

Note that the second component of ∇f is not affected by the perturbations in components 3, 4, 5, 6 of the unknown vector, while the third component of ∇f is not affected by perturbations in components 2, 4, 5, 6 of x , and so on. As in (8.15) and (8.16), we have for each component i that

$$\nabla f(x + p)_i = \nabla f(x + \epsilon(e_2 + e_3 + \dots + e_6))_i = \nabla f(x + \epsilon e_i)_i.$$

By applying the forward-difference formula (8.1) to each of these individual components, we then obtain

$$\frac{\partial^2 f}{\partial x_i^2}(x) \approx \frac{\nabla f(x + \epsilon e_i)_i - \nabla f(x)_i}{\epsilon} = \frac{\nabla f(x + \epsilon p)_i - \nabla f(x)_i}{\epsilon}, \quad i = 2, 3, \dots, 6.$$

By exploiting symmetry, we have been able to estimate the entire Hessian by evaluating ∇f only at x and two other points.

Again, graph-coloring techniques can be used to choose the perturbation vectors p economically. We use the *adjacency graph* in place of the intersection graph described earlier. The adjacency graph has n nodes, with arcs connecting nodes i and k whenever $i \neq k$ and $\partial^2 f(x)/(\partial x_i \partial x_k) \neq 0$ for some x . The requirements on the coloring scheme are a little more complicated than before, however. We require not only that connected nodes have different colors, but also that any path of length 3 through the graph contain at least three colors. In other words, if there exist nodes i_1, i_2, i_3, i_4 in the graph that are connected by arcs (i_1, i_2) , (i_2, i_3) , and (i_3, i_4) , then at least three different colors must be used in coloring these four nodes. See Coleman and Moré [69] for an explanation of this rule and for algorithms to compute valid colorings. The perturbation vectors are constructed as before: Whenever the nodes i_1, i_2, \dots, i_ℓ have the same color, we set the corresponding perturbation vector to be $p = \epsilon(e_{i_1} + e_{i_2} + \dots + e_{i_\ell})$.

8.2 AUTOMATIC DIFFERENTIATION

Automatic differentiation is the generic name for techniques that use the computational representation of a function to produce analytic values for the derivatives. Some techniques produce code for the derivatives at a general point x by manipulating the function code directly. Other techniques keep a record of the computations made during the evaluation of the function at a specific point x and then review this information to produce a set of derivatives at x .

Automatic differentiation techniques are founded on the observation that any function, no matter how complicated, is evaluated by performing a sequence of simple elementary operations involving just one or two arguments at a time. Two-argument operations include addition, multiplication, division, and the power operation a^b . Examples of single-argument operations include the trigonometric, exponential, and logarithmic functions. Another common ingredient of the various automatic differentiation tools is their use of the *chain rule*. This is the well-known rule from elementary calculus that says that if h is a function of the vector $y \in \mathbb{R}^m$, which is in turn a function of the vector $x \in \mathbb{R}^n$, we can write the derivative of h with respect to x as follows:

$$\nabla_x h(y(x)) = \sum_{i=1}^m \frac{\partial h}{\partial y_i} \nabla y_i(x). \quad (8.25)$$

See Appendix A for further details.

There are two basic modes of automatic differentiation: the *forward* and *reverse* modes. The difference between them can be illustrated by a simple example. We work through such

CHAPTER 9

Derivative-Free Optimization

Many practical applications require the optimization of functions whose derivatives are not available. Problems of this kind can be solved, in principle, by approximating the gradient (and possibly the Hessian) using finite differences (see Chapter 8), and using these approximate gradients within the algorithms described in earlier chapters. Even though this finite-difference approach is effective in some applications, it cannot be regarded a general-purpose technique for derivative-free optimization because the number of function evaluations required can be excessive and the approach can be unreliable in the presence of noise. (For the purposes of this chapter we define *noise* to be inaccuracy in the function evaluation.) Because of these shortcomings, various algorithms have been developed that

do not attempt to approximate the gradient. Rather, they use the function values at a set of sample points to determine a new iterate by some other means.

Derivative-free optimization (DFO) algorithms differ in the way they use the sampled function values to determine the new iterate. One class of methods constructs a linear or quadratic model of the objective function and defines the next iterate by seeking to minimize this model inside a trust region. We pay particular attention to these model-based approaches because they are related to the unconstrained minimization methods described in earlier chapters. Other widely used DFO methods include the simplex-reflection method of Nelder and Mead, pattern-search methods, conjugate-direction methods, and simulated annealing. In this chapter we briefly discuss these methods, with the exception of simulated annealing, which is a nondeterministic approach and has little in common with the other techniques discussed in this book.

Derivative-free optimization methods are not as well developed as gradient-based methods; current algorithms are effective only for small problems. Although most DFO methods have been adapted to handle simple types of constraints, such as bounds, the efficient treatment of general constraints is still the subject of investigation. Consequently, we limit our discussion to the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x). \quad (9.1)$$

Problems in which derivatives are not available arise often in practice. The evaluation of $f(x)$ can, for example, be the result of an experimental measurement or a stochastic simulation, with the underlying analytic form of f unknown. Even if the objective function f is known in analytic form, coding its derivatives may be time consuming or impractical. Automatic differentiation tools (Chapter 8) may not be applicable if $f(x)$ is provided only in the form of binary computer code. Even when the source code is available, these tools cannot be applied if the code is written in a combination of languages.

Methods for derivative-free optimization are often used (with mixed success) to minimize problems with nondifferentiable functions or to try to locate the global minimizer of a function. Since we do not treat nonsmooth optimization or global optimization in this book, we will restrict our attention to smooth problems in which f has a continuous derivative. We do, however, discuss the effects of noise in Sections 9.1 and 9.6.

9.1 FINITE DIFFERENCES AND NOISE

As mentioned above, an obvious DFO approach is to estimate the gradient by using finite differences and then employ a gradient-based method. This approach is sometimes successful and should always be considered, but the finite-difference estimates can be inaccurate when the objective function contains noise. We quantify the effect of noise in this section.

Noise can arise in function evaluations for various reasons. If $f(x)$ depends on a stochastic simulation, there will be a random error in the evaluated function because of the

finite number of trials in the simulation. When a differential equation solver or some other complex numerical procedure is needed to calculate f , small but nonzero error tolerances that are used during the calculations will produce noise in the value of f .

In many applications, then, the objective function f has the form

$$f(x) = h(x) + \phi(x), \quad (9.2)$$

where h is a smooth function and ϕ represents the noise. Note that we have written ϕ to be a function of x but in practice it need not be. For instance, if the evaluation of f depends on a simulation, the value of ϕ will generally differ at each evaluation, even at the same x . The form (9.2) is, however, useful for illustrating some of the difficulties caused by noise in gradient estimates and for developing algorithms for derivative-free optimization.

Given a difference interval ϵ , recall that the centered finite-difference approximation (8.7) to the gradient of f at x is defined as follows:

$$\nabla_{\epsilon} f(x) = \left[\frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} \right]_{i=1,2,\dots,n}, \quad (9.3)$$

where e_i is the i th unit vector (the vector whose only nonzero element is a 1 in the i th position). We wish to relate $\nabla_{\epsilon} f(x)$ to the gradient of the underlying smooth function $h(x)$, as a function of ϵ and the noise level. For this purpose we define the noise level η to be the largest value of ϕ in a box of edge length 2ϵ centered at x , that is,

$$\eta(x; \epsilon) = \sup_{\|z-x\|_{\infty} \leq \epsilon} |\phi(z)|. \quad (9.4)$$

By applying to the central difference formula (9.3) the argument that led to (8.5), we can establish the following result.

Lemma 9.1.

Suppose that $\nabla^2 h$ is Lipschitz continuous in a neighborhood of the box $\{z \mid \|z-x\|_{\infty} \leq \epsilon\}$ with Lipschitz constant L_h . Then we have

$$\|\nabla_{\epsilon} f(x) - \nabla h(x)\|_{\infty} \leq L_h \epsilon^2 + \frac{\eta(x; \epsilon)}{\epsilon}. \quad (9.5)$$

Thus the error in the approximation (9.3) comes from both the intrinsic finite difference approximation error (the $O(\epsilon^2)$ term) and the noise (the $\eta(x; \epsilon)/\epsilon$ term). If the noise dominates the difference interval ϵ , we cannot expect any accuracy at all in $\nabla_{\epsilon} f(x)$, so it will only be pure luck if $-\nabla_{\epsilon} f(x)$ turns out to be a direction of descent for f .

Instead of computing a tight cluster of function values around the current iterate, as required by a finite-difference approximation to the gradient, it may be preferable to separate these points more widely and use them to construct a model of the objective function. This

approach, which we consider in the next section and in Section 9.6, may be more robust to the presence of noise.

9.2 MODEL-BASED METHODS

Some of the most effective algorithms for unconstrained optimization described in the previous chapters compute steps by minimizing a quadratic model of the objective function f . The model is formed by using function and derivative information at the current iterate. When derivatives are not available, we may define the model m_k as the quadratic function that interpolates f at a set of appropriately chosen sample points. Since such a model is usually nonconvex, the model-based methods discussed in this chapter use a trust-region approach to compute the step.

Suppose that at the current iterate x_k we have a set of sample points $Y = \{y^1, y^2, \dots, y^q\}$, with $y^i \in \mathbb{R}^n$, $i = 1, 2, \dots, q$. We assume that x_k is an element of this set and that no point in Y has a lower function value than x_k . We wish to construct a quadratic model of the form

$$m_k(x_k + p) = c + g^T p + \frac{1}{2} p^T G p. \quad (9.6)$$

We cannot define $g = \nabla f(x_k)$ and $G = \nabla^2 f(x_k)$ because these derivatives are not available. Instead, we determine the scalar c , the vector $g \in \mathbb{R}^n$, and the symmetric matrix $G \in \mathbb{R}^{n \times n}$ by imposing the interpolation conditions

$$m_k(y^l) = f(y^l), \quad l = 1, 2, \dots, q. \quad (9.7)$$

Since there are $\frac{1}{2}(n+1)(n+2)$ coefficients in the model (9.6) (that is, the components of c , g , and G , taking into account the symmetry of G), the interpolation conditions (9.7) determine m_k uniquely only if

$$q = \frac{1}{2}(n+1)(n+2). \quad (9.8)$$

In this case, (9.7) can be written as a square linear system of equations in the coefficients of the model. If we choose the interpolation points y^1, y^2, \dots, y^q so that this linear system is nonsingular, the model m_k will be uniquely determined.

Once m_k has been formed, we compute a step p by approximately solving the trust-region subproblem

$$\min_p m_k(x_k + p), \quad \text{subject to } \|p\|_2 \leq \Delta, \quad (9.9)$$

for some trust-region radius $\Delta > 0$. We can use one of the techniques described in Chapter 4 to solve this subproblem. If $x_k + p$ gives a sufficient reduction in the objective function,

9.5 NELDER–MEAD METHOD

The Nelder–Mead simplex-reflection method has been a popular DFO method since its introduction in 1965 [223]. It takes its name from the fact that at any stage of the algorithm, we keep track of $n + 1$ points of interest in \mathbb{R}^n , whose convex hull forms a simplex. (The method has nothing to do with the simplex method for linear programming discussed in Chapter 13.) Given a simplex S with vertices $\{z_1, z_2, \dots, z_{n+1}\}$, we can define an associated matrix $V(S)$ by taking the n edges along V from one of its vertices (z_1 , say), as follows:

$$V(S) = [z_2 - z_1, z_3 - z_1, \dots, z_{n+1} - z_1].$$

The simplex is said to be *nondegenerate* or *nonsingular* if V is a nonsingular matrix. (For example, a simplex in \mathbb{R}^3 is nondegenerate if its four vertices are not coplanar.)

In a single iteration of the Nelder–Mead algorithm, we seek to remove the vertex with the worst function value and replace it with another point with a better value. The new point is obtained by reflecting, expanding, or contracting the simplex along the line joining the worst vertex with the centroid of the remaining vertices. If we cannot find a better point in this manner, we retain only the vertex with the *best* function value, and we shrink the simplex by moving all other vertices toward this value.

We specify a single step of the algorithm after some defining some notation. The $n + 1$ vertices of the current simplex are denoted by $\{x_1, x_2, \dots, x_{n+1}\}$, where we choose the ordering so that

$$f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1}).$$

The centroid of the best n points is denoted by

$$\bar{x} = \sum_{i=1}^n x_i.$$

Points along the line joining \bar{x} and the “worst” vertex x_{n+1} are denoted by

$$\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x}).$$

Procedure 9.5 (One Step of Nelder–Mead Simplex).

Compute the reflection point $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$;

if $f(x_1) \leq f_{-1} < f(x_n)$

(* reflected point is neither best nor worst in the new simplex *)

replace x_{n+1} by $\bar{x}(-1)$ and go to next iteration;

else if $f_{-1} < f(x_1)$


```

(* reflected point is better than the current best; try to
   go farther along this direction *)
Compute the expansion point  $\bar{x}(-2)$  and evaluate  $f_{-2} = f(\bar{x}(-2))$ ;
if  $f_{-2} < f_{-1}$ 
    replace  $x_{n+1}$  by  $x_{-2}$  and go to next iteration;
else
    replace  $x_{n+1}$  by  $x_{-1}$  and go to next iteration;
else if  $f_{-1} \geq f(x_n)$ 
    (* reflected point is still worse than  $x_n$ ; contract *)
    if  $f(x_n) \leq f_{-1} < f(x_{n+1})$ 
        (* try to perform “outside” contraction *)
        evaluate  $f_{-1/2} = \bar{x}(-1/2)$ ;
        if  $f_{-1/2} \leq f_{-1}$ 
            replace  $x_{n+1}$  by  $x_{-1/2}$  and go to next iteration;
        else
            (* try to perform “inside” contraction *)
            evaluate  $f_{1/2} = \bar{x}(1/2)$ ;
            if  $f_{1/2} < f_{n+1}$ 
                replace  $x_{n+1}$  by  $x_{1/2}$  and go to next iteration;
    (* neither outside nor inside contraction was acceptable;
       shrink the simplex toward  $x_1$  *)
    replace  $x_i \leftarrow (1/2)(x_1 + x_i)$  for  $i = 2, 3, \dots, n + 1$ ;

```

Procedure 9.5 is illustrated on a three-dimensional example in Figure 9.4. The worst current vertex is x_3 , and the possible replacement points are $\bar{x}(-1)$, $\bar{x}(-2)$, $\bar{x}(-\frac{1}{2})$, $\bar{x}(\frac{1}{2})$. If none of the replacement points proves to be satisfactory, the simplex is shrunk to the smaller triangle indicated by the dotted line, which retains the best vertex x_1 . The scalars t used in defining the candidate points $\bar{x}(t)$ have been assigned the specific (and standard) values -1 , -2 , $-\frac{1}{2}$, and $\frac{1}{2}$ in our description above. Different choices are also possible, subject to certain restrictions.

Practical performance of the Nelder–Mead algorithm is often reasonable, though stagnation has been observed to occur at nonoptimal points. Restarting can be used when stagnation is detected; see Kelley [178]. Note that unless the final shrinkage step is performed, the average function value

$$\frac{1}{n+1} \sum_{i=1}^{n+1} f(x_i) \quad (9.29)$$

will decrease at each step. When f is convex, even the shrinkage step is guaranteed not to increase the average function value.

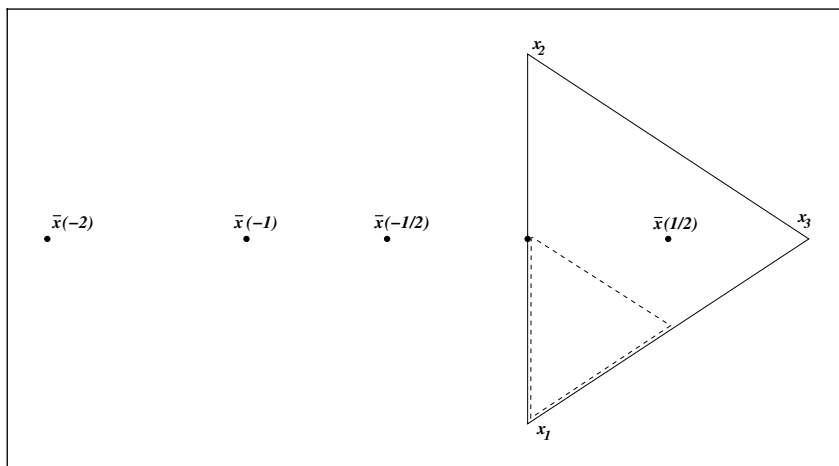


Figure 9.4 One step of the Nelder–Mead simplex method in \mathbb{R}^3 , showing current simplex (solid triangle with vertices x_1, x_2, x_3), reflection point $\bar{x}(-1)$, expansion point $\bar{x}(-2)$, inside contraction point $\bar{x}(\frac{1}{2})$, outside contraction point $\bar{x}(-\frac{1}{2})$, and shrunken simplex (dotted triangle).

A limited amount of convergence theory has been developed for the Nelder–Mead method in recent years; see, for example, Kelley [179] and Lagarias et al. [186].

9.6 IMPLICIT FILTERING

We now describe an algorithm designed for functions whose evaluations are modeled by (9.2), where h is smooth. This *implicit filtering* approach is, in its simplest form, a variant of the steepest descent algorithm with line search discussed in Chapter 3, in which the gradient ∇f_k is replaced by a finite difference estimate such as (9.3), with a difference parameter ϵ that may not be particularly small.

Implicit filtering works best on functions for which the noise level decreases as the iterates approach a solution. This situation may occur when we have control over the noise level, as is the case when f is obtained by solving a differential equation to a user-specified tolerance, or by running a stochastic simulation for a user-specified number of trials (where an increase in the number of trials usually produces a decrease in the noise). The implicit filtering algorithm decreases ϵ systematically (but, one hopes, not as rapidly as the decay in error) so as to maintain reasonable accuracy in $\nabla_\epsilon f(x)$, given the noise level at the current value of x . For each value of ϵ , it performs an inner loop that is simply an Armijo line search using the search direction $-\nabla_\epsilon f(x)$. If the inner loop is unable to find a satisfactory step length after backtracking at least a_{\max} times, we return to the outer loop, choose a smaller value of ϵ , and repeat. A formal specification follows.

CHAPTER *11*

Nonlinear Equations

In many applications we do not need to optimize an objective function explicitly, but rather to find values of the variables in a model that satisfy a number of given relationships. When these relationships take the form of n equalities—the same number of equality conditions as variables in the model—the problem is one of solving a system of *nonlinear equations*. We write this problem mathematically as

$$r(x) = 0, \tag{11.1}$$

where $r : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a vector function, that is,

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_n(x) \end{bmatrix}.$$

In this chapter, we assume that each function $r_i : \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1, 2, \dots, n$, is smooth. A vector x^* for which (11.1) is satisfied is called a *solution* or *root* of the nonlinear equations. A simple example is the system

$$r(x) = \begin{bmatrix} x_2^2 - 1 \\ \sin x_1 - x_2 \end{bmatrix} = 0,$$

which is a system of $n = 2$ equations with infinitely many solutions, two of which are $x^* = (3\pi/2, -1)^T$ and $x^* = (\pi/2, 1)^T$. In general, the system (11.1) may have no solutions, a unique solution, or many solutions.

The techniques for solving nonlinear equations overlap in their motivation, analysis, and implementation with optimization techniques discussed in earlier chapters. In both optimization and nonlinear equations, Newton's method lies at the heart of many important algorithms. Features such as line searches, trust regions, and inexact solution of the linear algebra subproblems at each iteration are important in both areas, as are other issues such as derivative evaluation and global convergence.

Because some important algorithms for nonlinear equations proceed by minimizing a sum of squares of the equations, that is,

$$\min_x \sum_{i=1}^n r_i^2(x),$$

there are particularly close connections with the nonlinear least-squares problem discussed in Chapter 10. The differences are that in nonlinear equations, the number of equations *equals* the number of variables (instead of exceeding the number of variables, as is typically the case in Chapter 10), and that we expect all equations to be satisfied at the solution, rather than just minimizing the sum of squares. This point is important because the nonlinear equations may represent physical or economic constraints such as conservation laws or consistency principles, which must hold exactly in order for the solution to be meaningful.

Many applications require us to solve a sequence of closely related nonlinear systems, as in the following example.

□ EXAMPLE 11.1 (RHEINBOLDT; SEE [212])

An interesting problem in control is to analyze the stability of an aircraft in response to the commands of the pilot. The following is a simplified model based on force-balance equations, in which gravity terms have been neglected.

The equilibrium equations for a particular aircraft are given by a system of 5 equations in 8 unknowns of the form

$$F(x) \equiv Ax + \phi(x) = 0, \quad (11.2)$$

where $F : \mathbb{R}^8 \rightarrow \mathbb{R}^5$, the matrix A is given by

$$A = \begin{bmatrix} -3.933 & 0.107 & 0.126 & 0 & -9.99 & 0 & -45.83 & -7.64 \\ 0 & -0.987 & 0 & -22.95 & 0 & -28.37 & 0 & 0 \\ 0.002 & 0 & -0.235 & 0 & 5.67 & 0 & -0.921 & -6.51 \\ 0 & 1.0 & 0 & -1.0 & 0 & -0.168 & 0 & 0 \\ 0 & 0 & -1.0 & 0 & -0.196 & 0 & -0.0071 & 0 \end{bmatrix},$$

and the nonlinear part is defined by

$$\phi(x) = \begin{bmatrix} -0.727x_2x_3 + 8.39x_3x_4 - 684.4x_4x_5 + 63.5x_4x_2 \\ 0.949x_1x_3 + 0.173x_1x_5 \\ -0.716x_1x_2 - 1.578x_1x_4 + 1.132x_4x_2 \\ -x_1x_5 \\ x_1x_4 \end{bmatrix}.$$

The first three variables x_1, x_2, x_3 , represent the rates of roll, pitch, and yaw, respectively, while x_4 is the incremental angle of attack and x_5 the sideslip angle. The last three variables x_6, x_7, x_8 are the controls; they represent the deflections of the elevator, aileron, and rudder, respectively.

For a given choice of the control variables x_6, x_7, x_8 we obtain a system of 5 equations and 5 unknowns. If we wish to study the behavior of the aircraft as the controls are changed, we need to solve a system of nonlinear equations with unknowns x_1, x_2, \dots, x_5 for each setting of the controls. □

Despite the many similarities between nonlinear equations and unconstrained and least-squares optimization algorithms, there are also some important differences. To obtain quadratic convergence in optimization we require second derivatives of the objective function, whereas knowledge of the first derivatives is sufficient in nonlinear equations.

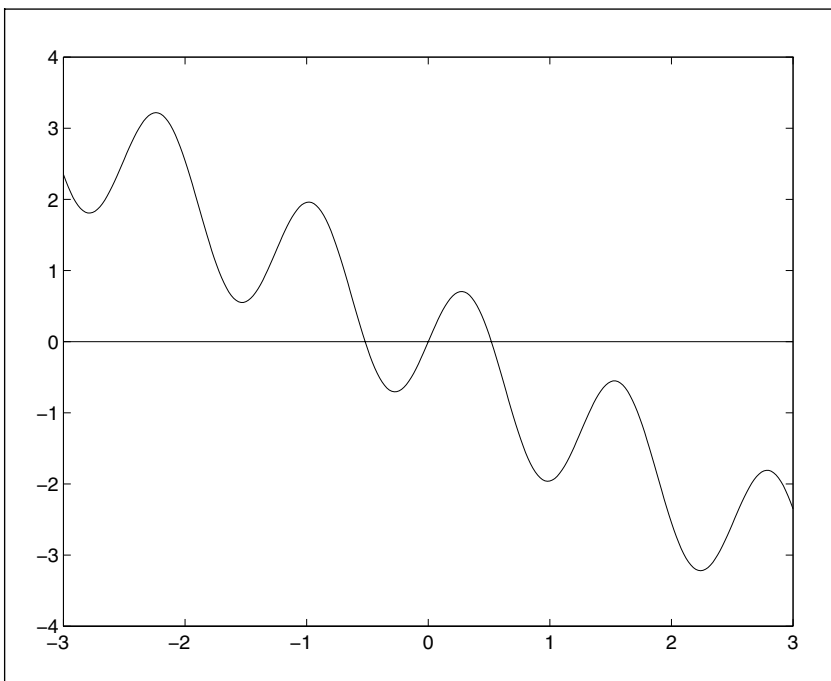


Figure 11.1 The function $r(x) = \sin(5x) - x$ has three roots.

Quasi-Newton methods are perhaps less useful in nonlinear equations than in optimization. In unconstrained optimization, the objective function is the natural choice of merit function that gauges progress towards the solution, but in nonlinear equations various merit functions can be used, all of which have some drawbacks. Line search and trust-region techniques play an equally important role in optimization, but one can argue that trust-region algorithms have certain theoretical advantages in solving nonlinear equations.

Some of the difficulties that arise in trying to solve nonlinear equations can be illustrated by a simple scalar example ($n = 1$). Suppose we have

$$r(x) = \sin(5x) - x, \quad (11.3)$$

as plotted in Figure 11.1. From this figure we see that there are three solutions of the problem $r(x) = 0$, also known as *roots of r* , located at zero and approximately ± 0.519148 . This situation of multiple solutions is similar to optimization problems where, for example, a function may have more than one local minimum. It is not *quite* the same, however: In the case of optimization, one of the local minima may have a lower function value than the others (making it a “better” solution), while in nonlinear equations all solutions are equally good from a mathematical viewpoint. (If the modeler decides that the solution

found by the algorithm makes no sense on physical grounds, their model may need to be reformulated.)

In this chapter we start by outlining algorithms related to Newton's method and examining their local convergence properties. Besides Newton's method itself, these include Broyden's quasi-Newton method, inexact Newton methods, and tensor methods. We then address global convergence, which is the issue of trying to force convergence to a solution from a remote starting point. Finally, we discuss a class of methods in which an "easy" problem—one to which the solution is well known—is gradually transformed into the problem $F(x) = 0$. In these so-called continuation (or homotopy) methods, we track the solution as the problem changes, with the aim of finishing up at a solution of $F(x) = 0$.

Throughout this chapter we make the assumption that the vector function r is continuously differentiable in the region \mathcal{D} containing the values of x we are interested in. In other words, the Jacobian $J(x)$ (the matrix of first partial derivatives of $r(x)$ defined in the Appendix and in (10.3)) exists and is continuous. We say that x^* satisfying $r(x^*) = 0$ is a *degenerate solution* if $J(x^*)$ is singular, and a *nondegenerate solution* otherwise.

11.1 LOCAL ALGORITHMS

NEWTON'S METHOD FOR NONLINEAR EQUATIONS

Recall from Theorem 2.1 that Newton's method for minimizing $f : \mathbb{R}^n \rightarrow \mathbb{R}$ forms a quadratic model function by taking the first three terms of the Taylor series approximation of f around the current iterate x_k . The Newton step is the vector that minimizes this model. In the case of nonlinear equations, Newton's method is derived in a similar way, but with a *linear* model, one that involves function values and first derivatives of the functions $r_i(x)$, $i = 1, 2, \dots, m$ at the current iterate x_k . We justify this strategy by referring to the following multidimensional variant of Taylor's theorem.

Theorem 11.1.

Suppose that $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is continuously differentiable in some convex open set \mathcal{D} and that x and $x + p$ are vectors in \mathcal{D} . We then have that

$$r(x + p) = r(x) + \int_0^1 J(x + tp)p \, dt. \quad (11.4)$$

We can define a linear model $M_k(p)$ of $r(x_k + p)$ by approximating the second term on the right-hand-side of (11.4) by $J(x_k)p$, and writing

$$M_k(p) \stackrel{\text{def}}{=} r(x_k) + J(x_k)p. \quad (11.5)$$

Newton's method, in its pure form, chooses the step p_k to be the vector for which $M_k(p_k) = 0$, that is, $p_k = -J(x_k)^{-1}r(x_k)$. We define it formally as follows.

Algorithm 11.1 (Newton's Method for Nonlinear Equations).

```
Choose  $x_0$ ;
for  $k = 0, 1, 2, \dots$ 
    Calculate a solution  $p_k$  to the Newton equations
```

$$J(x_k)p_k = -r(x_k); \quad (11.6)$$

```
     $x_{k+1} \leftarrow x_k + p_k$ ;
end (for)
```

We use a linear model to derive the Newton step, rather than a quadratic model as in unconstrained optimization, because the linear model normally has a solution and yields an algorithm with rapid convergence properties. In fact, Newton's method for unconstrained optimization (see (2.15)) can be derived by applying Algorithm 11.1 to the nonlinear equations $\nabla f(x) = 0$. We see also in Chapter 18 that sequential quadratic programming for equality-constrained optimization can be derived by applying Algorithm 11.1 to the nonlinear equations formed by the first-order optimality conditions (18.3) for this problem. Another connection is with the Gauss–Newton method for nonlinear least squares; the formula (11.6) is equivalent to (10.23) in the usual case in which $J(x_k)$ is nonsingular.

When the iterate x_k is close to a nondegenerate root x^* , Newton's method converges superlinearly, as we show in Theorem 11.2 below. Potential shortcomings of the method include the following.

- When the starting point is remote from a solution, Algorithm 11.1 can behave erratically. When $J(x_k)$ is singular, the Newton step may not even be defined.
- First-derivative information (the Jacobian matrix J) may be difficult to obtain.
- It may be too expensive to find and calculate the Newton step p_k exactly when n is large.
- The root x^* in question may be degenerate, that is, $J(x^*)$ may be singular.

An example of a degenerate problem is the scalar function $r(x) = x^2$, which has a single degenerate root at $x^* = 0$. Algorithm 11.1, when started from any nonzero x_0 , generates the sequence of iterates

$$x_k = \frac{1}{2^k}x_0,$$

which converges to the solution 0, but only at a linear rate.

As we show later in this chapter, Newton's method can be modified and enhanced in various ways to get around most of these problems. The variants we describe form the basis of much of the available software for solving nonlinear equations.

We summarize the local convergence properties of Algorithm 11.1 in the following theorem. For part of this result, we make use of a Lipschitz continuity assumption on the Jacobian, by which we mean that there is a constant β_L such that

$$\|J(x_0) - J(x_1)\| \leq \beta_L \|x_0 - x_1\|, \quad (11.7)$$

for all x_0 and x_1 in the domain in question.

Theorem 11.2.

Suppose that r is continuously differentiable in a convex open set $\mathcal{D} \subset \mathbb{R}^n$. Let $x^ \in \mathcal{D}$ be a nondegenerate solution of $r(x) = 0$, and let $\{x_k\}$ be the sequence of iterates generated by Algorithm 11.1. Then when $x_k \in \mathcal{D}$ is sufficiently close to x^* , we have*

$$x_{k+1} - x^* = o(\|x_k - x^*\|), \quad (11.8)$$

indicating local Q -superlinear convergence. When r is Lipschitz continuously differentiable near x^ , we have for all x_k sufficiently close to x^* that*

$$x_{k+1} - x^* = O(\|x_k - x^*\|^2), \quad (11.9)$$

indicating local Q -quadratic convergence.

PROOF. Since $r(x^*) = 0$, we have from Theorem 11.1 that

$$r(x_k) = r(x_k) - r(x^*) = J(x_k)(x_k - x^*) + w(x_k, x^*), \quad (11.10)$$

where

$$w(x_k, x^*) = \int_0^1 [J(x_k + t(x^* - x_k)) - J(x_k)](x_k - x^*) dt. \quad (11.11)$$

From (A.12) and continuity of J , we have

$$\begin{aligned} \|w(x_k, x^*)\| &= \left\| \int_0^1 [J(x^* + t(x^* - x_k)) - J(x_k)](x_k - x^*) dt \right\| \\ &\leq \int_0^1 \|J(x^* + t(x^* - x_k)) - J(x_k)\| \|x_k - x^*\| dt \\ &= o(\|x_k - x^*\|). \end{aligned} \quad (11.12)$$

Since $J(x^*)$ is nonsingular, there is a radius $\delta > 0$ and a positive constant β^* such that for all x in the ball $\mathcal{B}(x^*, \delta)$ defined by

$$\mathcal{B}(x^*, \delta) = \{x \mid \|x - x^*\| \leq \delta\}, \quad (11.13)$$

we have that

$$\|J(x)^{-1}\| \leq \beta^* \quad \text{and} \quad x \in \mathcal{D}. \quad (11.14)$$

Assuming that $x_k \in \mathcal{B}(x^*, \delta)$, and recalling the definition (11.6), we multiply both sides of (11.10) by $J(x_k)^{-1}$ to obtain

$$\begin{aligned} -p_k &= (x_k - x^*) + \|J(x_k)^{-1}\| o(\|x_k - x^*\|), \\ \Rightarrow x_k + p_k - x^* &= o(\|x_k - x^*\|), \\ \Rightarrow x_{k+1} - x^* &= o(\|x_k - x^*\|), \end{aligned} \quad (11.15)$$

which yields (11.8).

When the Lipschitz continuity assumption (11.7) is satisfied, we can obtain a sharper estimate for the remainder term $w(x_k, x^*)$ defined in (11.11). By using (11.7) in (11.12), we obtain

$$\|w(x_k, x^*)\| = O(\|x_k - x^*\|^2). \quad (11.16)$$

By multiplying (11.10) by $J(x_k)^{-1}$ as above, we obtain

$$-p_k - (x_k - x^*) = J(x_k)^{-1} w(x_k, x^*),$$

so the estimate (11.9) follows as in (11.15). □

INEXACT NEWTON METHODS

Instead of solving (11.6) exactly, inexact Newton methods use search directions p_k that satisfy the condition

$$\|r_k + J_k p_k\| \leq \eta_k \|r_k\|, \quad \text{for some } \eta_k \in [0, \eta], \quad (11.17)$$

where $\eta \in [0, 1)$ is a constant. As in Chapter 7, we refer to $\{\eta_k\}$ as the *forcing sequence*. Different methods make different choices of the forcing sequence, and they use different algorithms for finding the approximate solutions p_k . The general framework for this class of methods can be stated as follows.

Framework 11.2 (Inexact Newton for Nonlinear Equations).

Given $\eta \in [0, 1)$;

Choose x_0 ;

for $k = 0, 1, 2, \dots$

Choose forcing parameter $\eta_k \in [0, \eta]$;

Find a vector p_k that satisfies (11.17);

$x_{k+1} \leftarrow x_k + p_k$;

end (for)

The convergence theory for these methods depends only on the condition (11.17) and not on the particular technique used to calculate p_k . The most important methods in this class, however, make use of iterative techniques for solving linear systems of the form $Jp = -r$, such as GMRES (Saad and Schultz [273], Walker [302]) or other Krylov-space methods. Like the conjugate-gradient algorithm of Chapter 5 (which is not directly applicable here, since the coefficient matrix J is not symmetric positive definite), these methods typically require us to perform a matrix–vector multiplication of the form Jd for some d at each iteration, and to store a number of work vectors of length n . GMRES requires an additional vector to be stored at each iteration, so must be restarted periodically (often every 10 or 20 iterations) to keep memory requirements at a reasonable level.

The matrix–vector products Jd can be computed without explicit knowledge of the Jacobian J . A finite-difference approximation to Jd that requires one evaluation of $r(\cdot)$ is given by the formula (8.11). Calculation of Jd exactly (at least, to within the limits of finite-precision arithmetic) can be performed by using the forward mode of automatic differentiation, at a cost of at most a small multiple of an evaluation of $r(\cdot)$. Details of this procedure are given in Section 8.2.

We do not discuss the iterative methods for sparse linear systems here, but refer the interested reader to Kelley [177] and Saad [272] for comprehensive descriptions and implementations of the most interesting techniques. We prove a local convergence theorem for the method, similar to Theorem 11.2.

Theorem 11.3.

Suppose that r is continuously differentiable in a convex open set $\mathcal{D} \subset \mathbb{R}^n$. Let $x^ \in \mathcal{D}$ be a nondegenerate solution of $r(x) = 0$, and let $\{x_k\}$ be the sequence of iterates generated by the Framework 11.2. Then when $x_k \in \mathcal{D}$ is sufficiently close to x^* , the following are true:*

- (i) *If η in (11.17) is sufficiently small, the convergence of $\{x_k\}$ to x^* is Q-linear.*
- (ii) *If $\eta_k \rightarrow 0$, the convergence is Q-superlinear.*
- (iii) *If, in addition, $J(\cdot)$ is Lipschitz continuous in a neighborhood of x^* and $\eta_k = O(\|r_k\|)$, the convergence is Q-quadratic.*

PROOF. We first rewrite (11.17) as

$$J(x_k)p_k + r(x_k) = v_k, \quad \text{where } \|v_k\| \leq \eta_k \|r(x_k)\|. \quad (11.18)$$

Since x^* is a nondegenerate root, we have as in (11.14) that there is a radius $\delta > 0$ such that $\|J(x)^{-1}\| \leq \beta^*$ for some constant β^* and all $x \in \mathcal{B}(x^*, \delta)$. By multiplying both sides of (11.18) by $J(x_k)^{-1}$ and rearranging, we find that

$$\|p_k + J(x_k)^{-1}r(x_k)\| = \|J(x_k)^{-1}v_k\| \leq \beta^* \eta_k \|r(x_k)\|. \quad (11.19)$$

As in (11.10), we have that

$$r(x) = J(x)(x - x^*) + w(x, x^*), \quad (11.20)$$

where $\rho(x) \stackrel{\text{def}}{=} \|w(x, x^*)\|/\|x - x^*\| \rightarrow 0$ as $x \rightarrow x^*$. By reducing δ if necessary, we have from this expression that the following bound holds for all $x \in \mathcal{B}(x^*, \delta)$:

$$\|r(x)\| \leq 2\|J(x^*)\|\|x - x^*\| + o(\|x - x^*\|) \leq 4\|J(x^*)\|\|x - x^*\|. \quad (11.21)$$

We now set $x = x_k$ in (11.20), and use (11.19) and (11.21) to obtain

$$\begin{aligned} \|x_k + p_k - x^*\| &= \|p_k + J(x_k)^{-1}(r(x_k) - w(x_k, x^*))\| \\ &\leq \beta^* \eta_k \|r(x_k)\| + \|J(x_k)^{-1}\| \|w(x_k, x^*)\| \\ &\leq [4\|J(x^*)\|\beta^* \eta_k + \beta^* \rho(x_k)] \|x_k - x^*\|. \end{aligned} \quad (11.22)$$

By choosing x_k close enough to x^* that $\rho(x_k) \leq 1/(4\beta^*)$, and choosing $\eta = 1/(8\|J(x^*)\|\beta^*)$, we have that the term in square brackets in (11.22) is at most $1/2$. Hence, since $x_{k+1} = x_k + p_k$, this formula indicates Q-linear convergence of $\{x_k\}$ to x^* , proving part (i).

Part (ii) follows immediately from the fact that the term in brackets in (11.22) goes to zero as $x_k \rightarrow x^*$ and $\eta_k \rightarrow 0$. For part (iii), we combine the techniques above with the logic of the second part of the proof of Theorem 11.2. Details are left as an exercise. \square

BROYDEN'S METHOD

Secant methods, also known as quasi-Newton methods, do not require calculation of the Jacobian $J(x)$. Instead, they construct their own approximation to this matrix, updating it at each iteration so that it mimics the behavior of the true Jacobian J over the step just taken. The approximate Jacobian, which we denote at iteration k by B_k , is then used to construct a linear model analogous to (11.5), namely

$$M_k(p) = r(x_k) + B_k p. \quad (11.23)$$

We obtain the step by setting this model to zero. When B_k is nonsingular, we have the following explicit formula (cf. (11.6)):

$$p_k = -B_k^{-1} r(x_k). \quad (11.24)$$

The requirement that the approximate Jacobian should mimic the behavior of the true Jacobian can be specified as follows. Let s_k denote the step from x_k to x_{k+1} , and let y_k

be the corresponding change in r , that is,

$$s_k = x_{k+1} - x_k, \quad y_k = r(x_{k+1}) - r(x_k). \quad (11.25)$$

From Theorem 11.1, we have that s_k and y_k are related by the expression

$$y_k = \int_0^1 J(x_k + ts_k)s_k dt \approx J(x_{k+1})s_k + o(\|s_k\|). \quad (11.26)$$

We require the updated Jacobian approximation B_{k+1} to satisfy the following equation, which is known as the *secant equation*,

$$y_k = B_{k+1}s_k, \quad (11.27)$$

which ensures that B_{k+1} and $J(x_{k+1})$ have similar behavior along the direction s_k . (Note the similarity with the secant equation (6.6) in quasi-Newton methods for unconstrained optimization; the motivation is the same in both cases.) The secant equation does not say anything about how B_{k+1} should behave along directions orthogonal to s_k . In fact, we can view (11.27) as a system of n linear equations in n^2 unknowns, where the unknowns are the components of B_{k+1} , so for $n > 1$ the equation (11.27) does not determine all the components of B_{k+1} uniquely. (The scalar case of $n = 1$ gives rise to the scalar secant method; see (A.60).)

The most successful practical algorithm is Broyden's method, for which the update formula is

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)s_k^T}{s_k^T s_k}. \quad (11.28)$$

The Broyden update makes the smallest possible change to the Jacobian (as measured by the Euclidean norm $\|B_k - B_{k+1}\|_2$) that is consistent with (11.27), as we show in the following Lemma.

Lemma 11.4 (Dennis and Schnabel [92, Lemma 8.1.1]).

Among all matrices B satisfying $Bs_k = y_k$, the matrix B_{k+1} defined by (11.28) minimizes the difference $\|B - B_k\|$.

PROOF. Let B be any matrix that satisfies $Bs_k = y_k$. By the properties of the Euclidean norm (see (A.10)) and the fact that $\|ss^T/s^T s\| = 1$ for any vector s (see Exercise 11.1), we have

$$\begin{aligned} \|B_{k+1} - B_k\| &= \left\| \frac{(y_k - B_k s_k)s_k^T}{s_k^T s_k} \right\| \\ &= \left\| \frac{(B - B_k)s_k s_k^T}{s_k^T s_k} \right\| \leq \|B - B_k\| \left\| \frac{s_k s_k^T}{s_k^T s_k} \right\| = \|B - B_k\|. \end{aligned}$$

Hence, we have that

$$B_{k+1} \in \arg \min_{B: y_k = Bs_k} \|B - B_k\|,$$

and the result is proved. \square

In the specification of the algorithm below, we allow a line search to be performed along the search direction p_k , so that $s_k = \alpha p_k$ for some $\alpha > 0$ in the formula (11.25). (See below for details about line-search methods.)

Algorithm 11.3 (Broyden).

Choose x_0 and a nonsingular initial Jacobian approximation B_0 ;

for $k = 0, 1, 2, \dots$

 Calculate a solution p_k to the linear equations

$$B_k p_k = -r(x_k); \quad (11.29)$$

 Choose α_k by performing a line search along p_k ;

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$

$$s_k \leftarrow x_{k+1} - x_k;$$

$$y_k \leftarrow r(x_{k+1}) - r(x_k);$$

 Obtain B_{k+1} from the formula (11.28);

end (for)

Under certain assumptions, Broyden's method converges *superlinearly*, that is,

$$\|x_{k+1} - x^*\| = o(\|x_k - x^*\|). \quad (11.30)$$

This local convergence rate is fast enough for most practical purposes, though not as fast as the Q-quadratic convergence of Newton's method.

We illustrate the difference between the convergence rates of Newton's and Broyden's method with a small example. The function $r : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ defined by

$$r(x) = \begin{bmatrix} (x_1 + 3)(x_2^3 - 7) + 18 \\ \sin(x_2 e^{x_1} - 1) \end{bmatrix} \quad (11.31)$$

has a nondegenerate root at $x^* = (0, 1)^T$. We start both methods from the point $x_0 = (-0.5, 1.4)^T$, and use the exact Jacobian $J(x_0)$ at this point as the initial Jacobian approximation B_0 . Results are shown in Table 11.1.

Newton's method clearly exhibits Q-quadratic convergence, which is characterized by doubling of the exponent of the error at each iteration. Broyden's method takes twice as

Table 11.1 Convergence of Iterates in Broyden and Newton Methods

Iteration k	$\ x_k - x^*\ _2$	
	Broyden	Newton
0	0.64×10^0	0.64×10^0
1	0.62×10^{-1}	0.62×10^{-1}
2	0.52×10^{-3}	0.21×10^{-3}
3	0.25×10^{-3}	0.18×10^{-7}
4	0.43×10^{-4}	0.12×10^{-15}
5	0.14×10^{-6}	
6	0.57×10^{-9}	
7	0.18×10^{-11}	
8	0.87×10^{-15}	

Table 11.2 Convergence of Function Norms in Broyden and Newton Methods

Iteration k	$\ r(x_k)\ _2$	
	Broyden	Newton
0	0.74×10^1	0.74×10^1
1	0.59×10^0	0.59×10^0
2	0.20×10^{-2}	0.23×10^{-2}
3	0.21×10^{-2}	0.16×10^{-6}
4	0.37×10^{-3}	0.22×10^{-15}
5	0.12×10^{-5}	
6	0.49×10^{-8}	
7	0.15×10^{-10}	
8	0.11×10^{-18}	

many iterations as Newton's, and reduces the error at a rate that accelerates slightly towards the end. The function norms $\|r(x_k)\|$ approach zero at a similar rate to the iteration errors $\|x_k - x^*\|$. As in (11.10), we have that

$$r(x_k) = r(x_k) - r(x^*) \approx J(x^*)(x_k - x^*),$$

so by nonsingularity of $J(x^*)$, the norms of $r(x_k)$ and $(x_k - x^*)$ are bounded above and below by multiples of each other. For our example problem (11.31), convergence of the sequence of function norms in the two methods is shown in Table 11.2.

The convergence analysis of Broyden's method is more complicated than that of Newton's method. We state the following result without proof.

Theorem 11.5.

Suppose the assumptions of Theorem 11.2 hold. Then there are positive constants ϵ and δ such that if the starting point x_0 and the starting approximate Jacobian B_0 satisfy

$$\|x_0 - x^*\| \leq \delta, \quad \|B_0 - J(x^*)\| \leq \epsilon, \quad (11.32)$$

the sequence $\{x_k\}$ generated by Broyden's method (11.24), (11.28) is well-defined and converges *Q-superlinearly* to x^* .

The second condition in (11.32)—that the initial Jacobian approximation B_0 must be close to the true Jacobian at the solution $J(x^*)$ —is difficult to guarantee in practice. In contrast to the case of unconstrained minimization, a good choice of B_0 can be critical to the performance of the algorithm. Some implementations of Broyden's method recommend choosing B_0 to be $J(x_0)$, or some finite-difference approximation to this matrix.

The Broyden matrix B_k will be dense in general, even if the true Jacobian J is sparse. Therefore, when n is large, an implementation of Broyden's method that stores B_k as a full $n \times n$ matrix may be inefficient. Instead, we can use limited-memory methods in which B_k is stored implicitly in the form of a number of vectors of length n , while the system (11.29) is solved by a technique based on application of the Sherman–Morrison–Woodbury formula (A.28). These methods are similar to the ones described in Chapter 7 for large-scale unconstrained optimization.

TENSOR METHODS

In tensor methods, the linear model $M_k(p)$ used by Newton's method (11.5) is augmented with an extra term that aims to capture some of the nonlinear, higher-order, behavior of r . By doing so, it achieves more rapid and reliable convergence to degenerate roots, in particular, to roots x^* for which the Jacobian $J(x^*)$ has rank $n - 1$ or $n - 2$. We give a broad outline of the method here, and refer to Schnabel and Frank [277] for details.

We use $\hat{M}_k(p)$ to denote the model function on which tensor methods are based; this function has the form

$$\hat{M}_k(p) = r(x_k) + J(x_k)p + \frac{1}{2}T_k p p, \quad (11.33)$$

where T_k is a tensor defined by n^3 elements $(T_k)_{ijl}$ whose action on a pair of arbitrary vectors u and v in \mathbb{R}^n is defined by

$$(T_k u v)_i = \sum_{j=1}^n \sum_{l=1}^n (T_k)_{ijl} u_j v_l.$$

If we followed the reasoning behind Newton's method, we could consider building T_k from the *second* derivatives of r at the point x_k , that is,

$$(T_k)_{ijl} = [\nabla^2 r_i(x_k)]_{jl}.$$

For instance, in the example (11.31), we have that

$$\begin{aligned}(T(x)uv)_1 &= u^T \nabla^2 r_1(x)v = u^T \begin{bmatrix} 0 & 3x_2^2 \\ 3x_2^2 & 6x_2(x_1 + 3) \end{bmatrix} v \\ &= 3x_2^2(u_1v_2 + u_2v_1) + 6x_2(x_1 + 3)u_2v_2.\end{aligned}$$

However, use of the exact second derivatives is not practical in most instances. If we were to store this information explicitly, about $n^3/2$ memory locations would be needed, about n times the requirements of Newton's method. Moreover, there may be no vector p for which $\hat{M}_k(p) = 0$, so the step may not even be defined.

Instead, the approach described in [277] defines T_k in a way that requires little additional storage, but which gives \hat{M}_k some potentially appealing properties. Specifically, T_k is chosen so that $\hat{M}_k(p)$ interpolates the function $r(x_k + p)$ at some previous iterates visited by the algorithm. That is, we require that

$$\hat{M}_k(x_{k-j} - x_k) = r(x_{k-j}), \quad \text{for } j = 1, 2, \dots, q, \quad (11.34)$$

for some integer $q > 0$. By substituting from (11.33), we see that T_k must satisfy the condition

$$\frac{1}{2} T_k s_{jk} s_{jk} = r(x_{k-j}) - r(x_k) - J(x_k) s_{jk},$$

where

$$s_{jk} \stackrel{\text{def}}{=} x_{k-j} - x_k, \quad j = 1, 2, \dots, q.$$

In [277] it is shown that this condition can be ensured by choosing T_k so that its action on arbitrary vectors u and v is

$$T_k uv = \sum_{j=1}^q a_j (s_{jk}^T u) (s_{jk}^T v),$$

where a_j , $j = 1, 2, \dots, q$, are vectors of length n . The number of interpolating points q is typically chosen to be quite modest, usually less than \sqrt{n} . This T_k can be stored in $2nq$ locations, which contain the vectors a_j and s_{jk} for $j = 1, 2, \dots, q$. Note the connection between this idea and Broyden's method, which also chooses information in the model (albeit in the *first-order* part of the model) to interpolate the function value at the previous iterate.

This technique can be refined in various ways. The points of interpolation can be chosen to make the collection of directions s_{jk} more linearly independent. There may still not be a vector p for which $\hat{M}_k(p) = 0$, but we can instead take the step to be the vector that

minimizes $\|\hat{M}_k(p)\|_2^2$, which can be found by using a specialized least-squares technique. There is no assurance that the step obtained in this way is a descent direction for the merit function $\frac{1}{2}\|r(x)\|^2$ (which is discussed in the next section), and in this case it can be replaced by the standard Newton direction $-J_k^{-1}r_k$.

11.2 PRACTICAL METHODS

We now consider practical variants of the Newton-like methods discussed above, in which line-search and trust-region modifications to the steps are made in order to ensure better global convergence behavior.

MERIT FUNCTIONS

As mentioned above, neither Newton's method (11.6) nor Broyden's method (11.24), (11.28) with unit step lengths can be guaranteed to converge to a solution of $r(x) = 0$ unless they are started close to that solution. Sometimes, components of the unknown or function vector or the Jacobian will blow up. Another, more exotic, kind of behavior is *cycling*, where the iterates move between distinct regions of the parameter space without approaching a root. An example is the scalar function

$$r(x) = -x^5 + x^3 + 4x,$$

which has five nondegenerate roots. When started from the point $x_0 = 1$, Newton's method produces a sequence of iterates that oscillates between 1 and -1 (see Exercise 11.3) without converging to any of the roots.

The Newton and Broyden methods can be made more robust by using line-search and trust-region techniques similar to those described in Chapters 3 and 4. Before describing these techniques, we need to define a *merit function*, which is a scalar-valued function of x that indicates whether a new iterate is better or worse than the current iterate, in the sense of making progress toward a root of r . In unconstrained optimization, the objective function f is itself a natural merit function; most algorithms for minimizing f require a decrease in f at each iteration. In nonlinear equations, the merit function is obtained by combining the n components of the vector r in some way.

The most widely used merit function is the sum of squares, defined by

$$f(x) = \frac{1}{2}\|r(x)\|^2 = \frac{1}{2} \sum_{i=1}^n r_i^2(x). \quad (11.35)$$

(The factor $1/2$ is introduced for convenience.) Any root x^* of r obviously has $f(x^*) = 0$, and since $f(x) \geq 0$ for all x , each root is a minimizer of f . However, local minimizers of f are not roots of r if f is strictly positive at the point in question. Still, the merit function

CHAPTER *12*

Theory of Constrained Optimization

The second part of this book is about minimizing functions subject to constraints on the variables. A general formulation for these problems is

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}, \end{cases} \quad (12.1)$$

where f and the functions c_i are all smooth, real-valued functions on a subset of \mathbb{R}^n , and \mathcal{I} and \mathcal{E} are two finite sets of indices. As before, we call f the *objective* function, while c_i ,

$i \in \mathcal{E}$ are the *equality constraints* and $c_i, i \in \mathcal{I}$ are the *inequality constraints*. We define the *feasible set* Ω to be the set of points x that satisfy the constraints; that is,

$$\Omega = \{x \mid c_i(x) = 0, \quad i \in \mathcal{E}; \quad c_i(x) \geq 0, \quad i \in \mathcal{I}\}, \quad (12.2)$$

so that we can rewrite (12.1) more compactly as

$$\min_{x \in \Omega} f(x). \quad (12.3)$$

In this chapter we derive mathematical characterizations of the solutions of (12.3). As in the unconstrained case, we discuss optimality conditions of two types. *Necessary* conditions are conditions that must be satisfied by any solution point (under certain assumptions). *Sufficient* conditions are those that, if satisfied at a certain point x^* , guarantee that x^* is in fact a solution.

For the unconstrained optimization problem of Chapter 2, the optimality conditions were as follows:

Necessary conditions: Local unconstrained minimizers have $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ positive semidefinite.

Sufficient conditions: Any point x^* at which $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite is a strong local minimizer of f .

In this chapter, we derive analogous conditions to characterize the solutions of constrained optimization problems.

LOCAL AND GLOBAL SOLUTIONS

We have seen already that global solutions are difficult to find even when there are no constraints. The situation may be improved when we add constraints, since the feasible set might exclude many of the local minima and it may be comparatively easy to pick the global minimum from those that remain. However, constraints can also make things more difficult. As an example, consider the problem

$$\min (x_2 + 100)^2 + 0.01x_1^2, \quad \text{subject to } x_2 - \cos x_1 \geq 0, \quad (12.4)$$

illustrated in Figure 12.1. Without the constraint, the problem has the unique solution $(0, -100)^T$. With the constraint, there are local solutions near the points

$$x^{(k)} = (k\pi, -1)^T, \quad \text{for } k = \pm 1, \pm 3, \pm 5, \dots$$

Definitions of the different types of local solutions are simple extensions of the corresponding definitions for the unconstrained case, except that now we restrict consideration to the *feasible* points in the neighborhood of x^* . We have the following definition.

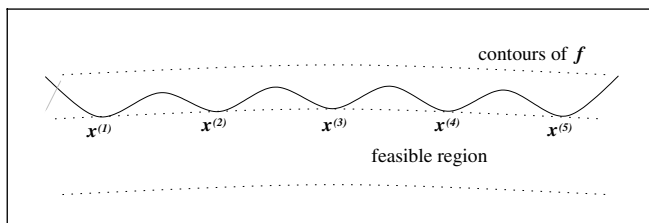


Figure 12.1 Constrained problem with many isolated local solutions.

A vector x^* is a *local solution* of the problem (12.3) if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} of x^* such that $f(x) \geq f(x^*)$ for $x \in \mathcal{N} \cap \Omega$.

Similarly, we can make the following definitions:

A vector x^* is a *strict local solution* (also called a *strong local solution*) if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} of x^* such that $f(x) > f(x^*)$ for all $x \in \mathcal{N} \cap \Omega$ with $x \neq x^*$.

A point x^* is an *isolated local solution* if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} of x^* such that x^* is the only local solution in $\mathcal{N} \cap \Omega$.

Note that isolated local solutions are strict, but that the reverse is not true (see Exercise 12.2).

SMOOTHNESS

Smoothness of objective functions and constraints is an important issue in characterizing solutions, just as in the unconstrained case. It ensures that the objective function and the constraints all behave in a reasonably predictable way and therefore allows algorithms to make good choices for search directions.

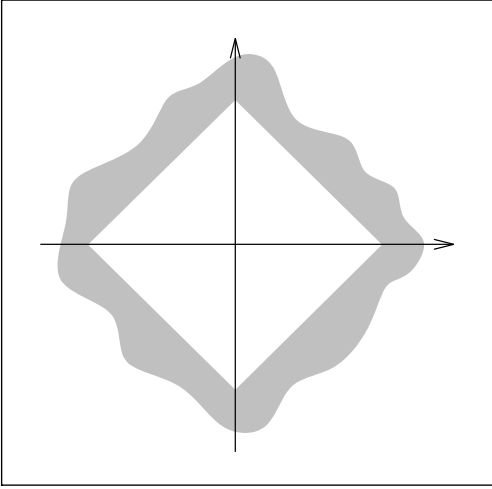
We saw in Chapter 2 that graphs of nonsmooth functions contain “kinks” or “jumps” where the smoothness breaks down. If we plot the feasible region for any given constrained optimization problem, we usually observe many kinks and sharp edges. Does this mean that the constraint functions that describe these regions are nonsmooth? The answer is often no, because the nonsmooth boundaries can often be described by a collection of smooth constraint functions. Figure 12.2 shows a diamond-shaped feasible region in \mathbb{R}^2 that could be described by the single nonsmooth constraint

$$\|x\|_1 = |x_1| + |x_2| \leq 1. \quad (12.5)$$

It can also be described by the following set of smooth (in fact, linear) constraints:

$$x_1 + x_2 \leq 1, \quad x_1 - x_2 \leq 1, \quad -x_1 + x_2 \leq 1, \quad -x_1 - x_2 \leq 1. \quad (12.6)$$

Each of the four constraints represents one edge of the feasible polytope. In general, the constraint functions are chosen so that each one represents a smooth piece of the boundary of Ω .

**Figure 12.2**

A feasible region with a nonsmooth boundary can be described by smooth constraints.

Nonsmooth, unconstrained optimization problems can sometimes be reformulated as smooth constrained problems. An example is the unconstrained minimization of a function

$$f(x) = \max(x^2, x), \quad (12.7)$$

which has kinks at $x = 0$ and $x = 1$, and the solution at $x^* = 0$. We obtain a smooth, constrained formulation of this problem by adding an artificial variable t and writing

$$\min t \quad \text{s.t.} \quad t \geq x, \quad t \geq x^2. \quad (12.8)$$

Reformulation techniques such as (12.6) and (12.8) are used often in cases where f is a maximum of a collection of functions or when f is a 1-norm or ∞ -norm of a vector function.

In the examples above we expressed inequality constraints in a slightly different way from the form $c_i(x) \geq 0$ that appears in the definition (12.1). However, any collection of inequality constraints with \geq and \leq and nonzero right-hand-sides can be expressed in the form $c_i(x) \geq 0$ by simple rearrangement of the inequality.

12.1 EXAMPLES

To introduce the basic principles behind the characterization of solutions of constrained optimization problems, we work through three simple examples. The discussion here is informal; the ideas introduced will be made rigorous in the sections that follow.

We start by noting one important item of terminology that recurs throughout the rest of the book.

Definition 12.1.

The active set $\mathcal{A}(x)$ at any feasible x consists of the equality constraint indices from \mathcal{E} together with the indices of the inequality constraints i for which $c_i(x) = 0$; that is,

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}.$$

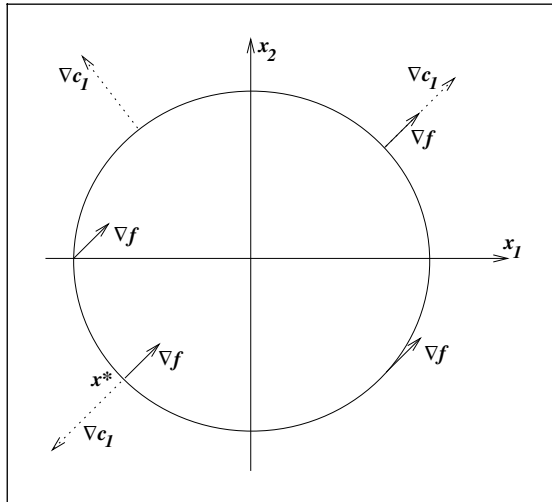
At a feasible point x , the inequality constraint $i \in \mathcal{I}$ is said to be *active* if $c_i(x) = 0$ and *inactive* if the strict inequality $c_i(x) > 0$ is satisfied.

A SINGLE EQUALITY CONSTRAINT**EXAMPLE 12.1**

Our first example is a two-variable problem with a single equality constraint:

$$\min x_1 + x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 2 = 0 \quad (12.9)$$

(see Figure 12.3). In the language of (12.1), we have $f(x) = x_1 + x_2$, $\mathcal{I} = \emptyset$, $\mathcal{E} = \{1\}$, and $c_1(x) = x_1^2 + x_2^2 - 2$. We can see by inspection that the feasible set for this problem is the circle of radius $\sqrt{2}$ centered at the origin—just the boundary of this circle, not its interior. The solution x^* is obviously $(-1, -1)^T$. From any other point on the circle, it is easy to find a way to move that *stays feasible* (that is, remains on the circle) while *decreasing* f . For instance, from the point $x = (\sqrt{2}, 0)^T$ any move in the clockwise direction around the circle has the desired effect.

**Figure 12.3**

Problem (12.9), showing constraint and function gradients at various feasible points.

We also see from Figure 12.3 that at the solution x^* , the *constraint normal* $\nabla c_1(x^*)$ is parallel to $\nabla f(x^*)$. That is, there is a scalar λ_1^* (in this case $\lambda_1^* = -1/2$) such that

$$\nabla f(x^*) = \lambda_1^* \nabla c_1(x^*). \quad (12.10)$$

□

We can derive (12.10) by examining first-order Taylor series approximations to the objective and constraint functions. To retain feasibility with respect to the function $c_1(x) = 0$, we require any small (but nonzero) step s to satisfy that $c_1(x + s) = 0$; that is,

$$0 = c_1(x + s) \approx c_1(x) + \nabla c_1(x)^T s = \nabla c_1(x)^T s. \quad (12.11)$$

Hence, the step s retains feasibility with respect to c_1 , to first order, when it satisfies

$$\nabla c_1(x)^T s = 0. \quad (12.12)$$

Similarly, if we want s to produce a decrease in f , we would have so that

$$0 > f(x + s) - f(x) \approx \nabla f(x)^T s,$$

or, to first order,

$$\nabla f(x)^T s < 0. \quad (12.13)$$

Existence of a small step s that satisfies both (12.12) and (12.13) strongly suggests existence of a direction d (where the size of d is *not* small; we could have $d \approx s/\|s\|$ to ensure that the norm of d is close to 1) with the same properties, namely

$$\nabla c_1(x)^T d = 0 \quad \text{and} \quad \nabla f(x)^T d < 0. \quad (12.14)$$

If, on the other hand, there is *no* direction d with the properties (12.14), then is it likely that we cannot find a small step s with the properties (12.12) and (12.13). In this case, x^* would appear to be a local minimizer.

By drawing a picture, the reader can check that the only way that a d satisfying (12.14) does *not* exist is if $\nabla f(x)$ and $\nabla c_1(x)$ are parallel, that is, if the condition $\nabla f(x) = \lambda_1 \nabla c_1(x)$ holds at x , for some scalar λ_1 . If in fact $\nabla f(x)$ and $\nabla c_1(x)$ are *not* parallel, we can set

$$\bar{d} = - \left(I - \frac{\nabla c_1(x) \nabla c_1(x)^T}{\|\nabla c_1(x)\|^2} \right) \nabla f(x); \quad d = \frac{\bar{d}}{\|\bar{d}\|}. \quad (12.15)$$

It is easy to verify that this d satisfies (12.14).

By introducing the *Lagrangian function*

$$\mathcal{L}(x, \lambda_1) = f(x) - \lambda_1 c_1(x), \quad (12.16)$$

and noting that $\nabla_x \mathcal{L}(x, \lambda_1) = \nabla f(x) - \lambda_1 \nabla c_1(x)$, we can state the condition (12.10) equivalently as follows: At the solution x^* , there is a scalar λ_1^* such that

$$\nabla_x \mathcal{L}(x^*, \lambda_1^*) = 0. \quad (12.17)$$

This observation suggests that we can search for solutions of the equality-constrained problem (12.9) by seeking stationary points of the Lagrangian function. The scalar quantity λ_1 in (12.16) is called a *Lagrange multiplier* for the constraint $c_1(x) = 0$.

Though the condition (12.10) (equivalently, (12.17)) appears to be *necessary* for an optimal solution of the problem (12.9), it is clearly not *sufficient*. For instance, in Example 12.1, condition (12.10) is satisfied at the point $x = (1, 1)^T$ (with $\lambda_1 = \frac{1}{2}$), but this point is obviously not a solution—in fact, it *maximizes* the function f on the circle. Moreover, in the case of equality-constrained problems, we cannot turn the condition (12.10) into a sufficient condition simply by placing some restriction on the sign of λ_1 . To see this, consider replacing the constraint $x_1^2 + x_2^2 - 2 = 0$ by its negative $2 - x_1^2 - x_2^2 = 0$ in Example 12.1. The solution of the problem is not affected, but the value of λ_1^* that satisfies the condition (12.10) changes from $\lambda_1^* = -\frac{1}{2}$ to $\lambda_1^* = \frac{1}{2}$.

A SINGLE INEQUALITY CONSTRAINT

□ EXAMPLE 12.2

This is a slight modification of Example 12.1, in which the equality constraint is replaced by an inequality. Consider

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0, \quad (12.18)$$

for which the feasible region consists of the circle of problem (12.9) and its interior (see Figure 12.4). Note that the constraint normal ∇c_1 points toward the interior of the feasible region at each point on the boundary of the circle. By inspection, we see that the solution is still $(-1, -1)^T$ and that the condition (12.10) holds for the value $\lambda_1^* = \frac{1}{2}$. However, this inequality-constrained problem differs from the equality-constrained problem (12.9) of Example 12.1 in that the sign of the Lagrange multiplier plays a significant role, as we now argue.

□

As before, we conjecture that a given feasible point x is *not* optimal if we can find a small step s that both retains feasibility and decreases the objective function f to first order. The main difference between problems (12.9) and (12.18) comes in the handling of the feasibility condition. As in (12.13), the step s improves the objective function, to first order, if $\nabla f(x)^T s < 0$. Meanwhile, s retains feasibility if

$$0 \leq c_1(x + s) \approx c_1(x) + \nabla c_1(x)^T s,$$

so, to first order, feasibility is retained if

$$c_1(x) + \nabla c_1(x)^T s \geq 0. \quad (12.19)$$

In determining whether a step s exists that satisfies both (12.13) and (12.19), we consider the following two cases, which are illustrated in Figure 12.4.

Case I: Consider first the case in which x lies *strictly inside* the circle, so that the strict inequality $c_1(x) > 0$ holds. In this case, *any* step vector s satisfies the condition (12.19), provided only that its length is sufficiently small. In fact, whenever $\nabla f(x) \neq 0$, we can obtain a step s that satisfies both (12.13) and (12.19) by setting

$$s = -\alpha \nabla f(x),$$

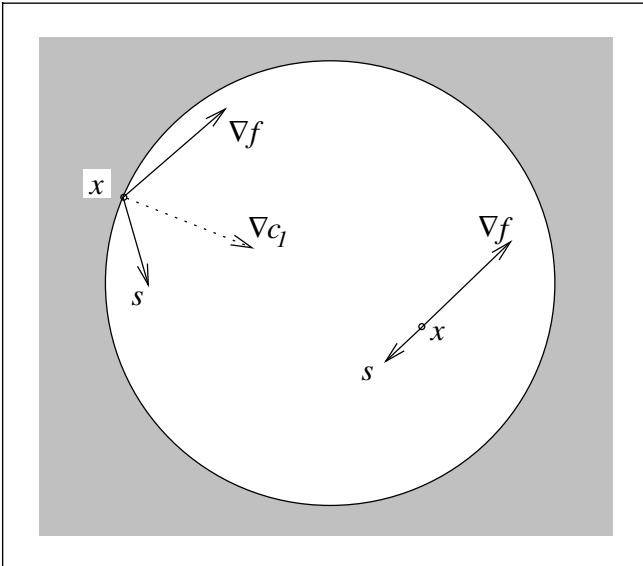


Figure 12.4 Improvement directions s from two feasible points x for the problem (12.18) at which the constraint is active and inactive, respectively.

for any positive scalar α sufficiently small. However, this definition does not give a step s with the required properties when

$$\nabla f(x) = 0, \quad (12.20)$$

Case II: Consider now the case in which x lies on the boundary of the circle, so that $c_1(x) = 0$. The conditions (12.13) and (12.19) therefore become

$$\nabla f(x)^T s < 0, \quad \nabla c_1(x)^T s \geq 0.$$

The first of these conditions defines an open half-space, while the second defines a closed half-space, as illustrated in Figure 12.5. It is clear from this figure that the intersection of these two regions is empty only when $\nabla f(x)$ and $\nabla c_1(x)$ point in the same direction, that is, when

$$\nabla f(x) = \lambda_1 \nabla c_1(x), \quad \text{for some } \lambda_1 \geq 0. \quad (12.21)$$

Note that the sign of the multiplier is significant here. If (12.10) were satisfied with a *negative* value of λ_1 , then $\nabla f(x)$ and $\nabla c_1(x)$ would point in opposite directions, and we see from Figure 12.5 that the set of directions that satisfy both (12.13) and (12.19) would make up an entire open half-plane.

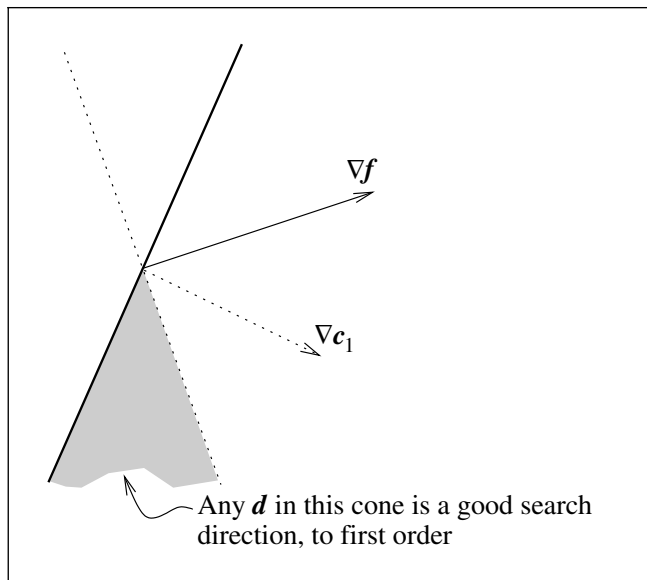


Figure 12.5 A direction d that satisfies both (12.13) and (12.19) lies in the intersection of a closed half-plane and an open half-plane.

The optimality conditions for both cases I and II can again be summarized neatly with reference to the Lagrangian function \mathcal{L} defined in (12.16). When no first-order feasible descent direction exists at some point x^* , we have that

$$\nabla_x \mathcal{L}(x^*, \lambda_1^*) = 0, \quad \text{for some } \lambda_1^* \geq 0, \quad (12.22)$$

where we also require that

$$\lambda_1^* c_1(x^*) = 0. \quad (12.23)$$

Condition (12.23) is known as a *complementarity condition*; it implies that the Lagrange multiplier λ_1 can be strictly positive *only when the corresponding constraint c_1 is active*. Conditions of this type play a central role in constrained optimization, as we see in the sections that follow. In case I, we have that $c_1(x^*) > 0$, so (12.23) requires that $\lambda_1^* = 0$. Hence, (12.22) reduces to $\nabla f(x^*) = 0$, as required by (12.20). In case II, (12.23) allows λ_1^* to take on a nonnegative value, so (12.22) becomes equivalent to (12.21).

TWO INEQUALITY CONSTRAINTS

□ EXAMPLE 12.3

Suppose we add an extra constraint to the problem (12.18) to obtain

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0, \quad x_2 \geq 0, \quad (12.24)$$

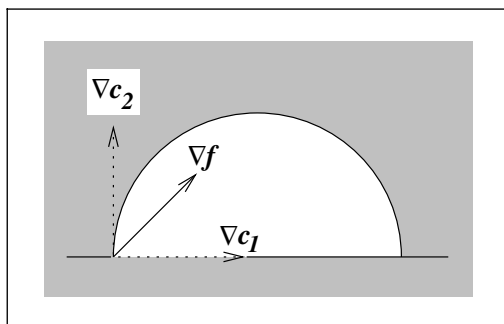
for which the feasible region is the half-disk illustrated in Figure 12.6. It is easy to see that the solution lies at $(-\sqrt{2}, 0)^T$, a point at which both constraints are active. By repeating the arguments for the previous examples, we would expect a direction d of first-order feasible descent to satisfy

$$\nabla c_i(x)^T d \geq 0, \quad i \in \mathcal{I} = \{1, 2\}, \quad \nabla f(x)^T d < 0. \quad (12.25)$$

However, it is clear from Figure 12.6 that no such direction can exist when $x = (-\sqrt{2}, 0)^T$. The conditions $\nabla c_i(x)^T d \geq 0$, $i = 1, 2$, are both satisfied only if d lies in the quadrant defined by $\nabla c_1(x)$ and $\nabla c_2(x)$, but it is clear by inspection that all vectors d in this quadrant satisfy $\nabla f(x)^T d \geq 0$.

Let us see how the Lagrangian and its derivatives behave for the problem (12.24) and the solution point $(-\sqrt{2}, 0)^T$. First, we include an additional term $\lambda_i c_i(x)$ in the Lagrangian for each additional constraint, so the definition of \mathcal{L} becomes

$$\mathcal{L}(x, \lambda) = f(x) - \lambda_1 c_1(x) - \lambda_2 c_2(x),$$

**Figure 12.6**

Problem (12.24), illustrating the gradients of the active constraints and objective at the solution.

where $\lambda = (\lambda_1, \lambda_2)^T$ is the vector of Lagrange multipliers. The extension of condition (12.22) to this case is

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \quad \text{for some } \lambda^* \geq 0, \quad (12.26)$$

where the inequality $\lambda^* \geq 0$ means that all components of λ^* are required to be nonnegative. By applying the complementarity condition (12.23) to both inequality constraints, we obtain

$$\lambda_1^* c_1(x^*) = 0, \quad \lambda_2^* c_2(x^*) = 0. \quad (12.27)$$

When $x^* = (-\sqrt{2}, 0)^T$, we have

$$\nabla f(x^*) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \nabla c_1(x^*) = \begin{bmatrix} 2\sqrt{2} \\ 0 \end{bmatrix}, \quad \nabla c_2(x^*) = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

so that it is easy to verify that $\nabla_x \mathcal{L}(x^*, \lambda^*) = 0$ when we select λ^* as follows:

$$\lambda^* = \begin{bmatrix} 1/(2\sqrt{2}) \\ 1 \end{bmatrix}.$$

Note that both components of λ^* are positive, so that (12.26) is satisfied.

We consider now some other feasible points that are *not* solutions of (12.24), and examine the properties of the Lagrangian and its gradient at these points.

For the point $x = (\sqrt{2}, 0)^T$, we again have that both constraints are active (see Figure 12.7). However, it is easy to identify vectors d that satisfy (12.25): $d = (-1, 0)^T$ is one such vector (there are many others). For this value of x it is easy to verify that the condition $\nabla_x \mathcal{L}(x, \lambda) = 0$ is satisfied only when $\lambda = (-1/(2\sqrt{2}), 1)^T$. Note that the first component λ_1 is negative, so that the conditions (12.26) are not satisfied at this point.

Finally, we consider the point $x = (1, 0)^T$, at which only the second constraint c_2 is active. Since any small step s away from this point will continue to satisfy $c_1(x + s) > 0$, we need to consider only the behavior of c_2 and f in determining whether s is indeed a feasible

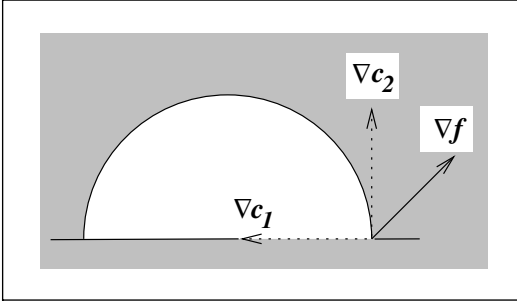


Figure 12.7
Problem (12.24), illustrating the gradients of the active constraints and objective at a nonoptimal point.

descent step. Using the same reasoning as in the earlier examples, we find that the direction of feasible descent d must satisfy

$$\nabla c_2(x)^T d \geq 0, \quad \nabla f(x)^T d < 0. \quad (12.28)$$

By noting that

$$\nabla f(x) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \nabla c_2(x) = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

it is easy to verify that the vector $d = \left(-\frac{1}{2}, \frac{1}{4}\right)^T$ satisfies (12.28) and is therefore a descent direction.

To show that optimality conditions (12.26) and (12.27) fail, we note first from (12.27) that since $c_1(x) > 0$, we must have $\lambda_1 = 0$. Therefore, in trying to satisfy $\nabla_x \mathcal{L}(x, \lambda) = 0$, we are left to search for a value λ_2 such that $\nabla f(x) - \lambda_2 \nabla c_2(x) = 0$. No such λ_2 exists, and thus this point fails to satisfy the optimality conditions. \square

12.2 TANGENT CONE AND CONSTRAINT QUALIFICATIONS

In this section we define the tangent cone $T_\Omega(x^*)$ to the closed convex set Ω at a point $x^* \in \Omega$, and also the set $\mathcal{F}(x^*)$ of first-order feasible directions at x^* . We also discuss *constraint qualifications*. In the previous section, we determined whether or not it was possible to take a feasible descent step away from a given feasible point x by examining the first derivatives of f and the constraint functions c_i . We used the first-order Taylor series expansion of these functions about x to form an approximate problem in which both objective and constraints are linear. This approach makes sense, however, only when the linearized approximation captures the essential geometric features of the feasible set near the point x in question. If, near x , the linearization is fundamentally different from the

feasible set (for instance, it is an entire plane, while the feasible set is a single point) then we cannot expect the linear approximation to yield useful information about the original problem. Hence, we need to make assumptions about the nature of the constraints c_i that are active at x to ensure that the linearized approximation is similar to the feasible set, near x . Constraint qualifications are assumptions that ensure similarity of the constraint set Ω and its linearized approximation, in a neighborhood of x^* .

Given a feasible point x , we call $\{z_k\}$ a *feasible sequence approaching x* if $z_k \in \Omega$ for all k sufficiently large and $z_k \rightarrow x$.

Later, we characterize a local solution of (12.1) as a point x at which all feasible sequences approaching x have the property that $f(z_k) \geq f(x)$ for all k sufficiently large, and we will derive practical, verifiable conditions under which this property holds. We lay the groundwork in this section by characterizing the directions in which we can step away from x while remaining feasible.

A *tangent* is a limiting direction of a feasible sequence.

Definition 12.2.

The vector d is said to be a *tangent* (or *tangent vector*) to Ω at a point x if there are a feasible sequence $\{z_k\}$ approaching x and a sequence of positive scalars $\{t_k\}$ with $t_k \rightarrow 0$ such that

$$\lim_{k \rightarrow \infty} \frac{z_k - x}{t_k} = d. \quad (12.29)$$

The set of all tangents to Ω at x^* is called the *tangent cone* and is denoted by $T_\Omega(x^*)$.

It is easy to see that the tangent cone is indeed a cone, according to the definition (A.36). If d is a tangent vector with corresponding sequences $\{z_k\}$ and $\{t_k\}$, then by replacing each t_k by $\alpha^{-1}t_k$, for any $\alpha > 0$, we find that $\alpha d \in T_\Omega(x^*)$ also. We obtain that $0 \in T_\Omega(x)$ by setting $z_k \equiv x$ in the definition of feasible sequence.

We turn now to the linearized feasible direction set, which we define as follows.

Definition 12.3.

Given a feasible point x and the active constraint set $\mathcal{A}(x)$ of Definition 12.1, the set of linearized feasible directions $\mathcal{F}(x)$ is

$$\mathcal{F}(x) = \left\{ d \mid \begin{array}{ll} d^T \nabla c_i(x) = 0, & \text{for all } i \in \mathcal{E}, \\ d^T \nabla c_i(x) \geq 0, & \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\}.$$

As with the tangent cone, it is easy to verify that $\mathcal{F}(x)$ is a cone, according to the definition (A.36).

It is important to note that the definition of tangent cone does not rely on the algebraic specification of the set Ω , only on its geometry. The linearized feasible direction set does, however, depend on the definition of the constraint functions c_i , $i \in \mathcal{E} \cup \mathcal{I}$.

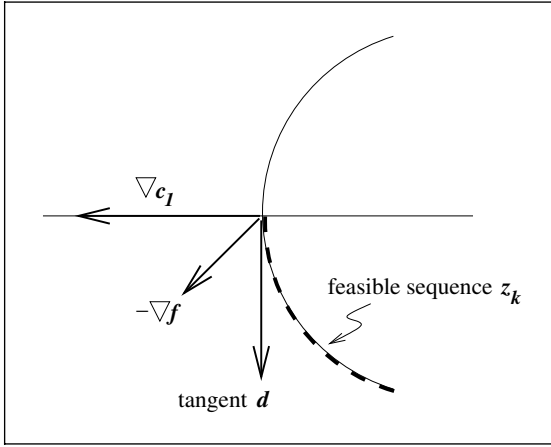


Figure 12.8
Constraint normal, objective gradient, and feasible sequence for problem (12.9).

We illustrate the tangent cone and the linearized feasible direction set by revisiting Examples 12.1 and 12.2.

□ **EXAMPLE 12.4** (EXAMPLE 12.1, REVISITED)

Figure 12.8 shows the problem (12.9), the equality-constrained problem in which the feasible set is a circle of radius $\sqrt{2}$, near the nonoptimal point $x = (-\sqrt{2}, 0)^T$. The figure also shows a feasible sequence approaching x . This sequence could be defined analytically by the formula

$$z_k = \begin{bmatrix} -\sqrt{2 - 1/k^2} \\ -1/k \end{bmatrix}. \quad (12.30)$$

By choosing $t_k = \|z_k - x\|$, we find that $d = (0, -1)^T$ is a tangent. Note that the objective function $f(x) = x_1 + x_2$ increases as we move along the sequence (12.30); in fact, we have $f(z_{k+1}) > f(z_k)$ for all $k = 2, 3, \dots$. It follows that $f(z_k) < f(x)$ for $k = 2, 3, \dots$, so x cannot be a solution of (12.9).

Another feasible sequence is one that approaches $x = (-\sqrt{2}, 0)^T$ from the opposite direction. Its elements are defined by

$$z_k = \begin{bmatrix} -\sqrt{2 - 1/k^2} \\ 1/k \end{bmatrix}.$$

It is easy to show that f decreases along this sequence and that the tangents corresponding to this sequence are $d = (0, \alpha)^T$. In summary, the tangent cone at $x = (-\sqrt{2}, 0)^T$ is $\{(0, d_2)^T \mid d_2 \in \mathbb{R}\}$.

For the definition (12.9) of this set, and Definition 12.3, we have that $d = (d_1, d_2)^T \in \mathcal{F}(x)$ if

$$0 = \nabla c_1(x)^T d = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}^T \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = -2\sqrt{2}d_1.$$

Therefore, we obtain $\mathcal{F}(x) = \{(0, d_2)^T \mid d_2 \in \mathbb{R}\}$. In this case, we have $T_\Omega(x) = \mathcal{F}(x)$.

Suppose that the feasible set is defined instead by the formula

$$\Omega = \{x \mid c_1(x) = 0\}, \quad \text{where } c_1(x) = (x_1^2 + x_2^2 - 2)^2 = 0. \quad (12.31)$$

(Note that Ω is the same, but its algebraic specification has changed.) The vector d belongs to the linearized feasible set if

$$0 = \nabla c_1(x)^T d = \begin{bmatrix} 4(x_1^2 + x_2^2 - 2)x_1 \\ 4(x_1^2 + x_2^2 - 2)x_2 \end{bmatrix}^T \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} d_1 \\ d_2 \end{bmatrix},$$

which is true for all $(d_1, d_2)^T$. Hence, we have $\mathcal{F}(x) = \mathbb{R}^2$, so for this algebraic specification of Ω , the tangent cone and linearized feasible sets differ. \square

\square EXAMPLE 12.5 (EXAMPLE 12.2, REVISITED)

We now reconsider problem (12.18) in Example 12.2. The solution $x = (-1, -1)^T$ is the same as in the equality-constrained case, but there is a much more extensive collection of feasible sequences that converge to any given feasible point (see Figure 12.9).

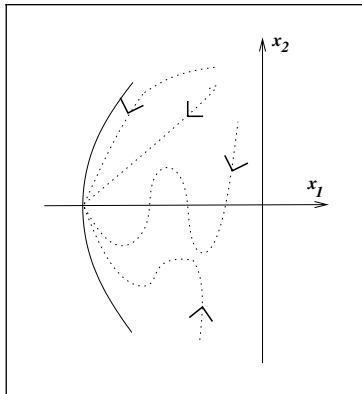


Figure 12.9

Feasible sequences converging to a particular feasible point for the region defined by $x_1^2 + x_2^2 \leq 2$.

From the point $x = (-\sqrt{2}, 0)^T$, the various feasible sequences defined above for the equality-constrained problem are still feasible for (12.18). There are also infinitely many feasible sequences that converge to $x = (-\sqrt{2}, 0)^T$ along a straight line from the interior of the circle. These sequences have the form

$$z_k = (-\sqrt{2}, 0)^T + (1/k)w,$$

where w is any vector whose first component is positive ($w_1 > 0$). The point z_k is feasible provided that $\|z_k\| \leq \sqrt{2}$, that is,

$$(-\sqrt{2} + w_1/k)^2 + (w_2/k)^2 \leq 2,$$

which is true when $k \geq (w_1^2 + w_2^2)/(2\sqrt{2}w_1)$. In addition to these straight-line feasible sequences, we can also define an infinite variety of sequences that approach $(-\sqrt{2}, 0)^T$ along a curve from the interior of the circle. To summarize, the tangent cone to this set at $(-\sqrt{2}, 0)^T$ is $\{(w_1, w_2)^T \mid w_1 \geq 0\}$.

For the definition (12.18) of this feasible set, we have from Definition 12.3 that $d \in \mathcal{F}(x)$ if

$$0 \leq \nabla c_1(x)^T d = \begin{bmatrix} -2x_1 \\ -2x_2 \end{bmatrix}^T \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = 2\sqrt{2}d_1.$$

Hence, we obtain $\mathcal{F}(x) = T_\Omega(x)$ for this particular algebraic specification of the feasible set. □

Constraint qualifications are conditions under which the linearized feasible set $\mathcal{F}(x)$ is similar to the tangent cone $T_\Omega(x)$. In fact, most constraint qualifications ensure that these two sets are identical. As mentioned earlier, these conditions ensure that the $\mathcal{F}(x)$, which is constructed by linearizing the algebraic description of the set Ω at x , captures the essential geometric features of the set Ω in the vicinity of x , as represented by $T_\Omega(x)$.

Revisiting Example 12.4, we see that both $T_\Omega(x)$ and $\mathcal{F}(x)$ consist of the vertical axis, which is qualitatively similar to the set $\Omega - \{x\}$ in the neighborhood of x . As a further example, consider the constraints

$$c_1(x) = 1 - x_1^2 - (x_2 - 1)^2 \geq 0, \quad c_2(x) = -x_2 \geq 0, \quad (12.32)$$

for which the feasible set is the single point $\Omega = \{(0, 0)^T\}$ (see Figure 12.10). For this point $x = (0, 0)^T$, it is obvious that the tangent cone is $T_\Omega(x) = \{(0, 0)^T\}$, since all feasible sequences approaching x must have $z_k = x = (0, 0)^T$ for all k sufficiently large. Moreover, it is easy to show that linearized approximation to the feasible set $\mathcal{F}(x)$ is

$$\mathcal{F}(x^*) = \{(d_1, 0)^T \mid d_1 \in \mathbb{R}\},$$

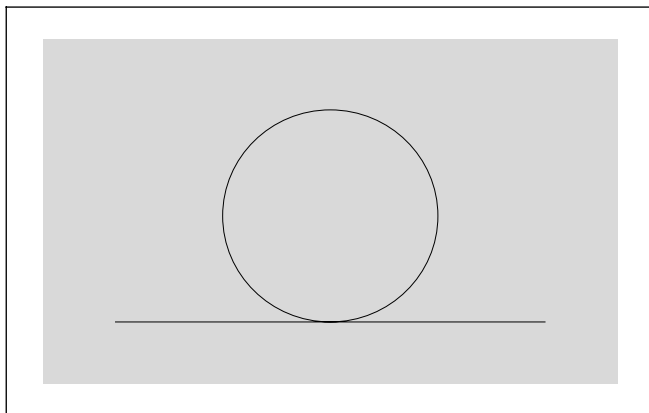


Figure 12.10 Problem (12.32), for which the feasible set is the single point of intersection between circle and line.

that is, the entire horizontal axis. In this case, the linearized feasible direction set does not capture the geometry of the feasible set, so constraint qualifications are not satisfied.

The constraint qualification most often used in the design of algorithms is the subject of the next definition.

Definition 12.4 (LICQ).

Given the point x and the active set $\mathcal{A}(x)$ defined in Definition 12.1, we say that the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(x), i \in \mathcal{A}(x)\}$ is linearly independent.

Note that this condition is *not* satisfied for the examples (12.32) and (12.31). In general, if LICQ holds, none of the active constraint gradients can be zero. We mention other constraint qualifications in Section 12.6.

12.3 FIRST-ORDER OPTIMALITY CONDITIONS

In this section, we state first-order necessary conditions for x^* to be a local minimizer and show how these conditions are satisfied on a small example. The proof of the result is presented in subsequent sections.

As a preliminary to stating the necessary conditions, we define the Lagrangian function for the general problem (12.1).

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x). \quad (12.33)$$

(We had previously defined special cases of this function for the examples of Section 12.1.)

The necessary conditions defined in the following theorem are called *first-order conditions* because they are concerned with properties of the gradients (first-derivative vectors) of the objective and constraint functions. These conditions are the foundation for many of the algorithms described in the remaining chapters of the book.

Theorem 12.1 (First-Order Necessary Conditions).

Suppose that x^* is a local solution of (12.1), that the functions f and c_i in (12.1) are continuously differentiable, and that the LICQ holds at x^* . Then there is a Lagrange multiplier vector λ^* , with components λ_i^* , $i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*)

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \quad (12.34a)$$

$$c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E}, \quad (12.34b)$$

$$c_i(x^*) \geq 0, \quad \text{for all } i \in \mathcal{I}, \quad (12.34c)$$

$$\lambda_i^* \geq 0, \quad \text{for all } i \in \mathcal{I}, \quad (12.34d)$$

$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E} \cup \mathcal{I}. \quad (12.34e)$$

The conditions (12.34) are often known as the *Karush–Kuhn–Tucker conditions*, or *KKT conditions* for short. The conditions (12.34e) are *complementarity conditions*; they imply that either constraint i is active or $\lambda_i^* = 0$, or possibly both. In particular, the Lagrange multipliers corresponding to inactive inequality constraints are zero, we can omit the terms for indices $i \notin \mathcal{A}(x^*)$ from (12.34a) and rewrite this condition as

$$0 = \nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla c_i(x^*). \quad (12.35)$$

A special case of complementarity is important and deserves its own definition.

Definition 12.5 (Strict Complementarity).

Given a local solution x^* of (12.1) and a vector λ^* satisfying (12.34), we say that the strict complementarity condition holds if exactly one of λ_i^* and $c_i(x^*)$ is zero for each index $i \in \mathcal{I}$. In other words, we have that $\lambda_i^* > 0$ for each $i \in \mathcal{I} \cap \mathcal{A}(x^*)$.

Satisfaction of the strict complementarity property usually makes it easier for algorithms to determine the active set $\mathcal{A}(x^*)$ and converge rapidly to the solution x^* .

For a given problem (12.1) and solution point x^* , there may be many vectors λ^* for which the conditions (12.34) are satisfied. When the LICQ holds, however, the optimal λ^* is unique (see Exercise 12.17).

The proof of Theorem 12.1 is quite complex, but it is important to our understanding of constrained optimization, so we present it in the next section. First, we illustrate the KKT conditions with another example.

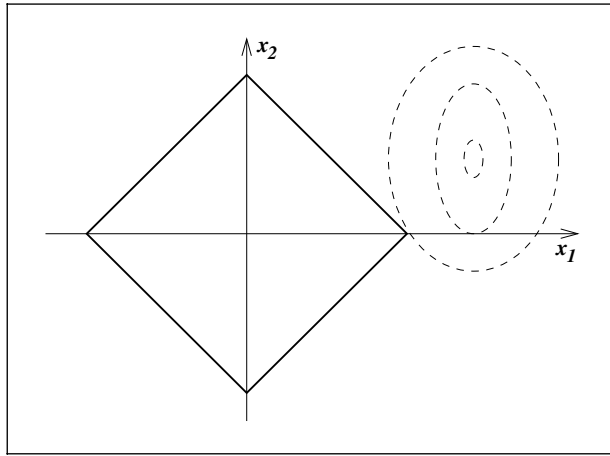


Figure 12.11 Inequality-constrained problem (12.36) with solution at $(1, 0)^T$.

□ EXAMPLE 12.6

Consider the feasible region illustrated in Figure 12.2 and described by the four constraints (12.6). By restating the constraints in the standard form of (12.1) and including an objective function, the problem becomes

$$\min_x \left(x_1 - \frac{3}{2} \right)^2 + \left(x_2 - \frac{1}{2} \right)^4 \quad \text{s.t.} \quad \begin{bmatrix} 1 - x_1 - x_2 \\ 1 - x_1 + x_2 \\ 1 + x_1 - x_2 \\ 1 + x_1 + x_2 \end{bmatrix} \geq 0. \quad (12.36)$$

It is fairly clear from Figure 12.11 that the solution is $x^* = (1, 0)^T$. The first and second constraints in (12.36) are active at this point. Denoting them by c_1 and c_2 (and the inactive constraints by c_3 and c_4), we have

$$\nabla f(x^*) = \begin{bmatrix} -1 \\ 1 \\ -\frac{1}{2} \end{bmatrix}, \quad \nabla c_1(x^*) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad \nabla c_2(x^*) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Therefore, the KKT conditions (12.34a)–(12.34e) are satisfied when we set

$$\lambda^* = \left(\frac{3}{4}, \frac{1}{4}, 0, 0 \right)^T.$$

□

12.4 FIRST-ORDER OPTIMALITY CONDITIONS: PROOF

We now develop a proof of Theorem 12.1. A number of key subsidiary results are required, so the development is quite long. However, a complete treatment is worthwhile, since these results are so fundamental to the field of optimization.

RELATING THE TANGENT CONE AND THE FIRST-ORDER FEASIBLE DIRECTION SET

The following key result uses a constraint qualification (LICQ) to relate the tangent cone of Definition 12.2 to the set \mathcal{F} of first-order feasible directions of Definition 12.3. In the proof below and in later results, we use the notation $A(x^*)$ to represent the matrix whose rows are the active constraint gradients at the optimal point, that is,

$$A(x^*)^T = [\nabla c_i(x^*)]_{i \in \mathcal{A}(x^*)}, \quad (12.37)$$

where the active set $\mathcal{A}(x^*)$ is defined as in Definition 12.1.

Lemma 12.2.

Let x^ be a feasible point. The following two statements are true.*

- (i) $T_\Omega(x^*) \subset \mathcal{F}(x^*)$.
- (ii) *If the LICQ condition is satisfied at x^* , then $\mathcal{F}(x^*) = T_\Omega(x^*)$.*

PROOF. Without loss of generality, let us assume that all the constraints $c_i(\cdot)$, $i = 1, 2, \dots, m$, are active at x^* . (We can arrive at this convenient ordering by simply dropping all inactive constraints—which are irrelevant in some neighborhood of x^* —and renumbering the active constraints that remain.)

To prove (i), let $\{z_k\}$ and $\{t_k\}$ be the sequences for which (12.29) is satisfied, that is,

$$\lim_{k \rightarrow \infty} \frac{z_k - x^*}{t_k} = d.$$

(Note in particular that $t_k > 0$ for all k .) From this definition, we have that

$$z_k = x^* + t_k d + o(t_k). \quad (12.38)$$

By taking $i \in \mathcal{E}$ and using Taylor's theorem, we have that

$$\begin{aligned} 0 &= \frac{1}{t_k} c_i(z_k) \\ &= \frac{1}{t_k} [c_i(x^*) + t_k \nabla c_i(x^*)^T d + o(t_k)] \\ &= \nabla c_i(x^*)^T d + \frac{o(t_k)}{t_k}. \end{aligned}$$

12.5 SECOND-ORDER CONDITIONS

So far, we have described first-order conditions—the KKT conditions—which tell us how the first derivatives of f and the active constraints c_i are related to each other at a solution x^* . When these conditions are satisfied, a move along any vector w from $\mathcal{F}(x^*)$ either increases the first-order approximation to the objective function (that is, $w^T \nabla f(x^*) > 0$), or else keeps this value the same (that is, $w^T \nabla f(x^*) = 0$).

What role do the *second* derivatives of f and the constraints c_i play in optimality conditions? We see in this section that second derivatives play a “tiebreaking” role. For the directions $w \in \mathcal{F}(x^*)$ for which $w^T \nabla f(x^*) = 0$, we cannot determine from first derivative information alone whether a move along this direction will increase or decrease the objective function f . Second-order conditions examine the second derivative terms in the Taylor series expansions of f and c_i , to see whether this extra information resolves the issue of increase or decrease in f . Essentially, the second-order conditions concern the curvature of the Lagrangian function in the “undecided” directions—the directions $w \in \mathcal{F}(x^*)$ for which $w^T \nabla f(x^*) = 0$.

Since we are discussing second derivatives, stronger smoothness assumptions are needed here than in the previous sections. For the purpose of this section, f and c_i , $i \in \mathcal{E} \cup \mathcal{I}$, are all assumed to be twice continuously differentiable.

Given $\mathcal{F}(x^*)$ from Definition 12.3 and some Lagrange multiplier vector λ^* satisfying the KKT conditions (12.34), we define the *critical cone* $\mathcal{C}(x^*, \lambda^*)$ as follows:

$$\mathcal{C}(x^*, \lambda^*) = \{w \in \mathcal{F}(x^*) \mid \nabla c_i(x^*)^T w = 0, \text{ all } i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0\}.$$

Equivalently,

$$w \in \mathcal{C}(x^*, \lambda^*) \Leftrightarrow \begin{cases} \nabla c_i(x^*)^T w = 0, & \text{for all } i \in \mathcal{E}, \\ \nabla c_i(x^*)^T w = 0, & \text{for all } i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^T w \geq 0, & \text{for all } i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* = 0. \end{cases} \quad (12.53)$$

The critical cone contains those directions w that would tend to “adhere” to the active inequality constraints even when we were to make small changes to the objective (those indices $i \in \mathcal{I}$ for which the Lagrange multiplier component λ_i^* is positive), as well as to the equality constraints. From the definition (12.53) and the fact that $\lambda_i^* = 0$ for all inactive components $i \in \mathcal{I} \setminus \mathcal{A}(x^*)$, it follows immediately that

$$w \in \mathcal{C}(x^*, \lambda^*) \Rightarrow \lambda_i^* \nabla c_i(x^*)^T w = 0 \text{ for all } i \in \mathcal{E} \cup \mathcal{I}. \quad (12.54)$$

Hence, from the first KKT condition (12.34a) and the definition (12.33) of the Lagrangian function, we have that

$$w \in \mathcal{C}(x^*, \lambda^*) \Rightarrow w^T \nabla f(x^*) = \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i^* w^T \nabla c_i(x^*) = 0. \quad (12.55)$$

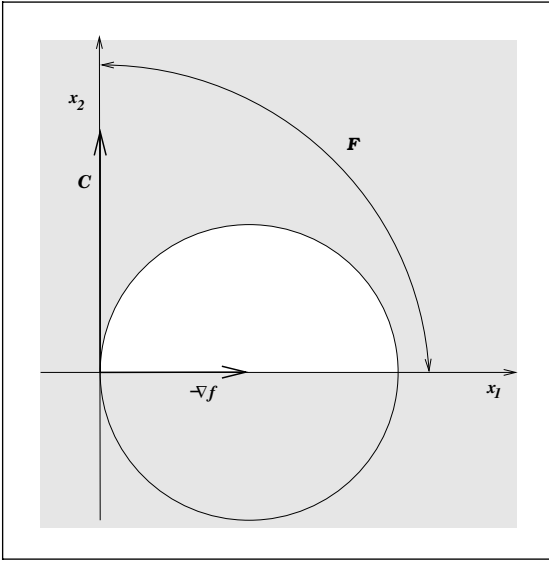


Figure 12.14
Problem (12.56), showing $\mathcal{F}(x^*)$ and $\mathcal{C}(x^*, \lambda^*)$.

Hence the critical cone $\mathcal{C}(x^*, \lambda^*)$ contains directions from $\mathcal{F}(x^*)$ for which it is not clear from first derivative information alone whether f will increase or decrease.

□ EXAMPLE 12.7

Consider the problem

$$\min x_1 \quad \text{subject to } x_2 \geq 0, \quad 1 - (x_1 - 1)^2 - x_2^2 \geq 0, \quad (12.56)$$

illustrated in Figure 12.14. It is not difficult to see that the solution is $x^* = (0, 0)^T$, with active set $\mathcal{A}(x^*) = \{1, 2\}$ and a unique optimal Lagrange multiplier $\lambda^* = (0, 0.5)^T$. Since the gradients of the active constraints at x^* are $(0, 1)^T$ and $(2, 0)^T$, respectively, the LICQ holds, so the optimal multiplier is unique. The linearized feasible set is then

$$\mathcal{F}(x^*) = \{d \mid d \geq 0\},$$

while the critical cone is

$$\mathcal{C}(x^*, \lambda^*) = \{(0, w_2)^T \mid w_2 \geq 0\}.$$



The first theorem defines a *necessary* condition involving the second derivatives: If x^* is a local solution, then the Hessian of the Lagrangian has nonnegative curvature along critical directions (that is, the directions in $\mathcal{C}(x^*, \lambda^*)$).

Theorem 12.5 (Second-Order Necessary Conditions).

Suppose that x^* is a local solution of (12.1) and that the LICQ condition is satisfied. Let λ^* be the Lagrange multiplier vector for which the KKT conditions (12.34) are satisfied. Then

$$w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w \geq 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*). \quad (12.57)$$

PROOF. Since x^* is a local solution, all feasible sequences $\{z_k\}$ approaching x^* must have $f(z_k) \geq f(x^*)$ for all k sufficiently large. Our approach in this proof is to construct a feasible sequence whose limiting direction is w and show that the property $f(z_k) \geq f(x^*)$ implies that (12.57) holds.

Since $w \in \mathcal{C}(x^*, \lambda^*) \subset \mathcal{F}(x^*)$, we can use the technique in the proof of Lemma 12.2 to choose a sequence $\{t_k\}$ of positive scalars and to construct a feasible sequence $\{z_k\}$ approaching x^* such that

$$\lim_{k \rightarrow \infty} \frac{z_k - x^*}{t_k} = w, \quad (12.58)$$

which we can write also as (12.58) that

$$z_k - x^* = t_k w + o(t_k). \quad (12.59)$$

Because of the construction technique for $\{z_k\}$, we have from formula (12.42) that

$$c_i(z_k) = t_k \nabla c_i(x^*)^T w, \quad \text{for all } i \in \mathcal{A}(x^*) \quad (12.60)$$

From (12.33), (12.60), and (12.54), we have

$$\begin{aligned} \mathcal{L}(z_k, \lambda^*) &= f(z_k) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i^* c_i(z_k) \\ &= f(z_k) - t_k \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla c_i(x^*)^T w \\ &= f(z_k), \end{aligned} \quad (12.61)$$

On the other hand, we can perform a Taylor series expansion to obtain an estimate of $\mathcal{L}(z_k, \lambda^*)$ near x^* . By using Taylor's theorem expression (2.6) and continuity of the Hessians $\nabla^2 f$ and $\nabla^2 c_i$, $i \in \mathcal{E} \cup \mathcal{I}$, we obtain

$$\begin{aligned} \mathcal{L}(z_k, \lambda^*) &= \mathcal{L}(x^*, \lambda^*) + (z_k - x^*)^T \nabla_x \mathcal{L}(x^*, \lambda^*) \\ &\quad + \frac{1}{2} (z_k - x^*)^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) (z_k - x^*) + o(\|z_k - x^*\|^2). \end{aligned} \quad (12.62)$$

By the complementarity conditions (12.34e), we have $\mathcal{L}(x^*, \lambda^*) = f(x^*)$. From (12.34a), the second term on the right-hand side is zero. Hence, using (12.59), we can rewrite (12.62) as

$$\mathcal{L}(z_k, \lambda^*) = f(x^*) + \frac{1}{2}t_k^2 w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) + o(t_k^2). \quad (12.63)$$

By substituting into (12.63), we obtain

$$f(z_k) = f(x^*) + \frac{1}{2}t_k^2 w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w + o(t_k^2). \quad (12.64)$$

If $w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w < 0$, then (12.64) would imply that $f(z_k) < f(x^*)$ for all k sufficiently large, contradicting the fact that x^* is a local solution. Hence, the condition (12.57) must hold, as claimed. \square

Sufficient conditions are conditions on f and $c_i, i \in \mathcal{E} \cup \mathcal{I}$, that ensure that x^* is a local solution of the problem (12.1). (They take the opposite tack to necessary conditions, which assume that x^* is a local solution and deduce properties of f and c_i , for the active indices i .) The second-order sufficient condition stated in the next theorem looks very much like the necessary condition just discussed, but it differs in that the constraint qualification is not required, and the inequality in (12.57) is replaced by a strict inequality.

Theorem 12.6 (Second-Order Sufficient Conditions).

Suppose that for some feasible point $x^ \in \mathbb{R}^n$ there is a Lagrange multiplier vector λ^* such that the KKT conditions (12.34) are satisfied. Suppose also that*

$$w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w > 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*), w \neq 0. \quad (12.65)$$

Then x^ is a strict local solution for (12.1).*

PROOF. First, note that the set $\bar{\mathcal{C}} = \{d \in \mathcal{C}(x^*, \lambda^*) \mid \|d\| = 1\}$ is a compact subset of $\mathcal{C}(x^*, \lambda^*)$, so by (12.65), the minimizer of $d^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) d$ over this set is a strictly positive number, say σ . Since $\mathcal{C}(x^*, \lambda^*)$ is a cone, we have that $(w/\|w\|) \in \bar{\mathcal{C}}$ if and only if $w \in \mathcal{C}(x^*, \lambda^*)$, $w \neq 0$. Therefore, condition (12.65) by

$$w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w \geq \sigma \|w\|^2, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*), \quad (12.66)$$

for $\sigma > 0$ defined as above. (Note that this inequality holds trivially for $w = 0$.)

We prove the result by showing that every feasible sequence $\{z_k\}$ approaching x^* has $f(z_k) \geq f(x^*) + (\sigma/4)\|z_k - x^*\|^2$, for all k sufficiently large. Suppose for contradiction that this is *not* the case, and that there is a sequence $\{z_k\}$ approaching x^* with

$$f(z_k) < f(x^*) + (\sigma/4)\|z_k - x^*\|^2, \quad \text{for all } k \text{ sufficiently large.} \quad (12.67)$$

By taking a subsequence if necessary, we can identify a limiting direction d such that

$$\lim_{k \rightarrow \infty} \frac{z_k - x^*}{\|z_k - x^*\|} = d. \quad (12.68)$$

We have from Lemma 12.2(i) and Definition 12.3 that $d \in \mathcal{F}(x^*)$. From (12.33) and the facts that $\lambda_i^* \geq 0$ and $c_i(z_k) \geq 0$ for $i \in \mathcal{I}$ and $c_i(z_k) = 0$ for $i \in \mathcal{E}$, we have that

$$\mathcal{L}(z_k, \lambda^*) = f(z_k) - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* c_i(z_k) \leq f(z_k), \quad (12.69)$$

while the Taylor series approximation (12.63) from the proof of Theorem 12.5 continues to hold.

If d were *not* in $\mathcal{C}(x^*, \lambda^*)$, we could identify some index $j \in \mathcal{A}(x^*) \cap \mathcal{I}$ such that the strict positivity condition

$$\lambda_j^* \nabla c_j(x^*)^T d > 0 \quad (12.70)$$

is satisfied, while for the remaining indices $i \in \mathcal{A}(x^*)$, we have

$$\lambda_i^* \nabla c_i(x^*)^T d \geq 0.$$

From Taylor's theorem and (12.68), we have for this particular value of j that

$$\begin{aligned} \lambda_j^* c_j(z_k) &= \lambda_j^* c_j(x^*) + \lambda_j^* \nabla c_j(x^*)^T (z_k - x^*) + o(\|z_k - x^*\|) \\ &= \|z_k - x^*\| \lambda_j^* \nabla c_j(x^*)^T d + o(\|z_k - x^*\|). \end{aligned}$$

Hence, from (12.69), we have that

$$\begin{aligned} \mathcal{L}(z_k, \lambda^*) &= f(z_k) - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* c_i(z_k) \\ &\leq f(z_k) - \lambda_j^* c_j(z_k) \\ &\leq f(z_k) - \|z_k - x^*\| \lambda_j^* \nabla c_j(x^*)^T d + o(\|z_k - x^*\|). \end{aligned} \quad (12.71)$$

From the Taylor series estimate (12.63), we have meanwhile that

$$\mathcal{L}(z_k, \lambda^*) = f(x^*) + O(\|z_k - x^*\|^2),$$

and by combining with (12.71), we obtain

$$f(z_k) \geq f(x^*) + \|z_k - x^*\| \lambda_j^* \nabla c_j(x^*)^T d + o(\|z_k - x^*\|).$$

Because of (12.70), this inequality is incompatible with (12.67). We conclude that $d \in \mathcal{C}(x^*, \lambda^*)$, and hence $d^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) d \geq \sigma$.

By combining the Taylor series estimate (12.63) with (12.69) and using (12.68), we obtain

$$\begin{aligned} f(z_k) &\geq f(x^*) + \frac{1}{2}(z_k - x^*)^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*)(z_k - x^*) + o(\|z_k - x^*\|^2) \\ &= f(x^*) + \frac{1}{2} d^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) d \|z_k - x^*\|^2 + o(\|z_k - x^*\|^2) \\ &\geq f(x^*) + (\sigma/2) \|z_k - x^*\|^2 + o(\|z_k - x^*\|^2). \end{aligned}$$

This inequality yields the contradiction to (12.67). We conclude that every feasible sequence $\{z_k\}$ approaching x^* must satisfy $f(z_k) \geq f(x^*) + (\sigma/4) \|z_k - x^*\|^2$, for all k sufficiently large, so x^* is a strict local solution. \square

\square **EXAMPLE 12.8** (EXAMPLE 12.2, ONE MORE TIME)

We now return to Example 12.2 to check the second-order conditions for problem (12.18). In this problem we have $f(x) = x_1 + x_2$, $c_1(x) = 2 - x_1^2 - x_2^2$, $\mathcal{E} = \emptyset$, and $\mathcal{I} = \{1\}$. The Lagrangian is

$$\mathcal{L}(x, \lambda) = (x_1 + x_2) - \lambda_1(2 - x_1^2 - x_2^2),$$

and it is easy to show that the KKT conditions (12.34) are satisfied by $x^* = (-1, -1)^T$, with $\lambda_1^* = \frac{1}{2}$. The Lagrangian Hessian at this point is

$$\nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) = \begin{bmatrix} 2\lambda_1^* & 0 \\ 0 & 2\lambda_1^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

This matrix is positive definite, so it certainly satisfies the conditions of Theorem 12.6. We conclude that $x^* = (-1, -1)^T$ is a strict local solution for (12.18). (In fact, it is the global solution of this problem, since, as we note later, this problem is a convex programming problem.) \square

\square **EXAMPLE 12.9**

For a more complex example, consider the problem

$$\min -0.1(x_1 - 4)^2 + x_2^2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 1 \geq 0, \quad (12.72)$$

in which we seek to minimize a nonconvex function over the *exterior* of the unit circle. Obviously, the objective function is not bounded below on the feasible region, since we can take the feasible sequence

$$\begin{bmatrix} 10 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 20 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 30 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 40 \\ 0 \end{bmatrix},$$

and note that $f(x)$ approaches $-\infty$ along this sequence. Therefore, no global solution exists, but it may still be possible to identify a strict local solution on the boundary of the constraint. We search for such a solution by using the KKT conditions (12.34) and the second-order conditions of Theorem 12.6.

By defining the Lagrangian for (12.72) in the usual way, it is easy to verify that

$$\nabla_x \mathcal{L}(x, \lambda) = \begin{bmatrix} -0.2(x_1 - 4) - 2\lambda_1 x_1 \\ 2x_2 - 2\lambda_1 x_2 \end{bmatrix}, \quad (12.73a)$$

$$\nabla_{xx}^2 \mathcal{L}(x, \lambda) = \begin{bmatrix} -0.2 - 2\lambda_1 & 0 \\ 0 & 2 - 2\lambda_1 \end{bmatrix}. \quad (12.73b)$$

The point $x^* = (1, 0)^T$ satisfies the KKT conditions with $\lambda_1^* = 0.3$ and the active set $\mathcal{A}(x^*) = \{1\}$. To check that the second-order sufficient conditions are satisfied at this point, we note that

$$\nabla c_1(x^*) = \begin{bmatrix} 2 \\ 0 \end{bmatrix},$$

so that the set \mathcal{C} defined in (12.53) is simply

$$\mathcal{C}(x^*, \lambda^*) = \{(0, w_2)^T \mid w_2 \in \mathbb{R}\}.$$

Now, by substituting x^* and λ^* into (12.73b), we have for any $w \in \mathcal{C}(x^*, \lambda^*)$ with $w \neq 0$ that $w_2 \neq 0$ and thus

$$w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w = \begin{bmatrix} 0 \\ w_2 \end{bmatrix}^T \begin{bmatrix} -0.4 & 0 \\ 0 & 1.4 \end{bmatrix} \begin{bmatrix} 0 \\ w_2 \end{bmatrix} = 1.4w_2^2 > 0.$$

Hence, the second-order sufficient conditions are satisfied, and we conclude from Theorem 12.6 that $(1, 0)^T$ is a strict local solution for (12.72). □

SECOND-ORDER CONDITIONS AND PROJECTED HESSIANS

The second-order conditions are sometimes stated in a form that is slightly weaker but easier to verify than (12.57) and (12.65). This form uses a two-sided projection of the Lagrangian Hessian $\nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*)$ onto subspaces that are related to $\mathcal{C}(x^*, \lambda^*)$.

The simplest case is obtained when the multiplier λ^* that satisfies the KKT conditions (12.34) is unique (as happens, for example, when the LICQ condition holds) *and* strict complementarity holds. In this case, the definition (12.53) of $\mathcal{C}(x^*, \lambda^*)$ reduces to

$$\mathcal{C}(x^*, \lambda^*) = \text{Null} \left[\nabla c_i(x^*)^T \right]_{i \in \mathcal{A}(x^*)} = \text{Null } A(x^*),$$

where $A(x^*)$ is defined as in (12.37). In other words, $\mathcal{C}(x^*, \lambda^*)$ is the null space of the matrix whose rows are the active constraint gradients at x^* . As in (12.39), we can define the matrix Z with full column rank whose columns span the space $\mathcal{C}(x^*, \lambda^*)$; that is,

$$\mathcal{C}(x^*, \lambda^*) = \{Zu \mid u \in \mathbb{R}^{|\mathcal{A}(x^*)|}\}.$$

Hence, the condition (12.57) in Theorem 12.5 can be restated as

$$u^T Z^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) Zu \geq 0 \quad \text{for all } u,$$

or, more succinctly,

$$Z^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) Z \quad \text{is positive semidefinite.}$$

Similarly, the condition (12.65) in Theorem 12.6 can be restated as

$$Z^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) Z \quad \text{is positive definite.}$$

As we show next, Z can be computed numerically, so that the positive (semi)definiteness conditions can actually be checked by forming these matrices and finding their eigenvalues.

One way to compute the matrix Z is to apply a QR factorization to the matrix of active constraint gradients whose null space we seek. In the simplest case above (in which the multiplier λ^* is unique and strictly complementary holds), we define $A(x^*)$ as in (12.37) and write the QR factorization of its transpose as

$$A(x^*)^T = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R, \quad (12.74)$$

where R is a square upper triangular matrix and Q is $n \times n$ orthogonal. If R is nonsingular, we can set $Z = Q_2$. If R is singular (indicating that the active constraint gradients are linearly dependent), a slight enhancement of this procedure that makes use of column pivoting during the QR procedure can be used to identify Z .

12.6 OTHER CONSTRAINT QUALIFICATIONS

We now reconsider constraint qualifications, the conditions discussed in Sections 12.2 and 12.4 that ensure that the linearized approximation to the feasible set Ω captures the essential shape of Ω in a neighborhood of x^* .

One situation in which the linearized feasible direction set $\mathcal{F}(x^*)$ is obviously an adequate representation of the actual feasible set occurs when all the active constraints are already linear; that is,

$$c_i(x) = a_i^T x + b_i, \quad (12.75)$$

for some $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. It is not difficult to prove a version of Lemma 12.2 for this situation.

Lemma 12.7.

Suppose that at some $x^ \in \Omega$, all active constraints $c_i(\cdot)$, $i \in \mathcal{A}(x^*)$, are linear functions. Then $\mathcal{F}(x^*) = T_\Omega(x^*)$.*

PROOF. We have from Lemma 12.2 (i) that $T_\Omega(x^*) \subset \mathcal{F}(x^*)$. To prove that $\mathcal{F}(x^*) \subset T_\Omega(x^*)$, we choose an arbitrary $w \in \mathcal{F}(x^*)$ and show that $w \in T_\Omega(x^*)$. By Definition 12.3 and the form (12.75) of the constraints, we have

$$\mathcal{F}(x^*) = \left\{ d \mid \begin{array}{ll} a_i^T d = 0, & \text{for all } i \in \mathcal{E}, \\ a_i^T d \geq 0, & \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\}.$$

First, note that there is a positive scalar \bar{t} such that the inactive constraints remain inactive at $x^* + tw$, for all $t \in [0, \bar{t}]$, that is,

$$c_i(x^* + tw) > 0, \quad \text{for all } i \in \mathcal{I} \setminus \mathcal{A}(x^*) \text{ and all } t \in [0, \bar{t}].$$

Now define the sequence z_k by

$$z_k = x^* + (\bar{t}/k)w, \quad k = 1, 2, \dots$$

Since $a_i^T w \geq 0$ for all $i \in \mathcal{I} \cap \mathcal{A}(x^*)$, we have

$$c_i(z_k) = c_i(z_k) - c_i(x^*) = a_i^T(z_k - x^*) = \frac{\bar{t}}{k} a_i^T w \geq 0, \quad \text{for all } i \in \mathcal{I} \cap \mathcal{A}(x^*),$$

so that z_k is feasible with respect to the active inequality constraints c_i , $i \in \mathcal{I} \cap \mathcal{A}(x^*)$. By the choice of \bar{t} , we find that z_k is also feasible with respect to the inactive inequality constraints

for all k sufficiently large. Hence, since f is continuously differentiable, we have from Taylor's theorem (2.4) that

$$f(z_k) - f(x^*) = t_k \nabla f(x^*)^T d + o(t_k) \geq 0.$$

By dividing by t_k and taking limits as $k \rightarrow \infty$, we have

$$\nabla f(x^*)^T d \geq 0.$$

Recall that d was an arbitrary member of $T_\Omega(x^*)$, so we have $-\nabla f(x^*)^T d \leq 0$ for all $d \in T_\Omega(x^*)$. We conclude from Definition 12.7 that $-\nabla f(x^*) \in N_\Omega(x^*)$. \square

This result suggests a close relationship between $N_\Omega(x^*)$ and the conic combination of active constraint gradients given by (12.50). When the linear independence constraint qualification holds, identical (to within a change of sign).

Lemma 12.9.

Suppose that the LICQ assumption (Definition 12.4) holds at x^ . Then the normal cone $N_\Omega(x^*)$ is simply $-N$, where N is the set defined in (12.50).*

PROOF. The proof follows from Farkas' Lemma (Lemma 12.4) and Definition 12.7 of $N_\Omega(x^*)$. From Lemma 12.4, we have that

$$g \in N \Rightarrow g^T d \geq 0 \text{ for all } d \in \mathcal{F}(x^*).$$

Since we have $\mathcal{F}(x^*) = T_\Omega(x^*)$ from Lemma 12.2, it follows by switching the sign of this expression that

$$g \in -N \Rightarrow g^T d \leq 0 \text{ for all } d \in T_\Omega(x^*).$$

We conclude from Definition 12.7 that $N_\Omega(x^*) = -N$, as claimed. \square

12.8 LAGRANGE MULTIPLIERS AND SENSITIVITY

The importance of Lagrange multipliers in optimality theory should be clear, but what of their intuitive significance? We show in this section that each Lagrange multiplier λ_i^* tells us something about the *sensitivity* of the optimal objective value $f(x^*)$ to the presence of the constraint c_i . To put it another way, λ_i^* indicates how hard f is “pushing” or “pulling” the solution x^* against the particular constraint c_i .

We illustrate this point with some informal analysis. When we choose an inactive constraint $i \notin \mathcal{A}(x^*)$ such that $c_i(x^*) > 0$, the solution x^* and function value $f(x^*)$ are

indifferent to whether this constraint is present or not. If we perturb c_i by a tiny amount, it will still be inactive and x^* will still be a local solution of the optimization problem. Since $\lambda_i^* = 0$ from (12.34e), the Lagrange multiplier indicates accurately that constraint i is not significant.

Suppose instead that constraint i is active, and let us perturb the right-hand-side of this constraint a little, requiring, say, that $c_i(x) \geq -\epsilon \|\nabla c_i(x^*)\|$ instead of $c_i(x) \geq 0$. Suppose that ϵ is sufficiently small that the perturbed solution $x^*(\epsilon)$ still has the same set of active constraints, and that the Lagrange multipliers are not much affected by the perturbation. (These conditions can be made more rigorous with the help of strict complementarity and second-order conditions.) We then find that

$$\begin{aligned} -\epsilon \|\nabla c_i(x^*)\| &= c_i(x^*(\epsilon)) - c_i(x^*) \approx (x^*(\epsilon) - x^*)^T \nabla c_i(x^*), \\ 0 &= c_j(x^*(\epsilon)) - c_j(x^*) \approx (x^*(\epsilon) - x^*)^T \nabla c_j(x^*), \\ &\text{for all } j \in \mathcal{A}(x^*) \text{ with } j \neq i. \end{aligned}$$

The value of $f(x^*(\epsilon))$, meanwhile, can be estimated with the help of (12.34a). We have

$$\begin{aligned} f(x^*(\epsilon)) - f(x^*) &\approx (x^*(\epsilon) - x^*)^T \nabla f(x^*) \\ &= \sum_{j \in \mathcal{A}(x^*)} \lambda_j^* (x^*(\epsilon) - x^*)^T \nabla c_j(x^*) \\ &\approx -\epsilon \|\nabla c_i(x^*)\| \lambda_i^*. \end{aligned}$$

By taking limits, we see that the family of solutions $x^*(\epsilon)$ satisfies

$$\frac{df(x^*(\epsilon))}{d\epsilon} = -\lambda_i^* \|\nabla c_i(x^*)\|. \quad (12.80)$$

A sensitivity analysis of this problem would conclude that if $\lambda_i^* \|\nabla c_i(x^*)\|$ is large, then the optimal value is sensitive to the placement of the i th constraint, while if this quantity is small, the dependence is not too strong. If λ_i^* is exactly zero for some active constraint, small perturbations to c_i in some directions will hardly affect the optimal objective value at all; the change is zero, to first order.

This discussion motivates the definition below, which classifies constraints according to whether or not their corresponding Lagrange multiplier is zero.

Definition 12.8.

Let x^ be a solution of the problem (12.1), and suppose that the KKT conditions (12.34) are satisfied. We say that an inequality constraint c_i is strongly active or binding if $i \in \mathcal{A}(x^*)$ and $\lambda_i^* > 0$ for some Lagrange multiplier λ^* satisfying (12.34). We say that c_i is weakly active if $i \in \mathcal{A}(x^*)$ and $\lambda_i^* = 0$ for all λ^* satisfying (12.34).*

Note that the analysis above is independent of scaling of the individual constraints. For instance, we might change the formulation of the problem by replacing some active

constraint c_i by $10c_i$. The new problem will actually be equivalent (that is, it has the same feasible set and same solution), but the optimal multiplier λ_i^* corresponding to c_i will be replaced by $\lambda_i^*/10$. However, since $\|\nabla c_i(x^*)\|$ is replaced by $10\|\nabla c_i(x^*)\|$, the product $\lambda_i^*\|\nabla c_i(x^*)\|$ does not change. If, on the other hand, we replace the objective function f by $10f$, the multipliers λ_i^* in (12.34) all will need to be replaced by $10\lambda_i^*$. Hence in (12.80) we see that the sensitivity of f to perturbations has increased by a factor of 10, which is exactly what we would expect.

12.9 DUALITY

In this section we present some elements of the duality theory for nonlinear programming. This theory is used to motivate and develop some important algorithms, including the augmented Lagrangian algorithms of Chapter 17. In its full generality, duality theory ranges beyond nonlinear programming to provide important insight into the fields of convex nonsmooth optimization and even discrete optimization. Its specialization to linear programming proved central to the development of that area; see Chapter 13. (We note that the discussion of linear programming duality in Section 13.1 can be read without consulting this section first.)

Duality theory shows how we can construct an alternative problem from the functions and data that define the original optimization problem. This alternative “dual” problem is related to the original problem (which is sometimes referred to in this context as the “primal” for purposes of contrast) in fascinating ways. In some cases, the dual problem is easier to solve computationally than the original problem. In other cases, the dual can be used to obtain easily a lower bound on the optimal value of the objective for the primal problem. As remarked above, the dual has also been used to design algorithms for solving the primal problem.

Our results in this section are mostly restricted to the special case of (12.1) in which there are no equality constraints and the objective f and the negatives of the inequality constraints $-c_i$ are all convex functions. For simplicity we assume that there are m inequality constraints labelled $1, 2, \dots, m$ and rewrite (12.1) as follows:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } c_i(x) \geq 0, \quad i = 1, 2, \dots, m.$$

If we assemble the constraints into a vector function

$$c(x) \stackrel{\text{def}}{=} (c_1(x), c_2(x), \dots, c_m(x))^T,$$

we can write the problem as

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } c(x) \geq 0, \quad (12.81)$$

for which the Lagrangian function (12.16) with Lagrange multiplier vector $\lambda \in \mathbb{R}^m$ is

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^T c(x).$$

We define the dual objective function $q : \mathbb{R}^n \rightarrow \mathbb{R}$ as follows:

$$q(\lambda) \stackrel{\text{def}}{=} \inf_x \mathcal{L}(x, \lambda). \quad (12.82)$$

In many problems, this infimum is $-\infty$ for some values of λ . We define the domain of q as the set of λ values for which q is finite, that is,

$$\mathcal{D} \stackrel{\text{def}}{=} \{\lambda \mid q(\lambda) > -\infty\}. \quad (12.83)$$

Note that calculation of the infimum in (12.82) requires finding the *global* minimizer of the function $\mathcal{L}(\cdot, \lambda)$ for the given λ which, as we have noted in Chapter 2, may be extremely difficult in practice. However, when f and $-c_i$ are convex functions and $\lambda \geq 0$ (the case in which we are most interested), the function $\mathcal{L}(\cdot, \lambda)$ is also convex. In this situation, all local minimizers are global minimizers (as we verify in Exercise 12.4), so computation of $q(\lambda)$ becomes a more practical proposition.

The dual problem to (12.81) is defined as follows:

$$\max_{\lambda \in \mathbb{R}^n} q(\lambda) \quad \text{subject to } \lambda \geq 0. \quad (12.84)$$

□ EXAMPLE 12.10

Consider the problem

$$\min_{(x_1, x_2)} 0.5(x_1^2 + x_2^2) \quad \text{subject to } x_1 - 1 \geq 0. \quad (12.85)$$

The Lagrangian is

$$\mathcal{L}(x_1, x_2, \lambda_1) = 0.5(x_1^2 + x_2^2) - \lambda_1(x_1 - 1).$$

If we hold λ_1 fixed, this is a convex function of $(x_1, x_2)^T$. Therefore, the infimum with respect to $(x_1, x_2)^T$ is achieved when the partial derivatives with respect to x_1 and x_2 are zero, that is,

$$x_1 - \lambda_1 = 0, \quad x_2 = 0.$$

By substituting these infimal values into $\mathcal{L}(x_1, x_2, \lambda_1)$ we obtain the dual objective (12.82):

$$q(\lambda_1) = 0.5(\lambda_1^2 + 0) - \lambda_1(\lambda_1 - 1) = -0.5\lambda_1^2 + \lambda_1.$$

Hence, the dual problem (12.84) is

$$\max_{\lambda_1 \geq 0} -0.5\lambda_1^2 + \lambda_1, \quad (12.86)$$

which clearly has the solution $\lambda_1 = 1$. □

In the remainder of this section, we show how the dual problem is related to (12.81). Our first result concerns concavity of q .

Theorem 12.10.

The function q defined by (12.82) is concave and its domain \mathcal{D} is convex.

PROOF. For any λ^0 and λ^1 in \mathbb{R}^m , any $x \in \mathbb{R}^n$, and any $\alpha \in [0, 1]$, we have

$$\mathcal{L}(x, (1 - \alpha)\lambda^0 + \alpha\lambda^1) = (1 - \alpha)\mathcal{L}(x, \lambda^0) + \alpha\mathcal{L}(x, \lambda^1).$$

By taking the infimum of both sides in this expression, using the definition (12.82), and using the results that the infimum of a sum is greater than or equal to the sum of infimums, we obtain

$$q((1 - \alpha)\lambda^0 + \alpha\lambda^1) \geq (1 - \alpha)q(\lambda^0) + \alpha q(\lambda^1),$$

confirming concavity of q . If both λ^0 and λ^1 belong to \mathcal{D} , this inequality implies that $q((1 - \alpha)\lambda^0 + \alpha\lambda^1) \geq -\infty$ also, and therefore $(1 - \alpha)\lambda^0 + \alpha\lambda^1 \in \mathcal{D}$, verifying convexity of \mathcal{D} . □

The optimal value of the dual problem (12.84) gives a lower bound on the optimal objective value for the primal problem (12.81). This observation is a consequence of the following *weak duality* result.

Theorem 12.11 (Weak Duality).

For any \bar{x} feasible for (12.81) and any $\bar{\lambda} \geq 0$, we have $q(\bar{\lambda}) \leq f(\bar{x})$.

PROOF.

$$q(\bar{\lambda}) = \inf_x f(x) - \bar{\lambda}^T c(x) \leq f(\bar{x}) - \bar{\lambda}^T c(\bar{x}) \leq f(\bar{x}),$$

where the final inequality follows from $\bar{\lambda} \geq 0$ and $c(\bar{x}) \geq 0$. □

For the remaining results, we note that the KKT conditions (12.34) specialized to (12.81) are as follows:

$$\nabla f(\bar{x}) - \nabla c(\bar{x})\bar{\lambda} = 0, \quad (12.87a)$$

$$c(\bar{x}) \geq 0, \quad (12.87b)$$

$$\bar{\lambda} \geq 0, \quad (12.87c)$$

$$\bar{\lambda}_i c_i(\bar{x}) = 0, \quad i = 1, 2, \dots, m, \quad (12.87d)$$

where $\nabla c(x)$ is the $n \times m$ matrix defined by $\nabla c(x) = [\nabla c_1(x), \nabla c_2(x), \dots, \nabla c_m(x)]$.

The next result shows that optimal Lagrange multipliers for (12.81) are solutions of the dual problem (12.84) under certain conditions. It is essentially due to Wolfe [309].

Theorem 12.12.

Suppose that \bar{x} is a solution of (12.81) and that f and $-c_i, i = 1, 2, \dots, m$ are convex functions on \mathbb{R}^n that are differentiable at \bar{x} . Then any $\bar{\lambda}$ for which $(\bar{x}, \bar{\lambda})$ satisfies the KKT conditions (12.87) is a solution of (12.84).

PROOF. Suppose that $(\bar{x}, \bar{\lambda})$ satisfies (12.87). We have from $\bar{\lambda} \geq 0$ that $\mathcal{L}(\cdot, \bar{\lambda})$ is a convex and differentiable function. Hence, for any x , we have

$$\mathcal{L}(x, \bar{\lambda}) \geq \mathcal{L}(\bar{x}, \bar{\lambda}) + \nabla_x \mathcal{L}(\bar{x}, \bar{\lambda})^T (x - \bar{x}) = \mathcal{L}(\bar{x}, \bar{\lambda}),$$

where the last equality follows from (12.87a). Therefore, we have

$$q(\bar{\lambda}) = \inf_x \mathcal{L}(x, \bar{\lambda}) = \mathcal{L}(\bar{x}, \bar{\lambda}) = f(\bar{x}) - \bar{\lambda}^T c(\bar{x}) = f(\bar{x}),$$

where the last equality follows from (12.87d). Since from Theorem 12.11, we have $q(\lambda) \leq f(\bar{x})$ for all $\lambda \geq 0$ it follows immediately from $q(\bar{\lambda}) = f(\bar{x})$ that $\bar{\lambda}$ is a solution of (12.84). \square

Note that if the functions are continuously differentiable and a constraint qualification such as LICQ holds at \bar{x} , then an optimal Lagrange multiplier is guaranteed to exist, by Theorem 12.1.

In Example 12.10, we see that $\lambda_1 = 1$ is both an optimal Lagrange multiplier for the problem (12.85) and a solution of (12.86). Note too that the optimal objective for both problems is 0.5.

We prove a partial converse of Theorem 12.12, which shows that solutions to the dual problem (12.84) can sometimes be used to derive solutions to the original problem (12.81). The essential condition is strict convexity of the function $\mathcal{L}(\cdot, \hat{\lambda})$ for a certain value $\hat{\lambda}$. We note that this condition holds if either f is strictly convex (as is the case in Example 12.10) or if c_i is strictly convex for some $i = 1, 2, \dots, m$ with $\hat{\lambda}_i > 0$.

Theorem 12.13.

Suppose that f and $-c_i, i = 1, 2, \dots, m$ are convex and continuously differentiable on \mathbb{R}^n . Suppose that \bar{x} is a solution of (12.81) at which LICQ holds. Suppose that $\hat{\lambda}$ solves (12.84) and that the infimum in $\inf_x \mathcal{L}(x, \hat{\lambda})$ is attained at \hat{x} . Assume further than $\mathcal{L}(\cdot, \hat{\lambda})$ is a strictly convex function. Then $\bar{x} = \hat{x}$ (that is, \hat{x} is the unique solution of (12.81)), and $f(\bar{x}) = \mathcal{L}(\hat{x}, \hat{\lambda})$.

PROOF. Assume for contradiction that $\bar{x} \neq \hat{x}$. From Theorem 12.1, because of the LICQ assumption, there exists $\bar{\lambda}$ satisfying (12.87). Hence, from Theorem 12.12, we have that $\bar{\lambda}$ also solves (12.84), so that

$$\mathcal{L}(\bar{x}, \bar{\lambda}) = q(\bar{\lambda}) = q(\hat{\lambda}) = \mathcal{L}(\hat{x}, \hat{\lambda}).$$

Because $\hat{x} = \arg \min_x \mathcal{L}(x, \hat{\lambda})$, we have from Theorem 2.2 that $\nabla_x \mathcal{L}(\hat{x}, \hat{\lambda}) = 0$. Moreover, by strict convexity of $\mathcal{L}(\cdot, \hat{\lambda})$, it follows that

$$\mathcal{L}(\bar{x}, \hat{\lambda}) - \mathcal{L}(\hat{x}, \hat{\lambda}) > \nabla_x \mathcal{L}(\hat{x}, \hat{\lambda})^T (\bar{x} - \hat{x}) = 0.$$

Hence, we have

$$\mathcal{L}(\bar{x}, \hat{\lambda}) > \mathcal{L}(\hat{x}, \hat{\lambda}) = \mathcal{L}(\bar{x}, \bar{\lambda}),$$

so in particular we have

$$-\hat{\lambda}^T c(\bar{x}) > -\bar{\lambda}^T c(\bar{x}) = 0,$$

where the final equality follows from (12.87d). Since $\hat{\lambda} \geq 0$ and $c(\bar{x}) \geq 0$, this yields the contradiction, and we conclude that $\hat{x} = \bar{x}$, as claimed. \square

In Example 12.10, at the dual solution $\lambda_1 = 1$, the infimum of $\mathcal{L}(x_1, x_2, \lambda_1)$ is achieved at $(x_1, x_2) = (1, 0)^T$, which is the solution of the original problem (12.85).

An slightly different form of duality that is convenient for computations, known as the *Wolfe dual* [309], can be stated as follows:

$$\max_{x, \lambda} \mathcal{L}(x, \lambda) \tag{12.88a}$$

$$\text{subject to } \nabla_x \mathcal{L}(x, \lambda) = 0, \quad \lambda \geq 0. \tag{12.88b}$$

The following results explains the relationship of the Wolfe dual to (12.81).

Theorem 12.14.

Suppose that f and $-c_i, i = 1, 2, \dots, m$ are convex and continuously differentiable on \mathbb{R}^n . Suppose that $(\bar{x}, \bar{\lambda})$ is a solution pair of (12.81) at which LICQ holds. Then $(\bar{x}, \bar{\lambda})$ solves the problem (12.88).

PROOF. From the KKT conditions (12.87) we have that $(\bar{x}, \bar{\lambda})$ satisfies (12.88b), and that $\mathcal{L}(\bar{x}, \bar{\lambda}) = f(\bar{x})$. Therefore for any pair (x, λ) that satisfies (12.88b) we have that

$$\begin{aligned}\mathcal{L}(\bar{x}, \bar{\lambda}) &= f(\bar{x}) \\ &\geq f(\bar{x}) - \lambda^T c(\bar{x}) \\ &= \mathcal{L}(\bar{x}, \lambda) \\ &\geq \mathcal{L}(x, \lambda) + \nabla_x \mathcal{L}(x, \lambda)^T (\bar{x} - x) \\ &= \mathcal{L}(x, \lambda),\end{aligned}$$

where the second inequality follows from the convexity of $\mathcal{L}(\cdot, \lambda)$. We have therefore shown that $(\bar{x}, \bar{\lambda})$ maximizes \mathcal{L} over the constraints (12.88b), and hence solves (12.88). \square

\square **EXAMPLE 12.11** (LINEAR PROGRAMMING)

An important special case of (12.81) is the linear programming problem

$$\min c^T x \quad \text{subject to} \quad Ax - b \geq 0, \quad (12.89)$$

for which the dual objective is

$$q(\lambda) = \inf_x [c^T x - \lambda^T (Ax - b)] = \inf_x [(c - A^T \lambda)^T x + b^T \lambda].$$

If $c - A^T \lambda \neq 0$, the infimum is clearly $-\infty$ (we can set x to be a large negative multiple of $-(c - A^T \lambda)$ to make q arbitrarily large and negative). When $c - A^T \lambda = 0$, on the other hand, the dual objective is simply $b^T \lambda$. In maximizing q , we can exclude λ for which $c - A^T \lambda \neq 0$ from consideration (the maximum obviously cannot be attained at a point λ for which $q(\lambda) = -\infty$). Hence, we can write the dual problem (12.84) as follows:

$$\max_{\lambda} b^T \lambda \quad \text{subject to} \quad A^T \lambda = c, \quad \lambda \geq 0. \quad (12.90)$$

The Wolfe dual of (12.89) can be written as

$$\max_{\lambda} c^T x - \lambda^T (Ax - b) \quad \text{subject to} \quad A^T \lambda = c, \quad \lambda \geq 0,$$

and by substituting the constraint $A^T \lambda - c = 0$ into the objective we obtain (12.90) again.

For some matrices A , the dual problem (12.90) may be computationally easier to solve than the original problem (12.89). We discuss the possibilities further in Chapter 13. \square

□ EXAMPLE 12.12 (CONVEX QUADRATIC PROGRAMMING)

Consider

$$\min \frac{1}{2}x^T Gx + c^T x \quad \text{subject to} \quad Ax - b \geq 0, \quad (12.91)$$

where G is a symmetric positive definite matrix. The dual objective for this problem is

$$q(\lambda) = \inf_x \mathcal{L}(x, \lambda) = \inf_x \frac{1}{2}x^T Gx + c^T x - \lambda^T (Ax - b). \quad (12.92)$$

Since G is positive definite, since $\mathcal{L}(\cdot, \lambda)$ is a strictly convex quadratic function, the infimum is achieved when $\nabla_x \mathcal{L}(x, \lambda) = 0$, that is,

$$Gx + c - A^T \lambda = 0. \quad (12.93)$$

Hence, we can substitute for x in the infimum expression and write the dual objective explicitly as follows:

$$q(\lambda) = -\frac{1}{2}(A^T \lambda - c)^T G^{-1}(A^T \lambda - c) + b^T \lambda.$$

Alternatively, we can write the Wolfe dual form (12.88) by retaining x as a variable and including the constraint (12.93) explicitly in the dual problem, to obtain

$$\begin{aligned} \max_{(\lambda, x)} \quad & \frac{1}{2}x^T Gx + c^T x - \lambda^T (Ax - b) \\ \text{subject to} \quad & Gx + c - A^T \lambda = 0, \quad \lambda \geq 0. \end{aligned} \quad (12.94)$$

To make it clearer that the objective is concave, we can use the constraint to substitute $(c - A^T \lambda)^T x = -x^T Gx$ in the objective, and rewrite the dual formulation as follows:

$$\max_{(\lambda, x)} -\frac{1}{2}x^T Gx + \lambda^T b, \quad \text{subject to} \quad Gx + c - A^T \lambda = 0, \quad \lambda \geq 0. \quad (12.95)$$

□

Note that the Wolfe dual form requires only positive *semidefiniteness* of G .

NOTES AND REFERENCES

The theory of constrained optimization is discussed in many books on numerical optimization. The discussion in Fletcher [101, Chapter 9] is similar to ours, though a little

terser, and includes additional material on duality. Bertsekas [19, Chapter 3] emphasizes the role of duality and discusses sensitivity of the solution with respect to the active constraints in some detail. The classic treatment of Mangasarian [198] is particularly notable for its thorough description of constraint qualifications. It also has an extensive discussion of theorems of the alternative [198, Chapter 2], placing Farkas' Lemma firmly in the context of other related results.

The KKT conditions were described in a 1951 paper of Kuhn and Tucker [185], though they were derived earlier (and independently) in an unpublished 1939 master's thesis of W. Karush. Lagrange multipliers and optimality conditions for general problems (including nonsmooth problems) are described in the deep and wide-ranging article of Rockafellar [270].

Duality theory for nonlinear programming is described in the books of Rockafellar [198] and Bertsekas [19]; the latter treatment is particularly extensive and general. The material in Section 12.9 is adapted from these sources.

We return to our claim that the set N defined by

$$N = \{By + Ct \mid y \geq 0\},$$

(where B and C are matrices of dimension $n \times m$ and $n \times p$, respectively, and y and t are vectors of appropriate dimensions; see (12.45)) is a closed set. This fact is needed in the proof of Lemma 12.4 to ensure that the solution of the projection subproblem (12.47) is well-defined. The following technical result is well known; the proof given below is due to R. Byrd.

Lemma 12.15.

The set N is closed.

PROOF. By splitting t into positive and negative parts, it is easy to see that

$$N = \left\{ \begin{bmatrix} B & C & -C \end{bmatrix} \begin{bmatrix} y \\ t^+ \\ t^- \end{bmatrix} \mid \begin{bmatrix} y \\ t^+ \\ t^- \end{bmatrix} \geq 0 \right\}.$$

Hence, we can assume without loss of generality that N has the form

$$N = \{By \mid y \geq 0\}.$$

Suppose that B has dimensions $n \times m$.

First, we show that for any $s \in N$, we can write $s = B_I y_I$ with $y_I \geq 0$, where $I \subset \{1, 2, \dots, m\}$, B_I is the column submatrix of B indexed by I with full column rank, and I has minimum cardinality. To prove this claim, we assume for contradiction that $K \subset \{1, 2, \dots, m\}$ is an index set with minimal cardinality such that $s = B_K y_K$, $y_K \geq 0$, yet

the columns of B_K are linearly dependent. Since K is minimal, y_K has no zero components. We then have a nonzero vector w such that $B_K w = 0$. Since $s = B_K(y_K + \tau w)$ for any τ , we can increase or decrease τ from 0 until one or more components of $y_K + \tau w$ become zero, while the other components remain positive. We define \bar{K} by removing the indices from K that correspond to zero components of $y_K + \tau w$, and define $\bar{y}_{\bar{K}}$ to be the vector of strictly positive components of $y_K + \tau w$. We then have that $s = B_{\bar{K}} \bar{y}_{\bar{K}}$ and $\bar{y}_{\bar{K}} \geq 0$, contradicting our assumption that K was the set of minimal cardinality with this property.

Now let $\{s^k\}$ be a sequence with $s^k \in N$ for all k and $s^k \rightarrow s$. We prove the lemma by showing that $s \in N$. By the claim of the previous paragraph, for all k we can write $s^k = B_{I_k} y_{I_k}^k$ with $y_{I_k}^k \geq 0$, I_k is minimal, and the columns of B_{I_k} are linearly independent. Since there are only finitely many possible choices of index set I_k , at least one index set occurs infinitely often in the sequence. By choosing such an index set I , we can take a subsequence if necessary and assume without loss of generality that $I_k \equiv I$ for all k . We then have that $s^k = A_I y_I^k$ with $y_I^k \geq 0$ and A_I has full column rank. Because of the latter property, we have that $A_I^T A_I$ is invertible, so that y_I^k is defined uniquely as follows:

$$y_I^k = (A_I^T A_I)^{-1} A_I^T s^k, \quad k = 0, 1, 2, \dots$$

By taking limits and using $s^k \rightarrow s$, we have that

$$y_I^k \rightarrow y_I \stackrel{\text{def}}{=} (A_I^T A_I)^{-1} A_I^T s,$$

and moreover $y_I \geq 0$, since $y_I^k \geq 0$ for all k . Hence we can write $s = B_I y_I$ with $y_I \geq 0$, and therefore $s \in N$. \square



EXERCISES





12.1 The following example from [268] with a single variable $x \in \mathbb{R}$ and a single equality constraint shows that strict local solutions are not necessarily isolated. Consider


$$\min_x x^2 \quad \text{subject to } c(x) = 0, \text{ where } c(x) = \begin{cases} x^6 \sin(1/x) = 0 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases} \quad (12.96)$$


- (a) Show that the constraint function is twice continuously differentiable at all x (including at $x = 0$) and that the feasible points are $x = 0$ and $x = 1/(k\pi)$ for all nonzero integers k .
- (b) Verify that each feasible point except $x = 0$ is an isolated local solution by showing that there is a neighborhood \mathcal{N} around each such point within which it is the only feasible point.

(c) Verify that $x = 0$ is a global solution and a strict local solution, but not an isolated local solution

 **12.2** Is an isolated local solution necessarily a strict local solution? Explain.


 **12.3** Does problem (12.4) have a finite or infinite number of local solutions? Use the first-order optimality conditions (12.34) to justify your answer.

 **12.4** If f is convex and the feasible region Ω is convex, show that local solutions of the problem (12.3) are also global solutions. Show that the set of global solutions is convex. (Hint: See Theorem 2.5.)

 **12.5** Let $v : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a smooth vector function and consider the unconstrained optimization problems of minimizing $f(x)$ where


$$f(x) = \|v(x)\|_\infty, \quad f(x) = \max_{i=1,2,\dots,m} v_i(x).$$


Reformulate these (generally nonsmooth) problems as smooth constrained optimization problems.


 **12.6** Can you perform a smooth reformulation as in the previous question when f is defined by


$$f(x) = \min_{i=1,2,\dots,m} f_i(x)?$$

(N.B. “min” not “max.”) Why or why not?

 **12.7** Show that the vector defined by (12.15) satisfies (12.14) when the first-order optimality condition (12.10) is not satisfied.

 **12.8** Verify that for the sequence $\{z_k\}$ defined by (12.30), the function $f(x) = x_1 + x_2$ satisfies $f(z_{k+1}) > f(z_k)$ for $k = 2, 3, \dots$ (Hint: Consider the trajectory $z(s) = (-\sqrt{2 - 1/s^2}, -1/s)^T$ and show that the function $h(s) \stackrel{\text{def}}{=} f(z(s))$ has $h'(s) > 0$ for all $s \geq 2$.)

 **12.9** Consider the problem (12.9). Specify *two* feasible sequences that approach the maximizing point $(1, 1)^T$, and show that neither sequence is a decreasing sequence for f .

 **12.10** Verify that neither the LICQ nor the MFCQ holds for the constraint set defined by (12.32) at $x^* = (0, 0)^T$.


 **12.11** Consider the feasible set Ω in \mathbb{R}^2 defined by $x_2 \geq 0, x_2 \leq x_1^2$.


(a) For $x^* = (0, 0)^T$, write down $T_\Omega(x^*)$ and $\mathcal{F}(x^*)$.

(b) Is LICQ satisfied at x^* ? Is MFCQ satisfied?

(c) If the objective function is $f(x) = -x_2$, verify that the KKT conditions (12.34) are satisfied at x^* .


(d) Find a feasible sequence $\{z_k\}$ approaching x^* with $f(z_k) < f(x^*)$ for all k .


 **12.12** It is trivial to construct an example of a feasible set and a feasible point x^* at which the LICQ is satisfied but the constraints are nonlinear. Give an example of the reverse situation, that is, where the active constraints are linear but the LICQ is not satisfied.

 **12.13** Show that for the feasible region defined by

$$\begin{aligned}(x_1 - 1)^2 + (x_2 - 1)^2 &\leq 2, \\ (x_1 - 1)^2 + (x_2 + 1)^2 &\leq 2, \\ x_1 &\geq 0,\end{aligned}$$

the MFCQ is satisfied at $x^* = (0, 0)^T$ but the LICQ is not satisfied.


 **12.14** Consider the half space defined by $H = \{x \in \mathbb{R}^n \mid a^T x + \alpha \geq 0\}$ where $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ are given. Formulate and solve the optimization problem for finding the point x in H that has the smallest Euclidean norm.

 **12.15** Consider the following modification of (12.36), where t is a parameter to be fixed prior to solving the problem:

$$\min_x \left(x_1 - \frac{3}{2} \right)^2 + (x_2 - t)^4 \quad \text{s.t.} \quad \begin{bmatrix} 1 - x_1 - x_2 \\ 1 - x_1 + x_2 \\ 1 + x_1 - x_2 \\ 1 + x_1 + x_2 \end{bmatrix} \geq 0. \quad (12.97)$$


(a) For what values of t does the point $x^* = (1, 0)^T$ satisfy the KKT conditions?


(b) Show that when $t = 1$, only the first constraint is active at the solution, and find the solution.

 **12.16** (Fletcher [101]) Solve the problem

$$\min_x x_1 + x_2 \quad \text{subject to} \quad x_1^2 + x_2^2 = 1$$


by eliminating the variable x_2 . Show that the choice of sign for a square root operation during the elimination process is critical; the “wrong” choice leads to an incorrect answer.

 **12.17** Prove that when the KKT conditions (12.34) and the LICQ are satisfied at a point x^* , the Lagrange multiplier λ^* in (12.34) is unique.

 **12.18** Consider the problem of finding the point on the parabola $y = \frac{1}{5}(x-1)^2$ that is closest to $(x, y) = (1, 2)$, in the Euclidean norm sense. We can formulate this problem as

$$\min f(x, y) = (x-1)^2 + (y-2)^2 \quad \text{subject to } (x-1)^2 = 5y.$$


- (a) Find all the KKT points for this problem. Is the LICQ satisfied?
- (b) Which of these points are solutions?
- (c) By directly substituting the constraint into the objective function and eliminating the variable x , we obtain an unconstrained optimization problem. Show that the solutions of this problem cannot be solutions of the original problem.


 **12.19** Consider the problem


$$\min_{x \in \mathbb{R}^2} f(x) = -2x_1 + x_2 \quad \text{subject to } \begin{cases} (1-x_1)^3 - x_2 & \geq 0 \\ x_2 + 0.25x_1^2 - 1 & \geq 0. \end{cases}$$

The optimal solution is $x^* = (0, 1)^T$, where both constraints are active.

- (a) Do the LICQ hold at this point?
- (b) Are the KKT conditions satisfied?
- (c) Write down the sets $\mathcal{F}(x^*)$ and $\mathcal{C}(x^*, \lambda^*)$.
- (d) Are the second-order necessary conditions satisfied? Are the second-order sufficient conditions satisfied?

 **12.20** Find the minima of the function $f(x) = x_1x_2$ on the unit circle $x_1^2 + x_2^2 = 1$. Illustrate this problem geometrically.

 **12.21** Find the *maxima* of $f(x) = x_1x_2$ over the unit disk defined by the inequality constraint $1 - x_1^2 - x_2^2 \geq 0$.

 **12.22** Show that for (12.1), the feasible set Ω is convex if $c_i, i \in \mathcal{E}$ are linear functions and $-c_i, i \in \mathcal{I}$ are convex functions.

CHAPTER *13*

Linear Programming: The Simplex Method

Dantzig's development of the simplex method in the late 1940s marks the start of the modern era in optimization. This method made it possible for economists to formulate large models and analyze them in a systematic and efficient way. Dantzig's discovery coincided with the development of the first electronic computers, and the simplex method became one of the earliest important applications of this new and revolutionary technology. From those days to the present, computer implementations of the simplex method have been continually improved and refined. They have benefited particularly from interactions with numerical analysis, a branch of mathematics that also came into its own with the appearance of electronic computers, and have now reached a high level of sophistication.

Today, linear programming and the simplex method continue to hold sway as the most widely used of all optimization tools. Since 1950, generations of workers in management, economics, finance, and engineering have been trained in the techniques of formulating linear models and solving them with simplex-based software. Often, the situations they model are actually nonlinear, but linear programming is appealing because of the advanced state of the software, guaranteed convergence to a global minimum, and the fact that uncertainty in the model makes a linear model more appropriate than an overly complex nonlinear model. Nonlinear programming may replace linear programming as the method of choice in some applications as the nonlinear software improves, and a new class of methods known as interior-point methods (see Chapter 14) has proved to be faster for some linear programming problems, but the continued importance of the simplex method is assured for the foreseeable future.

LINEAR PROGRAMMING

Linear programs have a linear objective function and linear constraints, which may include both equalities and inequalities. The feasible set is a polytope, a convex, connected set with flat, polygonal faces. The contours of the objective function are planar. Figure 13.1 depicts a linear program in two-dimensional space, in which the contours of the objective function are indicated by dotted lines. The solution in this case is unique—a single vertex. A simple reorientation of the polytope or the objective gradient c could however make the solution non-unique; the optimal value $c^T x$ could take on the same value over an entire edge. In higher dimensions, the set of optimal points can be a single vertex, an edge or face, or even the entire feasible set. The problem has no solution if the feasible set is empty (the *infeasible* case) or if the objective function is unbounded below on the feasible region (the *unbounded* case).

Linear programs are usually stated and analyzed in the following *standard form*:

$$\min c^T x, \quad \text{subject to } Ax = b, x \geq 0, \quad (13.1)$$

where c and x are vectors in \mathbb{R}^n , b is a vector in \mathbb{R}^m , and A is an $m \times n$ matrix. Simple devices can be used to transform any linear program to this form. For instance, given the problem

$$\min c^T x, \quad \text{subject to } Ax \leq b$$

(without any bounds on x), we can convert the inequality constraints to equalities by introducing a vector of *slack variables* z and writing

$$\min c^T x, \quad \text{subject to } Ax + z = b, z \geq 0. \quad (13.2)$$

This form is still not quite standard, since not all the variables are constrained to be

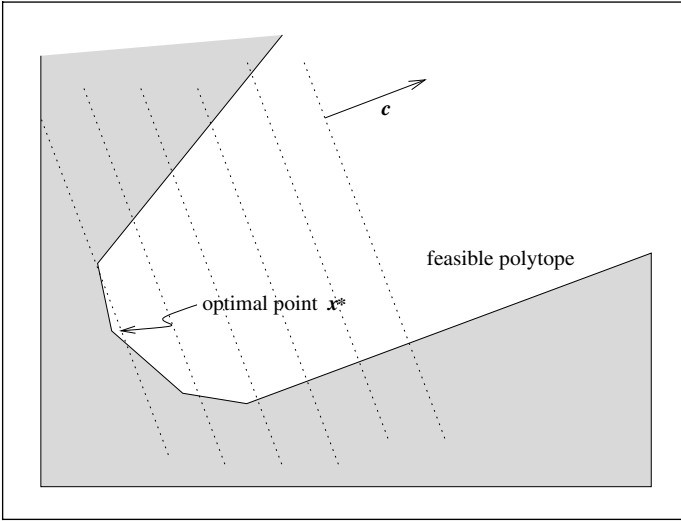


Figure 13.1 A linear program in two dimensions with solution at x^* .

nonnegative. We deal with this by *splitting* x into its nonnegative and nonpositive parts, $x = x^+ - x^-$, where $x^+ = \max(x, 0) \geq 0$ and $x^- = \max(-x, 0) \geq 0$. The problem (13.2) can now be written as

$$\min \begin{bmatrix} c \\ -c \\ 0 \end{bmatrix}^T \begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix}, \quad \text{s.t.} \quad \begin{bmatrix} A & -A & I \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix} = b, \quad \begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix} \geq 0,$$

which clearly has the same form as (13.1).

Inequality constraints of the form $x \leq u$ or $Ax \geq b$ always can be converted to equality constraints by adding or subtracting *slack variables* to make up the difference between the left- and right-hand sides. Hence,

$$\begin{aligned} x \leq u &\Leftrightarrow x + w = u, \quad w \geq 0, \\ Ax \geq b &\Leftrightarrow Ax - y = b, \quad y \geq 0. \end{aligned}$$

(When we subtract the variables from the left hand side, as in the second case, they are sometimes known as *surplus variables*.) We can also convert a “maximize” objective $\max c^T x$ into the “minimize” form of (13.1) by simply negating c to obtain: $\min (-c)^T x$.

We say that the linear program (13.1) is *infeasible* if the feasible set is empty. We say that the problem (13.1) is *unbounded* if the objective function is unbounded below on the feasible region, that is, there is a sequence of points x^k feasible for (13.1) such that $c^T x^k \downarrow -\infty$. (Of course, unbounded problems have no solution).

Many linear programs arise from models of transshipment and distribution networks. These problems have additional structure in their constraints; special-purpose simplex algorithms that exploit this structure are highly efficient. We do not discuss such problems further in this book, except to note that the subject is important and complex, and that a number of fine texts on the topic are available (see, for example, Ahuja, Magnanti, and Orlin [1]).

For the standard formulation (13.1), we will assume throughout that $m < n$. Otherwise, the system $Ax = b$ contains redundant rows, or is infeasible, or defines a unique point. When $m \geq n$, factorizations such as the QR or LU factorization (see Appendix A) can be used to transform the system $Ax = b$ to one with a coefficient matrix of full row rank.

13.1 OPTIMALITY AND DUALITY

OPTIMALITY CONDITIONS

Optimality conditions for the problem (13.1) can be derived from the theory of Chapter 12. Only the first-order conditions—the Karush–Kuhn–Tucker (KKT) conditions—are needed. Convexity of the problem ensures that these conditions are sufficient for a global minimum. We do not need to refer to the second-order conditions from Chapter 12, which are not informative in any case because the Hessian of the Lagrangian for (13.1) is zero.

The theory we developed in Chapter 12 make derivation of optimality and duality results for linear programming much easier than in other treatments, where this theory is developed more or less from scratch.

The KKT conditions follow from Theorem 12.1. As stated in Chapter 12, this theorem requires linear independence of the active constraint gradients (LICQ). However, as we noted in Section 12.6, the result continues to hold for *dependent* constraints provided they are linear, as is the case here.

We partition the Lagrange multipliers for the problem (13.1) into two vectors λ and s , where $\lambda \in \mathbb{R}^m$ is the multiplier vector for the equality constraints $Ax = b$, while $s \in \mathbb{R}^n$ is the multiplier vector for the bound constraints $x \geq 0$. Using the definition (12.33), we can write the Lagrangian function for (13.1) as

$$\mathcal{L}(x, \lambda, s) = c^T x - \lambda^T (Ax - b) - s^T x. \quad (13.3)$$

Applying Theorem 12.1, we find that the first-order necessary conditions for x^* to be a solution of (13.1) are that there exist vectors λ and s such that

$$A^T \lambda + s = c, \quad (13.4a)$$

$$Ax = b, \quad (13.4b)$$

$$x \geq 0, \quad (13.4c)$$

$$s \geq 0, \quad (13.4d)$$

$$x_i s_i = 0, \quad i = 1, 2, \dots, n. \quad (13.4e)$$

The complementarity condition (13.4e), which essentially says that at least one of the components x_i and s_i must be zero for each $i = 1, 2, \dots, n$, is often written in the alternative form $x^T s = 0$. Because of the nonnegativity conditions (13.4c), (13.4d), the two forms are identical.

Let (x^*, λ^*, s^*) denote a vector triple that satisfies (13.4). By combining the three equalities (13.4a), (13.4d), and (13.4e), we find that

$$c^T x^* = (A^T \lambda^* + s^*)^T x^* = (Ax^*)^T \lambda^* = b^T \lambda^*. \quad (13.5)$$

As we shall see in a moment, $b^T \lambda$ is the objective function for the dual problem to (13.1), so (13.5) indicates that the primal and dual objectives are equal for vector triples (x, λ, s) that satisfy (13.4).

It is easy to show directly that the conditions (13.4) are *sufficient* for x^* to be a global solution of (13.1). Let \bar{x} be any other feasible point, so that $A\bar{x} = b$ and $\bar{x} \geq 0$. Then

$$c^T \bar{x} = (A\lambda^* + s^*)^T \bar{x} = b^T \lambda^* + \bar{x}^T s^* \geq b^T \lambda^* = c^T x^*. \quad (13.6)$$

We have used (13.4) and (13.5) here; the inequality relation follows trivially from $\bar{x} \geq 0$ and $s^* \geq 0$. The inequality (13.6) tells us that no other feasible point can have a lower objective value than $c^T x^*$. We can say more: The feasible point \bar{x} is optimal if and only if

$$\bar{x}^T s^* = 0,$$

since otherwise the inequality in (13.6) is strict. In other words, when $s_i^* > 0$, then we must have $\bar{x}_i = 0$ for *all* solutions \bar{x} of (13.1).

THE DUAL PROBLEM

Given the data c , b , and A , which defines the problem (13.1), we can define another, closely related, problem as follows:

$$\max b^T \lambda, \quad \text{subject to } A^T \lambda \leq c. \quad (13.7)$$

This problem is called the *dual problem* for (13.1). In contrast, (13.1) is often referred to as the *primal*. We can restate (13.7) in a slightly different form by introducing a vector of *dual slack variables* s , and writing

$$\max b^T \lambda, \quad \text{subject to } A^T \lambda + s = c, \quad s \geq 0. \quad (13.8)$$

The variables (λ, s) in this problem are sometimes jointly referred to collectively as *dual variables*.

The primal and dual problems present two different viewpoints on the same data. Their close relationship becomes evident when we write down the KKT conditions for (13.7). Let us first restate (13.7) in the form

$$\min -b^T \lambda \quad \text{subject to } c - A^T \lambda \geq 0,$$

to fit the formulation (12.1) from Chapter 12. By using $x \in \mathbb{R}^n$ to denote the Lagrange multipliers for the constraints $A^T \lambda \leq c$, we see that the Lagrangian function is

$$\tilde{\mathcal{L}}(\lambda, x) = -b^T \lambda - x^T (c - A^T \lambda).$$

Using Theorem 12.1 again, we find the first-order necessary conditions for λ to be optimal for (13.7) to be that there exists x such that

$$Ax = b, \tag{13.9a}$$

$$A^T \lambda \leq c, \tag{13.9b}$$

$$x \geq 0, \tag{13.9c}$$

$$x_i (c - A^T \lambda)_i = 0, \quad i = 1, 2, \dots, n. \tag{13.9d}$$

Defining $s = c - A^T \lambda$ (as in (13.8)), we find that the conditions (13.9) and (13.4) are identical! The optimal Lagrange multipliers λ in the primal problem are the optimal variables in the dual problem, while the optimal Lagrange multipliers x in the dual problem are the optimal variables in the primal problem.

Analogously to (13.6), we can show that (13.9) are in fact *sufficient* conditions for a solution of the dual problem (13.7). Given x^* and λ^* satisfying these conditions (so that the triple $(x, \lambda, s) = (x^*, \lambda^*, c - A^T \lambda^*)$ satisfies (13.4)), we have for any other dual feasible point $\bar{\lambda}$ (with $A^T \bar{\lambda} \leq c$) that

$$\begin{aligned} b^T \bar{\lambda} &= (x^*)^T A^T \bar{\lambda} \\ &= (x^*)^T (A^T \bar{\lambda} - c) + c^T x^* \\ &\leq c^T x^* && \text{because } A^T \bar{\lambda} - c \leq 0 \text{ and } x^* \geq 0 \\ &= b^T \lambda^* && \text{from (13.5).} \end{aligned}$$

Hence λ^* achieves the maximum of the dual objective $b^T \lambda$ over the dual feasible region $A^T \lambda \leq c$, so it solves the dual problem (13.7).

The primal–dual relationship is symmetric; by taking the dual of the dual problem (13.7), we recover the primal problem (13.1). We leave the proof of this claim as an exercise.

Given a feasible vector x for the primal (satisfying $Ax = b$ and $x \geq 0$) and a feasible point (λ, s) for the dual (satisfying $A^T \lambda + s = c$, $s \geq 0$), we have as in (13.6) that

$$c^T x - b^T \lambda = (c - A^T \lambda)^T x = s^T x \geq 0. \tag{13.10}$$

Therefore we have $c^T x \geq b^T \lambda$ (that is, the dual objective is a lower bound on the primal objective) when both the primal and dual variables are feasible—a result known as *weak duality*.

The following *strong duality* result is fundamental to the theory of linear programming.

Theorem 13.1 (Strong Duality).

- (i) *If either problem (13.1) or (13.7) has a (finite) solution, then so does the other, and the objective values are equal.*
- (ii) *If either problem (13.1) or (13.7) is unbounded, then the other problem is infeasible.*

PROOF. For (i), suppose that (13.1) has a finite optimal solution x^* . It follows from Theorem 12.1 that there are vectors λ^* and s^* such that (x^*, λ^*, s^*) satisfies (13.4). We noted above that (13.4) and (13.9) are equivalent, and that (13.9) are sufficient conditions for λ^* to be a solution of the dual problem (13.7). Moreover, it follows from (13.5) that $c^T x^* = b^T \lambda^*$, as claimed.

A symmetric argument holds if we start by assuming that the dual problem (13.7) has a solution.

To prove (ii), suppose that the primal is unbounded, that is, there is a sequence of points $x^k, k = 1, 2, 3, \dots$ such that

$$c^T x^k \downarrow -\infty, \quad Ax^k = b, \quad x^k \geq 0.$$

Suppose too that the dual (13.7) is feasible, that is, there exists a vector $\bar{\lambda}$ such that $A^T \bar{\lambda} \leq c$. From the latter inequality together with $x^k \geq 0$, we have that $\bar{\lambda}^T Ax^k \leq c^T x^k$, and therefore

$$\bar{\lambda}^T b = \bar{\lambda}^T Ax^k \leq c^T x^k \downarrow -\infty,$$

yielding a contradiction. Hence, the dual must be infeasible.

A similar argument can be used to show that unboundedness of the dual implies infeasibility of the primal. \square

As we showed in the discussion following Theorem 12.1, the multiplier values λ and s for (13.1) indicate the sensitivity of the optimal objective value to perturbations in the constraints. In fact, the process of finding (λ, s) for a given optimal x is often called *sensitivity analysis*. Considering the case of perturbations to the vector b (the right-hand side in (13.1) and objective gradient in (13.7)), we can make an informal argument to illustrate the sensitivity. Suppose that this small change produces small perturbations in the primal and dual solutions, and that the vectors Δs and Δx have zeros in the same locations as s and x , respectively. Since x and s are complementary (see (13.4e)) it follows that

$$0 = x^T s = x^T \Delta s = (\Delta x)^T s = (\Delta x)^T \Delta s.$$

We have from Theorem 13.1 that the optimal objectives of the primal and dual problems are equal, for both the original and perturbed problems, so

$$c^T x = b^T \lambda, \quad c^T (x + \Delta x) = (b + \Delta b)^T (\lambda + \Delta \lambda).$$

Moreover, by feasibility of the perturbed solutions in the perturbed problems, we have

$$A(x + \Delta x) = b + \Delta b, \quad A^T \Delta \lambda = -\Delta s.$$

Hence, the change in optimal objective due to the perturbation is as follows:

$$\begin{aligned} c^T \Delta x &= (b + \Delta b)^T (\lambda + \Delta \lambda) - b^T \lambda \\ &= (b + \Delta b)^T \Delta \lambda + (\Delta b)^T \lambda \\ &= (x + \Delta x)^T A^T \Delta \lambda + (\Delta b)^T \lambda \\ &= (x + \Delta x)^T \Delta s + (\Delta b)^T \lambda \\ &= (\Delta b)^T \lambda. \end{aligned}$$

In particular, if $\Delta b = \epsilon e_j$, where e_j is the j th unit vector in \mathbb{R}^m , we have for all ϵ sufficiently small that

$$c^T \Delta x = \epsilon \lambda_j. \quad (13.11)$$

That is, the change in optimal objective is λ_j times the size of the perturbation to b_j , if the perturbation is small.

13.2 GEOMETRY OF THE FEASIBLE SET

BASES AND BASIC FEASIBLE POINTS

We assume for the remainder of the chapter that

$$\text{The matrix } A \text{ in (13.1) has full row rank.} \quad (13.12)$$

In practice, a preprocessing phase is applied to the user-supplied data to remove some redundancies from the given constraints and eliminate some of the variables. Reformulation by adding slack, surplus, and artificial variables can also result in A satisfying the property (13.12).

Each iterate generated by the simplex method is a *basic feasible point* of (13.1). A vector x is a basic feasible point if it is feasible and if there exists a subset \mathcal{B} of the index set $\{1, 2, \dots, n\}$ such that

- \mathcal{B} contains exactly m indices;
- $i \notin \mathcal{B} \Rightarrow x_i = 0$ (that is, the bound $x_i \geq 0$ can be inactive only if $i \in \mathcal{B}$);
- The $m \times m$ matrix B defined by

$$B = [A_i]_{i \in \mathcal{B}} \quad (13.13)$$

is nonsingular, where A_i is the i th column of A .

A set \mathcal{B} satisfying these properties is called a *basis* for the problem (13.1). The corresponding matrix B is called the *basis matrix*.

The simplex method's strategy of examining only basic feasible points will converge to a solution of (13.1) only if

- (a) the problem *has* basic feasible points; and
- (b) at least one such point is a *basic optimal point*, that is, a solution of (13.1) that is also a basic feasible point.

Happily, both (a) and (b) are true under reasonable assumptions, as the following result (sometimes known as the fundamental theorem of linear programming) shows.

Theorem 13.2.

- (i) If (13.1) has a nonempty feasible region, then there is at least one basic feasible point;
- (ii) If (13.1) has solutions, then at least one such solution is a basic optimal point.
- (iii) If (13.1) is feasible and bounded, then it has an optimal solution.

PROOF. Among all feasible vectors x , choose one with the minimal number of nonzero components, and denote this number by p . Without loss of generality, assume that the nonzeros are x_1, x_2, \dots, x_p , so we have

$$\sum_{i=1}^p A_i x_i = b.$$

Suppose first that the columns A_1, A_2, \dots, A_p are linearly dependent. Then we can express one of them (A_p , say) in terms of the others, and write

$$A_p = \sum_{i=1}^{p-1} A_i z_i, \quad (13.14)$$

for some scalars z_1, z_2, \dots, z_{p-1} . It is easy to check that the vector

$$x(\epsilon) = x + \epsilon(z_1, z_2, \dots, z_{p-1}, -1, 0, 0, \dots, 0)^T = x + \epsilon z \quad (13.15)$$

satisfies $Ax(\epsilon) = b$ for any scalar ϵ . In addition, since $x_i > 0$ for $i = 1, 2, \dots, p$, we also have $x_i(\epsilon) > 0$ for the same indices $i = 1, 2, \dots, p$ and all ϵ sufficiently small in magnitude. However, there is a value $\bar{\epsilon} \in (0, x_p]$ such that $x_i(\bar{\epsilon}) = 0$ for some $i = 1, 2, \dots, p$. Hence, $x(\bar{\epsilon})$ is feasible and has at most $p - 1$ nonzero components, contradicting our choice of p as the minimal number of nonzeros.

Therefore, columns A_1, A_2, \dots, A_p must be linearly independent, and so $p \leq m$. If $p = m$, we are done, since then x is a basic feasible point and \mathcal{B} is simply $\{1, 2, \dots, m\}$. Otherwise $p < m$ and, because A has full row rank, we can choose $m - p$ columns from among $A_{p+1}, A_{p+2}, \dots, A_n$ to build up a set of m linearly independent vectors. We construct \mathcal{B} by adding the corresponding indices to $\{1, 2, \dots, p\}$. The proof of (i) is complete.

The proof of (ii) is quite similar. Let x^* be a solution with a minimal number of nonzero components p , and assume again that $x_1^*, x_2^*, \dots, x_p^*$ are the nonzeros. If the columns A_1, A_2, \dots, A_p are linearly dependent, we define

$$x^*(\epsilon) = x^* + \epsilon z,$$

where z is chosen exactly as in (13.14), (13.15). It is easy to check that $x^*(\epsilon)$ will be feasible for all ϵ sufficiently small, both positive and negative. Hence, since x^* is optimal, we must have

$$c^T(x^* + \epsilon z) \geq c^T x^* \Rightarrow \epsilon c^T z \geq 0$$

for all ϵ sufficiently small (positive and negative). Therefore, $c^T z = 0$ and so $c^T x^*(\epsilon) = c^T x^*$ for all ϵ . The same logic as in the proof of (i) can be applied to find $\bar{\epsilon} > 0$ such that $x^*(\bar{\epsilon})$ is feasible and optimal, with at most $p - 1$ nonzero components. This contradicts our choice of p as the minimal number of nonzeros, so the columns A_1, A_2, \dots, A_p must be linearly independent. We can now apply the same reasoning as above to conclude that x^* is already a basic feasible point and therefore a basic optimal point.

The final statement (iii) is a consequence of finite termination of the simplex method. We comment on the latter property in the next section. \square

The terminology we use here is not quite standard, as the following table shows:

our terminology	terminology used elsewhere
basic feasible point	basic feasible solution
basic optimal point	optimal basic feasible solution

The standard terms arose because “solution” and “feasible solution” were originally used as synonyms for “feasible point.” However, as the discipline of optimization developed,

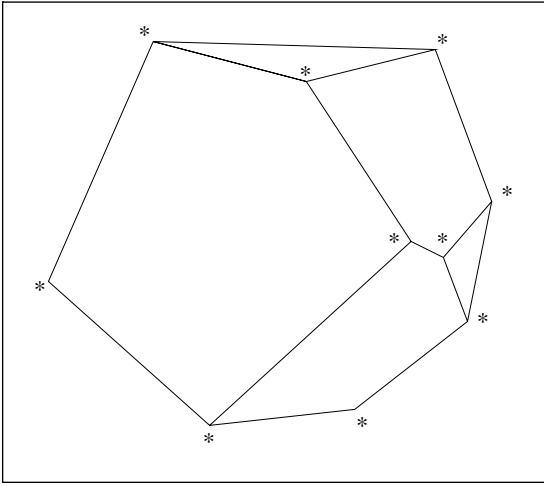


Figure 13.2
Vertices of a
three-dimensional polytope
(indicated by *).

the word “solution” took on a more specific and intuitive meaning (as in “solution to the problem”). We maintain consistency with the rest of the book by following this more modern usage.

VERTICES OF THE FEASIBLE POLYTOPE

The feasible set defined by the linear constraints is a polytope, and the *vertices* of this polytope are the points that do not lie on a straight line between two other points in the set. Geometrically, they are easily recognizable; see Figure 13.2. Algebraically, the vertices are exactly the basic feasible points defined above. We therefore have an important relationship between the algebraic and geometric viewpoints and a useful aid to understanding how the simplex method works.

Theorem 13.3.

All basic feasible points for (13.1) are vertices of the feasible polytope $\{x \mid Ax = b, x \geq 0\}$, and vice versa.

PROOF. Let x be a basic feasible point and assume without loss of generality that $\mathcal{B} = \{1, 2, \dots, m\}$. The matrix $B = [A_i]_{i=1,2,\dots,m}$ is therefore nonsingular, and

$$x_{m+1} = x_{m+2} = \dots = x_n = 0. \quad (13.16)$$

Suppose that x lies on a straight line between two other feasible points y and z . Then we can find $\alpha \in (0, 1)$ such that $x = \alpha y + (1 - \alpha)z$. Because of (13.16) and the fact that α and $1 - \alpha$ are both positive, we must have $y_i = z_i = 0$ for $i = m + 1, m + 2, \dots, n$. Writing $x_B = (x_1, x_2, \dots, x_m)^T$ and defining y_B and z_B likewise, we have from $Ax = Ay = Az = b$

that

$$Bx_B = By_B = Bz_B = b,$$

and so, by nonsingularity of B , we have $x_B = y_B = z_B$. Therefore, $x = y = z$, contradicting our assertion that y and z are two feasible points *other* than x . Therefore, x is a vertex.

Conversely, let x be a vertex of the feasible polytope, and suppose that the nonzero components of x are x_1, x_2, \dots, x_p . If the corresponding columns A_1, A_2, \dots, A_p are linearly dependent, then we can construct the vector $x(\epsilon) = x + \epsilon z$ as in (13.15). Since $x(\epsilon)$ is feasible for all ϵ with sufficiently small magnitude, we can define $\hat{\epsilon} > 0$ such that $x(\hat{\epsilon})$ and $x(-\hat{\epsilon})$ are both feasible. Since $x = x(0)$ obviously lies on a straight line between these two points, it cannot be a vertex. Hence our assertion that A_1, A_2, \dots, A_p are linearly dependent must be incorrect, so these columns must be linearly independent and $p \leq m$. If $p < m$, and since A has full row rank, we can add $m - p$ indices to $\{1, 2, \dots, p\}$ to form a basis \mathcal{B} , for which x is the corresponding basic feasible point. This completes our proof. \square

We conclude this discussion of the geometry of the feasible set with a definition of *degeneracy*. This term has a variety of meanings in optimization, as we discuss in Chapter 16. For the purposes of this chapter, we use the following definition.

Definition 13.1 (Degeneracy).

A basis \mathcal{B} is said to be degenerate if $x_i = 0$ for some $i \in \mathcal{B}$, where x is the basic feasible solution corresponding to \mathcal{B} . A linear program (13.1) is said to be degenerate if it has at least one degenerate basis.

13.3 THE SIMPLEX METHOD

OUTLINE

In this section we give a detailed description of the simplex method for (13.1). There are actually a number of variants the simplex method; the one described here is sometimes known as the *revised simplex method*. (We will describe an alternative known as the *dual simplex method* in Section 13.6.)

As we described above, all iterates of the simplex method are basic feasible points for (13.1) and therefore vertices of the feasible polytope. Most steps consist of a move from one vertex to an adjacent one for which the basis \mathcal{B} differs in exactly one component. On most steps (but not all), the value of the primal objective function $c^T x$ is decreased. Another type of step occurs when the problem is unbounded: The step is an edge along which the objective function is reduced, and along which we can move infinitely far without ever reaching a vertex.

The major issue at each simplex iteration is to decide which index to remove from the basis \mathcal{B} . Unless the step is a direction of unboundedness, a single index must be removed from \mathcal{B} and replaced by another from outside \mathcal{B} . We can gain some insight into how this decision is made by looking again at the KKT conditions (13.4).

From \mathcal{B} and (13.4), we can derive values for not just the primal variable x but also the dual variables (λ, s) , as we now show. First, define the nonbasic index set \mathcal{N} as the complement of \mathcal{B} , that is,

$$\mathcal{N} = \{1, 2, \dots, n\} \setminus \mathcal{B}. \quad (13.17)$$

Just as B is the basic matrix, whose columns are A_i for $i \in \mathcal{B}$, we use N to denote the nonbasic matrix $N = [A_i]_{i \in \mathcal{N}}$. We also partition the n -element vectors x, s , and c according to the index sets \mathcal{B} and \mathcal{N} , using the notation

$$\begin{aligned} x_{\mathcal{B}} &= [x_i]_{i \in \mathcal{B}}, & x_{\mathcal{N}} &= [x_i]_{i \in \mathcal{N}}, \\ s_{\mathcal{B}} &= [s_i]_{i \in \mathcal{B}}, & s_{\mathcal{N}} &= [s_i]_{i \in \mathcal{N}}, \\ c_{\mathcal{B}} &= [c_i]_{i \in \mathcal{B}}, & c_{\mathcal{N}} &= [c_i]_{i \in \mathcal{N}}. \end{aligned}$$

From the KKT condition (13.4b), we have that

$$Ax = Bx_{\mathcal{B}} + Nx_{\mathcal{N}} = b.$$

The primal variable x for this simplex iterate is defined as

$$x_{\mathcal{B}} = B^{-1}b, \quad x_{\mathcal{N}} = 0. \quad (13.18)$$

Since we are dealing only with basic feasible points, we know that B is nonsingular and that $x_{\mathcal{B}} \geq 0$, so this choice of x satisfies two of the KKT conditions: the equality constraints (13.4b) and the nonnegativity condition (13.4c).

We choose s to satisfy the complementarity condition (13.4e) by setting $s_{\mathcal{B}} = 0$. The remaining components λ and $s_{\mathcal{N}}$ can be found by partitioning this condition into $c_{\mathcal{B}}$ and $c_{\mathcal{N}}$ components and using $s_{\mathcal{B}} = 0$ to obtain

$$B^T \lambda = c_{\mathcal{B}}, \quad N^T \lambda + s_{\mathcal{N}} = c_{\mathcal{N}}. \quad (13.19)$$

Since B is square and nonsingular, the first equation uniquely defines λ as

$$\lambda = B^{-T} c_{\mathcal{B}}. \quad (13.20)$$

The second equation in (13.19) implies a value for $s_{\mathcal{N}}$:

$$s_{\mathcal{N}} = c_{\mathcal{N}} - N^T \lambda = c_{\mathcal{N}} - (B^{-1}N)^T c_{\mathcal{B}}. \quad (13.21)$$

Computation of the vector s_N is often referred to as *pricing*. The components of s_N are often called the *reduced costs* of the nonbasic variables x_N .

The only KKT condition that we have not enforced explicitly is the nonnegativity condition $s \geq 0$. The basic components s_B certainly satisfy this condition, by our choice $s_B = 0$. If the vector s_N defined by (13.21) also satisfies $s_N \geq 0$, we have found an optimal vector triple (x, λ, s) , so the algorithm can terminate and declare success. Usually, however, one or more of the components of s_N are negative. The new index to enter the basis \mathcal{B} —the *entering index*—is chosen to be *one of the indices* $q \in \mathcal{N}$ for which $s_q < 0$. As we show below, the objective $c^T x$ will decrease when we allow x_q to become positive if and only if (i) $s_q < 0$ and (ii) it is possible to increase x_q away from zero while maintaining feasibility of x . Our procedure for altering \mathcal{B} and changing x and s can be described accordingly as follows:

- allow x_q to increase from zero during the next step;
- fix all other components of x_N at zero, and figure out the effect of increasing x_q on the current basic vector x_B , given that we want to stay feasible with respect to the equality constraints $Ax = b$;
- keep increasing x_q until one of the components of x_B (x_p , say) is driven to zero, or determining that no such component exists (the unbounded case);
- remove index p (known as the *leaving index*) from \mathcal{B} and replace it with the entering index q .

This process of selecting entering and leaving indices, and performing the algebraic operations necessary to keep track of the values of the variables x , λ , and s , is sometimes known as *pivoting*.

We now formalize the pivoting procedure in algebraic terms. Since both the new iterate x^+ and the current iterate x should satisfy $Ax = b$, and since $x_N = 0$ and $x_i^+ = 0$ for $i \in \mathcal{N} \setminus \{q\}$, we have

$$Ax^+ = Bx_B^+ + A_q x_q^+ = Bx_B = Ax.$$

By multiplying this expression by B^{-1} and rearranging, we obtain

$$x_B^+ = x_B - B^{-1}A_q x_q^+. \quad (13.22)$$

Geometrically speaking, (13.22) is usually a move along an edge of the feasible polytope that decreases $c^T x$. We continue to move along this edge until a new vertex is encountered. At this vertex, a new constraint $x_p \geq 0$ must have become active, that is, one of the components x_p , $p \in \mathcal{B}$, has decreased to zero. We then remove this index p from the basis \mathcal{B} and replace it by q .

We now show how the step defined by (13.22) affects the value of $c^T x$. From (13.22), we have

$$c^T x^+ = c_B^T x_B^+ + c_q x_q^+ = c_B^T x_B - c_B^T B^{-1} A_q x_q^+ + c_q x_q^+. \quad (13.23)$$

From (13.20) we have $c_B^T B^{-1} = \lambda^T$, while from the second equation in (13.19), since $q \in \mathcal{N}$, we have $A_q^T \lambda = c_q - s_q$. Therefore,

$$c_B^T B^{-1} A_q x_q^+ = \lambda^T A_q x_q^+ = (c_q - s_q) x_q^+,$$

so by substituting in (13.23) we obtain

$$c^T x^+ = c_B^T x_B - (c_q - s_q) x_q^+ + c_q x_q^+ = c^T x + s_q x_q^+. \quad (13.24)$$

Since q was chosen to have $s_q < 0$, it follows that the step (13.22) produces a decrease in the primal objective function $c^T x$ whenever $x_q^+ > 0$.

It is possible that we can increase x_q^+ to ∞ without ever encountering a new vertex. In other words, the constraint $x_B^+ = x_B - B^{-1} A_q x_q^+ \geq 0$ holds for all positive values of x_q^+ . When this happens, the linear program is *unbounded*; the simplex method has identified a ray that lies entirely within the feasible polytope along which the objective $c^T x$ decreases to $-\infty$.

Figure 13.3 shows a path traversed by the simplex method for a problem in \mathbb{R}^2 . In this example, the optimal vertex x^* is found in three steps.

If the basis \mathcal{B} is *nondegenerate* (see Definition 13.1), then we are guaranteed that $x_q^+ > 0$, so we can be assured of a strict decrease in the objective function $c^T x$ at this step. If

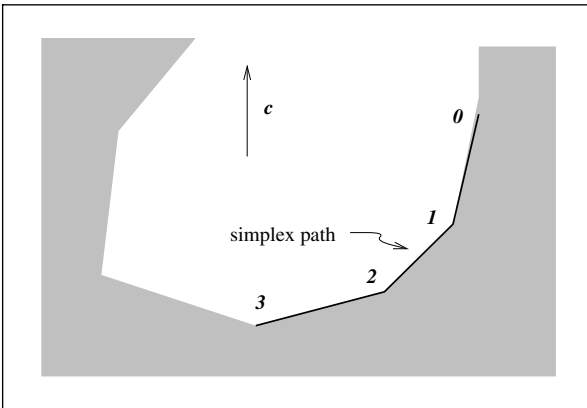


Figure 13.3
Simplex iterates for
a two-dimensional
problem.

the *problem* (13.1) is nondegenerate, we can ensure a decrease in $c^T x$ at *every* step, and can therefore prove the following result concerning termination of the simplex method.

Theorem 13.4.

Provided that the linear program (13.1) is nondegenerate and bounded, the simplex method terminates at a basic optimal point.

PROOF. The simplex method cannot visit the same basic feasible point x at two different iterations, because it attains a strict decrease at each iteration. Since the number of possible bases \mathcal{B} is finite (there are only a finite number of ways to choose a subset of m indices from $\{1, 2, \dots, n\}$), and since each basis defines a single basic feasible point, there are only a finite number of basic feasible points. Hence, the number of iterations is finite. Moreover, since the method is always able to take a step away from a nonoptimal basic feasible point, and since the problem is not unbounded, the method must terminate at a basic optimal point. \square

This result gives us a proof of Theorem 13.2 (iii) in the case in which the linear program is nondegenerate. The proof of finite termination is considerably more complex when nondegeneracy of (13.1) is not assumed, as we discuss at the end of Section 13.5.

A SINGLE STEP OF THE METHOD

We have covered most of the mechanics of taking a single step of the simplex method. To make subsequent discussions easier to follow, we summarize our description.

Procedure 13.1 (One Step of Simplex).

Given $\mathcal{B}, \mathcal{N}, x_{\mathcal{B}} = B^{-1}b \geq 0, x_{\mathcal{N}} = 0$;

Solve $B^T \lambda = c_{\mathcal{B}}$ for λ ,

Compute $s_{\mathcal{N}} = c_{\mathcal{N}} - N^T \lambda$; (* pricing *)

if $s_{\mathcal{N}} \geq 0$

stop; (* optimal point found *)

Select $q \in \mathcal{N}$ with $s_q < 0$ as the entering index;

Solve $Bd = A_q$ for d ;

if $d \leq 0$

stop; (* problem is unbounded *)

Calculate $x_q^+ = \min_{d_i > 0} (x_{\mathcal{B}})_i / d_i$, and use p to denote the minimizing i ;

Update $x_{\mathcal{B}}^+ = x_{\mathcal{B}} - dx_q^+, x_{\mathcal{N}}^+ = (0, \dots, 0, x_q^+, 0, \dots, 0)^T$;

Change \mathcal{B} by adding q and removing the basic variable corresponding to column p of B .

We illustrate this procedure with a simple example.

EXAMPLE 13.1

Consider the problem

$$\begin{aligned} \min \quad & -4x_1 - 2x_2 \quad \text{subject to} \\ & x_1 + x_2 + x_3 = 5, \\ & 2x_1 + (1/2)x_2 + x_4 = 8, \\ & x \geq 0. \end{aligned}$$

Suppose we start with the basis $\mathcal{B} = \{3, 4\}$, for which we have

$$x_B = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \end{bmatrix}, \quad \lambda = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad s_N = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \end{bmatrix},$$

and an objective value of $c^T x = 0$. Since both elements of s_N are negative, we could choose either 1 or 2 to be the entering variable. Suppose we choose $q = 1$. We obtain $d = (1, 2)^T$, so we cannot (yet) conclude that the problem is unbounded. By performing the ratio calculation, we find that $p = 2$ (corresponding to the index 4) and $x_1^+ = 4$. We update the basic and nonbasic index sets to $\mathcal{B} = \{3, 1\}$ and $\mathcal{N} = \{4, 2\}$, and move to the next iteration.

At the second iteration, we have

$$x_B = \begin{bmatrix} x_3 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \quad \lambda = \begin{bmatrix} 0 \\ -3/2 \end{bmatrix}, \quad s_N = \begin{bmatrix} s_4 \\ s_2 \end{bmatrix} = \begin{bmatrix} 3/2 \\ -5/4 \end{bmatrix},$$

with an objective value of -12 . We see that s_N has one negative component, corresponding to the index $q = 2$, so we select this index to enter the basis. We obtain $d = (3/2, -1/2)^T$, so again we do not detect unboundedness. Continuing, we find that the maximum value of x_2^+ is $4/3$, and that $p = 1$, which indicates that index 3 will leave the basis \mathcal{B} . We update the index sets to $\mathcal{B} = \{2, 1\}$ and $\mathcal{N} = \{4, 3\}$ and continue.

At the start of the third iteration, we have

$$x_B = \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 4/3 \\ 11/3 \end{bmatrix}, \quad \lambda = \begin{bmatrix} -5/3 \\ -2/3 \end{bmatrix}, \quad s_N = \begin{bmatrix} s_4 \\ s_3 \end{bmatrix} = \begin{bmatrix} 7/3 \\ 5/3 \end{bmatrix},$$

with an objective value of $c^T x = -41/3$. We see that $s_N \geq 0$, so the optimality test is satisfied, and we terminate. □

We need to flesh out Procedure 13.1 with specifics of three important aspects of the implementation:

- Linear algebra issues—maintaining an LU factorization of B that can be used to solve for λ and d .
- Selection of the entering index q from among the negative components of s_N . (In general, there are many such components.)
- Handling of degenerate bases and degenerate steps, in which it is not possible to choose a positive value of x_q^+ without violating feasibility.

Proper handling of these issues is crucial to the efficiency of a simplex implementation. We give some details in the next three sections.

13.4 LINEAR ALGEBRA IN THE SIMPLEX METHOD

We have to solve two linear systems involving the matrix B at each step; namely,

$$B^T \lambda = c_B, \quad Bd = A_q. \quad (13.25)$$

We never calculate the inverse basis matrix B^{-1} explicitly just to solve these systems. Instead, we calculate or maintain some factorization of B —usually an LU factorization—and use triangular substitutions with the factors to recover λ and d . It is less expensive to update the factorization than to calculate it afresh at each iteration because the basis matrix B changes by just a single column between iterations.

The standard factorization/updating procedures start with an LU factorization of B at the first iteration of the simplex algorithm. Since in practical applications B is large and sparse, its rows and columns are rearranged during the factorization to maintain both numerical stability and sparsity of the L and U factors. One successful pivot strategy that trades off between these two aims was proposed by Markowitz in 1957 [202]; it is still used as the basis of many practical sparse LU algorithms. Other considerations may also enter into our choice of row and column reordering of B . For example, it may help to improve the efficiency of the updating procedure if as many as possible of the leading columns of U contain just a single nonzero, on the diagonal. Many heuristics have been devised for choosing row and column permutations that produce this and other desirable structural features.

Let us assume for simplicity that row and column permutations are already incorporated in B , so that we write the initial LU factorization as

$$LU = B, \quad (13.26)$$

(L is unit lower triangular, U is upper triangular). The system $Bd = A_q$ can then be solved by the following two-step procedure:

$$L\bar{d} = A_q, \quad Ud = \bar{d}. \quad (13.27)$$

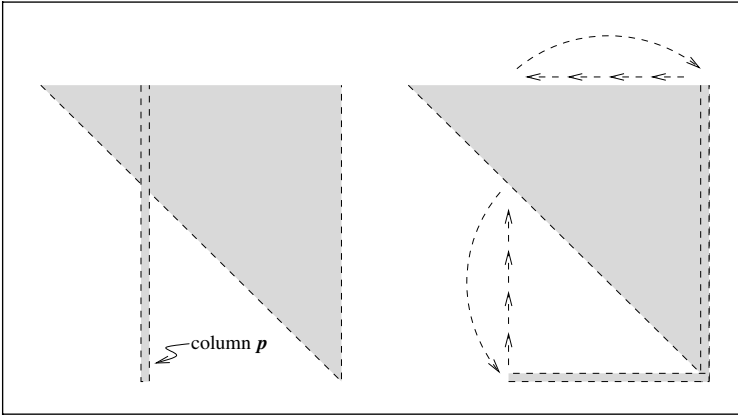


Figure 13.4 Left: $L^{-1}B^+$, which is upper triangular except for the column occupied by A_p . Right: After cyclic row and column permutation P_1 , the non-upper triangular part of $P_1L^{-1}B^+P_1^T$ appears in the last row.

Similarly, the system $B^T\lambda = c_B$ is solved by performing the following two triangular substitutions:

$$U^T\bar{\lambda} = c_B, \quad L^T\lambda = \bar{\lambda}.$$

We now discuss a procedure for updating the factors L and U after one step of the simplex method, when the index p is removed from the basis \mathcal{B} and replaced by the index q . The corresponding change to the basis matrix B is that the column B_p is removed from B and replaced by A_q . We call the resulting matrix B^+ and note that if we rewrite (13.26) as $U = L^{-1}B$, the modified matrix $L^{-1}B^+$ will be upper triangular except in column p . That is, $L^{-1}B^+$ has the form shown on the left in Figure 13.4.

We now perform a cyclic permutation that moves column p to the last column position m and moves columns $p+1, p+2, \dots, m$ one position to the left to make room for it. If we apply the same permutation to rows p through m , the net effect is to move the non-upper triangular part to the last row of the matrix, as shown in Figure 13.4. If we denote the permutation matrix by P_1 , the matrix illustrated at right in Figure 13.4 is $P_1L^{-1}B^+P_1^T$.

Finally, we perform sparse Gaussian elimination on the matrix $P_1L^{-1}B^+P_1^T$ to restore upper triangular form. That is, we find L_1 and U_1 (lower and upper triangular, respectively) such that

$$P_1L^{-1}B^+P_1^T = L_1U_1. \quad (13.28)$$

It is easy to show that L_1 and U_1 have a simple form. The lower triangular matrix L_1 differs from the identity only in the last row, while U_1 is identical to $P_1L^{-1}B^+P_1^T$ except that the (m, m) element is changed and the off-diagonal elements in the last row are eliminated.

We give details of this process for the case of $m = 5$. Using the notation

$$L^{-1}B = U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ & u_{22} & u_{23} & u_{24} & u_{25} \\ & & u_{33} & u_{34} & u_{35} \\ & & & u_{44} & u_{45} \\ & & & & u_{55} \end{bmatrix}, \quad L^{-1}A_q = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix},$$

and supposing that $p = 2$ (so that the second column is replaced by $L^{-1}A_q$), we have

$$L^{-1}B^+ = \begin{bmatrix} u_{11} & w_1 & u_{13} & u_{14} & u_{15} \\ & w_2 & u_{23} & u_{24} & u_{25} \\ & w_3 & u_{33} & u_{34} & u_{35} \\ & w_4 & & u_{44} & u_{45} \\ & w_5 & & & u_{55} \end{bmatrix}.$$

After the cyclic permutation P_1 , we have

$$P_1 L^{-1} B^+ P_1^T = \begin{bmatrix} u_{11} & u_{13} & u_{14} & u_{15} & w_1 \\ & u_{33} & u_{34} & u_{35} & w_3 \\ & & u_{44} & u_{45} & w_4 \\ & & & u_{55} & w_5 \\ & u_{23} & u_{24} & u_{25} & w_2 \end{bmatrix}. \quad (13.29)$$

The factors L_1 and U_1 are now as follows:

$$L_1 = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ 0 & l_{52} & l_{53} & l_{54} & 1 \end{bmatrix}, \quad U_1 = \begin{bmatrix} u_{11} & u_{13} & u_{14} & u_{15} & w_1 \\ & u_{33} & u_{34} & u_{35} & w_3 \\ & & u_{44} & u_{45} & w_4 \\ & & & u_{55} & w_5 \\ & & & & \hat{w}_2 \end{bmatrix}, \quad (13.30)$$

for certain values of l_{52} , l_{53} , l_{54} , and \hat{w}_2 (see Exercise 13.10).

The result of this updating process is the factorization (13.28), which we can rewrite as follows:

$$B^+ = L^+ U^+, \quad \text{where } L^+ = L P_1^T L_1, \quad U^+ = U_1 P_1. \quad (13.31)$$

There is no need to calculate L^+ and U^+ explicitly. Rather, the nonzero elements in L_1 and the last column of U_1 , and the permutation information in P_1 , can be stored in compact form, so that triangular substitutions involving L^+ and U^+ can be performed by applying a number of permutations and sparse triangular substitutions involving these factors. The factorization updates from subsequent simplex steps are stored and applied in a similar fashion.

The procedure we have just outlined is due to Forrest and Tomlin [110]. It is quite efficient, because it requires the storage of little data at each update and does not require much movement of data in memory. Its major disadvantage is possible numerical instability. Large elements in the factors of a matrix are a sure indicator of instability, and the multipliers in the L_1 factor (l_{52} in (13.30), for example) may be very large. An earlier scheme of Bartels and Golub [12] allowed swapping of rows to avoid these problems. For instance, if $|u_{33}| < |u_{23}|$ in (13.29), we could swap rows 2 and 5 to ensure that the subsequent multiplier l_{52} in the L_1 factor does not exceed 1 in magnitude. This improved stability comes at a price: The lower right corner of the upper triangular factor may become more dense during each update.

Although the update information for each iteration (the permutation matrices and the sparse triangular factors) can often be stored in a highly compact form, the total amount of space may build up to unreasonable levels after many such updates have been performed. As the number of updates builds up, so does the time needed to solve for the vectors d and λ in Procedure 13.1. If an unstable updating procedure is used, numerical errors may also come into play, blocking further progress by the simplex algorithm. For all these reasons, most simplex implementations periodically calculate a fresh LU factorization of the current basis matrix B and discard the accumulated updates. The new factorization uses the same permutation strategies that we apply to the very first factorization, which balance the requirements of stability, sparsity, and structure.

13.5 OTHER IMPORTANT DETAILS

PRICING AND SELECTION OF THE ENTERING INDEX

There are usually many negative components of s_N at each step. How do we choose one of these to become the index that enters the basis? Ideally, we would like to choose the sequence of entering indices q that gets us to the solution x^* in the fewest possible steps, but we rarely have the global perspective needed to implement this strategy. Instead, we use more shortsighted but practical strategies that obtain a significant decrease in $c^T x$ on just the present iteration. There is usually a tradeoff between the effort spent on finding a good entering index and the amount of decrease in $c^T x$ resulting from this choice. Different pivot strategies resolve this tradeoff in different ways.

Dantzig's original selection rule is one of the simplest. It chooses q such that s_q is the most negative component of $s_N = N^T \lambda$. This rule, which is motivated by (13.24), gives the maximum improvement in $c^T x$ per unit increase in the entering variable x_q . A large

reduction in $c^T x$ is not guaranteed, however. It could be that we can increase x_q^+ only a tiny amount from zero (or not at all) before reaching the next vertex.

Calculation of the entire vector s_N from (13.21) requires a multiplication by N^T , which can be expensive when the matrix N is very large. *Partial pricing* strategies calculate only a subvector of s_N and make the choice of entering variable from among the negative entries in this subvector. To give all the indices in \mathcal{N} a chance to enter the basis, these strategies cycle through the nonbasic elements, periodically changing the subvector of s_N they evaluate so that no nonbasic index is ignored for too long.

Neither of these strategies guarantees that we can make a substantial move along the chosen edge before reaching a new vertex. *Multiple pricing* strategies are more thorough: For a small subset of indices $q \in \mathcal{N}$, they evaluate s_q and, if $s_q < 0$, the maximum value of x_q^+ that maintains feasibility of x^+ and the consequent change $s_q x_q^+$ in the objective function (see (13.24)). Calculation of x_q^+ requires evaluation of $d = B^{-1}A_q$ as in Procedure 13.1, which is not cheap. Subsequent iterations deal with this same index subset until we reach an iteration at which all s_q are nonnegative for q in the subset. At this point, the full vector s_N is computed, a new subset of nonbasic indices is chosen, and the cycle begins again. This approach has the advantage that the columns of the matrix N outside the current subset of priced components need not be accessed at all, so memory access in the implementation is quite localized.

Naturally, it is possible to devise heuristics that combine partial and multiple pricing in various imaginative ways.

A sophisticated rule known as *steepest edge* chooses the “most downhill” direction from among all the candidates—the one that produces the largest decrease in $c^T x$ per unit distance moved along the edge. (By contrast, Dantzig’s rule maximizes the decrease in $c^T x$ per unit change in x_q^+ , which is not the same thing, as a small change in x_q^+ can correspond to a large distance moved along the edge.) During the pivoting step, the overall change in x is

$$x^+ = \begin{bmatrix} x_B^+ \\ x_N^+ \end{bmatrix} = \begin{bmatrix} x_B \\ x_N \end{bmatrix} + \begin{bmatrix} -B^{-1}A_q \\ e_q \end{bmatrix} x_q^+ = x + \eta_q x_q^+, \quad (13.32)$$

where e_q is the unit vector with a 1 in the position corresponding to the index $q \in \mathcal{N}$ and zeros elsewhere, and the vector η_q is defined as

$$\eta_q = \begin{bmatrix} -B^{-1}A_q \\ e_q \end{bmatrix} = \begin{bmatrix} -d \\ e_q \end{bmatrix}; \quad (13.33)$$

see (13.25). The change in $c^T x$ per unit step along η_q is given by

$$\frac{c^T \eta_q}{\|\eta_q\|}. \quad (13.34)$$

The steepest-edge rule chooses $q \in \mathcal{N}$ to minimize this quantity.

If we had to compute each η_i by solving $Bd = A_i$ for each $i \in \mathcal{N}$, the steepest-edge strategy would be prohibitively expensive. Goldfarb and Reid [134] showed that the measure (13.34) of edge steepness for all indices $i \in \mathcal{N}$ can, in fact, be updated quite economically at each iteration. We outline their steepest-edge procedure by showing how each $c^T \eta_i$ and $\|\eta_i\|$ can be updated at the current iteration.

First, note that we already know the numerator $c^T \eta_i$ in (13.34) without calculating η_i , because by taking the inner product of (13.32) with c and using (13.24), we have that $c^T \eta_i = s_i$. To investigate the change in denominator $\|\eta_i\|$ at this step, we define $\gamma_i = \|\eta_i\|^2$, where this quantity is defined before and after the update as follows:

$$\gamma_i = \|\eta_i\|^2 = \|B^{-1}A_i\|^2 + 1, \quad (13.35a)$$

$$\gamma_i^+ = \|\eta_i^+\|^2 = \|(B^+)^{-1}A_i\|^2 + 1. \quad (13.35b)$$

Assume without loss of generality that the entering column A_q replaces the first column of the basis matrix B (that is, $p = 1$), and that this column corresponds to the index t . We can then express the update to B as follows:

$$B^+ = B + (A_q - A_t)e_1^T = B + (A_q - Be_1)e_1^T, \quad (13.36)$$

where $e_1 = (1, 0, 0, \dots, 0)^T$. By applying the Sherman–Morrison formula (A.27) to the rank-one update formula in (13.36), we obtain

$$(B^+)^{-1} = B^{-1} - \frac{(B^{-1}A_q - e_1)e_1^T B^{-1}}{1 + e_1^T (B^{-1}A_q - e_1)} = B^{-1} - \frac{(d - e_1)e_1^T B^{-1}}{e_1^T d},$$

where again we have used the fact that $d = B^{-1}A_q$ (see (13.25)). Therefore, we have that

$$(B^+)^{-1}A_i = B^{-1}A_i - \frac{e_1^T B^{-1}A_i}{e_1^T d}(d - e_1).$$

By substituting for $(B^+)^{-1}A_i$ in (13.35) and performing some simple manipulation, we obtain

$$\gamma_i^+ = \gamma_i - 2 \left(\frac{e_1^T B^{-1}A_i}{e_1^T d} \right) A_i^T B^{-T} d + \left(\frac{e_1^T B^{-1}A_i}{e_1^T d} \right)^2 \gamma_q. \quad (13.37)$$

Once we solve the following two linear systems to obtain \hat{d} and r :

$$B^T \hat{d} = d, \quad B^T r = e_1. \quad (13.38)$$

The formula (13.37) then becomes

$$\gamma_i^+ = \gamma_i - 2 \left(\frac{r^T A_i}{r^T A_q} \right) \hat{d}^T A_i + \left(\frac{r^T A_i}{r^T A_q} \right)^2 \gamma_q. \quad (13.39)$$

Hence, the entire set of γ_i values, for $i \in \mathcal{N}$ with $i \neq q$, can be calculated by solving the two systems (13.38) and then evaluating the inner products $r^T A_i$ and $\hat{d}^T A_i$, for each i .

The steepest-edge strategy does not guarantee that we can take a long step before reaching another vertex, but it has proved to be highly effective in practice.

STARTING THE SIMPLEX METHOD

The simplex method requires a basic feasible starting point x and a corresponding initial basis $\mathcal{B} \subset \{1, 2, \dots, n\}$ with $|\mathcal{B}| = m$ such that the basis matrix B defined by (13.13) is nonsingular and $x_{\mathcal{B}} = B^{-1}b \geq 0$ and $x_{\mathcal{N}} = 0$. The problem of finding this initial point and basis may itself be nontrivial—in fact, its difficulty is equivalent to that of actually solving a linear program. We describe here the *two-phase approach* that is commonly used to deal with this difficulty in practical implementations.

In Phase I of this approach we set up an auxiliary linear program based on the data of (13.1), and solve it with the simplex method. The Phase-I problem is designed so that an initial basis and initial basic feasible point is trivial to find, and so that its solution gives a basic feasible initial point for the second phase. In Phase II, a second linear program similar to the original problem (13.1) is solved, with the Phase-I solution as a starting point. The solution of the original problem (13.1) can be extracted easily from the solution of the Phase-II problem.

In Phase I we introduce artificial variables z into (13.1) and redefine the objective function to be the sum of these artificial variables, as follows:

$$\min e^T z, \quad \text{subject to } Ax + Ez = b, (x, z) \geq 0, \quad (13.40)$$

where $z \in \mathbb{R}^m$, $e = (1, 1, \dots, 1)^T$, and E is a diagonal matrix whose diagonal elements are

$$E_{jj} = +1 \text{ if } b_j \geq 0, \quad E_{jj} = -1 \text{ if } b_j < 0.$$

It is easy to see that the point (x, z) defined by

$$x = 0, \quad z_j = |b_j|, \quad j = 1, 2, \dots, m, \quad (13.41)$$

is a basic feasible point for (13.40). Obviously, this point satisfies the constraints in (13.40), while the initial basis matrix B is simply the diagonal matrix E , which is clearly nonsingular.

At any feasible point for (13.40), the artificial variables z represent the amounts by which the constraints $Ax = b$ are violated by the x component. The objective function is

simply the sum of these violations, so by minimizing this sum we are forcing x to become feasible for the original problem (13.1). It is not difficult to see that the Phase-I problem (13.40) has an optimal objective value of zero if and only if the original problem (13.1) is feasible, by using the following argument: If there exists a vector (\tilde{x}, \tilde{z}) that is feasible for (13.40) such that $e^T \tilde{z} = 0$, we must have $\tilde{z} = 0$, and therefore $A\tilde{x} = b$ and $\tilde{x} \geq 0$, so \tilde{x} is feasible for (13.1). Conversely, if \tilde{x} is feasible for (13.1), then the point $(\tilde{x}, 0)$ is feasible for (13.40) with an objective value of 0. Since the objective in (13.40) is obviously nonnegative at all feasible points, then $(\tilde{x}, 0)$ must be optimal for (13.40), verifying our claim.

In Phase I, we apply the simplex method to (13.40) from the initial point (13.41). This linear program cannot be unbounded, because its objective function is bounded below by 0, so the simplex method will terminate at an optimal point (assuming that it does not cycle; see below). If the objective $e^T \tilde{z}$ is positive at this solution, we conclude by the argument above that the original problem (13.1) is infeasible. Otherwise, the simplex method identifies a point (\tilde{x}, \tilde{z}) with $e^T \tilde{z} = 0$, which is also a basic feasible point for the following *Phase-II problem*:

$$\min c^T x \quad \text{subject to } Ax + z = b, \quad x \geq 0, \quad 0 \geq z \geq 0. \quad (13.42)$$

Note that this problem differs from (13.40) in that the objective function is replaced by the original objective $c^T x$, while upper bounds of 0 have been imposed on z . In fact, (13.42) is equivalent to (13.1), because any solution (and indeed any feasible point) must have $z = 0$. We need to retain the artificial variables z in Phase II, however, since some components of z may still be present in the optimal basis from Phase I that we are using as the initial basis for (13.42), though of course the values \tilde{z}_j of these components must be zero. In fact, we can modify (13.42) to include *only* those components of z that are present in the optimal basis for (13.40).

The problem (13.42) is not quite in standard form because of the two-sided bounds on z . However, it is easy to modify the simplex method described above to handle upper and lower bounds on the variables (we omit the details). We can customize the simplex algorithm slightly by deleting each component of z from the problem (13.42) as soon as it is swapped out of the basis. This strategy ensures that components of z do not repeatedly enter and leave the basis, thereby avoiding unnecessary simplex iterations.

If (x^*, z^*) is a basic solution of (13.42), it must have $z^* = 0$, and so x^* is a solution of (13.1). In fact, x^* is a basic feasible point for (13.1), though this claim is not completely obvious because the final basis \mathcal{B} for the Phase-II problem may still contain components of z^* , making it unsuitable as an optimal basis for (13.1). Since A has full row rank, however, we can construct an optimal basis for (13.1) in a postprocessing phase: Extract from \mathcal{B} any components of z that are present, and replace them with nonbasic components of x in a way that maintains nonsingularity of the submatrix B defined by (13.13).

A final point to note is that in many problems we do not need to add a complete set of m artificial variables to form the Phase-I problem. This observation is particularly relevant when slack and surplus variables have already been added to the problem formulation, as

in (13.2), to obtain a linear program with inequality constraints in standard form (13.1). Some of these slack/surplus variables can play the roles of artificial variables, making it unnecessary to include such variables explicitly.

We illustrate this point with the following example.

□ **EXAMPLE 13.2**

Consider the inequality-constrained linear program defined by

$$\begin{aligned} \min \quad & 3x_1 + x_2 + x_3 \quad \text{subject to} \\ & 2x_1 + x_2 + x_3 \leq 2, \\ & x_1 - x_2 - x_3 \leq -1, \\ & x \geq 0. \end{aligned}$$

By adding slack variables to both inequality constraints, we obtain the following equivalent problem in standard form:

$$\begin{aligned} \min \quad & 3x_1 + x_2 + x_3 \quad \text{subject to} \\ & 2x_1 + x_2 + x_3 + x_4 = 2, \\ & x_1 - x_2 - x_3 + x_5 = -1, \\ & x \geq 0. \end{aligned}$$

By inspection, it is easy to see that the vector $x = (0, 0, 0, 2, 0)$ is feasible with respect to the first linear constraint and the lower bound $x \geq 0$, though it does not satisfy the second constraint. Hence, in forming the Phase-I problem, we add just a single artificial variable z_2 to the second constraint and obtain

$$\min z_2 \quad \text{subject to} \tag{13.43}$$

$$2x_1 + x_2 + x_3 + x_4 = 2, \tag{13.44}$$

$$x_1 - x_2 - x_3 + x_5 - z_2 = -1, \tag{13.45}$$

$$(x, z_2) \geq 0. \tag{13.46}$$

It is easy to see that the vector $(x, z_2) = ((0, 0, 0, 2, 0), 1)$ is feasible with respect to (13.43). In fact, it is a basic feasible point, since the corresponding basis matrix B is

$$B = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

which is clearly nonsingular. In this example, the variable x_4 plays the role of artificial variable for the first constraint. There was no need to add an explicit artificial variable z_1 . □

DEGENERATE STEPS AND CYCLING

As noted above, the simplex method may encounter situations in which for the entering index q , we cannot set x_q^+ any greater than zero in (13.22) without violating the nonnegativity condition $x^+ \geq 0$. By referring to Procedure 13.1, we see that these situations arise when there is i with $(x_B)_i = 0$ and $d_i < 0$, where d is defined by (13.25). Steps of this type are called *degenerate steps*. On such steps, the components of x do not change and, therefore, the objective function $c^T x$ does not decrease. However, the steps may still be useful because they change the basis \mathcal{B} (by replacing one index), and the updated \mathcal{B} may be closer to the optimal basis. In other words, the degenerate step may be laying the groundwork for reductions in $c^T x$ on later steps.

Sometimes, however, a phenomenon known as *cycling* can occur. After a number of successive degenerate steps, we may return to an earlier basis \mathcal{B} . If we continue to apply the algorithm from this point using the same rules for selecting entering and leaving indices, we will repeat the same cycle ad infinitum, never converging.

Cycling was once thought to be a rare phenomenon, but in recent times it has been observed frequently in the large linear programs that arise as relaxations of integer programming problems. Since integer programs are an important source of linear programs, practical simplex codes usually incorporate a cycling avoidance strategy.

In the remainder of this section, we describe a *perturbation strategy* and its close relative, the *lexicographic strategy*.

Suppose that a degenerate basis is encountered at some simplex iteration, at which the basis is $\hat{\mathcal{B}}$ and the basis matrix is \hat{B} , say. We consider a modified linear program in which we add a small perturbation to the right-hand side of the constraints in (13.1), as follows:

$$b(\epsilon) = b + \hat{B} \begin{bmatrix} \epsilon \\ \epsilon^2 \\ \vdots \\ \epsilon^m \end{bmatrix},$$

where ϵ is a very small positive number. This perturbation in b induces a perturbation in the components of the basic solution vector; we have

$$x_{\hat{B}}(\epsilon) = x_{\hat{B}} + \begin{bmatrix} \epsilon \\ \epsilon^2 \\ \vdots \\ \epsilon^m \end{bmatrix}. \quad (13.47)$$

Retaining the perturbation for subsequent iterations, we see that subsequent basic solutions have the form

$$x_B(\epsilon) = x_B + B^{-1}\hat{B} \begin{bmatrix} \epsilon \\ \epsilon^2 \\ \vdots \\ \epsilon^m \end{bmatrix} = x_B + \sum_{k=1}^m (B^{-1}\hat{B})_{.k} \epsilon^k, \quad (13.48)$$

where $(B^{-1}\hat{B})_{.k}$ denotes the k th column of $B^{-1}\hat{B}$ and x_B represents the basic solution for the unperturbed right-hand side b .

From (13.47), we have that for all ϵ sufficiently small (but positive), $(x_B(\epsilon))_i > 0$ for all i . Hence, the basis is nondegenerate for the perturbed problem, and we can perform a step of the simplex method that produces a nonzero (but tiny) decrease in the objective.

Indeed, if we retain the perturbation over all subsequent iterations, and provided that the initial choice of ϵ was small enough, we claim that *all* subsequent bases visited by the algorithm are nondegenerate. We prove this claim by contradiction, by assuming that there is some basis matrix B such that $(x_B(\epsilon))_i = 0$ for some i and all ϵ sufficiently small. From (13.48), we see that this can happen only when $(x_B)_i = 0$ and $(B^{-1}\bar{B})_{ik} = 0$ for $k = 1, 2, \dots, m$. The latter relation implies that the i th row of $B^{-1}\bar{B}$ is zero, which cannot occur, because both B and \bar{B} are nonsingular.

We conclude that, provided the initial choice of ϵ is sufficiently small to ensure nondegeneracy of all subsequent bases, no basis is visited more than once by the simplex method and therefore, by the same logic as in the proof of Theorem 13.4, the method terminates finitely at a solution of the perturbed problem. The perturbation can be removed in a postprocessing phase, by resetting $x_B = B^{-1}b$ for the final basis B and the original right-hand side b .

The question remains of how to choose ϵ small enough at the point at which the original degenerate basis \hat{B} is encountered. The *lexicographic strategy* finesses this issue by not making an explicit choice of ϵ , but rather keeping track of the dependence of each basic variable on each power of ϵ . When it comes to selecting the leaving variable, it chooses the index p that minimizes $(x_B(\epsilon))_i/d_i$ over all variables in the basis, for all sufficiently small ϵ . (The choice of p is uniquely defined by this procedure, as we can show by an argument similar to the one above concerning nondegeneracy of each basis.) We can extend the pivot procedure slightly to update the dependence of each basic variable on the powers of ϵ at each iteration, including the variable x_q that has just entered the basis.

13.6 THE DUAL SIMPLEX METHOD

Here we describe another variant of the simplex method that is useful in a variety of situations and is often faster on many practical problems than the variant described above. This *dual*

suppose that the following two equality constraints are present in the problem:

$$3x_2 = 6, \quad x_2 + 4x_5 = 10.$$

The first of these constraints is a row singleton, which we can use to set $x_2 = 2$ and eliminate this variable and constraint. After substitution, the second constraint becomes $4x_5 = 10 - x_2 = 8$, which is again a row singleton. We can therefore set $x_5 = 2$ and eliminate this variable and constraint as well.

Relatively little information about presolving techniques has appeared in the literature, in part because they have commercial value as an important component of linear programming software.

13.8 WHERE DOES THE SIMPLEX METHOD FIT?

In linear programming, as in all optimization problems in which inequality constraints are present, the fundamental task of the algorithm is to determine which of these constraints are active at the solution (see Definition 12.1 and which are inactive. The simplex method belongs to a general class of algorithms for constrained optimization known as *active set methods*, which explicitly maintain estimates of the active and inactive index sets that are updated at each step of the algorithm. (At each iteration, the basis \mathcal{B} is our current estimate of the inactive set, that is, the set of indices i for which we suspect that $x_i > 0$ at the solution of the linear program.) Like most active set methods, the simplex method makes only modest changes to these index sets at each step; a single index is exchanged between \mathcal{B} into \mathcal{N} .

Active set algorithms for quadratic programming, bound-constrained optimization, and nonlinear programming use the same basic strategy as simplex of making an explicit estimate of the active set and taking a step toward the solution of a reduced problem in which the constraints in this estimated active set are satisfied as equalities. When nonlinearity enters the problem, many of the features that make the simplex method so effective no longer apply. For example, it is no longer true in general that at least $n - m$ of the bounds $x \geq 0$ are active at the solution, and the specialized linear algebra techniques described in Section 13.5 no longer apply. Nevertheless, the simplex method is rightly viewed as the antecedent of the active set class of methods for constrained optimization.

One undesirable feature of the simplex method attracted attention from its earliest days. Though highly efficient on almost all practical problems (the method generally requires at most $2m$ to $3m$ iterations, where m is the row dimension of the constraint matrix in (13.1)), there are pathological problems on which the algorithm performs very poorly. Klee and Minty [182] presented an n -dimensional problem whose feasible polytope has 2^n vertices, for which the simplex method visits every single vertex before reaching the optimal point! This example verified that the complexity of the simplex method is *exponential*;

roughly speaking, its running time may be an exponential function of the dimension of the problem. For many years, theoreticians searched for a linear programming algorithm that has *polynomial complexity*, that is, an algorithm in which the running time is bounded by a polynomial function of the amount of storage required to define the problem. In the late 1970s, Khachiyan [180] described an *ellipsoid method* that indeed has polynomial complexity but turned out to be impractical. In the mid-1980s, Karmarkar [175] described a polynomial algorithm that approaches the solution through the interior of the feasible polytope rather than working its way around the boundary as the simplex method does. Karmarkar's announcement marked the start of intense research in the field of *interior-point methods*, which are the subject of the next chapter.

NOTES AND REFERENCES

The standard reference for the simplex method is Dantzig's book [86]. Later excellent texts include Chvátal [61] and Vanderbei [293].

Further information on steepest-edge pivoting can be found in Goldfarb and Reid [134] and Goldfarb and Forrest [133].

An alternative procedure for performing the Phase-I calculation of an initial basis was described by Wolfe [310]. This technique does not require artificial variables to be introduced in the problem formulation, but rather starts at any point x that satisfies $Ax = b$ with at most m nonzero components in x . (Note that we do not require the basic part x_b to consist of all positive components.) Phase I then consists in solving the problem

$$\min_x \sum_{x_i < 0} -x_i \quad \text{subject to } Ax = b,$$

and terminating when an objective value of 0 is attained. This problem is *not* a linear program—its objective is only piecewise linear—but it can be solved by the simplex method nonetheless. The key is to *redefine* the cost vector f at each iteration x such that $f_i = -1$ for $x_i < 0$ and $f_i = 0$ otherwise.



EXERCISES




13.1 Convert the following linear program to standard form:


$$\max_{x,y} c^T x + d^T y \quad \text{subject to } A_1 x = b_1, \quad A_2 x + B_2 y \leq b_2, \quad l \leq y \leq u,$$


where there are no explicit bounds on x .



13.2 Verify that the dual of (13.8) is the original primal problem (13.1).

 **13.3** Complete the proof of Theorem 13.1 by showing that if the dual (13.7) is unbounded above, the primal (13.1) must be infeasible.


 **13.4** Theorem 13.1 does not exclude the possibility that both primal and dual are infeasible. Give a simple linear program for which such is the case.


 **13.5** Show that the dual of the linear program

$$\min c^T x \quad \text{subject to } Ax \geq b, \quad x \geq 0,$$

is

$$\max b^T \lambda \quad \text{subject to } A^T \lambda \leq c, \quad \lambda \geq 0.$$


 **13.6** Show that when $m \leq n$ and the rows of A are linearly dependent in (13.1), then the matrix B in (13.13) is singular, and therefore there are no basic feasible points.


 **13.7** Consider the overdetermined linear system $Ax = b$ with m rows and n columns ($m > n$). When we apply Gaussian elimination with complete pivoting to A , we obtain

$$PAQ = L \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix},$$

where P and Q are permutation matrices, L is $m \times m$ lower triangular, U_{11} is $\bar{m} \times \bar{m}$ upper triangular and nonsingular, U_{12} is $\bar{m} \times (n - \bar{m})$, and $\bar{m} \leq n$ is the rank of A .


- (a) Show that the system $Ax = b$ is feasible if the last $m - \bar{m}$ components of $L^{-1}Pb$ are zero, and infeasible otherwise.
- (b) When $\bar{m} = n$, find the unique solution of $Ax = b$.
- (c) Show that the reduced system formed from the first \bar{m} rows of PA and the first \bar{m} components of Pb is equivalent to $Ax = b$ (i.e., a solution of one system also solves the other).


 **13.8** Verify formula (13.37).

 **13.9** Consider the following linear program:

$$\begin{aligned} \min \quad & -5x_1 - x_2 \quad \text{subject to} \\ & x_1 + x_2 \leq 5, \\ & 2x_1 + (1/2)x_2 \leq 8, \\ & x \geq 0. \end{aligned}$$

- (a) Add slack variables x_3 and x_4 to convert this problem to standard form.
- (b) Using Procedure 13.1, solve this problem using the simplex method, showing at each step the basis and the vectors λ , s_N , and x_B , and the value of the objective function. (The initial choice of \mathcal{B} for which $x_B \geq 0$ should be obvious once you have added the slacks in part (a).)

 **13.10** Calculate the values of l_{52} , l_{53} , l_{54} , and \hat{w}_2 in (13.30), by equating the last row of $L_1 U_1$ to the last row of the matrix in (13.29).

 **13.11** By extending the procedure (13.27) appropriately, show how the factorization (13.31) can be used to solve linear systems with coefficient matrix B^+ efficiently.

CHAPTER *14*

Linear Programming: Interior-Point Methods

In the 1980s it was discovered that many large linear programs could be solved efficiently by using formulations and algorithms from nonlinear programming and nonlinear equations. One characteristic of these methods was that they required all iterates to satisfy the inequality constraints in the problem *strictly*, so they became known as interior-point methods. By the early 1990s, a subclass of interior-point methods known as *primal-dual methods* had distinguished themselves as the most efficient practical approaches, and proved to be strong competitors to the simplex method on large problems. These methods are the focus of this chapter.

while the final three possibilities are obtained by making exactly two constraints active (that is, $\mathcal{W} = \{1, 2\}$, $\mathcal{W} = \{1, 3\}$, and $\mathcal{W} = \{2, 3\}$). We consider three of these cases in detail.

- $\mathcal{W} = \{2\}$; that is, only the constraint $x = 0$ is active. If we minimize f enforcing only this constraint, we obtain the point $(0, 1/2)^T$. A check of the KKT conditions (12.34) shows that no matter how we choose the Lagrange multipliers, we cannot satisfy all these conditions at $(0, 1/2)^T$. (We must have $\lambda_1 = \lambda_3 = 0$ to satisfy (12.34e), which implies that we must set $\lambda_2 = -2$ to satisfy (12.34a); but this value of λ_2 violates the condition (12.34d).)
- $\mathcal{W} = \{1, 3\}$, which yields the single feasible point $(3, 0)^T$. Since constraint 2 is inactive at this point, we have $\lambda_2 = 0$, so by solving (12.34a) for the other Lagrange multipliers, we obtain $\lambda_1 = -16$ and $\lambda_3 = -16.5$. These values are negative, so they violate (12.34d), and $x = (3, 0)^T$ cannot be a solution of (15.1).
- $\mathcal{W} = \{1\}$. Solving the equality-constrained problem in which the first constraint is active, we obtain $(x, y)^T = (1.953, 0.089)^T$ with Lagrange multiplier $\lambda_1 = 0.411$. It is easy to see that by setting $\lambda_2 = \lambda_3 = 0$, the remaining KKT conditions (12.34) are satisfied, so we conclude that this is a KKT point. Furthermore, it is easy to show that the second-order sufficient conditions are satisfied, as the Hessian of the Lagrangian is positive definite.



Even for this small example, we see that it is exhausting to consider all possible choices for \mathcal{W} . Figure 15.1 suggests, however, that some choices of \mathcal{W} can be eliminated from consideration if we make use of knowledge of the functions that define the problem, and their derivatives. In fact, the active set methods described in Chapter 16 use this kind of information to make a series of educated guesses for the working set, avoiding choices of \mathcal{W} that obviously will not lead to a solution of (15.1).

A different approach is followed by interior-point (or barrier) methods discussed in Chapter 19. These methods generate iterates that stay away from the boundary of the feasible region defined by the inequality constraints. As the solution of the nonlinear program is approached, the barrier effects are weakened to permit an increasingly accurate estimate of the solution. In this manner, interior-point methods avoid the combinatorial difficulty of nonlinear programming.

15.3 ELIMINATION OF VARIABLES

When dealing with constrained optimization problems, it is natural to try to use the constraints to eliminate some of the variables from the problem, to obtain a simpler problem with fewer degrees of freedom. Elimination techniques must be used with care, however, as they may alter the problem or introduce ill conditioning.

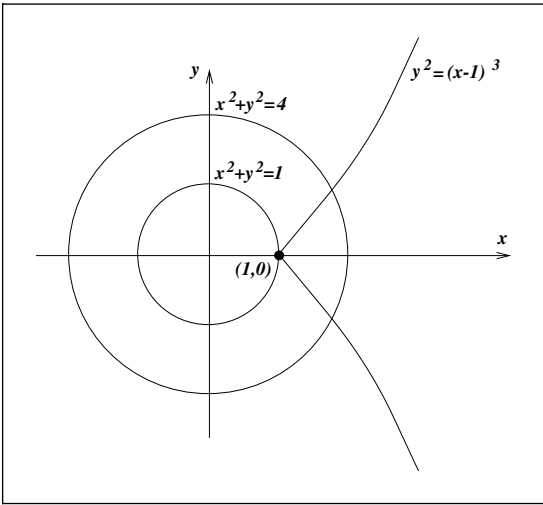


Figure 15.2
The danger of nonlinear
elimination.

We begin with an example in which it is safe and convenient to eliminate variables. In the problem

$$\begin{aligned} \min f(x) = f(x_1, x_2, x_3, x_4) \quad \text{subject to} \quad & x_1 + x_3^2 - x_4x_3 = 0, \\ & -x_2 + x_4 + x_3^2 = 0, \end{aligned}$$

there is no risk in setting

$$x_1 = x_4x_3 - x_3^2, \quad x_2 = x_4 + x_3^2,$$

to obtain a function of two variables

$$h(x_3, x_4) = f(x_4x_3 - x_3^2, x_4 + x_3^2, x_3, x_4),$$

which we can minimize using the unconstrained optimization techniques described in earlier chapters.

The dangers of nonlinear elimination are illustrated in the following example.

EXAMPLE 15.2 (FLETCHER [101])

Consider the problem

$$\min x^2 + y^2 \quad \text{subject to} \quad (x - 1)^3 = y^2.$$

The contours of the objective function and the constraints are illustrated in Figure 15.2, which shows that the solution is $(x, y) = (1, 0)$.

We attempt to solve this problem by eliminating y . By doing so, we obtain

$$h(x) = x^2 + (x - 1)^3.$$

Clearly, $h(x) \rightarrow -\infty$ as $x \rightarrow -\infty$. By blindly applying this transformation we may conclude that the problem is unbounded, but this view ignores the fact that the constraint $(x - 1)^3 = y^2$ implicitly imposes the bound $x \geq 1$ that is active at the solution. Hence, if we wish to eliminate y , we should explicitly introduce the bound $x \geq 1$ into the problem. □

This example shows that the use of nonlinear equations to eliminate variables may result in errors that can be difficult to trace. For this reason, nonlinear elimination is not used by most optimization algorithms. Instead, many algorithms linearize the constraints and apply elimination techniques to the simplified problem. We now describe systematic procedures for performing variable elimination using linear constraints.

SIMPLE ELIMINATION USING LINEAR CONSTRAINTS

We consider the minimization of a nonlinear function subject to a set of linear equality constraints,

$$\min f(x) \quad \text{subject to } Ax = b, \quad (15.6)$$

where A is an $m \times n$ matrix with $m \leq n$. Suppose for simplicity that A has full row rank. (If such is not the case, we find either that the problem is inconsistent or that some of the constraints are redundant and can be deleted without affecting the solution of the problem.) Under this assumption, we can find a subset of m columns of A that is linearly independent. If we gather these columns into an $m \times m$ matrix B and define an $n \times n$ permutation matrix P that swaps these columns to the first m column positions in A , we can write

$$AP = [B \mid N], \quad (15.7)$$

where N denotes the $n - m$ remaining columns of A . (The notation here is consistent with that of Chapter 13, where we discussed similar concepts in the context of linear programming.) We define the subvectors $x_B \in \mathbb{R}^m$ and $x_N \in \mathbb{R}^{n-m}$ as follows:

$$\begin{bmatrix} x_B \\ x_N \end{bmatrix} = P^T x, \quad (15.8)$$

and call x_B the *basic variables* and B the *basis matrix*. Noting that $PP^T = I$, we can rewrite the constraint $Ax = b$ as

$$b = Ax = AP(P^Tx) = Bx_B + Nx_N.$$

By rearranging this formula, we deduce that the basic variables can be expressed as follows:

$$x_B = B^{-1}b - B^{-1}Nx_N. \quad (15.9)$$

We can therefore compute a feasible point for the constraints $Ax = b$ by choosing *any* value of x_N and then setting x_B according to the formula (15.9). The problem (15.6) is therefore equivalent to the unconstrained problem

$$\min_{x_N} h(x_N) \stackrel{\text{def}}{=} f\left(P \begin{bmatrix} B^{-1}b - B^{-1}Nx_N \\ x_N \end{bmatrix}\right). \quad (15.10)$$

We refer to the substitution in (15.9) as *simple elimination of variables*.

This discussion shows that a nonlinear optimization problem with linear equality constraints is, from a mathematical point of view, the same as an unconstrained problem.

□ EXAMPLE 15.3

Consider the problem

$$\min \sin(x_1 + x_2) + x_3^2 + \frac{1}{3}(x_4 + x_5^4 + x_6/2) \quad (15.11a)$$

$$\begin{aligned} \text{subject to} \quad & 8x_1 - 6x_2 + x_3 + 9x_4 + 4x_5 = 6 \\ & 3x_1 + 2x_2 - x_4 + 6x_5 + 4x_6 = -4. \end{aligned} \quad (15.11b)$$

By defining the permutation matrix P so as to reorder the components of x as $x^T = (x_3, x_6, x_1, x_2, x_4, x_5)^T$, we find that the coefficient matrix AP is

$$AP = \left[\begin{array}{cc|cc} 1 & 0 & 8 & -6 & 9 & 4 \\ 0 & 4 & 3 & 2 & -1 & 6 \end{array} \right].$$

The basis matrix B is diagonal and therefore easy to invert. We obtain from (15.9) that

$$\begin{bmatrix} x_3 \\ x_6 \end{bmatrix} = - \begin{bmatrix} 8 & -6 & 9 & 4 \\ 3 & 1 & -1 & 3 \\ 4 & 2 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} 6 \\ -1 \end{bmatrix}. \quad (15.12)$$

By substituting for x_3 and x_6 in (15.11a), the problem becomes

$$\begin{aligned} \min_{x_1, x_2, x_4, x_5} \quad & \sin(x_1 + x_2) + (8x_1 - 6x_2 + 9x_4 + 4x_5 - 6)^2 \\ & + \frac{1}{3}(x_4 + x_5^4 - [(1/2) + (3/8)x_1 + (1/4)x_2 - (1/8)x_4 + (3/4)x_5]). \end{aligned} \quad (15.13)$$

We could have chosen two other columns of the coefficient matrix A (that is, two variables other than x_3 and x_6) as the basis for elimination in the system (15.11b), but the matrix $B^{-1}N$ would not have been so simple. □

A set of m independent columns can be selected, in general, by means of Gaussian elimination. In the parlance of linear algebra, we can compute the row echelon form of the matrix and choose the pivot columns as the columns of the basis B . Ideally, we would like B to be easy to factor and well conditioned. A technique that suits these purposes is a sparse Gaussian elimination approach that attempts to preserve sparsity while keeping rounding errors under control. A well-known implementation of this algorithm is MA48 from the HSL library [96]. As we discuss below, however, there is no guarantee that the Gaussian elimination process will identify the best choice of basis matrix.

There is an interesting interpretation of the simple elimination-of-variables approach that we have just described. To simplify the notation, we will assume from now on that the coefficient matrix is already given to us so that the basic columns appear in the first m positions, that is, $P = I$.

From (15.8) and (15.9) we see that any feasible point x for the linear constraints in (15.6) can be written as

$$\begin{bmatrix} x_B \\ x_N \end{bmatrix} = x = Yb + Zx_N, \quad (15.14)$$

where

$$Y = \begin{bmatrix} B^{-1} \\ 0 \end{bmatrix}, \quad Z = \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}. \quad (15.15)$$

Note that Z has $n - m$ linearly independent columns (because of the presence of the identity matrix in the lower block) and that it satisfies $AZ = 0$. Therefore, Z is a *basis for the null space* of A . In addition, the columns of Y and the columns of Z form a linearly independent set. We note also from (15.15), (15.7) that Yb is a particular solution of the linear constraints $Ax = b$.

In other words, the simple elimination technique expresses feasible points as the sum of a particular solution of $Ax = b$ (the first term in (15.14)) plus a displacement along the

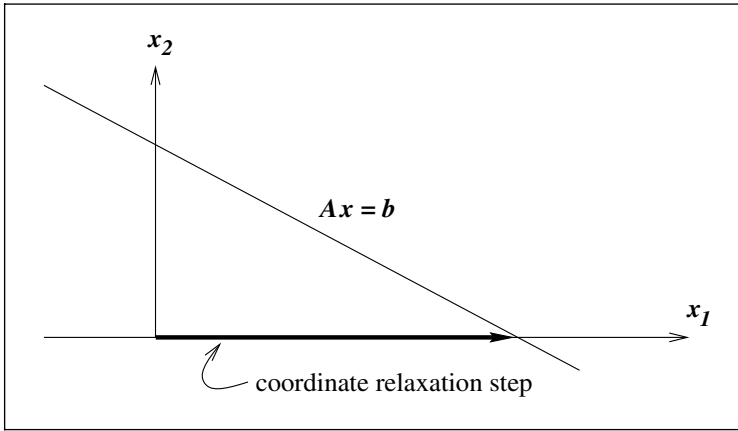


Figure 15.3 Simple elimination, showing the coordinate relaxation step obtained by choosing the basis to be the first column of A .

null space of the constraints (the second term in (15.14)). The relations (15.14), (15.15) indicate that the particular Yb solution is obtained by holding $n - m$ components of x at zero while relaxing the other m components (the ones in x_b) until they reach the constraints. The particular solution Yb is sometimes known as the coordinate relaxation step. In Figure 15.3, we see the coordinate relaxation step Yb obtained by choosing the basis matrix B to be the first column of A . If we were to choose B to be the second column of A , the coordinate relaxation step would lie along the x_2 axis.

Simple elimination is inexpensive but can give rise to numerical instabilities. If the feasible set in Figure 15.3 consisted of a line that was almost parallel to the x_1 axis, the coordinate relaxation along this axis would be very large in magnitude. We would then be computing x as the difference of very large vectors, giving rise to numerical cancellation. In that situation it would be preferable to choose a particular solution along the x_2 axis, that is, to select a different basis. Selection of the best basis is, therefore, not a straightforward task in general. To overcome the dangers of an excessively large coordinate relaxation step, we could define the particular solution Yb as the minimum-norm step to the constraints. This approach is a special case of more general elimination strategies, which we now describe.

GENERAL REDUCTION STRATEGIES FOR LINEAR CONSTRAINTS

To generalize (15.14) and (15.15), we choose matrices $Y \in \mathbb{R}^{n \times m}$ and $Z \in \mathbb{R}^{n \times (n-m)}$ with the following properties:

$$[Y \mid Z] \in \mathbb{R}^{n \times n} \text{ is nonsingular, } AZ = 0. \quad (15.16)$$

These properties indicate that, as in (15.15), the columns of Z are a basis for the null space of A . Since A has full row rank, so does $A[Y \mid Z] = [AY \mid 0]$, so it follows that the $m \times m$ matrix AY is nonsingular. We now express any solution of the linear constraints $Ax = b$ as

$$x = Yx_Y + Zx_Z, \quad (15.17)$$

for some vectors $x_Y \in \mathbb{R}^m$ and $x_Z \in \mathbb{R}^{n-m}$. By substituting (15.17) into the constraints $Ax = b$, we obtain

$$Ax = (AY)x_Y = b;$$

hence by nonsingularity of AY , x_Y can be written explicitly as

$$x_Y = (AY)^{-1}b. \quad (15.18)$$

By substituting this expression into (15.17), we conclude that any vector x of the form

$$x = Y(AY)^{-1}b + Zx_Z \quad (15.19)$$

satisfies the constraints $Ax = b$ for any choice of $x_Z \in \mathbb{R}^{n-m}$. Therefore, the problem (15.6) can be restated equivalently as the following unconstrained problem

$$\min_{x_Z} f(Y(AY)^{-1}b + Zx_Z). \quad (15.20)$$

Ideally, we would like to choose Y in such a way that the matrix AY is as well conditioned as possible, since it needs to be factorized to give the particular solution $Y(AY)^{-1}b$. We can do this by computing Y and Z by means of a QR factorization of A^T , which has the form

$$A^T \Pi = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad (15.21)$$

where $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is orthogonal. The submatrices Q_1 and Q_2 have orthonormal columns and are of dimension $n \times m$ and $n \times (n - m)$, while R is $m \times m$ upper triangular and nonsingular and Π is an $m \times m$ permutation matrix. (See the discussion following (A.24) in the Appendix for further details.) We now define

$$Y = Q_1, \quad Z = Q_2, \quad (15.22)$$

so that the columns of Y and Z form an *orthonormal basis* of \mathbb{R}^n . If we expand (15.21) and do a little rearrangement, we obtain

$$AY = \Pi R^T, \quad AZ = 0.$$

Therefore, Y and Z have the desired properties, and the condition number of AY is the same as that of R , which in turn is the same as that of A itself. From (15.19) we see that any solution of $Ax = b$ can be expressed as

$$x = Q_1 R^{-T} \Pi^T b + Q_2 x_z,$$

for some vector x_z . The computation $R^{-T} \Pi^T b$ can be carried out inexpensively, at the cost of a single triangular substitution.

A simple computation shows that the particular solution $Q_1 R^{-T} \Pi^T b$ can also be written as $A^T (AA^T)^{-1} b$. This vector is the solution of the following problem:

$$\min \|x\|^2 \quad \text{subject to } Ax = b;$$

that is, it is the minimum-norm solution of $Ax = b$. See Figure 15.5 for an illustration of this step.

Elimination via the orthogonal basis (15.22) is ideal from the point of view of numerical stability. The main cost associated with this reduction strategy is in computing the QR factorization (15.21). Unfortunately, for problems in which A is large and sparse, a sparse QR factorization can be much more costly to compute than the sparse Gaussian elimination strategy used in simple elimination. Therefore, other elimination strategies have been developed that seek a compromise between these two techniques; see Exercise 15.7.

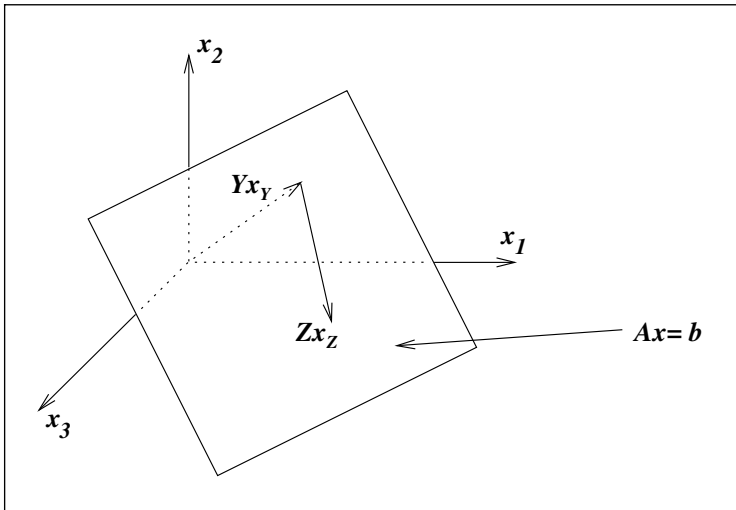


Figure 15.4 General elimination: Case in which $A \in \mathbb{R}^{1 \times 3}$, showing the particular solution and a step in the null space of A .

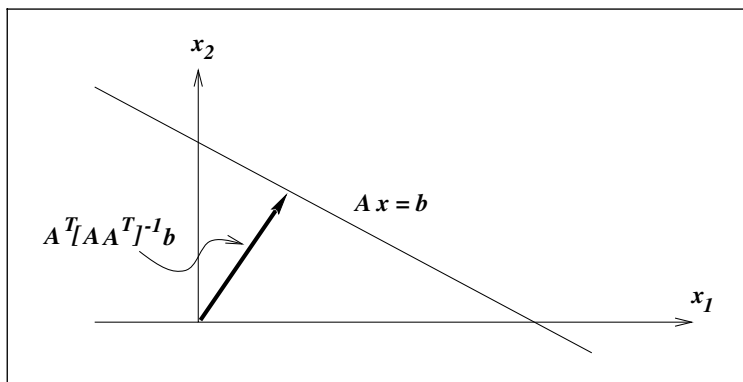


Figure 15.5 The minimum-norm step.

EFFECT OF INEQUALITY CONSTRAINTS

Elimination of variables is not always beneficial if inequality constraints are present alongside the equalities. For instance, if problem (15.11) had the additional constraint $x \geq 0$, then after eliminating the variables x_3 and x_6 , we would be left with the problem of minimizing the function in (15.13) subject to the constraints

$$\begin{aligned} (x_1, x_2, x_4, x_5) &\geq 0, \\ 8x_1 - 6x_2 + 9x_4 + 4x_5 &\leq 6, \\ (3/4)x_1 + (1/2)x_2 - (1/4)x_4 + (3/2)x_5 &\leq -1. \end{aligned}$$

Hence, the cost of eliminating the equality constraints (15.11b) is to make the inequalities more complicated than the simple bounds $x \geq 0$. For many algorithms, this transformation will not yield any benefit.

If, however, problem (15.11) included the general inequality constraint $3x_1 + 2x_3 \geq 1$, the elimination (15.12) would transform the problem into one of minimizing the function in (15.13) subject to the inequality constraint

$$-13x_1 + 12x_2 - 18x_4 - 8x_5 \geq -11. \quad (15.23)$$

In this case, the inequality constraint would not become much more complicated after elimination of the equality constraints, so it is probably worthwhile to perform the elimination.

15.4 MERIT FUNCTIONS AND FILTERS

Suppose that an algorithm for solving the nonlinear programming problem (15.1) generates a step that reduces the objective function but increases the violation of the constraints. Should we accept this step?

This question is not easy to answer. We must look for a way to balance the twin (often competing) goals of reducing the objective function and satisfying the constraints. Merit functions and filters are two approaches for achieving this balance. In a typical constrained optimization algorithm, a step p will be accepted only if it leads to a sufficient reduction in the merit function ϕ or if it is acceptable to the filter. These concepts are explained in the rest of the section.

MERIT FUNCTIONS

In unconstrained optimization, the objective function f is the natural choice for the merit function. All the unconstrained optimization methods described in this book require that f be decreased at each step (or at least within a certain number of iterations). In feasible methods for constrained optimization in which the starting point and all subsequent iterates satisfy all the constraints in the problem, the objective function is still an appropriate merit function. On the other hand, algorithms that allow iterates to violate the constraints require some means to assess the quality of the steps and iterates. The merit function in this case combines the objective with measures of constraint violation.

A popular choice of merit function for the nonlinear programming problem (15.1) is the ℓ_1 penalty function defined by

$$\phi_1(x; \mu) = f(x) + \mu \sum_{i \in \mathcal{E}} |c_i(x)| + \mu \sum_{i \in \mathcal{I}} [c_i(x)]^-, \quad (15.24)$$

where we use the notation $[z]^- = \max\{0, -z\}$. The positive scalar μ is the *penalty parameter*, which determines the weight that we assign to constraint satisfaction relative to minimization of the objective. The ℓ_1 merit function ϕ_1 is not differentiable because of the presence of the absolute value and $[\cdot]^-$ functions, but it has the important property of being *exact*.

Definition 15.1 (Exact Merit Function).

A merit function $\phi(x; \mu)$ is exact if there is a positive scalar μ^ such that for any $\mu > \mu^*$, any local solution of the nonlinear programming problem (15.1) is a local minimizer of $\phi(x; \mu)$.*

We show in Theorem 17.3 that, under certain assumptions, the ℓ_1 merit function $\phi_1(x; \mu)$ is exact and that the threshold value μ^* is given by

$$\mu^* = \max\{|\lambda_i^*|, i \in \mathcal{E} \cup \mathcal{I}\},$$

where the λ_i^* denote the Lagrange multipliers associated with an optimal solution x^* . Since the optimal Lagrange multipliers are, however, not known in advance, algorithms based on the ℓ_1 merit function contain rules for adjusting the penalty parameter whenever there is reason to believe that it is not large enough (or is excessively large). These rules depend on the choice of optimization algorithm and are discussed in the next chapters.

Another useful merit function is the exact ℓ_2 function, which for equality-constrained problems takes the form

$$\phi_2(x; \mu) = f(x) + \mu \|c(x)\|_2. \quad (15.25)$$

This function is nondifferentiable because the 2-norm term is not squared; its derivative is not defined at x for which $c(x) = 0$.

Some merit functions are both smooth and exact. To ensure that both properties hold, we must include additional terms in the merit function. For equality-constrained problems, *Fletcher's augmented Lagrangian* is given by

$$\phi_F(x; \mu) = f(x) - \lambda(x)^T c(x) + \frac{1}{2} \mu \sum_{i \in \mathcal{E}} c_i(x)^2, \quad (15.26)$$

where $\mu > 0$ is the penalty parameter and

$$\lambda(x) = [A(x)A(x)^T]^{-1} A(x) \nabla f(x). \quad (15.27)$$

(Here $A(x)$ denotes the Jacobian of $c(x)$.) Although this merit function has some interesting theoretical properties, it has practical limitations, including the expense of solving for $\lambda(x)$ in (15.27).

A quite different merit function is the (standard) augmented Lagrangian in x and λ , which for equality-constrained problems has the form

$$\mathcal{L}_A(x, \lambda; \mu) = f(x) - \lambda^T c(x) + \frac{1}{2} \mu \|c(x)\|_2^2. \quad (15.28)$$

We assess the acceptability of a trial point (x^+, λ^+) by comparing the value of $\mathcal{L}_A(x^+, \lambda^+; \mu)$ with the value at the current iterate, (x, λ) . Strictly speaking, \mathcal{L}_A is not a merit function in the sense that a solution (x^*, λ^*) of the nonlinear programming problem is not in general a minimizer of $\mathcal{L}_A(x, \lambda; \mu)$ but only a stationary point. Although some sequential quadratic programming methods use \mathcal{L}_A successfully as a merit function by adaptively modifying μ and λ , we will not consider its use as a merit function further. Instead, we will focus primarily on the nonsmooth exact penalty functions ϕ_1 and ϕ_2 .

A trial step $x^+ = x + \alpha p$ generated by a line search algorithm will be accepted if it produces a *sufficient decrease* in the merit function $\phi(x; \mu)$. One way to define this concept is analogous to the condition (3.4) used in unconstrained optimization, where the amount

of decrease is not too small relative to the predicted change in the function over the step. The ℓ_1 and ℓ_2 merit functions are not differentiable, but they have a directional derivative. (See (A.51) for background on directional derivatives.) We write the directional derivative of $\phi(x; \mu)$ in the direction p as

$$D(\phi(x; \mu); p).$$

In a line search method, the sufficient decrease condition requires the steplength parameter $\alpha > 0$ to be small enough that the inequality

$$\phi(x + \alpha p; \mu) \leq \phi(x; \mu) + \eta \alpha D(\phi(x; \mu); p), \quad (15.29)$$

is satisfied for some $\eta \in (0, 1)$.

Trust-region methods typically use a quadratic model $q(p)$ to estimate the value of the merit function ϕ after a step p ; see Section 18.5. The sufficient decrease condition can be stated in terms of a decrease in this model, as follows

$$\phi(x + p; \mu) \leq \phi(x; \mu) - \eta(q(0) - q(p)), \quad (15.30)$$

for some $\eta \in (0, 1)$. (The final term in (15.30) is positive, because the step p is computed to decrease the model q .)

FILTERS

Filter techniques are step acceptance mechanisms based on ideas from multiobjective optimization. Our derivation starts with the observation that nonlinear programming has two goals: minimization of the objective function and the satisfaction of the constraints. If we define a measure of infeasibility as

$$h(x) = \sum_{i \in \mathcal{E}} |c_i(x)| + \sum_{i \in \mathcal{I}} [c_i(x)]^-, \quad (15.31)$$

we can write these two goals as

$$\min_x f(x) \quad \text{and} \quad \min_x h(x). \quad (15.32)$$

Unlike merit functions, which combine both problems into a single minimization problem, filter methods keep the two goals in (15.32) separate. Filter methods accept a trial step x^+ as a new iterate if the pair $(f(x^+), h(x^+))$ is not *dominated* by a previous pair $(f_l, h_l) = (f(x_l), h(x_l))$ generated by the algorithm. These concepts are defined as follows.

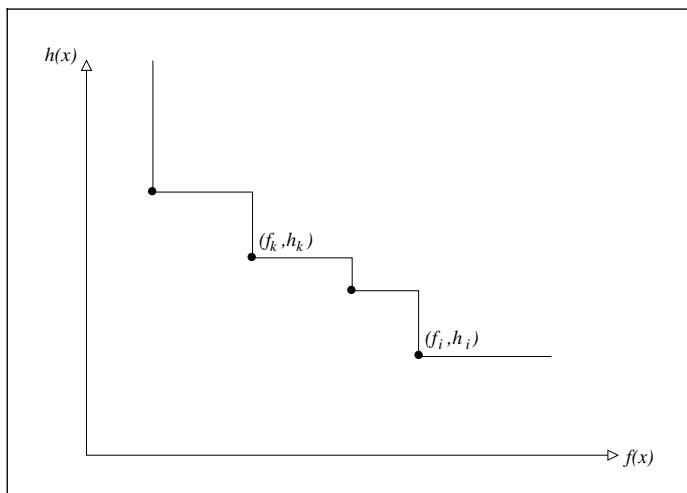


Figure 15.6 Graphical illustration of a filter with four pairs.

Definition 15.2.

- (a) A pair (f_k, h_k) is said to dominate another pair (f_l, h_l) if both $f_k \leq f_l$ and $h_k \leq h_l$.
- (b) A filter is a list of pairs (f_l, h_l) such that no pair dominates any other.
- (c) An iterate x_k is said to be acceptable to the filter if (f_k, h_k) is not dominated by any pair in the filter.

When an iterate x_k is acceptable to the filter, we (normally) add (f_k, h_k) to the filter and remove any pairs that are dominated by (f_k, h_k) . Figure 15.6 shows a filter where each pair (f_l, h_l) in the filter is represented as a black dot. Every point in the filter creates an (infinite) rectangular region, and their union defines the set of pairs not acceptable to the filter. More specifically, a trial point x^+ is acceptable to the filter if (f^+, h^+) lies below or to the left of the solid line in Figure 15.6.

To compare the filter and merit function approaches, we plot in Figure 15.7 the contour line of the set of pairs (f, h) such that $f + \mu h = f_k + \mu h_k$, where x_k is the current iterate. The region to the left of this line corresponds to the set of pairs that reduce the merit function $\phi(x; \mu) = f(x) + \mu h(x)$; clearly this set is quite different from the set of points acceptable to the filter.

If a trial step $x^+ = x_k + \alpha_k p_k$ generated by a line search method gives a pair (f^+, h^+) that is acceptable to the filter, we set $x_{k+1} = x^+$; otherwise, a backtracking line search is performed. In a trust-region method, if the step is not acceptable to the filter, the trust region is reduced, and a new step is computed.

Several enhancements to this filter technique are needed to obtain global convergence and good practical performance. We need to ensure, first of all, that we do not accept a point whose (f, h) pair is very close to the current pair (f_k, h_k) or to another pair in the filter. We

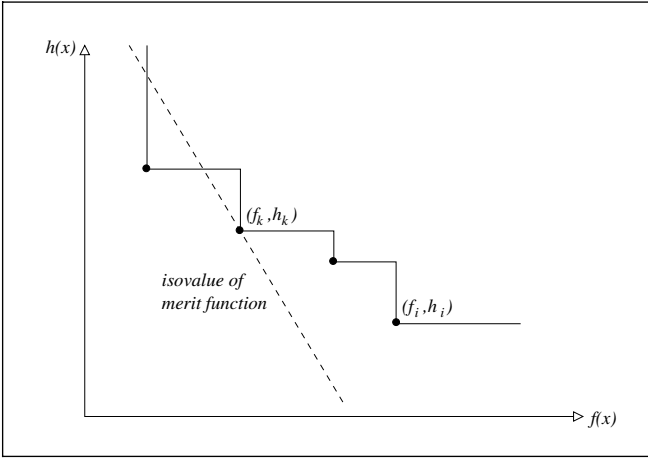


Figure 15.7 Comparing the filter and merit function techniques.

do so by modifying the acceptability criterion and imposing a sufficient decrease condition. A trial iterate x^+ is acceptable to the filter if, for all pairs (f_j, h_j) in the filter, we have that

$$f(x^+) \leq f_j - \beta h_j \quad \text{or} \quad h(x^+) \leq h_j - \beta h_j, \quad (15.33)$$

for $\beta \in (0, 1)$. Although this condition is effective in practice using, say $\beta = 10^{-5}$, for purposes of analysis it may be advantageous to replace the first inequality by

$$f(x^+) \leq f_j - \beta h^+.$$

A second enhancement addresses some problematic aspects of the filter mechanism. Under certain circumstances, the search directions generated by line search methods may require arbitrarily small steplengths α_k to be acceptable to the filter. This phenomenon can cause the algorithm to stall and fail. To guard against this situation, if the backtracking line search generates a steplength that is smaller than a given threshold α_{\min} , the algorithm switches to a *feasibility restoration phase*, which we describe below. Similarly, in a trust-region method, if a sequence of trial steps is rejected by the filter, the trust-region radius may be decreased so much that the trust-region subproblem becomes infeasible (see Section 18.5). In this case, too, the feasibility restoration phase is invoked. (Other mechanisms could be employed to handle this situation, but as we discuss below, the feasibility restoration phase can help the algorithm achieve other useful goals.)

The feasibility restoration phase aims exclusively to reduce the constraint violation, that is, to find an approximate solution to the problem

$$\min_x h(x).$$

Although $h(x)$ defined by (15.31) is not smooth, we show in Chapter 17 how to minimize it using a smooth constrained optimization subproblem. This phase terminates at an iterate that has a sufficiently small value of h and is compatible with the filter.

We now present a framework for filter methods that assumes that iterates are generated by a trust-region method; see Section 18.5 for a discussion of trust-region methods for constrained optimization.

Algorithm 15.1 (General Filter Method).

```

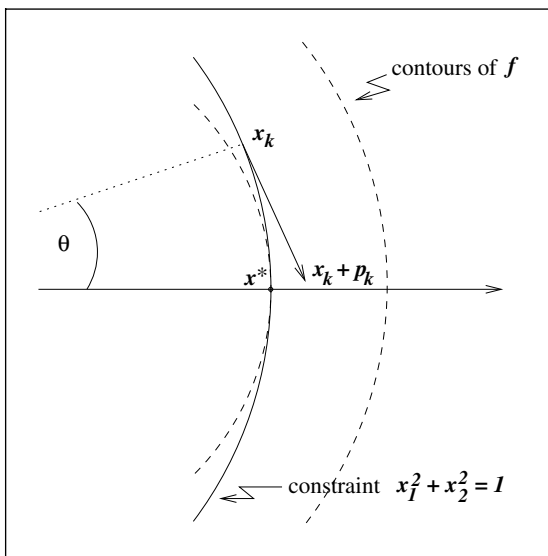
Choose a starting point  $x_0$  and an initial trust-region radius  $\Delta_0$ ;
Set  $k \leftarrow 0$ ;
repeat until a convergence test is satisfied
    if the step-generation subproblem is infeasible
        Compute  $x_{k+1}$  using the feasibility restoration phase;
    else
        Compute a trial iterate  $x^+ = x_k + p_k$ ;
        if  $(f^+, h^+)$  is acceptable to the filter
            Set  $x_{k+1} = x^+$  and add  $(f_{k+1}, h_{k+1})$  to the filter;
            Choose  $\Delta_{k+1}$  such that  $\Delta_{k+1} \geq \Delta_k$ ;
            Remove all pairs from the filter that are dominated
                by  $(f_{k+1}, h_{k+1})$ ;
        else
            Reject the step, set  $x_{k+1} = x_k$ ;
            Choose  $\Delta_{k+1} < \Delta_k$ ;
        end if
    end if
     $k \leftarrow k + 1$ ;
end repeat

```

Other enhancements of this simple filter framework are used in practice; they depend on the choice of algorithm and will be discussed in subsequent chapters.

15.5 THE MARATOS EFFECT

Some algorithms based on merit functions or filters may fail to converge rapidly because they reject steps that make good progress toward a solution. This undesirable phenomenon is often called the *Maratos effect*, because it was first observed by Maratos [199]. It is illustrated by the following example, in which steps p_k , which would yield quadratic convergence if accepted, cause an increase both in the objective function value and the constraint violation.

**Figure 15.8**

Maratos Effect: Example 15.4. Note that the constraint is no longer satisfied after the step from x_k to $x_k + p_k$, and the objective value has increased.

□ **EXAMPLE 15.4** (POWELL [255])

Consider the problem

$$\min f(x_1, x_2) = 2(x_1^2 + x_2^2 - 1) - x_1, \quad \text{subject to} \quad x_1^2 + x_2^2 - 1 = 0. \quad (15.34)$$

One can verify (see Figure 15.8) that the optimal solution is $x^* = (1, 0)^T$, that the corresponding Lagrange multiplier is $\lambda^* = \frac{3}{2}$, and that $\nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) = I$.

Let us consider an iterate x_k of the form $x_k = (\cos \theta, \sin \theta)^T$, which is feasible for any value of θ . Suppose that our algorithm computes the following step:

$$p_k = \begin{pmatrix} \sin^2 \theta \\ -\sin \theta \cos \theta \end{pmatrix}, \quad (15.35)$$

which yields a trial point

$$x_k + p_k = \begin{pmatrix} \cos \theta + \sin^2 \theta \\ \sin \theta (1 - \cos \theta) \end{pmatrix}.$$

By using elementary trigonometric identities, we have that

$$\|x_k + p_k - x^*\|_2 = 2 \sin^2(\theta/2), \quad \|x_k - x^*\|_2 = 2 |\sin(\theta/2)|,$$

and therefore

$$\frac{\|x_k + p_k - x^*\|_2}{\|x_k - x^*\|_2^2} = \frac{1}{2}.$$

Hence, this step approaches the solution at a rate consistent with Q-quadratic convergence. However, we have that

$$\begin{aligned} f(x_k + p_k) &= \sin^2 \theta - \cos \theta > -\cos \theta = f(x_k), \\ c(x_k + p_k) &= \sin^2 \theta > c(x_k) = 0, \end{aligned}$$

so that, as can be seen in Figure 15.8, both the objective function value and the constraint violation increase over this step. This behavior occurs for any nonzero value of θ , even if the initial point is arbitrarily close to the solution. □

On the example above, any algorithm that requires reduction of a merit function of the form

$$\phi(x; \mu) = f(x) + \mu h(c(x)),$$

where $h(\cdot)$ is a nonnegative function satisfying $h(0) = 0$, will reject the good step (15.35). (Examples of such merit functions include the ϕ_1 and ϕ_2 penalty functions.) The step (15.35) will also be rejected by the filter mechanism described above because the pair $(f(x_k + p_k), h(x_k + p_k))$ is dominated by (f_k, h_k) . Therefore, all these approaches will suffer from the Maratos effect.

If no remedial measures are taken, the Maratos effect can slow optimization methods by interfering with good steps away from the solution and by preventing superlinear convergence. Strategies for avoiding the Maratos effect include the following.

1. We can use a merit function that does not suffer from the Maratos effect. An example is Fletcher's augmented Lagrangian function (15.26).
2. We can use a second-order correction in which we add to p_k a step \hat{p}_k , which is computed at $c(x_k + p_k)$ and which decreases the constraint violation.
3. We can allow the merit function ϕ to increase on certain iterations; that is, we can use a nonmonotone strategy.

We discuss the last two approaches in the next section.

15.6 SECOND-ORDER CORRECTION AND NONMONOTONE TECHNIQUES

By adding a correction term that decreases the constraint violation, various algorithms are able to overcome the difficulties associated with the Maratos effect. We describe this technique with respect to the equality-constrained problem, in which the constraint is $c(x) = 0$, where $c : \mathbb{R}^n \rightarrow \mathbb{R}^{|\mathcal{E}|}$.

Given a step p_k , the second-order correction step \hat{p}_k is defined to be

$$\hat{p}_k = -A_k^T (A_k A_k^T)^{-1} c(x_k + p_k), \quad (15.36)$$

where $A_k = A(x_k)$ is the Jacobian of c at x_k . Note that \hat{p}_k has the property that it satisfies a linearization of the constraint c at the point $x_k + p_k$, that is,

$$A_k \hat{p}_k + c(x_k + p_k) = 0.$$

In fact, \hat{p}_k is the minimum-norm solution of this equation. (A different interpretation of the second-order correction is given in Section 18.3.)

The effect of the correction step \hat{p}_k is to decrease the quantity $\|c(x)\|$ to the order of $\|x_k - x^*\|^3$, provided the primary step p_k satisfies $A_k p_k + c(x_k) = 0$. This estimate indicates that the step from x_k to $x_k + p_k + \hat{p}_k$ will decrease the merit function, at least near the solution. The cost of this enhancement includes the additional evaluation of the constraint function c at $x_k + p_k$ and the linear algebra required to calculate the step \hat{p}_k from (15.36).

We now describe an algorithm that uses a merit function together with a line-search strategy and a second-order correction step. We assume that the search direction p_k and the penalty parameter μ_k are computed so that p_k is a descent direction for the merit function, that is, $D(\phi(x_k; \mu); p_k) < 0$. In Chapters 18 and 19, we discuss how to accomplish these goals. The key feature of the algorithm is that, if the full step $\alpha_k = 1$ does not produce satisfactory descent in the merit function, we try the second-order correction step *before* backtracking along the original direction p_k .

Algorithm 15.2 (Generic Algorithm with Second-Order Correction).

Choose parameters $\eta \in (0, 0.5)$ and τ_1, τ_2 with $0 < \tau_1 < \tau_2 < 1$;

Choose initial point x_0 ; set $k \leftarrow 0$;

repeat until a convergence test is satisfied:

 Compute a search direction p_k ;

 Set $\alpha_k \leftarrow 1$, **newpoint** \leftarrow **false**;

while **newpoint** = **false**

if $\phi(x_k + \alpha_k p_k; \mu) \leq \phi(x_k; \mu) + \eta \alpha_k D(\phi(x_k; \mu); p_k)$

 Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$;

 Set **newpoint** \leftarrow **true**;

the added complexity is worthwhile because the approach has good practical performance. A potential advantage of the watchdog technique over the second-order correction strategy is that it may require fewer evaluations of the constraint functions. In the best case, most of the steps will be full steps, and there will rarely be a need to return to an earlier point.

NOTES AND REFERENCES

Techniques for eliminating linear constraints are described, for example, in Fletcher [101] and Gill, Murray, and Wright [131]. For a thorough discussion of merit functions see Boggs and Tolle [33] and Conn, Gould, and Toint [74]. Some of the earliest references on nonmonotone methods include Grippo, Lampariello and Lucidi [158], and Chamberlain et al [57]; see [74] for a review of nonmonotone techniques and an extensive list of references. The concept of a filter was introduced by Fletcher and Leyffer [105]; our discussion of filters is based on that paper. Second-order correction steps are motivated and discussed in Fletcher [101].



EXERCISES

15.1 In Example 15.1, consider these three choices of the working set: $\mathcal{W} = \{3\}$, $\mathcal{W} = \{1, 2\}$, $\mathcal{W} = \{2, 3\}$. Show that none of these working sets are the optimal active set for (15.5).

15.2 For the problem in Example 15.3, perform simple elimination of the variables x_2 and x_5 to obtain an unconstrained problem in the remaining variables x_1 , x_3 , x_4 , and x_6 . Similarly to (15.12), express the eliminated variables explicitly in terms of the retained variables.

15.3 Do the following problems have solutions? Explain.

$$\min x_1 + x_2 \quad \text{subject to } x_1^2 + x_2^2 = 2, \quad 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1;$$


$$\min x_1 + x_2 \quad \text{subject to } x_1^2 + x_2^2 \leq 1, \quad x_1 + x_2 = 3;$$

$$\min x_1 x_2 \quad \text{subject to } x_1 + x_2 = 2.$$

15.4 Show that if in Example 15.2 we eliminate x in terms of y , then the correct solution of the problem is obtained by performing unconstrained minimization.

15.5 Show that the basis matrices (15.15) are linearly independent.

15.6 Show that the particular solution $Q_1 R^{-T} \Pi^T b$ of $Ax = b$ is identical to $A^T(AA^T)^{-1}b$.


 **15.7** In this exercise we compute basis matrices that attempt to compromise between the orthonormal basis (15.22) and simple elimination (15.15). We assume that the basis matrix is given by the first m columns of A , so that $P = I$ in (15.7), and define

$$Y = \begin{bmatrix} I \\ (B^{-1}N)^T \end{bmatrix}, \quad Z = \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}. \quad (15.37)$$

- (a) Show that the columns of Y and Z are no longer of norm 1 and that the relations $AZ = 0$ and $Y^T Z = 0$ hold. Therefore, the columns of Y and Z form a linearly independent set, showing that (15.37) is a valid choice of the basis matrices.
- (b) Show that the particular solution $Y(AY)^{-1}b$ defined by this choice of Y is, as in the orthogonal factorization approach, the minimum-norm solution of $Ax = b$. More specifically, show that

$$Y(AY)^{-1} = A^T(AA^T)^{-1}.$$

It follows that the matrix $Y(AY)^{-1}$ is independent of the choice of basis matrix B in (15.7), and its conditioning is determined by that of A alone. (Note, however, that the matrix Z still depends explicitly on B , so a careful choice of B is needed to ensure well conditioning in this part of the computation.)

 **15.8** Verify that by adding the inequality constraint $3x_1 + 2x_3 \geq 1$ to the problem (15.11), the elimination (15.12) transforms the problem into one of minimizing the function (15.13) subject to the inequality constraint (15.23).

CHAPTER *16*

Quadratic Programming

An optimization problem with a quadratic objective function and linear constraints is called a quadratic program. Problems of this type are important in their own right, and they also arise as subproblems in methods for general constrained optimization, such as sequential quadratic programming (Chapter 18), augmented Lagrangian methods (Chapter 17), and interior-point methods (Chapter 19).

The general quadratic program (QP) can be stated as

$$\min_x \quad q(x) = \frac{1}{2}x^T Gx + x^T c \quad (16.1a)$$

$$\text{subject to} \quad a_i^T x = b_i, \quad i \in \mathcal{E}, \quad (16.1b)$$

$$a_i^T x \geq b_i, \quad i \in \mathcal{I}, \quad (16.1c)$$

where G is a symmetric $n \times n$ matrix, \mathcal{E} and \mathcal{I} are finite sets of indices, and c , x , and $\{a_i\}$, $i \in \mathcal{E} \cup \mathcal{I}$, are vectors in \mathbb{R}^n . Quadratic programs can always be solved (or shown to be infeasible) in a finite amount of computation, but the effort required to find a solution depends strongly on the characteristics of the objective function and the number of inequality constraints. If the Hessian matrix G is positive semidefinite, we say that (16.1) is a *convex QP*, and in this case the problem is often similar in difficulty to a linear program. (*Strictly convex QPs* are those in which G is positive definite.) *Nonconvex QPs*, in which G is an indefinite matrix, can be more challenging because they can have several stationary points and local minima.

In this chapter we focus primarily on convex quadratic programs. We start by considering an interesting application of quadratic programming.

□ **EXAMPLE 16.1** (PORTFOLIO OPTIMIZATION)

Every investor knows that there is a tradeoff between risk and return: To increase the expected return on investment, an investor must be willing to tolerate greater risks. Portfolio theory studies how to model this tradeoff given a collection of n possible investments with returns r_i , $i = 1, 2, \dots, n$. The returns r_i are usually not known in advance and are often assumed to be random variables that follow a normal distribution. We can characterize these variables by their expected value $\mu_i = E[r_i]$ and their variance $\sigma_i^2 = E[(r_i - \mu_i)^2]$. The variance measures the fluctuations of the variable r_i about its mean, so that larger values of σ_i indicate riskier investments. The returns are not in general independent, and we can define correlations between pairs of returns as follows:

$$\rho_{ij} = \frac{E[(r_i - \mu_i)(r_j - \mu_j)]}{\sigma_i \sigma_j}, \quad \text{for } i, j = 1, 2, \dots, n.$$

The correlation measures the tendency of the return on investments i and j to move in the same direction. Two investments whose returns tend to rise and fall together have a positive correlation; the nearer ρ_{ij} is to 1, the more closely the two investments track each other. Investments whose returns tend to move in opposite directions have a negative correlation.

An investor constructs a portfolio by putting a fraction x_i of the available funds into investment i , for $i = 1, 2, \dots, n$. Assuming that all available funds are invested and that short-selling is not allowed, the constraints are $\sum_{i=1}^n x_i = 1$ and $x \geq 0$. The return on the

portfolio is given by

$$R = \sum_{i=1}^n x_i r_i. \quad (16.2)$$

To measure the desirability of the portfolio, we need to obtain measures of its expected return and variance. The expected return is simply

$$E[R] = E\left[\sum_{i=1}^n x_i r_i\right] = \sum_{i=1}^n x_i E[r_i] = x^T \mu,$$

while the variance is given by

$$\text{Var}[R] = E[(R - E[R])^2] = \sum_{i=1}^n \sum_{j=1}^n x_i x_j \sigma_i \sigma_j \rho_{ij} = x^T G x,$$

where the $n \times n$ symmetric positive semidefinite matrix G defined by

$$G_{ij} = \rho_{ij} \sigma_i \sigma_j$$

is called the *covariance matrix*.

Ideally, we would like to find a portfolio for which the expected return $x^T \mu$ is large while the variance $x^T G x$ is small. In the model proposed by Markowitz [201], we combine these two aims into a single objective function with the aid of a “risk tolerance parameter” denoted by κ , and we solve the following problem to find the optimal portfolio:

$$\max x^T \mu - \kappa x^T G x, \quad \text{subject to} \quad \sum_{i=1}^n x_i = 1, \quad x \geq 0.$$

The value chosen for the nonnegative parameter κ depends on the preferences of the individual investor. Conservative investors, who place more emphasis on minimizing risk in their portfolio, would choose a large value of κ to increase the weight of the variance measure in the objective function. More daring investors, who are prepared to take on more risk in the hope of a higher expected return, would choose a smaller value of κ .

The difficulty in applying this portfolio optimization technique to real-life investing lies in defining the expected returns, variances, and correlations for the investments in question. Financial professionals often combine historical data with their own insights and expectations to produce values of these quantities.



16.1 EQUALITY-CONSTRAINED QUADRATIC PROGRAMS

We begin our discussion of algorithms for quadratic programming by considering the case in which only equality constraints are present. Techniques for this special case are applicable also to problems with inequality constraints since, as we see later in this chapter, some algorithms for general QP require the solution of an equality-constrained QP at each iteration.

PROPERTIES OF EQUALITY-CONSTRAINED QPs

For simplicity, we write the equality constraints in matrix form and state the equality-constrained QP as follows:

$$\min_x q(x) \stackrel{\text{def}}{=} \frac{1}{2}x^T Gx + x^T c \quad (16.3a)$$

$$\text{subject to } Ax = b, \quad (16.3b)$$

where A is the $m \times n$ Jacobian of constraints (with $m \leq n$) whose rows are a_i^T , $i \in \mathcal{E}$ and b is the vector in \mathbb{R}^m whose components are b_i , $i \in \mathcal{E}$. For the present, we assume that A has full row rank (rank m) so that the constraints (16.3b) are consistent. (In Section 16.8 we discuss the case in which A is rank deficient.)

The first-order necessary conditions for x^* to be a solution of (16.3) state that there is a vector λ^* such that the following system of equations is satisfied:

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}. \quad (16.4)$$

These conditions are a consequence of the general result for first-order optimality conditions, Theorem 12.1. As in Chapter 12, we call λ^* the vector of Lagrange multipliers. The system (16.4) can be rewritten in a form that is useful for computation by expressing x^* as $x^* = x + p$, where x is some estimate of the solution and p is the desired step. By introducing this notation and rearranging the equations, we obtain

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ h \end{bmatrix}, \quad (16.5)$$

where

$$h = Ax - b, \quad g = c + Gx, \quad p = x^* - x. \quad (16.6)$$

The matrix in (16.5) is called the Karush–Kuhn–Tucker (KKT) matrix, and the following result gives conditions under which it is nonsingular. As in Chapter 15, we use Z to

denote the $n \times (n - m)$ matrix whose columns are a basis for the null space of A . That is, Z has full rank and satisfies $AZ = 0$.

Lemma 16.1.

Let A have full row rank, and assume that the reduced-Hessian matrix $Z^T G Z$ is positive definite. Then the KKT matrix

$$K = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \quad (16.7)$$

is nonsingular, and hence there is a unique vector pair (x^, λ^*) satisfying (16.4).*

PROOF. Suppose there are vectors w and v such that

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = 0. \quad (16.8)$$

Since $Aw = 0$, we have from (16.8) that

$$0 = \begin{bmatrix} w \\ v \end{bmatrix}^T \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = w^T G w.$$

Since w lies in the null space of A , it can be written as $w = Zu$ for some vector $u \in \mathbb{R}^{n-m}$. Therefore, we have

$$0 = w^T G w = u^T Z^T G Z u,$$

which by positive definiteness of $Z^T G Z$ implies that $u = 0$. Therefore, $w = 0$, and by (16.8), $A^T v = 0$. Full row rank of A then implies that $v = 0$. We conclude that equation (16.8) is satisfied only if $w = 0$ and $v = 0$, so the matrix is nonsingular, as claimed. \square

\square **EXAMPLE 16.2**

Consider the quadratic programming problem

$$\begin{aligned} \min q(x) &= 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3, \\ \text{subject to} \quad &x_1 + x_3 = 3, \quad x_2 + x_3 = 0. \end{aligned} \quad (16.9)$$

We can write this problem in the form (16.3) by defining

$$G = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}, \quad c = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 0 \end{bmatrix}.$$

The solution x^* and optimal Lagrange multiplier vector λ^* are given by

$$x^* = (2, -1, 1)^T, \quad \lambda^* = (3, -2)^T.$$

In this example, the matrix G is positive definite, and the null-space basis matrix can be defined as in (15.15), giving

$$Z = (-1, -1, 1)^T. \quad (16.10)$$



We have seen that when the conditions of Lemma 16.1 are satisfied, there is a unique vector pair (x^*, λ^*) that satisfies the first-order necessary conditions for (16.3). In fact, the second-order sufficient conditions (see Theorem 12.6) are also satisfied at (x^*, λ^*) , so x^* is a strict local minimizer of (16.3). In fact, we can use a direct argument to show that x^* is a *global* solution of (16.3).

Theorem 16.2.

Let A have full row rank and assume that the reduced-Hessian matrix $Z^T G Z$ is positive definite. Then the vector x^ satisfying (16.4) is the unique global solution of (16.3).*

PROOF. Let x be any other feasible point (satisfying $Ax = b$), and as before, let p denote the difference $x^* - x$. Since $Ax^* = Ax = b$, we have that $Ap = 0$. By substituting into the objective function (16.3a), we obtain

$$\begin{aligned} q(x) &= \frac{1}{2}(x^* - p)^T G(x^* - p) + c^T(x^* - p) \\ &= \frac{1}{2}p^T Gp - p^T Gx^* - c^T p + q(x^*). \end{aligned} \quad (16.11)$$

From (16.4) we have that $Gx^* = -c + A^T \lambda^*$, so from $Ap = 0$ we have that

$$p^T Gx^* = p^T(-c + A^T \lambda^*) = -p^T c.$$

By substituting this relation into (16.11), we obtain

$$q(x) = \frac{1}{2}p^T Gp + q(x^*).$$

Since p lies in the null space of A , we can write $p = Zu$ for some vector $u \in \mathbb{R}^{n-m}$, so that

$$q(x) = \frac{1}{2}u^T Z^T G Z u + q(x^*).$$

By positive definiteness of $Z^T G Z$, we conclude that $q(x) > q(x^*)$ except when $u = 0$, that is, when $x = x^*$. Therefore, x^* is the unique global solution of (16.3). \square

When the reduced Hessian matrix $Z^T G Z$ is positive semidefinite with zero eigenvalues, the vector x^* satisfying (16.4) is a local minimizer but not a strict local minimizer. If the reduced Hessian has negative eigenvalues, then x^* is only a stationary point, not a local minimizer.

16.2 DIRECT SOLUTION OF THE KKT SYSTEM

In this section we discuss efficient methods for solving the KKT system (16.5). The first important observation is that if $m \geq 1$, the KKT matrix is always indefinite. We define the inertia of a symmetric matrix K to be the scalar triple that indicates the numbers n_+ , n_- , and n_0 of positive, negative, and zero eigenvalues, respectively, that is,

$$\text{inertia}(K) = (n_+, n_-, n_0).$$

The following result characterizes the inertia of the KKT matrix.

Theorem 16.3.

Let K be defined by (16.7), and suppose that A has rank m . Then

$$\text{inertia}(K) = \text{inertia}(Z^T G Z) + (m, m, 0).$$

Therefore, if $Z^T G Z$ is positive definite, $\text{inertia}(K) = (n, m, 0)$.

The proof of this result is given in [111], for example. Note that the assumptions of this theorem are satisfied by Example 16.2. Hence, if we construct the 5×5 matrix K using the data of this example, we obtain $\text{inertia}(K) = (3, 2, 0)$.

Knowing that the KKT system is indefinite, we now describe the main direct techniques used to solve (16.5).

FACTORING THE FULL KKT SYSTEM

One option for solving (16.5) is to perform a triangular factorization on the full KKT matrix and then perform backward and forward substitution with the triangular factors. Because of indefiniteness, we cannot use the Cholesky factorization. We could use Gaussian

elimination with partial pivoting (or a sparse variant thereof) to obtain the L and U factors, but this approach has the disadvantage that it ignores the symmetry.

The most effective strategy in this case is to use a *symmetric indefinite factorization*, which we have discussed in Chapter 3 and the Appendix. For a general symmetric matrix K , this factorization has the form

$$P^T K P = L B L^T, \quad (16.12)$$

where P is a permutation matrix, L is unit lower triangular, and B is block-diagonal with either 1×1 or 2×2 blocks. The symmetric permutations defined by the matrix P are introduced for numerical stability of the computation and, in the case of large sparse K , for maintaining sparsity. The computational cost of the symmetric indefinite factorization (16.12) is typically about half the cost of sparse Gaussian elimination.

To solve (16.5), we first compute the factorization (16.12) of the coefficient matrix. We then perform the following sequence of operations to arrive at the solution:

$$\begin{aligned} &\text{solve } Lz = P^T \begin{bmatrix} g \\ h \end{bmatrix} && \text{to obtain } z; \\ &\text{solve } B\hat{z} = z && \text{to obtain } \hat{z}; \\ &\text{solve } L^T \bar{z} = \hat{z} && \text{to obtain } \bar{z}; \\ &\text{set } \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = P\bar{z}. \end{aligned}$$

Since multiplications with the permutation matrices P and P^T can be performed by simply rearranging vector components, they are inexpensive. Solution of the system $B\hat{z} = z$ entails solving a number of small 1×1 and 2×2 systems, so the number of operations is a small multiple of the system dimension $(m + n)$, again inexpensive. Triangular substitutions with L and L^T are more costly. Their precise cost depends on the amount of sparsity, but is usually significantly less than the cost of performing the factorization (16.12).

This approach of factoring the full $(n + m) \times (n + m)$ KKT matrix (16.7) is quite effective on many problems. It may be expensive, however, when the heuristics for choosing the permutation matrix P are not able to maintain sparsity in the L factor, so that L becomes much more dense than the original coefficient matrix.

SCHUR-COMPLEMENT METHOD

Assuming that G is positive definite, we can multiply the first equation in (16.5) by AG^{-1} and then subtract the second equation to obtain a linear system in the vector λ^* alone:

$$(AG^{-1}A^T)\lambda^* = (AG^{-1}g - h). \quad (16.13)$$

We solve this symmetric positive definite system for λ^* and then recover p from the first equation in (16.5) by solving

$$Gp = A^T \lambda^* - g. \quad (16.14)$$

This approach requires us to perform operations with G^{-1} , as well as to compute the factorization of the $m \times m$ matrix $AG^{-1}A^T$. Therefore, it is most useful when:

- G is well conditioned and easy to invert (for instance, when G is diagonal or block-diagonal); or
- G^{-1} is known explicitly through a quasi-Newton updating formula; or
- the number of equality constraints m is small, so that the number of backsolves needed to form the matrix $AG^{-1}A^T$ is not too large.

The name “Schur-Complement method” derives from the fact that, by applying block Gaussian elimination to (16.7) using G as the pivot, we obtain the block upper triangular system

$$\begin{bmatrix} G & A^T \\ 0 & -AG^{-1}A^T \end{bmatrix}. \quad (16.15)$$

In linear algebra terminology, the matrix $AG^{-1}A^T$ is the Schur complement of G in the matrix K of (16.7). By applying this block elimination technique to the system (16.5), and performing a block backsolve, we obtain (16.13), (16.14).

We can use an approach like the Schur-complement method to derive an explicit inverse formula for the KKT matrix in (16.5). This formula is

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix}^{-1} = \begin{bmatrix} C & E \\ E^T & F \end{bmatrix}, \quad (16.16)$$

with

$$\begin{aligned} C &= G^{-1} - G^{-1}A^T(AG^{-1}A^T)^{-1}AG^{-1}, \\ E &= G^{-1}A^T(AG^{-1}A^T)^{-1}, \\ F &= -(AG^{-1}A^T)^{-1}. \end{aligned}$$

The solution of (16.5) can be obtained by multiplying its right-hand side by this inverse matrix. If we take advantage of common expressions, and group the terms appropriately, we recover the approach (16.13), (16.14).

NULL-SPACE METHOD

The null-space method does not require nonsingularity of G and therefore has wider applicability than the Schur-complement method. It assumes only that the conditions of Lemma 16.1 hold, namely, that A has full row rank and that $Z^T G Z$ is positive definite. However, it requires knowledge of the null-space basis matrix Z . Like the Schur-complement method, it exploits the block structure in the KKT system to decouple (16.5) into two smaller systems.

Suppose that we partition the vector p in (16.5) into two components, as follows:

$$p = Y p_y + Z p_z, \quad (16.17)$$

where Z is the $n \times (n - m)$ null-space matrix, Y is any $n \times m$ matrix such that $[Y \mid Z]$ is nonsingular, p_y is an m -vector, and p_z is an $(n - m)$ -vector. The matrices Y and Z were discussed in Section 15.3, where Figure 15.4 shows that $Y x_y$ is a particular solution of $Ax = b$, while $Z x_z$ is a displacement along these constraints.

By substituting p into the second equation of (16.5) and recalling that $AZ = 0$, we obtain

$$(AY)p_y = -h. \quad (16.18)$$

Since A has rank m and $[Y \mid Z]$ is $n \times n$ nonsingular, the product $A[Y \mid Z] = [AY \mid 0]$ has rank m . Therefore, AY is a nonsingular $m \times m$ matrix, and p_y is well determined by the equations (16.18). Meanwhile, we can substitute (16.17) into the first equation of (16.5) to obtain

$$-GYp_y - GZp_z + A^T \lambda^* = g$$

and multiply by Z^T to obtain

$$(Z^T G Z)p_z = -Z^T G Y p_y - Z^T g. \quad (16.19)$$

This system can be solved by performing a Cholesky factorization of the reduced-Hessian matrix $Z^T G Z$ to determine p_z . We therefore can compute the total step $p = Y p_y + Z p_z$. To obtain the Lagrange multiplier, we multiply the first block row in (16.5) by Y^T to obtain the linear system

$$(AY)^T \lambda^* = Y^T (g + Gp), \quad (16.20)$$

which can be solved for λ^* .

□ EXAMPLE 16.3

Consider the problem (16.9) given in Example 16.2. We can choose

$$Y = \begin{bmatrix} 2/3 & -1/3 \\ -1/3 & 2/3 \\ 1/3 & 1/3 \end{bmatrix}$$

and set Z as in (16.10). Note that $AY = I$.

Suppose we have $x = (0, 0, 0)^T$ in (16.6). Then

$$h = Ax - b = -b, \quad g = c + Gx = c = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}.$$

Simple calculation shows that

$$p_Y = \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \quad p_Z = \begin{bmatrix} 0 \end{bmatrix},$$

so that

$$p = x^* - x = Yp_Y + Zp_Z = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}.$$

After recovering λ^* from (16.20), we conclude that

$$x^* = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}, \quad \lambda^* = \begin{bmatrix} 3 \\ -2 \end{bmatrix}.$$

□

The null-space approach can be very effective when the number of degrees of freedom $n - m$ is small. Its main limitation lies in the need for the null-space matrix Z which, as we have seen in Chapter 15, can be expensive to compute in some large problems. The matrix Z is not uniquely defined and, if it is poorly chosen, the reduced system (16.19) may become ill conditioned. If we choose Z to have orthonormal columns, as is normally done in software for small and medium-sized problems, then the conditioning of $Z^T G Z$ is at least as good as that of G itself. When A is large and sparse, however, an orthonormal Z is expensive to

compute, so for practical reasons we are often forced to use one of the less reliable choices of Z described in Chapter 15.

It is difficult to give hard and fast rules about the relative effectiveness of null-space and Schur-complement methods, because factors such as fill-in during computation of Z vary significantly even among problems of the same dimension. In general, we can recommend the Schur-complement method if G is positive definite and $AG^{-1}A^T$ can be computed relatively cheaply (because G is easy to invert or because m is small relative to n). Otherwise, the null-space method is often preferable, in particular when it is much more expensive to compute factors of G than to compute the null-space matrix Z and the factors of $Z^T GZ$.

16.3 ITERATIVE SOLUTION OF THE KKT SYSTEM

An alternative to the direct factorization techniques discussed in the previous section is to use an iterative method to solve the KKT system (16.5). Iterative methods are suitable for solving very large systems and often lend themselves well to parallelization. The conjugate gradient (CG) method is not recommended for solving the full system (16.5) as written, because it can be unstable on systems that are not positive definite. Better options are Krylov methods for general linear or symmetric indefinite systems. Candidates include the GMRES, QMR, and LSQR methods; see the Notes and References at the end of the chapter. Other iterative methods can be derived from the null-space approach by applying the conjugate gradient method to the reduced system (16.19). Methods of this type are key to the algorithms of Chapters 18 and 19, and are discussed in the remainder of this section. We assume throughout that $Z^T GZ$ is positive definite.

CG APPLIED TO THE REDUCED SYSTEM

We begin our discussion of iterative null-space methods by deriving the underlying equations in the notation of the equality-constrained QP (16.3). Expressing the solution of the quadratic program (16.3) as

$$x^* = Yx_Y + Zx_Z, \quad (16.21)$$

for some vectors $x_Z \in \mathbb{R}^{n-m}$, $x_Y \in \mathbb{R}^m$, the constraints $Ax = b$ yield

$$AYx_Y = b, \quad (16.22)$$

which determines the vector x_Y . In Chapter 15, various practical choices of Y are described, some of which allow (16.22) to be solved economically. Substituting (16.21) into (16.3), we see that x_Z solves the unconstrained reduced problem

$$\min_{x_Z} \frac{1}{2} x_Z^T Z^T GZ x_Z + x_Z^T c_Z,$$

We suppose now that the preconditioning has the general form of (16.27) and (16.28d). When H is nonsingular, we can compute g^+ as follows:

$$g^+ = Pr^+, \quad \text{where} \quad P = H^{-1} (I - A^T (AH^{-1}A^T)^{-1} AH^{-1}). \quad (16.33)$$

Otherwise, when $z^T H z \neq 0$ for all nonzero z with $Az = 0$, we can find g^+ as the solution of the system

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} g^+ \\ v^+ \end{bmatrix} = \begin{bmatrix} r^+ \\ 0 \end{bmatrix}. \quad (16.34)$$

While (16.33) is unappealing when H^{-1} does not have a simple form, (16.34) is a useful generalization of (16.32). A “perfect” preconditioner is obtained by taking $H = G$, but other choices for H are also possible, provided that $Z^T H Z$ is positive definite. The matrix in (16.34) is often called a *constraint preconditioner*.

None of these procedures for computing the projection makes use of a null-space basis Z ; only the factorization of matrices involving A is required. Significantly, all these forms allow us to compute an initial point satisfying $Ax = b$. The operator $g^+ = P_r r^+$ relies on a factorization of AA^T from which we can compute $x = A^T (AA^T)^{-1} b$, while factorizations of the system matrices in (16.32) and (16.34) allow us to find a suitable x by solving

$$\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}.$$

Therefore we can compute an initial point for Algorithm 16.2 at the cost of one backsolve, using the factorization of the system needed to perform the projection operators.

We point out that these approaches for computing g^+ can give rise to significant round-off errors, so the use of iterative refinement is recommended to improve accuracy.

16.4 INEQUALITY-CONSTRAINED PROBLEMS

In the remainder of the chapter we discuss several classes of algorithms for solving convex quadratic programs that contain both inequality and equality constraints. *Active-set methods* have been widely used since the 1970s and are effective for small- and medium-sized problems. They allow for efficient detection of unboundedness and infeasibility and typically return an accurate estimate of the optimal active set. *Interior-point methods* are more recent, having become popular in the 1990s. They are well suited for large problems but may not be the most effective when a series of related QPs must be solved. We also study a special

type of active-set methods called a *gradient projection method*, which is most effective when the only constraints in the problem are bounds on the variables.

OPTIMALITY CONDITIONS FOR INEQUALITY-CONSTRAINED PROBLEMS

We begin our discussion with a brief review of the optimality conditions for inequality-constrained quadratic programming, then discuss some of the less obvious properties of the solutions.

Theorem 12.1 can be applied to (16.1) by noting that the Lagrangian for this problem is

$$\mathcal{L}(x, \lambda) = \frac{1}{2}x^T Gx + x^T c - \sum_{i \in \mathcal{I} \cup \mathcal{E}} \lambda_i (a_i^T x - b_i). \quad (16.35)$$

As in Definition 12.1, the active set $\mathcal{A}(x^*)$ consists of the indices of the constraints for which equality holds at x^* :

$$\mathcal{A}(x^*) = \{i \in \mathcal{E} \cup \mathcal{I} \mid a_i^T x^* = b_i\}. \quad (16.36)$$

By specializing the KKT conditions (12.34) to this problem, we find that any solution x^* of (16.1) satisfies the following first-order conditions, for some Lagrange multipliers λ_i^* , $i \in \mathcal{A}(x^*)$:

$$Gx^* + c - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* a_i = 0, \quad (16.37a)$$

$$a_i^T x^* = b_i, \quad \text{for all } i \in \mathcal{A}(x^*), \quad (16.37b)$$

$$a_i^T x^* \geq b_i, \quad \text{for all } i \in \mathcal{I} \setminus \mathcal{A}(x^*), \quad (16.37c)$$

$$\lambda_i^* \geq 0, \quad \text{for all } i \in \mathcal{I} \cap \mathcal{A}(x^*). \quad (16.37d)$$

A technical point: In Theorem 12.1 we assumed that the linear independence constraint qualification (LICQ) was satisfied. As mentioned in Section 12.6, this theorem still holds if we replace LICQ by other constraint qualifications, such as linearity of the constraints, which is certainly satisfied for quadratic programming. Hence, in the optimality conditions for quadratic programming given above, we need not assume that the active constraints are linearly independent at the solution.

For convex QP, when G is positive semidefinite, the conditions (16.37) are in fact sufficient for x^* to be a global solution, as we now prove.

Theorem 16.4.

If x^ satisfies the conditions (16.37) for some λ_i^* , $i \in \mathcal{A}(x^*)$, and G is positive semidefinite, then x^* is a global solution of (16.1).*

PROOF. If x is any other feasible point for (16.1), we have that $a_i^T x = b_i$ for all $i \in \mathcal{E}$ and $a_i^T x \geq b_i$ for all $i \in \mathcal{A}(x^*) \cap \mathcal{I}$. Hence, $a_i^T(x - x^*) = 0$ for all $i \in \mathcal{E}$ and $a_i^T(x - x^*) \geq 0$ for all $i \in \mathcal{A}(x^*) \cap \mathcal{I}$. Using these relationships, together with (16.37a) and (16.37d), we have that

$$(x - x^*)^T(Gx^* + c) = \sum_{i \in \mathcal{E}} \lambda_i^* a_i^T(x - x^*) + \sum_{i \in \mathcal{A}(x^*) \cap \mathcal{I}} \lambda_i^* a_i^T(x - x^*) \geq 0. \quad (16.38)$$

By elementary manipulation, we find that

$$\begin{aligned} q(x) &= q(x^*) + (x - x^*)^T(Gx^* + c) + \frac{1}{2}(x - x^*)^T G(x - x^*) \\ &\geq q(x^*) + \frac{1}{2}(x - x^*)^T G(x - x^*) \\ &\geq q(x^*), \end{aligned}$$

where the first inequality follows from (16.38) and the second inequality follows from positive semidefiniteness of G . We have shown that $q(x) \geq q(x^*)$ for any feasible x , so x^* is a global solution. \square

By a trivial modification of this proof, we see that x^* is actually the unique global solution when G is positive definite.

We can also apply the theory from Section 12.5 to derive second-order optimality conditions for (16.1). Second-order sufficient conditions for x^* to be a local minimizer are satisfied if $Z^T G Z$ is positive definite, where Z is defined to be a null-space basis matrix for the active constraint Jacobian matrix, which is the matrix whose rows are a_i^T for all $i \in \mathcal{A}(x^*)$. In this case, x^* is a strict local solution, according to Theorem 12.6.

When G is not positive definite, the general problem (16.1) may have more than one strict local solution. As mentioned above, such problems are called “nonconvex QPs” or “indefinite QPs,” and they cause some complications for algorithms. Examples of indefinite QPs are illustrated in Figure 16.1. On the left we have plotted the feasible region and the contours of a quadratic objective $q(x)$ in which G has one positive and one negative eigenvalue. We have indicated by $+$ or $-$ that the function tends toward plus or minus infinity in that direction. Note that x^{**} is a local maximizer, x^* a local minimizer, and the center of the box is a stationary point. The picture on the right in Figure 16.1, in which both eigenvalues of G are negative, shows a global maximizer at \tilde{x} and local minimizers at x_* and x_{**} .

DEGENERACY

A second property that causes difficulties for some algorithms is *degeneracy*. Confusingly, this term has been given a variety of meanings. It refers to situations in which

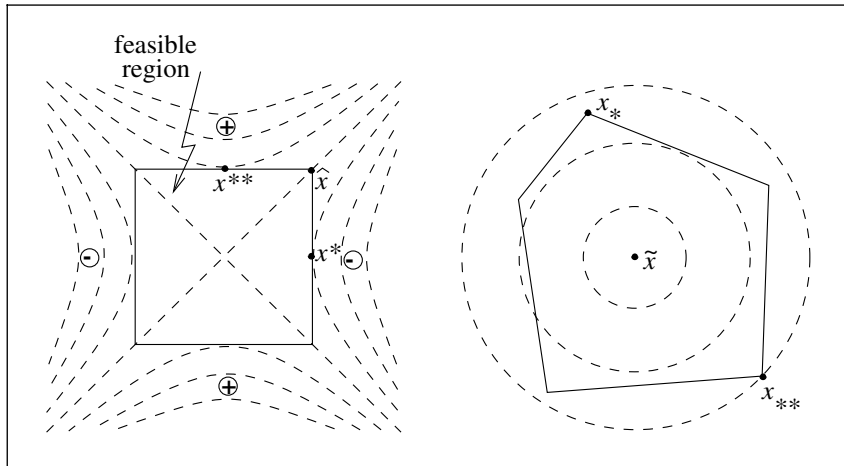


Figure 16.1 Nonconvex quadratic programs.

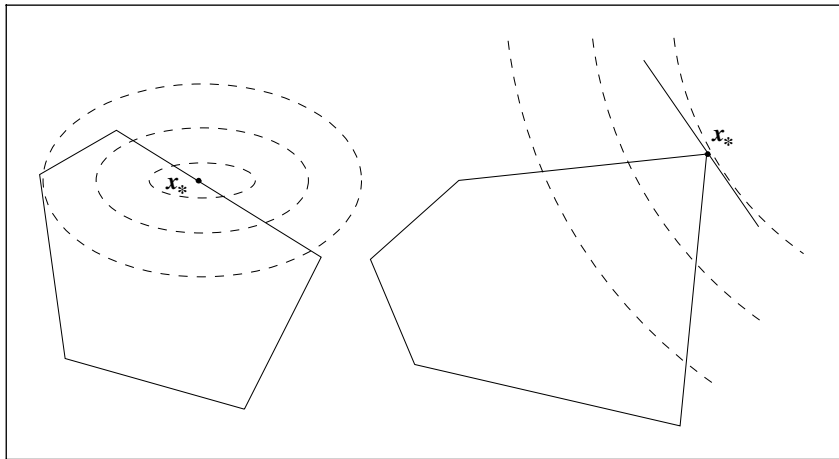


Figure 16.2 Degenerate solutions of quadratic programs.

- (a) the active constraint gradients a_i , $i \in \mathcal{A}(x^*)$, are linearly dependent at the solution x^* , and/or
- (b) the strict complementarity condition of Definition 12.5 fails to hold, that is, there is some index $i \in \mathcal{A}(x^*)$ such that all Lagrange multipliers satisfying (16.37) have $\lambda_i^* = 0$. (Such constraints are *weakly active* according to Definition 12.8.)

Two examples of degeneracy are shown in Figure 16.2. In the left-hand picture, there is a single active constraint at the solution x_* , which is also an unconstrained minimizer of the objective function. In the notation of (16.37a), we have that $Gx_* + c = 0$, so that

the lone Lagrange multiplier must be zero. In the right-hand picture, three constraints are active at the solution x_* . Since each of the three constraint gradients is a vector in \mathbb{R}^2 , they must be linearly dependent.

Lack of strict complementarity is also illustrated by the problem

$$\min x_1^2 + (x_2 + 1)^2 \quad \text{subject to } x \geq 0,$$

which has a solution at $x^* = 0$ at which both constraints are active. Strict complementarity does not hold at x^* because the Lagrange multiplier associated with the active constraint $x_1 \geq 0$ is zero.

Degeneracy can cause problems for algorithms for two main reasons. First, linear dependence of the active constraint gradients can cause numerical difficulties in the step computation because certain matrices that we need to factor become rank deficient. Second, when the problem contains weakly active constraints, it is difficult for the algorithm to determine whether these constraints are active at the solution. In the case of active-set methods and gradient projection methods (described below), this indecisiveness can cause the algorithm to zigzag as the iterates move on and off the weakly active constraints on successive iterations. Safeguards must be used to prevent such behavior.

16.5 ACTIVE-SET METHODS FOR CONVEX QPs

We now describe active-set methods for solving quadratic programs of the form (16.1) containing equality and inequality constraints. We consider only the convex case, in which the matrix G in (16.1a) is positive semidefinite. The case in which G is an indefinite matrix raises complications in the algorithms and is outside the scope of this book. We refer to Gould [147] for a discussion of nonconvex QPs.

If the contents of the *optimal* active set (16.36) were known in advance, we could find the solution x^* by applying one of the techniques for equality-constrained QP of Sections 16.2 and 16.3 to the problem

$$\min_x q(x) = \frac{1}{2}x^T Gx + x^T c \quad \text{subject to} \quad a_i^T x = b_i, \quad i \in \mathcal{A}(x^*).$$

Of course, we usually do not have prior knowledge of $\mathcal{A}(x^*)$ and, as we now see, determination of this set is the main challenge facing algorithms for inequality-constrained QP.

We have already encountered an active-set approach for linear programming in Chapter 13, namely, the simplex method. In essence, the simplex method starts by making a guess of the optimal active set, then repeatedly uses gradient and Lagrange multiplier information to drop one index from the current estimate of $\mathcal{A}(x^*)$ and add a new index, until optimality

is detected. Active-set methods for QP differ from the simplex method in that the iterates (and the solution x^*) are not necessarily vertices of the feasible region.

Active-set methods for QP come in three varieties: *primal*, *dual*, and *primal-dual*. We restrict our discussion to primal methods, which generate iterates that remain feasible with respect to the primal problem (16.1) while steadily decreasing the objective function $q(x)$.

Primal active-set methods find a step from one iterate to the next by solving a quadratic subproblem in which some of the inequality constraints (16.1c), and all the equality constraints (16.1b), are imposed as equalities. This subset is referred to as the *working set* and is denoted at the k th iterate x_k by \mathcal{W}_k . An important requirement we impose on \mathcal{W}_k is that the gradients a_i of the constraints in the working set be linearly independent, even when the full set of active constraints at that point has linearly dependent gradients.

Given an iterate x_k and the working set \mathcal{W}_k , we first check whether x_k minimizes the quadratic q in the subspace defined by the working set. If not, we compute a step p by solving an equality-constrained QP subproblem in which the constraints corresponding to the working set \mathcal{W}_k are regarded as equalities and all other constraints are temporarily disregarded. To express this subproblem in terms of the step p , we define

$$p = x - x_k, \quad g_k = Gx_k + c.$$

By substituting for x into the objective function (16.1a), we find that

$$q(x) = q(x_k + p) = \frac{1}{2}p^T Gp + g_k^T p + \rho_k,$$

where $\rho_k = \frac{1}{2}x_k^T Gx_k + c^T x_k$ is independent of p . Since we can drop ρ_k from the objective without changing the solution of the problem, we can write the QP subproblem to be solved at the k th iteration as follows:

$$\min_p \quad \frac{1}{2}p^T Gp + g_k^T p \tag{16.39a}$$

$$\text{subject to} \quad a_i^T p = 0, \quad i \in \mathcal{W}_k. \tag{16.39b}$$

We denote the solution of this subproblem by p_k . Note that for each $i \in \mathcal{W}_k$, the value of $a_i^T x$ does not change as we move along p_k , since we have $a_i^T(x_k + \alpha p_k) = a_i^T x_k = b_i$ for all α . Since the constraints in \mathcal{W}_k were satisfied at x_k , they are also satisfied at $x_k + \alpha p_k$, for any value of α . Since G is positive definite, the solution of (16.39) can be computed by any of the techniques described in Section 16.2.

Supposing for the moment that the optimal p_k from (16.39) is nonzero, we need to decide how far to move along this direction. If $x_k + p_k$ is feasible with respect to all the constraints, we set $x_{k+1} = x_k + p_k$. Otherwise, we set

$$x_{k+1} = x_k + \alpha_k p_k, \tag{16.40}$$

where the step-length parameter α_k is chosen to be the largest value in the range $[0, 1]$ for which all constraints are satisfied. We can derive an explicit definition of α_k by considering what happens to the constraints $i \notin \mathcal{W}_k$, since the constraints $i \in \mathcal{W}_k$ will certainly be satisfied regardless of the choice of α_k . If $a_i^T p_k \geq 0$ for some $i \notin \mathcal{W}_k$, then for all $\alpha_k \geq 0$ we have $a_i^T(x_k + \alpha_k p_k) \geq a_i^T x_k \geq b_i$. Hence, constraint i will be satisfied for all nonnegative choices of the step-length parameter. Whenever $a_i^T p_k < 0$ for some $i \notin \mathcal{W}_k$, however, we have that $a_i^T(x_k + \alpha_k p_k) \geq b_i$ only if

$$\alpha_k \leq \frac{b_i - a_i^T x_k}{a_i^T p_k}.$$

To maximize the decrease in q , we want α_k to be as large as possible in $[0, 1]$ subject to retaining feasibility, so we obtain the following definition:

$$\alpha_k \stackrel{\text{def}}{=} \min \left(1, \min_{i \notin \mathcal{W}_k, a_i^T p_k < 0} \frac{b_i - a_i^T x_k}{a_i^T p_k} \right). \quad (16.41)$$

We call the constraints i for which the minimum in (16.41) is achieved the *blocking constraints*. (If $\alpha_k = 1$ and no new constraints are active at $x_k + \alpha_k p_k$, then there are no blocking constraints on this iteration.) Note that it is quite possible for α_k to be zero, because we could have $a_i^T p_k < 0$ for some constraint i that is active at x_k but not a member of the current working set \mathcal{W}_k .

If $\alpha_k < 1$, that is, the step along p_k was blocked by some constraint not in \mathcal{W}_k , a new working set \mathcal{W}_{k+1} is constructed by adding one of the blocking constraints to \mathcal{W}_k .

We continue to iterate in this manner, adding constraints to the working set until we reach a point \hat{x} that minimizes the quadratic objective function over its current working set $\hat{\mathcal{W}}$. It is easy to recognize such a point because the subproblem (16.39) has solution $p = 0$. Since $p = 0$ satisfies the optimality conditions (16.5) for (16.39), we have that

$$\sum_{i \in \hat{\mathcal{W}}} a_i \hat{\lambda}_i = g = G\hat{x} + c, \quad (16.42)$$

for some Lagrange multipliers $\hat{\lambda}_i$, $i \in \hat{\mathcal{W}}$. It follows that \hat{x} and $\hat{\lambda}$ satisfy the first KKT condition (16.37a), if we define the multipliers corresponding to the inequality constraints that are not in the working set to be zero. Because of the control imposed on the step length, \hat{x} is also feasible with respect to all the constraints, so the second and third KKT conditions (16.37b) and (16.37c) are satisfied at this point.

We now examine the signs of the multipliers corresponding to the inequality constraints in the working set, that is, the indices $i \in \hat{\mathcal{W}} \cap \mathcal{I}$. If these multipliers are all nonnegative, the fourth KKT condition (16.37d) is also satisfied, so we conclude that \hat{x} is a KKT point for the original problem (16.1). In fact, since G is positive semidefinite, we have

from Theorem 16.4 that \hat{x} is a global solution of (16.1). (As noted after Theorem 16.4, \hat{x} is a strict local minimizer and the unique global solution if G is positive definite.)

If, on the other hand, one or more of the multipliers $\hat{\lambda}_j$, $j \in \hat{\mathcal{W}} \cap \mathcal{I}$, is negative, the condition (16.37d) is not satisfied and the objective function $q(\cdot)$ may be decreased by dropping one of these constraints, as shown in Section 12.3. Thus, we remove an index j corresponding to one of the negative multipliers from the working set and solve a new subproblem (16.39) for the new step. We show in the following theorem that this strategy produces a direction p at the next iteration that is feasible with respect to the dropped constraint. We continue to assume that the constraint gradients a_i for i in the working set are linearly independent. After the algorithm has been fully stated, we discuss how this property can be maintained.

Theorem 16.5.

Suppose that the point \hat{x} satisfies first-order conditions for the equality-constrained subproblem with working set $\hat{\mathcal{W}}$; that is, equation (16.42) is satisfied along with $a_i^T \hat{x} = b_i$ for all $i \in \hat{\mathcal{W}}$. Suppose, too, that the constraint gradients a_i , $i \in \hat{\mathcal{W}}$, are linearly independent and that there is an index $j \in \hat{\mathcal{W}}$ such that $\hat{\lambda}_j < 0$. Let p be the solution obtained by dropping the constraint j and solving the following subproblem:

$$\min_p \frac{1}{2} p^T G p + (G\hat{x} + c)^T p, \quad (16.43a)$$

$$\text{subject to } a_i^T p = 0, \text{ for all } i \in \hat{\mathcal{W}} \text{ with } i \neq j. \quad (16.43b)$$

Then p is a feasible direction for constraint j , that is, $a_j^T p \geq 0$. Moreover, if p satisfies second-order sufficient conditions for (16.43), then we have that $a_j^T p > 0$, and that p is a descent direction for $q(\cdot)$.

PROOF. Since p solves (16.43), we have from the results of Section 16.1 that there are multipliers $\tilde{\lambda}_i$, for all $i \in \hat{\mathcal{W}}$ with $i \neq j$, such that

$$\sum_{i \in \hat{\mathcal{W}}, i \neq j} \tilde{\lambda}_i a_i = Gp + (G\hat{x} + c). \quad (16.44)$$

In addition, we have by second-order necessary conditions that if Z is a null-space basis vector for the matrix

$$[a_i^T]_{i \in \hat{\mathcal{W}}, i \neq j},$$

then $Z^T G Z$ is positive semidefinite. Clearly, p has the form $p = Z p_z$ for some vector p_z , so it follows that $p^T G p \geq 0$.

We have made the assumption that \hat{x} and $\hat{\mathcal{W}}$ satisfy the relation (16.42). By subtracting (16.42) from (16.44), we obtain

$$\sum_{i \in \hat{\mathcal{W}}, i \neq j} (\tilde{\lambda}_i - \hat{\lambda}_i) a_i - \hat{\lambda}_j a_j = Gp. \quad (16.45)$$

By taking inner products of both sides with p and using the fact that $a_i^T p = 0$ for all $i \in \hat{\mathcal{W}}$ with $i \neq j$, we have that

$$-\hat{\lambda}_j a_j^T p = p^T Gp. \quad (16.46)$$

Since $p^T Gp \geq 0$ and $\hat{\lambda}_j < 0$ by assumption, it follows that $a_j^T p \geq 0$.

If the second-order sufficient conditions of Section 12.5 are satisfied, we have that $Z^T GZ$ defined above is positive definite. From (16.46), we can have $a_j^T p = 0$ only if $p^T Gp = p_z^T Z^T GZ p_z = 0$, which happens only if $p_z = 0$ and $p = 0$. But if $p = 0$, then by substituting into (16.45) and using linear independence of a_i for $i \in \hat{\mathcal{W}}$, we must have that $\hat{\lambda}_j = 0$, which contradicts our choice of j . We conclude that $p^T Gp > 0$ in (16.46), and therefore $a_j^T p > 0$ whenever p satisfies the second-order sufficient conditions for (16.43).

The claim that p is a descent direction for $q(\cdot)$ is proved in Theorem 16.6 below. \square

While any index j for which $\hat{\lambda}_j < 0$ usually will yield a direction p along which the algorithm can make progress, the most negative multiplier is often chosen in practice (and in the algorithm specified below). This choice is motivated by the sensitivity analysis given in Chapter 12, which shows that the rate of decrease in the objective function when one constraint is removed is proportional to the magnitude of the Lagrange multiplier for that constraint. As in linear programming, however, the step along the resulting direction may be short (as when it is blocked by a new constraint), so the amount of decrease in q is not guaranteed to be greater than for other possible choices of j .

We conclude with a result that shows that whenever p_k obtained from (16.39) is nonzero and satisfies second-order sufficient optimality conditions for the current working set, it is a direction of strict descent for $q(\cdot)$.

Theorem 16.6.

Suppose that the solution p_k of (16.39) is nonzero and satisfies the second-order sufficient conditions for optimality for that problem. Then the function $q(\cdot)$ is strictly decreasing along the direction p_k .

PROOF. Since p_k satisfies the second-order conditions, that is, $Z^T GZ$ is positive definite for the matrix Z whose columns are a basis of the null space of the constraints (16.39b), we have by applying Theorem 16.2 to (16.39) that p_k is the unique global solution of (16.39). Since $p = 0$ is also a feasible point for (16.39), its objective value in (16.39a) must be larger

than that of p_k , so we have

$$\frac{1}{2}p_k^T G p_k + g_k^T p_k < 0.$$

Since $p_k^T G p_k \geq 0$ by convexity, this inequality implies that $g_k^T p_k < 0$. Therefore, we have

$$q(x_k + \alpha_k p_k) = q(x_k) + \alpha g_k^T p_k + \frac{1}{2}\alpha^2 p_k^T G p_k < q(x_k),$$

for all $\alpha > 0$ sufficiently small. □

When G is positive definite—the *strictly* convex case—the second-order sufficient conditions are satisfied for *all* feasible subproblems of the form (16.39). Hence, it follows from the result above that we obtain a strict decrease in $q(\cdot)$ whenever $p_k \neq 0$. This fact is significant when we discuss finite termination of the algorithm.

SPECIFICATION OF THE ACTIVE-SET METHOD FOR CONVEX QP

Having described the active-set algorithm for convex QP, we now present the following formal specification. We assume that the objective function q is bounded in the feasible set (16.1b), (16.1c).

Algorithm 16.3 (Active-Set Method for Convex QP).

```

Compute a feasible starting point  $x_0$ ;
Set  $\mathcal{W}_0$  to be a subset of the active constraints at  $x_0$ ;
for  $k = 0, 1, 2, \dots$ 
    Solve (16.39) to find  $p_k$ ;
    if  $p_k = 0$ 
        Compute Lagrange multipliers  $\hat{\lambda}_i$  that satisfy (16.42),
            with  $\hat{\mathcal{W}} = \mathcal{W}_k$ ;
        if  $\hat{\lambda}_i \geq 0$  for all  $i \in \mathcal{W}_k \cap \mathcal{I}$ 
            stop with solution  $x^* = x_k$ ;
        else
             $j \leftarrow \arg \min_{j \in \mathcal{W}_k \cap \mathcal{I}} \hat{\lambda}_j$ ;
             $x_{k+1} \leftarrow x_k$ ;  $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\}$ ;
        else (*  $p_k \neq 0$  *)
            Compute  $\alpha_k$  from (16.41);
             $x_{k+1} \leftarrow x_k + \alpha_k p_k$ ;
            if there are blocking constraints
                Obtain  $\mathcal{W}_{k+1}$  by adding one of the blocking
                    constraints to  $\mathcal{W}_k$ ;
            else
                 $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k$ ;
    end (for)
```


Various techniques can be used to determine an initial feasible point. One such is to use the “Phase I” approach for linear programming described in Chapter 13. Though no significant modifications are needed to generalize this method from linear programming to quadratic programming, we describe a variant here that allows the user to supply an initial estimate \tilde{x} of the vector x . This estimate need not be feasible, but a good choice based on knowledge of the QP may reduce the work needed in the Phase I step.

Given \tilde{x} , we define the following feasibility linear program:

$$\begin{aligned} & \min_{(x,z)} e^T z \\ & \text{subject to } a_i^T x + \gamma_i z_i = b_i, \quad i \in \mathcal{E}, \\ & \quad a_i^T x + \gamma_i z_i \geq b_i, \quad i \in \mathcal{I}, \\ & \quad z \geq 0, \end{aligned}$$

where $e = (1, 1, \dots, 1)^T$, $\gamma_i = -\text{sign}(a_i^T \tilde{x} - b_i)$ for $i \in \mathcal{E}$, and $\gamma_i = 1$ for $i \in \mathcal{I}$. A feasible initial point for this problem is then

$$x = \tilde{x}, \quad z_i = |a_i^T \tilde{x} - b_i| \quad (i \in \mathcal{E}), \quad z_i = \max(b_i - a_i^T \tilde{x}, 0) \quad (i \in \mathcal{I}).$$

It is easy to verify that if \tilde{x} is feasible for the original problem (16.1), then $(\tilde{x}, 0)$ is optimal for the feasibility subproblem. In general, if the original problem has feasible points, then the optimal objective value in the subproblem is zero, and any solution of the subproblem yields a feasible point for the original problem. The initial working set \mathcal{W}_0 for Algorithm 16.3 can be found by taking a linearly independent subset of the active constraints at the solution of the feasibility problem.

An alternative approach is a penalty (or “big M ”) method, which does away with the “Phase I” and instead includes a measure of infeasibility in the objective that is guaranteed to be zero at the solution. That is, we introduce a scalar artificial variable η into (16.1) to measure the constraint violation, and we solve the problem

$$\begin{aligned} & \min_{(x,\eta)} \frac{1}{2} x^T G x + x^T c + M\eta, \\ & \text{subject to } (a_i^T x - b_i) \leq \eta, \quad i \in \mathcal{E}, \\ & \quad -(a_i^T x - b_i) \leq \eta, \quad i \in \mathcal{E}, \\ & \quad b_i - a_i^T x \leq \eta, \quad i \in \mathcal{I}, \\ & \quad 0 \leq \eta, \end{aligned} \tag{16.47}$$

for some large positive value of M . It can be shown by applying the theory of exact penalty functions (see Chapter 17) that whenever there exist feasible points for the original problem (16.1), then for all M sufficiently large, the solution of (16.47) will have $\eta = 0$, with an x component that is a solution for (16.1).

Our strategy is to use some heuristic to choose a value of M and solve (16.47) by the usual means. If the solution we obtain has a positive value of η , we increase M and try again. Note that a feasible point is easy to obtain for the subproblem (16.47): We set $x = \tilde{x}$ (where, as before, \tilde{x} is the user-supplied initial guess) and choose η large enough that all the constraints in (16.47) are satisfied. This approach is, in fact, an exact penalty method using the ℓ_∞ norm; see Chapter 17.

A variant of (16.47) that penalizes the ℓ_1 norm of the constraint violation rather than the ℓ_∞ norm is as follows:

$$\begin{aligned} \min_{(x,s,t,v)} \quad & \frac{1}{2}x^T Gx + x^T c + Me_{\mathcal{E}}^T(s+t) + Me_{\mathcal{I}}^T v \\ \text{subject to} \quad & a_i^T x - b_i + s_i - t_i = 0, \quad i \in \mathcal{E}, \\ & a_i^T x - b_i + v_i \geq 0, \quad i \in \mathcal{I}, \\ & s \geq 0, \quad t \geq 0, \quad v \geq 0. \end{aligned} \tag{16.48}$$

Here, $e_{\mathcal{E}}$ is the vector $(1, 1, \dots, 1)^T$ of length $|\mathcal{E}|$; similarly for $e_{\mathcal{I}}$. The slack variables s_i , t_i , and v_i soak up any infeasibility in the constraints.

In the following example we use subscripts on the vectors x and p to denote their components, and we use superscripts to indicate the iteration index. For example, x_1 denotes the first component, while x^4 denotes the fourth iterate of the vector x .

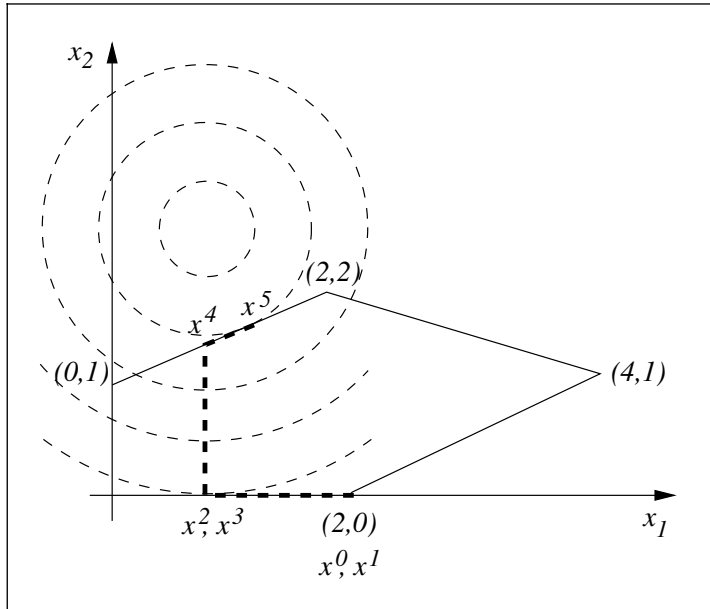


Figure 16.3 Iterates of the active-set method.

EXAMPLE 16.4

We apply Algorithm 16.3 to the following simple 2-dimensional problem illustrated in Figure 16.3.

$$\min_x q(x) = (x_1 - 1)^2 + (x_2 - 2.5)^2 \quad (16.49a)$$

$$\text{subject to } x_1 - 2x_2 + 2 \geq 0, \quad (16.49b)$$

$$-x_1 - 2x_2 + 6 \geq 0, \quad (16.49c)$$

$$-x_1 + 2x_2 + 2 \geq 0, \quad (16.49d)$$

$$x_1 \geq 0, \quad (16.49e)$$

$$x_2 \geq 0. \quad (16.49f)$$

We refer the constraints, in order, by indices 1 through 5. For this problem it is easy to determine a feasible initial point; say $x^0 = (2, 0)^T$. Constraints 3 and 5 are active at this point, and we set $\mathcal{W}_0 = \{3, 5\}$. (Note that we could just as validly have chosen $\mathcal{W}_0 = \{5\}$ or $\mathcal{W}_0 = \{3\}$ or even $\mathcal{W} = \emptyset$; each choice would lead the algorithm to perform somewhat differently.)

Since x^0 lies on a vertex of the feasible region, it is obviously a minimizer of the objective function q with respect to the working set \mathcal{W}_0 ; that is, the solution of (16.39) with $k = 0$ is $p = 0$. We can then use (16.42) to find the multipliers $\hat{\lambda}_3$ and $\hat{\lambda}_5$ associated with the active constraints. Substitution of the data from our problem into (16.42) yields

$$\begin{bmatrix} -1 \\ 2 \end{bmatrix} \hat{\lambda}_3 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \hat{\lambda}_5 = \begin{bmatrix} 2 \\ -5 \end{bmatrix},$$

which has the solution $(\hat{\lambda}_3, \hat{\lambda}_5) = (-2, -1)$.

We now remove constraint 3 from the working set, because it has the most negative multiplier, and set $\mathcal{W}_1 = \{5\}$. We begin iteration 1 by finding the solution of (16.39) for $k = 1$, which is $p^1 = (-1, 0)^T$. The step-length formula (16.41) yields $\alpha_1 = 1$, and the new iterate is $x^2 = (1, 0)^T$.

There are no blocking constraints, so that $\mathcal{W}_2 = \mathcal{W}_1 = \{5\}$, and we find at the start of iteration 2 that the solution of (16.39) is $p^2 = 0$. From (16.42) we deduce that the Lagrange multiplier for the lone working constraint is $\hat{\lambda}_5 = -5$, so we drop 5 from the working set to obtain $\mathcal{W}_3 = \emptyset$.

Iteration 3 starts by solving the unconstrained problem, to obtain the solution $p^3 = (0, 2.5)^T$. The formula (16.41) yields a step length of $\alpha_3 = 0.6$ and a new iterate $x^4 = (1, 1.5)^T$. There is a single blocking constraint (constraint 1), so we obtain $\mathcal{W}_4 = \{1\}$. The solution of (16.39) for $k = 4$ is then $p^4 = (0.4, 0.2)^T$, and the new step length is 1. There are no blocking constraints on this step, so the next working set is unchanged: $\mathcal{W}_5 = \{1\}$. The new iterate is $x^5 = (1.4, 1.7)^T$.

Finally, we solve (16.39) for $k = 5$ to obtain a solution $p^5 = 0$. The formula (16.42) yields a multiplier $\hat{\lambda}_1 = 0.8$, so we have found the solution. We set $x^* = (1.4, 1.7)^T$ and terminate. □

FURTHER REMARKS ON THE ACTIVE-SET METHOD

We noted above that there is flexibility in the choice of the initial working set and that each initial choice leads to a different iteration sequence. When the initial active constraints have independent gradients, as above, we can include them all in \mathcal{W}_0 . Alternatively, we can select a subset. For instance, if in the example above we have chosen $\mathcal{W}_0 = \{3\}$, the first iterate would have yielded $p^0 = (0.2, 0.1)^T$ and a new iterate of $x^1 = (2.2, 0.1)^T$. If we had chosen $\mathcal{W}_0 = \{5\}$, we would have moved immediately to the new iterate $x^1 = (1, 0)^T$, without first performing the operation of dropping the index 3, as is done in the example. If we had selected $\mathcal{W}_0 = \emptyset$, we would have obtained $p^1 = (-1, 2.5)^T$, $\alpha_1 = \frac{2}{3}$, a new iterate of $x^1 = (\frac{4}{3}, \frac{5}{3})^T$, and a new working set of $\mathcal{W}_1 = \{1\}$. The solution x^* would have been found on the next iteration.

Even if the initial working set \mathcal{W}_0 coincides with the initial active set, the sets \mathcal{W}_k and $\mathcal{A}(x^k)$ may differ at later iterations. For instance, when a particular step encounters more than one blocking constraint, just one of them is added to the working set, so the identification between \mathcal{W}_k and $\mathcal{A}(x^k)$ is broken. Moreover, subsequent iterates differ in general according to what choice is made.

We require the constraint gradients in \mathcal{W}_0 to be linearly independent, and our strategy for modifying the working set ensures that this same property holds for all subsequent working sets \mathcal{W}_k . When we encounter a blocking constraint on a particular step, its constraint normal cannot be a linear combination of the normals a_i in the current working set (see Exercise 16.18). Hence, linear independence is maintained after the blocking constraint is added to the working set. On the other hand, deletion of an index from the working set cannot introduce linear dependence.

The strategy of removing the constraint corresponding to the most negative Lagrange multiplier often works well in practice but has the disadvantage that it is susceptible to the scaling of the constraints. (By multiplying constraint i by some factor $\beta > 0$ we do not change the geometry of the optimization problem, but we introduce a scaling of $1/\beta$ to the corresponding multiplier λ_i .) Choice of the most negative multiplier is analogous to Dantzig's original pivot rule for the simplex method in linear programming (see Chapter 13) and, as we noted there, strategies that are less sensitive to scaling often give better results. We do not discuss this advanced topic further.

We note that the strategy of adding or deleting at most one constraint at each iteration of the Algorithm 16.3 places a natural lower bound on the number of iterations needed to reach optimality. Suppose, for instance, that we have a problem in which m inequality constraints are active at the solution x^* but that we start from a point x^0 that is strictly

feasible with respect to all the inequality constraints. In this case, the algorithm will need at least m iterations to move from x^0 to x^* . Even more iterations will be required if the algorithm adds some constraint j to the working set at some iteration, only to remove it at a later step.

FINITE TERMINATION OF ACTIVE-SET ALGORITHM ON STRICTLY CONVEX QPs

It is not difficult to show that, under certain assumptions, Algorithm 16.3 converges for strictly convex QPs, that is, it identifies the solution x^* in a finite number of iterations. This claim is certainly true if we assume that the method always takes a nonzero step length α_k whenever the direction p_k computed from (16.39) is nonzero. Our argument proceeds as follows:

- If the solution of (16.39) is $p_k = 0$, the current point x_k is the unique global minimizer of $q(\cdot)$ for the working set \mathcal{W}_k ; see Theorem 16.6. If it is not the solution of the original problem (16.1) (that is, at least one of the Lagrange multipliers is negative), Theorems 16.5 and 16.6 together show that the step p_{k+1} computed after a constraint is dropped will be a strict decrease direction for $q(\cdot)$. Therefore, because of our assumption $\alpha_k > 0$, we have that the value of q is lower than $q(x_k)$ at all subsequent iterations. It follows that the algorithm can never return to the working set \mathcal{W}_k , because subsequent iterates have values of q that are lower than the global minimizer for this working set.
- The algorithm encounters an iterate k for which $p_k = 0$ solves (16.39) at least on every n th iteration. To demonstrate this claim, we note that for any k at which $p_k \neq 0$, either we have $\alpha_k = 1$ (in which case we reach the minimizer of q on the current working set \mathcal{W}_k , so that the next iteration will yield $p_{k+1} = 0$), or else a constraint is added to the working set \mathcal{W}_k . If the latter situation occurs repeatedly, then after at most n iterations the working set will contain n indices, which correspond to n linearly independent vectors. The solution of (16.39) will then be $p_k = 0$, since only the zero vector will satisfy the constraints (16.39b).
- Taken together, the two statements above indicate that the algorithm finds the global minimum of q on its current working set periodically (at least once every n iterations) and that, having done so, it never visits this particular working set again. It follows that, since there are only a finite number of possible working sets, the algorithm cannot iterate forever. Eventually, it encounters a minimizer for a current working set that satisfies optimality conditions for (16.1), and it terminates with a solution.

The assumption that we can always take a nonzero step along a nonzero descent direction p_k calculated from (16.39) guarantees that the algorithm does not undergo *cycling*. This term refers to the situation in which a sequence of consecutive iterations results in no movement in iterate x , while the working set \mathcal{W}_k undergoes deletions and additions of indices

and eventually repeats itself. That is, for some integers k and $l \geq 1$, we have that $x^k = x^{k+l}$ and $\mathcal{W}_k = \mathcal{W}_{k+l}$. At each iterate in the cycle, a constraint is dropped (as in Theorem 16.5), but a new constraint $i \notin \mathcal{W}_k$ is encountered immediately without any movement along the computed direction p . Procedures for handling degeneracy and cycling in quadratic programming are similar to those for linear programming discussed in Chapter 13; we do not discuss them here. Most QP implementations simply ignore the possibility of cycling.

UPDATING FACTORIZATIONS

We have seen that the step computation in the active-set method given in Algorithm 16.3 requires the solution of the equality-constrained subproblem (16.39). As mentioned at the beginning of this chapter, this computation amounts to solving the KKT system (16.5). Since the working set can change by just one index at every iteration, the KKT matrix differs in at most one row and one column from the previous iteration's KKT matrix. Indeed, G remains fixed, whereas the matrix A of constraint gradients corresponding to the current working set may change through addition and/or deletion of a single row.

It follows from this observation that we can compute the matrix factors needed to solve (16.39) at the current iteration by updating the factors computed at the previous iteration, rather than recomputing them from scratch. These updating techniques are crucial to the efficiency of active-set methods.

We limit our discussion to the case in which the step is computed with the null-space method (16.17)–(16.20). Suppose that A has m linearly independent rows and assume that the bases Y and Z are defined by means of a QR factorization of A (see Section 15.3 for details). Thus

$$A^T \Pi = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} \quad (16.50)$$

(see (15.21)), where Π is a permutation matrix; R is square, upper triangular and nonsingular; $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is $n \times n$ orthogonal; and Q_1 and R both have m columns while Q_2 has $n - m$ columns. As noted in Chapter 15, we can choose Z to be simply the orthonormal matrix Q_2 .

Suppose that one constraint is added to the working set at the next iteration, so that the new constraint matrix is $\bar{A}^T = \begin{bmatrix} A^T & a \end{bmatrix}$, where a is a column vector of length n such that \bar{A}^T retains full column rank. As we now show, there is an economical way to update the Q and R factors in (16.50) to obtain new factors (and hence a new null-space basis matrix \bar{Z} , with $n - m - 1$ columns) for the expanded matrix \bar{A} . Note first that, since $Q_1 Q_1^T + Q_2 Q_2^T = I$, we have

$$\bar{A}^T \begin{bmatrix} \Pi & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} A^T \Pi & a \end{bmatrix} = Q \begin{bmatrix} R & Q_1^T a \\ 0 & Q_2^T a \end{bmatrix}. \quad (16.51)$$

We can now define an orthogonal matrix \hat{Q} that transforms the vector $Q_2^T a$ to a vector in which all elements except the first are zero. That is, we have

$$\hat{Q}(Q_2^T a) = \begin{bmatrix} \gamma \\ 0 \end{bmatrix},$$

where γ is a scalar. (Since \hat{Q} is orthogonal, we have $\|Q_2^T a\| = |\gamma|$.) From (16.51) we now have

$$\bar{A}^T \begin{bmatrix} \Pi & 0 \\ 0 & 1 \end{bmatrix} = Q \begin{bmatrix} R & Q_1^T a \\ 0 & \hat{Q}^T \begin{bmatrix} \gamma \\ 0 \end{bmatrix} \end{bmatrix} = Q \begin{bmatrix} I & 0 \\ 0 & \hat{Q}^T \end{bmatrix} \begin{bmatrix} R & Q_1^T a \\ 0 & \gamma \\ 0 & 0 \end{bmatrix}.$$

This factorization has the form

$$\bar{A}^T \bar{\Pi} = \bar{Q} \begin{bmatrix} \bar{R} \\ 0 \end{bmatrix},$$

where

$$\bar{\Pi} = \begin{bmatrix} \Pi & 0 \\ 0 & 1 \end{bmatrix}, \quad \bar{Q} = Q \begin{bmatrix} I & 0 \\ 0 & \hat{Q}^T \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \hat{Q}^T \end{bmatrix}, \quad \bar{R} = \begin{bmatrix} R & Q_1^T a \\ 0 & \gamma \end{bmatrix}.$$

We can therefore choose \bar{Z} to be the last $n - m - 1$ columns of $Q_2 \hat{Q}^T$. If we know Z explicitly and need an explicit representation of \bar{Z} , we need to account for the cost of obtaining \hat{Q} and the cost of forming the product $Q_2 \hat{Q}^T = Z \hat{Q}^T$. Because of the special structure of \hat{Q} , this cost is of order $n(n - m)$, compared to the cost of computing (16.50) from scratch, which is of order $n^2 m$. The updating strategy is less expensive, especially when the null space is small (that is, when $n - m \ll n$).

An updating technique can also be designed for the case in which a row is removed from A . This operation has the effect of deleting a column from R in (16.50), thus disturbing the upper triangular property of this matrix by introducing a number of nonzeros on the diagonal immediately below the main diagonal of the matrix. Upper triangularity can be restored by applying a sequence of plane rotations. These rotations introduce a number of inexpensive transformations into the first m columns of Q , and the updated null-space matrix is obtained by selecting the last $n - m + 1$ columns from this matrix after the transformations are complete. The new null-space basis in this case has the form

$$\bar{Z} = \begin{bmatrix} \bar{z} & Z \end{bmatrix}, \quad (16.52)$$

that is, the current matrix Z is augmented by a single column. The total cost of this operation varies with the location of the removed column in A but is in general cheaper

than recomputing a QR factorization from scratch. For details of these procedures, see Gill et al. [124, Section 5].

We now consider the reduced Hessian. Because of the special form of (16.39), we have $h = 0$ in (16.5), and the step p_v given in (16.18) is zero. Thus from (16.19), the null-space component p_z is the solution of

$$(Z^T G Z)p_z = -Z^T g. \quad (16.53)$$

We can sometimes find ways of updating the factorization of the reduced Hessian $Z^T G Z$ after Z has changed. Suppose that we have the Cholesky factorization of the current reduced Hessian, written as

$$Z^T G Z = L L^T,$$

and that at the next step Z changes as in (16.52), gaining a column after deletion of a constraint. A series of inexpensive, elementary operations can be used to transform the Cholesky factor L into the new factor \bar{L} for the new reduced Hessian $\bar{Z}^T G \bar{Z}$.

A variety of other simplifications are possible. For example, as discussed in Section 16.7, we can update the reduced gradient $Z^T g$ at the same time as we update Z to \bar{Z} .

16.6 INTERIOR-POINT METHODS

The interior-point approach can be applied to convex quadratic programs through a simple extension of the linear-programming algorithms described in Chapter 14. The resulting primal-dual algorithms are easy to describe and are quite efficient on many types of problems. Extensions of interior-point methods to nonconvex problems are discussed in Chapter 19.

For simplicity, we restrict our attention to convex quadratic programs with inequality constraints, which we write as follows:

$$\min_x \quad q(x) = \frac{1}{2}x^T G x + x^T c \quad (16.54a)$$

$$\text{subject to} \quad A x \geq b, \quad (16.54b)$$

where G is symmetric and positive semidefinite and where the $m \times n$ matrix A and right-hand side b are defined by

$$A = [a_i]_{i \in \mathcal{I}}, \quad b = [b_i]_{i \in \mathcal{I}}, \quad \mathcal{I} = \{1, 2, \dots, m\}.$$

(If equality constraints are also present, they can be accommodated with simple extensions to the approaches described below.) Rewriting the KKT conditions (16.37) in this notation,

While gradient projection methods can be applied in principle to problems with general linear constraints, significant computation may be required to perform the projection onto the feasible set in such cases. For example, if the constraint set is defined as $a_i^T x \geq b_i$, $i \in \mathcal{I}$, we must solve the following convex quadratic program to compute the projection of a given point \bar{x} onto this set:

$$\max_x \|x - \bar{x}\|^2 \quad \text{subject to } a_i^T x \geq b_i \text{ for all } i \in \mathcal{I}.$$

The expense of solving this “projection subproblem” may approach the cost of solving the original quadratic program, so it is usually not economical to apply gradient projection to this case.

When we use duality to replace a strictly convex quadratic program with its dual (see Example 12.12), the gradient projection method may be useful in solving the bound-constrained dual problem, which is formulated in terms of the Lagrange multipliers λ as follows:

$$\max_{\lambda} \tilde{q}(\lambda) = -\frac{1}{2}(A^T \lambda - c)^T G^{-1}(A^T \lambda - c)^T + b^T \lambda, \quad \text{subject to } \lambda \geq 0.$$

(Note that the dual is conventionally written as a maximization problem; we can equivalently minimize $-\tilde{q}(\lambda)$ and note that this transformed problem is convex.) This approach is most useful when G has a simple form, for example, a diagonal or block-diagonal matrix.

16.8 PERSPECTIVES AND SOFTWARE

Active-set methods for convex quadratic programming are implemented in QPOPT [126], VE09 [142], BQPD [103], and QPA [148]. Several commercial interior-point solvers for QP are available, including CPLEX [172], XPRESS-MP [159] and MOSEK [5]. The code QPB [146] uses a two-phase interior-point method that can handle convex and nonconvex problems. OOPS [139] and OOQP [121] are object-oriented interior-point codes that allow the user to customize the linear algebra techniques to the particular structure of the data for an application. Some nonlinear programming interior-point packages, such as LOQO [294] and KNITRO [46], are also effective for convex and nonconvex quadratic programming.

The numerical comparison of active-set and interior-point methods for convex quadratic programming reported in [149] indicates that interior-point methods are generally much faster on large problems. If a warm start is required, however, active-set methods may be generally preferable. Although considerable research has been focused on improving the warm-start capabilities of interior-point methods, the full potential of such techniques is now yet known.

We have assumed in this chapter that all equality-constrained quadratic programs have linearly independent constraints, that is, the $m \times n$ constraint Jacobian matrix A has rank m .

If redundant constraints are present, they can be detected by forming a SVD or rank-revealing QR factorization of A^T , and then removed from the formulation. When A is larger, sparse Gaussian elimination techniques can be applied to A^T instead, but they are less reliable.

The KNITRO and OOPS software packages provide the option of solving the primal-dual equations (16.63) by means of the projected CG iteration of Algorithm 16.2.

We have not considered active-set methods for the case in which the Hessian matrix G is indefinite because these methods can be quite complicated to describe and it is not well understood how to adapt them to the large dimensional case. We make some comments here on the principal techniques.

Algorithm 16.3, the active-set method for convex QP, can be adapted to this indefinite case by modifying the computation of the search direction and step length in certain situations. To explain the need for the modification, we consider the computation of a step by a null-space method, that is, $p = Zp_z$, where p_z is given by (16.53). If the reduced Hessian $Z^T G Z$ is positive definite, then this step p points to the minimizer of the subproblem (16.39), and the logic of the iteration need not be changed. If $Z^T G Z$ has negative eigenvalues, however, p points only to a saddle point of (16.39) and is therefore not always a suitable step. Instead, we seek an alternative direction s_z that is a direction of *negative curvature* for $Z^T G Z$. We then have that

$$q(x + \alpha Zs_z) \rightarrow -\infty \quad \text{as } \alpha \rightarrow \infty. \quad (16.75)$$

Additionally, we change the sign of s_z if necessary to ensure that Zs_z is a non-ascent direction for q at the current point x , that is, $\nabla q(x)^T Zs_z \leq 0$. By moving along the direction Zs_z , we will encounter a constraint that can be added to the working set for the next iteration. (If we don't find such a constraint, the problem is unbounded.) If the reduced Hessian for the new working set is not positive definite, we repeat this process until enough constraints have been added to make the reduced Hessian positive definite. A difficulty with this general approach, however, is that if we allow the reduced Hessian to have several negative eigenvalues, it is difficult to make these methods efficient when the reduced Hessian changes from one working set to the next.

Inertia controlling methods are a practical class of algorithms for indefinite QP that never allow the reduced Hessian to have more than one negative eigenvalue. As in the convex case, there is a preliminary phase in which a feasible starting point x_0 is found. We place the additional demand on x_0 that it be either a vertex (in which case the reduced Hessian is the null matrix) or a constrained stationary point at which the reduced Hessian is positive definite. At each iteration, the algorithm will either add or remove a constraint from the working set. If a constraint is added, the reduced Hessian is of smaller dimension and must remain positive definite or be the null matrix. Therefore, an indefinite reduced Hessian can arise only when one of the constraints is removed from the working set, which happens only when the current point is a minimizer with respect to the current working set. In this case, we will choose the new search direction to be a direction of negative curvature for the reduced Hessian.

Various algorithms for indefinite QP differ in the way that indefiniteness is detected, in the computation of the negative curvature direction, and in the handling of the working set; see Fletcher [99] and Gill and Murray [126].

NOTES AND REFERENCES

The problem of determining whether a feasible point for a nonconvex QP (16.1) is a global minimizer is NP-hard (Murty and Kabadi [219]); so is the problem of determining whether a given point is a local minimizer (Vavasis [296, Theorem 5.1]). Various algorithms for convex QP with polynomial convexity are discussed in Nesterov and Nemirovskii [226].

The portfolio optimization problem was formulated by Markowitz [201].

For a discussion on the QMR, LSQR, and GMRES methods see, for example, [136, 272, 290]. The idea of using the projection (16.30) in the CG method dates back to at least Polyak [238]. The alternative (16.34), and its special case (16.32), are proposed in Coleman [64]. Although it can give rise to substantial rounding errors, they can be corrected by iterative refinement; see Gould et al. [143]. More recent studies on preconditioning of the projected CG method include Keller et al. [176] and Lukšan and Vlček [196].

For further discussion on the gradient projection method see, for example, Conn, Gould, and Toint [70] and Burke and Moré [44].

In some areas of application, the KKT matrix (16.7) not only is sparse but also contains special structure. For instance, the quadratic programs that arise in many control problems have banded matrices G and A (see Wright [315]), which can be exploited by interior-point methods via a suitable symmetric reordering of K . When active-set methods are applied to this problem, however, the advantages of bandedness and sparsity are lost after just a few updates of the factorization.

Further details of interior-point methods for convex quadratic programming can be found in Wright [316] and Vanderbei [293]. The first inertia-controlling method for indefinite quadratic programming was proposed by Fletcher [99]. See also Gill et al. [129] and Gould [142] for a discussion of methods for general quadratic programming.



EXERCISES




16.1

- (a) Solve the following quadratic program and illustrate it geometrically.

$$\begin{aligned} \min f(x) &= 2x_1 + 3x_2 + 4x_1^2 + 2x_1x_2 + x_2^2, \\ \text{subject to } x_1 - x_2 &\geq 0, \quad x_1 + x_2 \leq 4, \quad x_1 \leq 3. \end{aligned}$$

- (b) If the objective function is redefined as $q(x) = -f(x)$, does the problem have a finite minimum? Are there local minimizers?

 **16.2** The problem of finding the shortest distance from a point x_0 to the hyperplane $\{x \mid Ax = b\}$, where A has full row rank, can be formulated as the quadratic program

$$\min \frac{1}{2}(x - x_0)^T(x - x_0) \quad \text{subject to} \quad Ax = b.$$


Show that the optimal multiplier is


$$\lambda^* = (AA^T)^{-1}(b - Ax_0)$$


and that the solution is


$$x^* = x_0 + A^T(AA^T)^{-1}(b - Ax_0).$$


Show that in the special case in which A is a row vector, the shortest distance from x_0 to the solution set of $Ax = b$ is $|b - Ax_0|/\|A\|_2$.


 **16.3** Use Theorem 12.1 to verify that the first-order necessary conditions for (16.3) are given by (16.4).


 **16.4** Suppose that G is positive semidefinite in (16.1) and that x^* satisfies the KKT conditions (16.37) for some $\lambda_i^*, i \in \mathcal{A}(x^*)$. Suppose in addition that second-order sufficient conditions are satisfied, that is, $Z^T G Z$ is positive definite where the columns of Z span the null space of the active constraint Jacobian matrix. Show that x^* is in fact the unique global solution for (16.1), that is, $q(x) > q(x^*)$ for all feasible x with $x \neq x^*$.


 **16.5** Verify that the inverse of the KKT matrix is given by (16.16).


 **16.6** Use Theorem 12.6 to show that if the conditions of Lemma 16.1 hold, then the second-order sufficient conditions for (16.3) are satisfied by the vector pair (x^*, λ^*) that satisfies (16.4).

 **16.7** Consider (16.3) and suppose that the projected Hessian matrix $Z^T G Z$ has a negative eigenvalue; that is, $u^T Z^T G Z u < 0$ for some vector u . Show that if there exists any vector pair (x^*, λ^*) that satisfies (16.4), then the point x^* is only a stationary point of (16.3) and not a local minimizer. (Hint: Consider the function $q(x^* + \alpha Z u)$ for $\alpha \neq 0$, and use an expansion like that in the proof of Theorem 16.2.)

 **16.8** By using the QR factorization and a permutation matrix, show that for a full-rank $m \times n$ matrix A (with $m < n$) one can find an orthogonal matrix Q and an $m \times m$ upper triangular matrix \hat{U} such that $AQ = \begin{bmatrix} 0 & \hat{U} \end{bmatrix}$. (Hint: Start by applying the standard QR factorization to A^T .)


 **16.9** Verify that the first-order conditions for optimality of (16.1) are equivalent to (16.37) when we make use of the active-set definition (16.36).

 **16.10** For each of the alternative choices of initial working set \mathcal{W}_0 in the example (16.49) (that is, $\mathcal{W}_0 = \{3\}$, $\mathcal{W}_0 = \{5\}$, and $\mathcal{W}_0 = \emptyset$) work through the first two iterations of Algorithm 16.3.

 **16.11** Program Algorithm 16.3, and use it to solve the problem

$$\begin{aligned} \min \quad & x_1^2 + 2x_2^2 - 2x_1 - 6x_2 - 2x_1x_2 \\ \text{subject to} \quad & \frac{1}{2}x_1 + \frac{1}{2}x_2 \leq 1, \quad -x_1 + 2x_2 \leq 2, \quad x_1, x_2 \geq 0. \end{aligned}$$

Choose three initial starting points: one in the interior of the feasible region, one at a vertex, and one at a non-vertex point on the boundary of the feasible region.


 **16.12** Show that the operator P defined by (16.27) is independent of the choice of null-space basis Z . (Hint: First show that any null-space basis Z can be written as $Z = QB$ where Q is an orthogonal basis and B is a nonsingular matrix.)


 **16.13**

- (a) Show that the computation of the preconditioned residual g^+ in (16.28d) can be performed with (16.29) or (16.30).
- (b) Show that we can also perform this computation by solving the system (16.32).
- (c) Verify (16.33).


 **16.14**

- (a) Show that if $Z^T G Z$ is positive definite, then the denominator in (16.28a) is nonzero.
- (b) Show that if $Z^T r = r_z \neq 0$ and $Z^T H Z$ is positive definite, then the denominator in (16.28e) is nonzero.

 **16.15** Consider problem (16.3), and assume that A has full row rank and that Z is a basis for the null space of A . Prove that there are no finite solutions if $Z^T G Z$ has negative eigenvalues.


 **16.16**


- (a) Assume that $A \neq 0$. Show that the KKT matrix (16.7) is indefinite.
- (b) Prove that if the KKT matrix (16.7) is nonsingular, then A must have full rank.


 **16.17** Consider the quadratic program


$$\begin{aligned} \max \quad & 6x_1 + 4x_2 - 13 - x_1^2 - x_2^2, \\ \text{subject to} \quad & x_1 + x_2 \leq 3, \quad x_1 \geq 0, \quad x_2 \geq 0. \end{aligned} \tag{16.76}$$

First solve it graphically, and then use your program implementing the active-set method given in Algorithm 16.3.

 **16.18** Using (16.39) and (16.41), explain briefly why the gradient of each blocking constraint cannot be a linear combination of the constraint gradients in the current working set \mathcal{W}_k .


 **16.19** Let W be an $n \times n$ symmetric matrix, and suppose that Z is of dimension $n \times t$. Suppose that $Z^T W Z$ is positive definite and that \bar{Z} is obtained by removing a column from Z . Show that $\bar{Z}^T W \bar{Z}$ is positive definite.


 **16.20** Find a null-space basis matrix Z for the equality-constrained problem defined by (16.74a), (16.74b).

 **16.21** Write down KKT conditions for the following convex quadratic program with mixed equality and inequality constraints:

$$\min q(x) = \frac{1}{2}x^T Gx + x^T c \quad \text{subject to} \quad Ax \geq b, \quad \bar{A}x = \bar{b},$$


where G is symmetric and positive semidefinite. Use these conditions to derive an analogue of the generic primal-dual step (16.58) for this problem.


 **16.22** Explain why for a bound-constrained problems the number of possible active sets is at most 3^n .

 **16.23**

(a) Show that the primal-dual system (16.58) can be solved using the augmented system (16.61) or the normal equations (16.62). Describe in detail how all the components $(\Delta x, \Delta y, \Delta \lambda)$ are computed.

(b) Verify (16.65).

 **16.24** Program Algorithm 16.4 and use it to solve problem (16.76). Set all initial variables to be the vector $e = (1, 1, \dots, 1)^T$.

 **16.25** Let $\bar{x} \in R^n$ be given, and let x^* be the solution of the projection problem


$$\min \|x - \bar{x}\|^2 \quad \text{subject to} \quad l \leq x \leq u. \quad (16.77)$$

For simplicity, assume that $-\infty < l_i < u_i < \infty$ for all $i = 1, 2, \dots, n$. Show that the solution of this problem coincides with the projection formula given by (16.69) that is, show that $x^* = P(\bar{x}, l, u)$. (Hint: Note that the problem is separable.)

 **16.26** Consider the bound-constrained quadratic problem (16.68) with

$$G = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad l = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad u = \begin{bmatrix} 5 \\ 3 \end{bmatrix}. \quad (16.78)$$

Suppose $x^0 = (0, 2)^T$. Find $\bar{t}_1, \bar{t}_2, t_1, t_2, p^1, p^2$ and $x(t_1), x(t_2)$. Find the minimizer of $q(x(t))$.

 **16.27** Consider the search for the one dimensional minimizer of the function $q(x(t))$ defined by (16.73). There are 9 possible cases since f, f', f'' can each be positive, negative, or zero. For each case, determine the location of the minimizer. Verify that the rules described in Section 16.7 hold.

CHAPTER *17*

Penalty and Augmented Lagrangian Methods

Some important methods for constrained optimization replace the original problem by a sequence of subproblems in which the constraints are represented by terms added to the objective. In this chapter we describe three approaches of this type. The *quadratic penalty* method adds a multiple of the square of the violation of each constraint to the objective. Because of its simplicity and intuitive appeal, this approach is used often in practice, although it has some important disadvantages. In *nonsmooth exact penalty* methods, a single unconstrained problem (rather than a sequence) takes the place of the original constrained problem. Using these penalty functions, we can often find a solution by performing a single

CHAPTER *18*

Sequential Quadratic Programming

One of the most effective methods for nonlinearly constrained optimization generates steps by solving quadratic subproblems. This sequential quadratic programming (SQP) approach can be used both in line search and trust-region frameworks, and is appropriate for small or large problems. Unlike linearly constrained Lagrangian methods (Chapter 17), which are effective when most of the constraints are linear, SQP methods show their strength when solving problems with significant nonlinearities in the constraints.

All the methods considered in this chapter are active-set methods; a more descriptive title for this chapter would perhaps be “Active-Set Methods for Nonlinear Programming.”

In Chapter 14 we study interior-point methods for nonlinear programming, a competing approach for handling inequality-constrained problems.

There are two types of active-set SQP methods. In the IQP approach, a general inequality-constrained quadratic program is solved at each iteration, with the twin goals of computing a step and generating an estimate of the optimal active set. EQP methods decouple these computations. They first compute an estimate of the optimal active set, then solve an equality-constrained quadratic program to find the step. In this chapter we study both IQP and EQP methods.

Our development of SQP methods proceeds in two stages. First, we consider local methods that motivate the SQP approach and allow us to introduce the step computation techniques in a simple setting. Second, we consider practical line search and trust-region methods that achieve convergence from remote starting points. Throughout the chapter we give consideration to the algorithmic demands of solving large problems.

18.1 LOCAL SQP METHOD

We begin by considering the equality-constrained problem

$$\min f(x) \tag{18.1a}$$

$$\text{subject to } c(x) = 0, \tag{18.1b}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $c : \mathbb{R}^n \rightarrow \mathbb{R}^m$ are smooth functions. The idea behind the SQP approach is to model (18.1) at the current iterate x_k by a quadratic programming subproblem, then use the minimizer of this subproblem to define a new iterate x_{k+1} . The challenge is to design the quadratic subproblem so that it yields a good step for the nonlinear optimization problem. Perhaps the simplest derivation of SQP methods, which we present now, views them as an application of Newton's method to the KKT optimality conditions for (18.1).

From (12.33), we know that the Lagrangian function for this problem is $\mathcal{L}(x, \lambda) = f(x) - \lambda^T c(x)$. We use $A(x)$ to denote the Jacobian matrix of the constraints, that is,

$$A(x)^T = [\nabla c_1(x), \nabla c_2(x), \dots, \nabla c_m(x)], \tag{18.2}$$

where $c_i(x)$ is the i th component of the vector $c(x)$. The first-order (KKT) conditions (12.34) of the equality-constrained problem (18.1) can be written as a system of $n + m$ equations in the $n + m$ unknowns x and λ :

$$F(x, \lambda) = \begin{bmatrix} \nabla f(x) - A(x)^T \lambda \\ c(x) \end{bmatrix} = 0. \tag{18.3}$$

Any solution (x^*, λ^*) of the equality-constrained problem (18.1) for which $A(x^*)$ has full

rank satisfies (18.3). One approach that suggests itself is to solve the nonlinear equations (18.3) by using Newton's method, as described in Chapter 11.

The Jacobian of (18.3) with respect to x and λ is given by

$$F'(x, \lambda) = \begin{bmatrix} \nabla_{xx}^2 \mathcal{L}(x, \lambda) & -A(x)^T \\ A(x) & 0 \end{bmatrix}. \quad (18.4)$$

The Newton step from the iterate (x_k, λ_k) is thus given by

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix}, \quad (18.5)$$

where p_k and p_λ solve the Newton–KKT system

$$\begin{bmatrix} \nabla_{xx}^2 \mathcal{L}_k & -A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f_k + A_k^T \lambda_k \\ -c_k \end{bmatrix}. \quad (18.6)$$

This Newton iteration is well defined when the KKT matrix in (18.6) is nonsingular. We saw in Chapter 16 that this matrix is nonsingular if the following assumption holds at $(x, \lambda) = (x_k, \lambda_k)$.

Assumptions 18.1.

- (a) The constraint Jacobian $A(x)$ has full row rank;
- (b) The matrix $\nabla_{xx}^2 \mathcal{L}(x, \lambda)$ is positive definite on the tangent space of the constraints, that is, $d^T \nabla_{xx}^2 \mathcal{L}(x, \lambda) d > 0$ for all $d \neq 0$ such that $A(x)d = 0$.

The first assumption is the linear independence constraint qualification discussed in Chapter 12 (see Definition 12.4), which we assume throughout this chapter. The second condition holds whenever (x, λ) is close to the optimum (x^*, λ^*) and the second-order sufficient condition is satisfied at the solution (see Theorem 12.6). The Newton iteration (18.5), (18.6) can be shown to be quadratically convergent under these assumptions (see Theorem 18.4) and constitutes an excellent algorithm for solving equality-constrained problems, provided that the starting point is close enough to x^* .

SQP FRAMEWORK

There is an alternative way to view the iteration (18.5), (18.6). Suppose that at the iterate (x_k, λ_k) we model problem (18.1) using the quadratic program

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.7a)$$

$$\text{subject to} \quad A_k p + c_k = 0. \quad (18.7b)$$

If Assumptions 18.1 hold, this problem has a unique solution (p_k, l_k) that satisfies

$$\nabla_{xx}^2 \mathcal{L}_k p_k + \nabla f_k - A_k^T l_k = 0, \quad (18.8a)$$

$$A_k p_k + c_k = 0. \quad (18.8b)$$

The vectors p_k and l_k can be identified with the solution of the Newton equations (18.6). If we subtract $A_k^T \lambda_k$ from both sides of the first equation in (18.6), we obtain

$$\begin{bmatrix} \nabla_{xx}^2 \mathcal{L}_k & -A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f_k \\ -c_k \end{bmatrix}. \quad (18.9)$$

Hence, by nonsingularity of the coefficient matrix, we have that $\lambda_{k+1} = l_k$ and that p_k solves (18.7) and (18.6).

The new iterate (x_{k+1}, λ_{k+1}) can therefore be defined either as the solution of the quadratic program (18.7) or as the iterate generated by Newton's method (18.5), (18.6) applied to the optimality conditions of the problem. Both viewpoints are useful. The Newton point of view facilitates the analysis, whereas the SQP framework enables us to derive practical algorithms and to extend the technique to the inequality-constrained case.

We now state the SQP method in its simplest form.

Algorithm 18.1 (Local SQP Algorithm for solving (18.1)).

```

Choose an initial pair  $(x_0, \lambda_0)$ ; set  $k \leftarrow 0$ ;
repeat until a convergence test is satisfied
    Evaluate  $f_k, \nabla f_k, \nabla_{xx}^2 \mathcal{L}_k, c_k$ , and  $A_k$ ;
    Solve (18.7) to obtain  $p_k$  and  $l_k$ ;
    Set  $x_{k+1} \leftarrow x_k + p_k$  and  $\lambda_{k+1} \leftarrow l_k$ ;
end (repeat)
```

We note in passing that, in the objective (18.7a) of the quadratic program, we could replace the linear term $\nabla f_k^T p$ by $\nabla_x \mathcal{L}(x_k, \lambda_k)^T p$, since the constraint (18.7b) makes the two choices equivalent. In this case, (18.7a) is a quadratic approximation of the Lagrangian function. This fact provides a motivation for our choice of the quadratic model (18.7): We first replace the nonlinear program (18.1) by the problem of minimizing the Lagrangian subject to the equality constraints (18.1b), then make a quadratic approximation to the Lagrangian and a linear approximation to the constraints to obtain (18.7).

INEQUALITY CONSTRAINTS

The SQP framework can be extended easily to the general nonlinear programming problem

$$\min f(x) \quad (18.10a)$$

$$\text{subject to } c_i(x) = 0, \quad i \in \mathcal{E}, \quad (18.10b)$$

$$c_i(x) \geq 0, \quad i \in \mathcal{I}. \quad (18.10c)$$

To model this problem we now linearize both the inequality and equality constraints to obtain

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.11a)$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \quad (18.11b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \quad (18.11c)$$

We can use one of the algorithms for quadratic programming described in Chapter 16 to solve this problem. The new iterate is given by $(x_k + p_k, \lambda_{k+1})$ where p_k and λ_{k+1} are the solution and the corresponding Lagrange multiplier of (18.11). A local SQP method for (18.10) is thus given by Algorithm 18.1 with the modification that the step is computed from (18.11).

In this IQP approach the set of active constraints \mathcal{A}_k at the solution of (18.11) constitutes our guess of the active set at the solution of the nonlinear program. If the SQP method is able to correctly identify this optimal active set (and not change its guess at a subsequent iteration) then it will act like a Newton method for equality-constrained optimization and will converge rapidly. The following result gives conditions under which this desirable behavior takes place. Recall that strict complementarity is said to hold at a solution pair (x^*, λ^*) if there is no index $i \in \mathcal{I}$ such that $\lambda_i^* = c_i(x^*) = 0$.

Theorem 18.1 (Robinson [267]).

Suppose that x^ is a local solution of (18.10) at which the KKT conditions are satisfied for some λ^* . Suppose, too, that the linear independence constraint qualification (LICQ) (Definition 12.4), the strict complementarity condition (Definition 12.5), and the second-order sufficient conditions (Theorem 12.6) hold at (x^*, λ^*) . Then if (x_k, λ_k) is sufficiently close to (x^*, λ^*) , there is a local solution of the subproblem (18.11) whose active set \mathcal{A}_k is the same as the active set $\mathcal{A}(x^*)$ of the nonlinear program (18.10) at x^* .*

It is also remarkable that, far from the solution, the SQP approach is usually able to improve the estimate of the active set and guide the iterates toward a solution; see Section 18.7.

18.2 PREVIEW OF PRACTICAL SQP METHODS

IQP AND EQP

There are two ways of designing SQP methods for solving the general nonlinear programming problem (18.10). The first is the approach just described, which solves at

every iteration the quadratic subprogram (18.11), taking the active set at the solution of this subproblem as a guess of the optimal active set. This approach is referred to as the IQP (inequality-constrained QP) approach; it has proved to be quite successful in practice. Its main drawback is the expense of solving the general quadratic program (18.11), which can be high when the problem is large. As the iterates of the SQP method converge to the solution, however, solving the quadratic subproblem becomes economical if we use information from the previous iteration to make a good guess of the optimal solution of the current subproblem. This *warm-start* strategy is described below.

The second approach selects a subset of constraints at each iteration to be the so-called working set, and solves only equality-constrained subproblems of the form (18.7), where the constraints in the working sets are imposed as equalities and all other constraints are ignored. The working set is updated at every iteration by rules based on Lagrange multiplier estimates, or by solving an auxiliary subproblem. This EQP (equality-constrained QP) approach has the advantage that the equality-constrained quadratic subproblems are less expensive to solve than (18.11) in the large-scale case.

An example of an EQP method is the sequential linear-quadratic programming (SLQP) method discussed in Section 18.5. This approach constructs a linear program by omitting the quadratic term $p^T \nabla_{xx}^2 \mathcal{L}_k p$ from (18.11a) and adding a trust-region constraint $\|p\|_\infty \leq \Delta_k$ to the subproblem. The active set of the resulting linear programming subproblem is taken to be the working set for the current iteration. The method then fixes the constraints in the working set and solves an equality-constrained quadratic program (with the term $p^T \nabla_{xx}^2 \mathcal{L}_k p$ reinserted) to obtain the SQP step. Another successful EQP method is the gradient projection method described in Section 16.7 in the context of bound constrained quadratic programs. In this method, the working set is determined by minimizing a quadratic model along the path obtained by projecting the steepest descent direction onto the feasible region.

ENFORCING CONVERGENCE

To be practical, an SQP method must be able to converge from remote starting points and on nonconvex problems. We now outline how the local SQP strategy can be adapted to meet these goals.

We begin by drawing an analogy with unconstrained optimization. In its simplest form, the Newton iteration for minimizing a function f takes a step to the minimizer of the quadratic model

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2 f_k p.$$

This framework is useful near the solution, where the Hessian $\nabla^2 f(x_k)$ is normally positive definite and the quadratic model has a well defined minimizer. When x_k is not close to the solution, however, the model function m_k may not be convex. Trust-region methods ensure that the new iterate is always well defined and useful by restricting the candidate step p_k

to some neighborhood of the origin. Line search methods modify the Hessian in $m_k(p)$ to make it positive definite (possibly replacing it by a quasi-Newton approximation B_k), to ensure that p_k is a descent direction for the objective function f .

Similar strategies are used to globalize SQP methods. If $\nabla_{xx}^2 \mathcal{L}_k$ is positive definite on the tangent space of the active constraints, the quadratic subproblem (18.7) has a unique solution. When $\nabla_{xx}^2 \mathcal{L}_k$ does not have this property, line search methods either replace it by a positive definite approximation B_k or modify $\nabla_{xx}^2 \mathcal{L}_k$ directly during the process of matrix factorization. In all these cases, the subproblem (18.7) becomes well defined, but the modifications may introduce unwanted distortions in the model.

Trust-region SQP methods add a constraint to the subproblem, limiting the step to a region within which the model (18.7) is considered reliable. These methods are able to handle indefinite Hessians $\nabla_{xx}^2 \mathcal{L}_k$. The inclusion of the trust region may, however, cause the subproblem to become infeasible, and the procedures for handling this situation complicate the algorithms and increase their computational cost. Due to these tradeoffs, neither of the two SQP approaches—line search or trust-region—is currently regarded as clearly superior to the other.

The technique used to accept or reject steps also impacts the efficiency of SQP methods. In unconstrained optimization, the merit function is simply the objective f , and it remains fixed throughout the minimization procedure. For constrained problems, we use devices such as a merit function or a filter (see Section 15.4). The parameters or entries used in these devices must be updated in a way that is compatible with the step produced by the SQP method.

18.3 ALGORITHMIC DEVELOPMENT

In this section we expand on the ideas of the previous section and describe various ingredients needed to produce practical SQP algorithms. We focus on techniques for ensuring that the subproblems are always feasible, on alternative choices for the Hessian of the quadratic model, and on step-acceptance mechanisms.

HANDLING INCONSISTENT LINEARIZATIONS

A possible difficulty with SQP methods is that the linearizations (18.11b), (18.11c) of the nonlinear constraints may give rise to an infeasible subproblem. Consider, for example, the case in which $n = 1$ and the constraints are $x \leq 1$ and $x^2 \geq 4$. When we linearize these constraints at $x_k = 1$, we obtain the inequalities

$$-p \geq 0 \quad \text{and} \quad 2p - 3 \geq 0,$$

which are inconsistent.

To overcome this difficulty, we can reformulate the nonlinear program (18.10) as the ℓ_1 penalty problem

$$\min_{x, v, w, t} \quad f(x) + \mu \sum_{i \in \mathcal{E}} (v_i + w_i) + \mu \sum_{i \in \mathcal{I}} t_i \quad (18.12a)$$

$$\text{subject to} \quad c_i(x) = v_i - w_i, \quad i \in \mathcal{E}, \quad (18.12b)$$

$$c_i(x) \geq -t_i, \quad i \in \mathcal{I}, \quad (18.12c)$$

$$v, w, t \geq 0, \quad (18.12d)$$

for some positive choice of the penalty parameter μ . The quadratic subproblem (18.11) associated with (18.12) is always feasible. As discussed in Chapter 17, if the nonlinear problem (18.10) has a solution x^* that satisfies certain regularity assumptions, and if the penalty parameter μ is sufficiently large, then x^* (along with $v_i^* = w_i^* = 0, i \in \mathcal{E}$ and $t_i^* = 0, i \in \mathcal{I}$) is a solution of the penalty problem (18.12). If, on the other hand, there is no feasible solution to the nonlinear problem and μ is large enough, then the penalty problem (18.12) usually determines a stationary point of the infeasibility measure. The choice of μ has been discussed in Chapter 17 and is considered again in Section 18.5. The SNOPT software package [127] uses the formulation (18.12), which is sometimes called the *elastic mode*, to deal with inconsistencies of the linearized constraints.

Other procedures for relaxing the constraints are presented in Section 18.5 in the context of trust-region methods.

FULL QUASI-NEWTON APPROXIMATIONS

The Hessian of the Lagrangian $\nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k)$ is made up of second derivatives of the objective function and constraints. In some applications, this information is not easy to compute, so it is useful to consider replacing the Hessian $\nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k)$ in (18.11a) by a quasi-Newton approximation. Since the BFGS and SR1 formulae have proved to be successful in the context of unconstrained optimization, we can employ them here as well.

The update for B_k that results from the step from iterate k to iterate $k + 1$ makes use of the vectors s_k and y_k defined as follows:

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}). \quad (18.13)$$

We compute the new approximation B_{k+1} using the BFGS or SR1 formulae given, respectively, by (6.19) and (6.24). We can view this process as the application of quasi-Newton updating to the case in which the objective function is given by the Lagrangian $\mathcal{L}(x, \lambda)$ (with λ fixed). This viewpoint immediately reveals the strengths and weaknesses of this approach.

If $\nabla_{xx}^2 \mathcal{L}$ is positive definite in the region where the minimization takes place, then BFGS quasi-Newton approximations B_k will reflect some of the curvature information of the problem, and the iteration will converge robustly and rapidly, just as in the unconstrained BFGS method. If, however, $\nabla_{xx}^2 \mathcal{L}$ contains negative eigenvalues, then the BFGS approach

of approximating it with a positive definite matrix may be problematic. BFGS updating requires that s_k and y_k satisfy the curvature condition $s_k^T y_k > 0$, which may not hold when s_k and y_k are defined by (18.13), even when the iterates are close to the solution.

To overcome this difficulty, we could *skip* the BFGS update if the condition

$$s_k^T y_k \geq \theta s_k^T B_k s_k \quad (18.14)$$

is not satisfied, where θ is a positive parameter (10^{-2} , say). This strategy may, on occasion, yield poor performance or even failure, so it cannot be regarded as adequate for general-purpose algorithms.

A more effective modification ensures that the update is always well defined by modifying the definition of y_k .

Procedure 18.2 (Damped BFGS Updating).

Given: symmetric and positive definite matrix B_k ;

Define s_k and y_k as in (18.13) and set

$$r_k = \theta_k y_k + (1 - \theta_k) B_k s_k,$$

where the scalar θ_k is defined as

$$\theta_k = \begin{cases} 1 & \text{if } s_k^T y_k \geq 0.2 s_k^T B_k s_k, \\ (0.8 s_k^T B_k s_k) / (s_k^T B_k s_k - s_k^T y_k) & \text{if } s_k^T y_k < 0.2 s_k^T B_k s_k; \end{cases} \quad (18.15)$$

Update B_k as follows:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_k r_k^T}{s_k^T r_k}. \quad (18.16)$$

The formula (18.16) is simply the standard BFGS update formula, with y_k replaced by r_k . It guarantees that B_{k+1} is positive definite, since it is easy to show that when $\theta_k \neq 1$ we have

$$s_k^T r_k = 0.2 s_k^T B_k s_k > 0. \quad (18.17)$$

To gain more insight into this strategy, note that the choice $\theta_k = 0$ gives $B_{k+1} = B_k$, while $\theta_k = 1$ gives the (possibly indefinite) matrix produced by the unmodified BFGS update. A value $\theta_k \in (0, 1)$ thus produces a matrix that interpolates the current approximation B_k and the one produced by the unmodified BFGS formula. The choice of θ_k ensures that the new approximation stays close enough to the current approximation B_k to ensure positive definiteness.

Damped BFGS updating often works well but it, too, can behave poorly on difficult problems. It still fails to address the underlying problem that the Lagrangian Hessian may not be positive definite. For this reason, SR1 updating may be more appropriate, and is indeed a good choice for trust-region SQP methods. An SR1 approximation to the Hessian of the Lagrangian is obtained by applying formula (6.24) with s_k and y_k defined by (18.13), using the safeguards described in Chapter 6. Line search methods cannot, however, accept indefinite Hessian approximations and would therefore need to modify the SR1 formula, possibly by adding a sufficiently large multiple of the identity matrix; see the discussion around (19.25).

All quasi-Newton approximations B_k discussed above are dense $n \times n$ matrices that can be expensive to store and manipulate in the large-scale case. Limited-memory updating is useful in this context and is often implemented in software packages. (See (19.29) for an implementation of limited-memory BFGS in a constrained optimization algorithm.)

REDUCED-HESSIAN QUASI-NEWTON APPROXIMATIONS

When we examine the KKT system (18.9) for the equality-constrained problem (18.1), we see that the part of the step p_k in the range space of A_k^T is completely determined by the second block row $A_k p_k = -c_k$. The Lagrangian Hessian $\nabla_{xx}^2 \mathcal{L}_k$ affects only the part of p_k in the orthogonal subspace, namely, the null space of A_k . It is reasonable, therefore, to consider quasi-Newton methods that find approximations to only that part of $\nabla_{xx}^2 \mathcal{L}_k$ that affects the component of p_k in the null space of A_k . In this section, we consider quasi-Newton methods based on these reduced-Hessian approximations. Our focus is on equality-constrained problems in this section, as existing SQP methods for the full problem (18.10) use reduced-Hessian approaches only after an equality-constrained subproblem has been generated.

To derive reduced-Hessian methods, we consider solution of the step equations (18.9) by means of the null space approach of Section 16.2. In that section, we defined matrices Y_k and Z_k whose columns span the range space of A_k^T and the null space of A_k , respectively. By writing

$$p_k = Y_k p_Y + Z_k p_Z, \quad (18.18)$$

and substituting into (18.9), we obtain the following system to be solved for p_Y and p_Z :

$$(A_k Y_k) p_Y = -c_k, \quad (18.19a)$$

$$(Z_k^T \nabla_{xx}^2 \mathcal{L}_k Z_k) p_Z = -Z_k^T \nabla_{xx}^2 \mathcal{L}_k Y_k p_Y - Z_k^T \nabla f_k. \quad (18.19b)$$

From the first block of equations in (18.9) we see that the Lagrange multipliers λ_{k+1} , which are sometimes called QP multipliers, can be obtained by solving

$$(A_k Y_k)^T \lambda_{k+1} = Y_k^T (\nabla f_k + \nabla_{xx}^2 \mathcal{L}_k p_k). \quad (18.20)$$

We can avoid computation of the Hessian $\nabla_{xx}^2 \mathcal{L}_k$ by introducing several approximations in the null-space approach. First, we delete the term involving p_k from the right-hand-side of (18.20), thereby decoupling the computations of p_k and λ_{k+1} and eliminating the need for $\nabla_{xx}^2 \mathcal{L}_k$ in this term. This simplification can be justified by observing that p_k converges to zero as we approach the solution, whereas ∇f_k normally does not. Therefore, the multipliers computed in this manner will be good estimates of the QP multipliers near the solution. More specifically, if we choose $Y_k = A_k^T$ (which is a valid choice for Y_k when A_k has full row rank; see (15.16)), we obtain

$$\hat{\lambda}_{k+1} = (A_k A_k^T)^{-1} A_k \nabla f_k. \quad (18.21)$$

These are called the *least-squares multipliers* because they can also be derived by solving the problem

$$\min_{\lambda} \|\nabla_x \mathcal{L}(x_k, \lambda)\|_2^2 = \|\nabla f_k - A_k^T \lambda\|_2^2. \quad (18.22)$$

This observation shows that the least-squares multipliers are useful even when the current iterate is far from the solution, because they seek to satisfy the first-order optimality condition in (18.3) as closely as possible. Conceptually, the use of least-squares multipliers transforms the SQP method from a primal-dual iteration in x and λ to a purely primal iteration in the x variable alone.

Our second simplification of the null-space approach is to remove the cross term $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Y_k p_Y$ in (18.19b), thereby yielding the simpler system

$$(Z_k^T \nabla_{xx}^2 \mathcal{L}_k Z_k) p_Z = -Z_k^T \nabla f_k. \quad (18.23)$$

This approach has the advantage that it needs to approximate only the matrix $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Z_k$, not the $(n - m) \times m$ cross-term matrix $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Y_k$, which is a relatively large matrix when $m \gg n - m$. Dropping the cross term is justified when $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Z_k$ is replaced by a quasi-Newton approximation because the normal component p_Y usually converges to zero faster than the tangential component p_Z , thereby making (18.23) a good approximation of (18.19b).

Having dispensed with the partial Hessian $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Y_k$, we discuss how to approximate the remaining part $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Z_k$. Suppose we have just taken a step $\alpha_k p_k = x_{k+1} - x_k = \alpha_k Z_k p_Z + \alpha_k Y_k p_Y$. By Taylor's theorem, writing $\nabla_{xx}^2 \mathcal{L}_{k+1} = \nabla_{xx}^2 \mathcal{L}(x_{k+1}, \lambda_{k+1})$, we have

$$\nabla_{xx}^2 \mathcal{L}_{k+1} \alpha_k p_k \approx \nabla_x \mathcal{L}(x_k + \alpha_k p_k, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}).$$

By premultiplying by Z_k^T , we have

$$\begin{aligned} & Z_k^T \nabla_{xx}^2 \mathcal{L}_{k+1} Z_k \alpha_k p_Z \\ & \approx -Z_k^T \nabla_{xx}^2 \mathcal{L}_{k+1} Y_k \alpha_k p_Y + Z_k^T [\nabla_x \mathcal{L}(x_k + \alpha_k p_k, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})]. \end{aligned} \quad (18.24)$$

If we drop the cross term $Z_k^T \nabla_{xx}^2 \mathcal{L}_{k+1} Y_k \alpha_k p_Y$ (using the rationale discussed earlier), we see that the secant equation for M_k can be defined by

$$M_{k+1} s_k = y_k, \quad (18.25)$$

where s_k and y_k are given by

$$s_k = \alpha_k p_Z, \quad y_k = Z_k^T [\nabla_x \mathcal{L}(x_k + \alpha_k p_k, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})]. \quad (18.26)$$

We then apply the BFGS or SR1 formulae, using these definitions for the correction vectors s_k and y_k , to define the new approximation M_{k+1} . An advantage of this reduced-Hessian approach, compared to full-Hessian quasi-Newton approximations, is that the reduced Hessian is much more likely to be positive definite, even when the current iterate is some distance from the solution. When using the BFGS formula, the safeguarding mechanism discussed above will be required less often in line search implementations.

MERIT FUNCTIONS

SQP methods often use a merit function to decide whether a trial step should be accepted. In line search methods, the merit function controls the size of the step; in trust-region methods it determines whether the step is accepted or rejected and whether the trust-region radius should be adjusted. A variety of merit functions have been used in SQP methods, including nonsmooth penalty functions and augmented Lagrangians. We limit our discussion to exact, nonsmooth merit functions typified by the ℓ_1 merit function discussed in Chapters 15 and 17.

For the purpose of step computation and evaluation of a merit function, inequality constraints $c(x) \geq 0$ are often converted to the form

$$\bar{c}(x, s) = c(x) - s = 0,$$

where $s \geq 0$ is a vector of slacks. (The condition $s \geq 0$ is typically not monitored by the merit function.) Therefore, in the discussion that follows we assume that all constraints are in the form of equalities, and we focus our attention on problem (18.1).

The ℓ_1 merit function for (18.1) takes the form

$$\phi_1(x; \mu) = f(x) + \mu \|c(x)\|_1. \quad (18.27)$$

In a line search method, a step $\alpha_k p_k$ will be accepted if the following sufficient decrease condition holds:

$$\phi_1(x_k + \alpha_k p_k; \mu_k) \leq \phi_1(x_k, \mu_k) + \eta \alpha_k D(\phi_1(x_k; \mu); p_k), \quad \eta \in (0, 1), \quad (18.28)$$

where $D(\phi_1(x_k; \mu); p_k)$ denotes the directional derivative of ϕ_1 in the direction p_k . This requirement is analogous to the Armijo condition (3.4) for unconstrained optimization provided that p_k is a descent direction, that is, $D(\phi_1(x_k; \mu); p_k) < 0$. This descent condition holds if the penalty parameter μ is chosen sufficiently large, as we show in the following result.

Theorem 18.2.

Let p_k and λ_{k+1} be generated by the SQP iteration (18.9). Then the directional derivative of ϕ_1 in the direction p_k satisfies

$$D(\phi_1(x_k; \mu); p_k) = \nabla f_k^T p_k - \mu \|c_k\|_1. \quad (18.29)$$

Moreover, we have that

$$D(\phi_1(x_k; \mu); p_k) \leq -p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k - (\mu - \|\lambda_{k+1}\|_\infty) \|c_k\|_1. \quad (18.30)$$

PROOF. By applying Taylor's theorem (see (2.5)) to f and c_i , $i = 1, 2, \dots, m$, we obtain

$$\begin{aligned} \phi_1(x_k + \alpha p; \mu) - \phi_1(x_k; \mu) &= f(x_k + \alpha p) - f_k + \mu \|c(x_k + \alpha p)\|_1 - \mu \|c_k\|_1 \\ &\leq \alpha \nabla f_k^T p + \gamma \alpha^2 \|p\|^2 + \mu \|c_k + \alpha A_k p\|_1 - \mu \|c_k\|_1, \end{aligned}$$

where the positive constant γ bounds the second-derivative terms in f and c . If $p = p_k$ is given by (18.9), we have that $A_k p_k = -c_k$, so for $\alpha \leq 1$ we have that

$$\phi_1(x_k + \alpha p_k; \mu) - \phi_1(x_k; \mu) \leq \alpha [\nabla f_k^T p_k - \mu \|c_k\|_1] + \alpha^2 \gamma \|p_k\|^2.$$

By arguing similarly, we also obtain the following lower bound:

$$\phi_1(x_k + \alpha p_k; \mu) - \phi_1(x_k; \mu) \geq \alpha [\nabla f_k^T p_k - \mu \|c_k\|_1] - \alpha^2 \gamma \|p_k\|^2.$$

Taking limits, we conclude that the directional derivative of ϕ_1 in the direction p_k is given by

$$D(\phi_1(x_k; \mu); p_k) = \nabla f_k^T p_k - \mu \|c_k\|_1, \quad (18.31)$$

which proves (18.29). The fact that p_k satisfies the first equation in (18.9) implies that

$$D(\phi_1(x_k; \mu); p_k) = -p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k + p_k^T A_k^T \lambda_{k+1} - \mu \|c_k\|_1.$$

From the second equation in (18.9), we can replace the term $p_k^T A_k^T \lambda_{k+1}$ in this expression by $-c_k^T \lambda_{k+1}$. By making this substitution in the expression above and invoking the inequality

$$-c_k^T \lambda_{k+1} \leq \|c_k\|_1 \|\lambda_{k+1}\|_\infty,$$

we obtain (18.30). □

It follows from (18.30) that p_k will be a descent direction for ϕ_1 if $p_k \neq 0$, $\nabla_{xx}^2 \mathcal{L}_k$ is positive definite and

$$\mu > \|\lambda_{k+1}\|_\infty. \quad (18.32)$$

(A more detailed analysis shows that this assumption on $\nabla_{xx}^2 \mathcal{L}_k$ can be relaxed; we need only the reduced Hessian $Z_k^T \nabla_{xx}^2 \mathcal{L}_k Z_k$ to be positive definite.)

One strategy for choosing the new value of the penalty parameter μ in $\phi_1(x; \mu)$ at every iteration is to increase the previous value, if necessary, so as to satisfy (18.32), with some margin. It has been observed, however, that this strategy may select inappropriate values of μ and often interferes with the progress of the iteration.

An alternative approach, based on (18.29), is to require that the directional derivative be sufficiently negative in the sense that

$$D(\phi_1(x_k; \mu); p_k) = \nabla f_k^T p_k - \mu \|c_k\|_1 \leq -\rho \mu \|c_k\|_1,$$

for some $\rho \in (0, 1)$. This inequality holds if

$$\mu \geq \frac{\nabla f_k^T p_k}{(1 - \rho) \|c_k\|_1}. \quad (18.33)$$

This choice is not dependent on the Lagrange multipliers and performs adequately in practice.

A more effective strategy for choosing μ , which is appropriate both in the line search and trust-region contexts, considers the effect of the step on a model of the merit function. We define a (piecewise) quadratic model of ϕ_1 by

$$q_\mu(p) = f_k + \nabla f_k^T p + \frac{\sigma}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p + \mu m(p), \quad (18.34)$$

where

$$m(p) = \|c_k + A_k p\|_1,$$

and σ is a parameter to be defined below. After computing a step p_k , we choose the penalty parameter μ large enough that

$$q_\mu(0) - q_\mu(p_k) \geq \rho \mu [m(0) - m(p_k)], \quad (18.35)$$

for some parameter $\rho \in (0, 1)$. It follows from (18.34) and (18.7b) that inequality (18.35) is satisfied for

$$\mu \geq \frac{\nabla f_k^T p_k + (\sigma/2) p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k}{(1 - \rho) \|c_k\|_1}. \quad (18.36)$$

If the value of μ from the previous iteration of the SQP method satisfies (18.36), it is left unchanged. Otherwise, μ is increased so that it satisfies this inequality with some margin.

The constant σ is used to handle the case in which the Hessian $\nabla_{xx}^2 \mathcal{L}_k$ is not positive definite. We define σ as

$$\sigma = \begin{cases} 1 & \text{if } p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (18.37)$$

It is easy to verify that, if μ satisfies (18.36), this choice of σ ensures that $D(\phi_1(x_k; \mu); p_k) \leq -\rho \mu \|c_k\|_1$, so that p_k is a descent direction for the merit function ϕ_1 . This conclusion is not always valid if $\sigma = 1$ and $p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k < 0$. By comparing (18.33) and (18.36) we see that, when $\sigma > 0$, the strategy based on (18.35) selects a larger penalty parameter, thus placing more weight on the reduction of the constraints. This property is advantageous if the step p_k decreases the constraints but increases the objective, for in this case the step has a better chance of being accepted by the merit function.

SECOND-ORDER CORRECTION

In Chapter 15, we showed by means of Example 15.4 that many merit functions can impede progress of an optimization algorithm, a phenomenon known as the Maratos effect. We now show that the step analyzed in that example is, in fact, produced by an SQP method.

□ **EXAMPLE 18.1** (EXAMPLE 15.4, REVISITED)

Consider problem (15.34). At the iterate $x_k = (\cos \theta, \sin \theta)^T$, let us compute a search direction p_k by solving the SQP subproblem (18.7) with $\nabla_{xx}^2 \mathcal{L}_k$ replaced by $\nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) = I$. Since

$$f_k = -\cos \theta, \quad \nabla f_k = \begin{bmatrix} 4 \cos \theta - 1 \\ 4 \sin \theta \end{bmatrix}, \quad A_k^T = \begin{bmatrix} 2 \cos \theta \\ 2 \sin \theta \end{bmatrix},$$

the quadratic subproblem (18.7) takes the form

$$\begin{aligned} \min_p \quad & (4 \cos \theta - 1)p_1 + 4 \sin \theta p_2 + \frac{1}{2}p_1^2 + \frac{1}{2}p_2^2 \\ \text{subject to} \quad & p_2 + \cot \theta p_1 = 0. \end{aligned}$$

By solving this subproblem, we obtain the direction

$$p_k = \begin{bmatrix} \sin^2 \theta \\ -\sin \theta \cos \theta \end{bmatrix}, \quad (18.38)$$

which coincides with (15.35). □

We mentioned in Section 15.4 that the difficulties associated with the Maratos effect can be overcome by means of a *second-order correction*. There are various ways of applying this technique; we describe one possible implementation next.

Suppose that the SQP method has computed a step p_k from (18.11). If this step yields an increase in the merit function ϕ_1 , a possible cause is that our linear approximations to the constraints are not sufficiently accurate. To overcome this deficiency, we could re-solve (18.11) with the linear terms $c_i(x_k) + \nabla c_i(x_k)^T p$ replaced by quadratic approximations,

$$c_i(x_k) + \nabla c_i(x_k)^T p + \frac{1}{2} p^T \nabla^2 c_i(x_k) p. \quad (18.39)$$

However, even if the Hessians of the constraints are individually available, the resulting quadratically constrained subproblem may be too difficult to solve. Instead, we evaluate the constraint values at the new point $x_k + p_k$ and make use of the following approximations. By Taylor's theorem, we have

$$c_i(x_k + p_k) \approx c_i(x_k) + \nabla c_i(x_k)^T p_k + \frac{1}{2} p_k^T \nabla^2 c_i(x_k) p_k. \quad (18.40)$$

Assuming that the (still unknown) second-order step p will not be too different from p_k , we can approximate the last term in (18.39) as follows:

$$p^T \nabla^2 c_i(x_k) p = p_k^T \nabla^2 c_i(x_k) p_k. \quad (18.41)$$

By making this substitution in (18.39) and using (18.40), we obtain the second-order correction subproblem

$$\begin{aligned} \min_p \quad & \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ \text{subject to} \quad & \nabla c_i(x_k)^T p + d_i = 0, \quad i \in \mathcal{E}, \\ & \nabla c_i(x_k)^T p + d_i \geq 0, \quad i \in \mathcal{I}, \end{aligned}$$

where

$$d_i = c_i(x_k + p_k) - \nabla c_i(x_k)^T p_k, \quad i \in \mathcal{E} \cup \mathcal{I}.$$

The second-order correction step requires evaluation of the constraints $c_i(x_k + p_k)$ for $i \in \mathcal{E} \cup \mathcal{I}$, and therefore it is preferable not to apply it every time the merit function increases. One strategy is to use it only if the increase in the merit function is accompanied by an increase in the constraint norm.

It can be shown that when the step p_k is generated by the SQP method (18.11) then, near a solution satisfying second-order sufficient conditions, the algorithm above takes either the full step p_k or the corrected step $p_k + \hat{p}_k$. The merit function does not interfere with the iteration, so superlinear convergence is attained, as in the local algorithm.

18.4 A PRACTICAL LINE SEARCH SQP METHOD

From the discussion in the previous section, we can see that there is a wide variety of line search SQP methods that differ in the way the Hessian approximation is computed, in the step acceptance mechanism, and in other algorithmic features. We now incorporate some of these ideas into a concrete, practical SQP algorithm for solving the nonlinear programming problem (18.10). To keep the description simple, we will not include a mechanism such as (18.12) to ensure the feasibility of the subproblem, or a second-order correction step. Rather, the search direction is obtained simply by solving the subproblem (18.11). We also assume that the quadratic program (18.11) is convex, so that we can solve it by means of the active-set method for quadratic programming (Algorithm 16.3) described in Chapter 16.

Algorithm 18.3 (Line Search SQP Algorithm).

Choose parameters $\eta \in (0, 0.5)$, $\tau \in (0, 1)$, and an initial pair (x_0, λ_0) ;

Evaluate $f_0, \nabla f_0, c_0, A_0$;

If a quasi-Newton approximation is used, choose an initial $n \times n$ symmetric positive definite Hessian approximation B_0 , otherwise compute $\nabla_{xx}^2 \mathcal{L}_0$;

repeat until a convergence test is satisfied

 Compute p_k by solving (18.11); let $\hat{\lambda}$ be the corresponding multiplier;

 Set $p_\lambda \leftarrow \hat{\lambda} - \lambda_k$;

 Choose μ_k to satisfy (18.36) with $\sigma = 1$;

 Set $\alpha_k \leftarrow 1$;

while $\phi_1(x_k + \alpha_k p_k; \mu_k) > \phi_1(x_k; \mu_k) + \eta \alpha_k D_1(\phi(x_k; \mu_k) p_k)$

 Reset $\alpha_k \leftarrow \tau_\alpha \alpha_k$ for some $\tau_\alpha \in (0, \tau]$;

end (while)

 Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k p_\lambda$;

 Evaluate $f_{k+1}, \nabla f_{k+1}, c_{k+1}, A_{k+1}$, (and possibly $\nabla_{xx}^2 \mathcal{L}_{k+1}$);

 If a quasi-Newton approximation is used, set

$s_k \leftarrow \alpha_k p_k$ and $y_k \leftarrow \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})$,

 and obtain B_{k+1} by updating B_k using a quasi-Newton formula;

end (repeat)

We can achieve significant savings in the solution of the quadratic subproblem by warm-start procedures. For example, we can initialize the working set for each QP subproblem to be the final active set from the previous SQP iteration.

We have not given particulars of the quasi-Newton approximation in Algorithm 18.3. We could use, for example, a limited-memory BFGS approach that is suitable for large-scale problems. If we use an exact Hessian $\nabla_{xx}^2 \mathcal{L}_k$, we assume that it is modified as necessary to be positive definite on the null space of the equality constraints.

Instead of a merit function, we could employ a filter (see Section 15.4) in the inner “while” loop to determine the steplength α_k . As discussed in Section 15.4, a feasibility restoration phase is invoked if a trial steplength generated by the backtracking line search is

smaller than a given threshold. Regardless of whether a merit function or filter are used, a mechanism such as second-order correction can be incorporated to overcome the Maratos effect.

18.5 TRUST-REGION SQP METHODS

Trust-region SQP methods have several attractive properties. Among them are the facts that they do not require the Hessian matrix $\nabla_{xx}^2 \mathcal{L}_k$ in (18.11) to be positive definite, they control the quality of the steps even in the presence of Hessian and Jacobian singularities, and they provide a mechanism for enforcing global convergence. Some implementations follow an IQP approach and solve an inequality-constrained subproblem, while others follow an EQP approach.

The simplest way to formulate a trust-region SQP method is to add a trust-region constraint to subproblem (18.11), as follows:

$$\min_p f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.43a)$$

$$\text{subject to } \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \quad (18.43b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}, \quad (18.43c)$$

$$\|p\| \leq \Delta_k. \quad (18.43d)$$

Even if the constraints (18.43b), (18.43c) are compatible, this problem may not always have a solution because of the trust-region constraint (18.43d). We illustrate this fact in Figure 18.1 for a problem that contains only one equality constraint whose linearization is represented by the solid line. In this example, any step p that satisfies the linearized constraint must lie outside the trust region, which is indicated by the circle of radius Δ_k . As we see from this example, a consistent system of equalities and inequalities may not have a solution if we restrict the norm of the solution.

To resolve the possible conflict between the linear constraints (18.43b), (18.43c) and the trust-region constraint (18.43d), it is not appropriate simply to increase Δ_k until the set of steps p satisfying the linear constraints intersects the trust region. This approach would defeat the purpose of using the trust region in the first place as a way to define a region within which we trust the model (18.43a)–(18.43c) to accurately reflect the behavior of the objective and constraint functions. Analytically, it would harm the convergence properties of the algorithm.

A more appropriate viewpoint is that there is no reason to satisfy the linearized constraints exactly at every step; rather, we should aim to improve the feasibility of these constraints at each step and to satisfy them exactly only if the trust-region constraint permits it. This point of view is the basis of the three classes of methods discussed in this section: relaxation methods, penalty methods, and filter methods.

the correction vectors (18.26), and the initial approximation M_0 is symmetric and positive definite. If we make the assumption that the null space bases Z_k used to define the correction vectors (18.26) vary smoothly, then we can apply Theorem 18.7 to show that x_k converges two-step superlinearly.

18.8 PERSPECTIVES AND SOFTWARE

SQP methods are most efficient if the number of active constraints is nearly as large as the number of variables, that is, if the number of free variables is relatively small. They require few evaluations of the functions, in comparison with augmented Lagrangian methods, and can be more robust on badly scaled problems than the nonlinear interior-point methods described in the next chapter. It is not known at present whether the IQP or EQP approach will prove to be more effective for large problems. Current research focuses on widening the class of problems that can be solved with SQP and SLQP approaches.

Two established SQP software packages are SNOPT [128] and FILTERSQP [105]. The former code follows a line search approach, while the latter implements a trust-region strategy using a filter for step acceptance. The SLQP approach of Section 18.5 is implemented in KNITRO/ACTIVE [49]. All three packages include mechanisms to ensure that the subproblems are always feasible and to guard against rank-deficient constraint Jacobians. SNOPT uses the penalty (or elastic) mode (18.12), which is invoked if the SQP subproblem is infeasible or if the Lagrange multiplier estimates become very large in norm. FILTERSQP includes a feasibility restoration phase that, in addition to promoting convergence, provides rapid identification of convergence to infeasible points. KNITRO/ACTIVE implements a penalty method using the update strategy of Algorithm 18.5.

There is no established implementation of the $S\ell_1$ QP approach, but prototype implementations have shown promise. The CONOPT [9] package implements a generalized reduced gradient method as well as an SQP method.

Quasi-Newton approximations to the Hessian of the Lagrangian $\nabla_{xx}^2 \mathcal{L}_k$ are often used in practice. BFGS updating is generally less effective for constrained problems than in the unconstrained case because of the requirement of maintaining a positive definite approximation to an underlying matrix that often does not have this property. Nevertheless, the BFGS and limited-memory BFGS approximations implemented in SNOPT and KNITRO perform adequately in practice. KNITRO also offers an SR1 option that may be more effective than the BFGS option, but the question of how best to implement full quasi-Newton approximations for constrained optimization requires further investigation. The RSQP package [13] implements an SQP method that maintains a quasi-Newton approximation to the reduced Hessian.

The Maratos effect, if left unattended, can significantly slow optimization algorithms that use nonsmooth merit functions or filters. However, selective application of second-order correction steps adequately resolves the difficulties in practice.

Trust-region implementations of the gradient projection method include TRON [192] and LANCELOT [72]. Both codes use a conjugate gradient iteration to perform the subspace minimization and apply an incomplete Cholesky preconditioner. Gradient projection methods in which the Hessian approximation is defined by limited-memory BFGS updating are implemented in LBFGS-B [322] and BLMVM [17]. The properties of limited-memory BFGS matrices can be exploited to perform the projected gradient search and subspace minimization efficiently. SPG [23] implements the gradient projection method using a nonmonotone line search.

NOTES AND REFERENCES

SQP methods were first proposed in 1963 by Wilson [306] and were developed in the 1970s by Garcia-Palomares and Mangasarian [117], Han [163, 164], and Powell [247, 250, 249], among others. Trust-region variants are studied by Vardi [295], Celis, Dennis, and Tapia [56], and Byrd, Schnabel, and Shultz [55]. See Boggs and Tolle [33] and Gould, Orban, and Toint [147] for literature surveys.


The SLQP approach was proposed by Fletcher and Sainz de la Maza [108] and was further developed by Chin and Fletcher [59] and Byrd et al. [49]. The latter paper discusses how to update the LP trust region and many other details of implementation. The technique for updating the penalty parameter implemented in Algorithm 18.5 is discussed in [49, 47]. The Sl_1QP method was proposed by Fletcher; see [101] for a complete discussion of this method.


Some analysis shows that several—but not all—of the good properties of BFGS updating are preserved by damped BFGS updating. Numerical experiments exposing the weakness of the approach are reported by Powell [254]. Second-order correction strategies were proposed by Coleman and Conn [65], Fletcher [100], Gabay [116], and Mayne and Polak [204]. The watchdog technique was proposed by Chamberlain et al. [57] and other nonmonotone strategies are described by Bonnans et al. [36]. For a comprehensive discussion of second-order correction and nonmonotone techniques, see the book by Conn, Gould, and Toint [74].


Two filter SQP algorithms are described by Fletcher and Leyffer [105] and Fletcher, Leyffer, and Toint [106]. It is not yet known whether the filter strategy has advantages over merit functions. Both approaches are undergoing development and improved implementations can be expected in the future. Theorem 18.3 is proved by Powell [252] and Theorem 18.5 by Boggs, Tolle, and Wang [34].



EXERCISES

 **18.1** Show that in the quadratic program (18.7) we can replace the linear term $\nabla f_k^T p$ by $\nabla_x \mathcal{L}(x_k, \lambda_k)^T p$ without changing the solution.

 **18.2** Prove Theorem 18.4.

 **18.3** Write a program that implements Algorithm 18.1. Use it to solve the problem


$$\min e^{x_1 x_2 x_3 x_4 x_5} - \frac{1}{2}(x_1^3 + x_2^3 + 1)^2 \quad (18.66)$$


$$\text{subject to } x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0, \quad (18.67)$$


$$x_2 x_3 - 5 x_4 x_5 = 0, \quad (18.68)$$


$$x_1^3 + x_2^3 + 1 = 0. \quad (18.69)$$


Use the starting point $x_0 = (-1.71, 1.59, 1.82, -0.763, -0.763)^T$. The solution is $x^* = (-1.8, 1.7, 1.9, -0.8, -0.8)^T$.


 **18.4** Show that the damped BFGS updating satisfies (18.17).

 **18.5** Consider the constraint $x_1^2 + x_2^2 = 1$. Write the linearized constraints (18.7b) at the following points: $(0, 0)^T$, $(0, 1)^T$, $(0.1, 0.02)^T$, $-(0.1, 0.02)^T$.

 **18.6** Prove Theorem 18.2 for the case in which the merit function is given by $\phi(x; \mu) = f(x) + \mu \|c(x)\|_q$, where $q > 0$. Use this lemma to show that the condition that ensures descent is given by $\mu > \|\lambda_{k+1}\|_r$, where $r > 0$ satisfies $r^{-1} + q^{-1} = 1$.

 **18.7** Write a program that implements the reduced-Hessian method given by (18.18), (18.19a), (18.21), (18.23). Use your program to solve the problem given in Exercise 18.3.

 **18.8** Show that the constraints (18.50b)–(18.50e) are always consistent.

 **18.9** Show that the feasibility problem (18.45a)–(18.45b) always has a solution v_k lying in the range space of A_k^T . Hint: First show that if the trust-region constraint (18.45b) is active, v_k lies in the range space of A_k^T . Next, show that if the trust region is inactive, the minimum-norm solution of (18.45a) lies in the range space of A_k^T .

CHAPTER *19*

Interior-Point Methods for Nonlinear Programming

Interior-point (or barrier) methods have proved to be as successful for nonlinear optimization as for linear programming, and together with active-set SQP methods, they are currently considered the most powerful algorithms for large-scale nonlinear programming. Some of the key ideas, such as primal-dual steps, carry over directly from the linear programming case, but several important new challenges arise. These include the treatment of nonconvexity, the strategy for updating the barrier parameter in the presence of nonlinearities, and the need to ensure progress toward the solution. In this chapter we describe two classes of interior-point methods that have proved effective in practice.

APPENDIX

A

Background Material

A.1 ELEMENTS OF LINEAR ALGEBRA

VECTORS AND MATRICES

In this book we work exclusively with vectors and matrices whose components are real numbers. Vectors are usually denoted by lowercase roman characters, and matrices by uppercase roman characters. The space of real vectors of length n is denoted by \mathbb{R}^n , while the space of real $m \times n$ matrices is denoted by $\mathbb{R}^{m \times n}$.

Given a vector $x \in \mathbb{R}^n$, we use x_i to denote its i th component. We invariably assume that x is a *column* vector, that is,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

The transpose of x , denoted by x^T is the row vector

$$x^T = [x_1 \quad x_2 \quad \cdots \quad x_n],$$

and is often also written with parentheses as $x = (x_1, x_2, \dots, x_n)$. We write $x \geq 0$ to indicate componentwise nonnegativity, that is, $x_i \geq 0$ for all $i = 1, 2, \dots, n$, while $x > 0$ indicates that $x_i > 0$ for all $i = 1, 2, \dots, n$.

Given $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$, the standard inner product is $x^T y = \sum_{i=1}^n x_i y_i$.

Given a matrix $A \in \mathbb{R}^{m \times n}$, we specify its components by double subscripts as A_{ij} , for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. The transpose of A , denoted by A^T , is the $n \times m$ matrix whose components are A_{ji} . The matrix A is said to be *square* if $m = n$. A square matrix is *symmetric* if $A = A^T$.

A square matrix A is *positive definite* if there is a positive scalar α such that

$$x^T A x \geq \alpha x^T x, \quad \text{for all } x \in \mathbb{R}^n. \quad (\text{A.1})$$

It is *positive semidefinite* if

$$x^T A x \geq 0, \quad \text{for all } x \in \mathbb{R}^n.$$

We can recognize that a symmetric matrix is positive definite by computing its eigenvalues and verifying that they are all positive, or by performing a Cholesky factorization. Both techniques are discussed further in later sections.

The diagonal of the matrix $A \in \mathbb{R}^{m \times n}$ consists of the elements A_{ii} , for $i = 1, 2, \dots, \min(m, n)$. The matrix $A \in \mathbb{R}^{m \times n}$ is *lower triangular* if $A_{ij} = 0$ whenever $i < j$; that is, all elements above the diagonal are zero. It is *upper triangular* if $A_{ij} = 0$ whenever $i > j$; that is, all elements below the diagonal are zero. A is *diagonal* if $A_{ij} = 0$ whenever $i \neq j$.

The identity matrix, denoted by I , is the square diagonal matrix whose diagonal elements are all 1.

A square $n \times n$ matrix A is *nonsingular* if for any vector $b \in \mathbb{R}^n$, there exists $x \in \mathbb{R}^n$ such that $Ax = b$. For nonsingular matrices A , there exists a unique $n \times n$ matrix B such that $AB = BA = I$. We denote B by A^{-1} and call it the *inverse* of A . It is not hard to show that the inverse of A^T is the transpose of A^{-1} .

A square matrix Q is *orthogonal* if it has the property that $QQ^T = Q^T Q = I$. In other words, the inverse of an orthogonal matrix is its transpose.

NORMS

For a vector $x \in \mathbb{R}^n$, we define the following norms:

$$\|x\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^n |x_i|, \quad (\text{A.2a})$$

$$\|x\|_2 \stackrel{\text{def}}{=} \left(\sum_{i=1}^n x_i^2 \right)^{1/2} = (x^T x)^{1/2}, \quad (\text{A.2b})$$

$$\|x\|_\infty \stackrel{\text{def}}{=} \max_{i=1, \dots, n} |x_i|. \quad (\text{A.2c})$$

The norm $\|\cdot\|_2$ is often called the *Euclidean norm*. We sometimes refer to $\|\cdot\|_1$ as the ℓ_1 norm and to $\|\cdot\|_\infty$ as the ℓ_∞ norm. All these norms measure the length of the vector in some sense, and they are equivalent in the sense that each one is bounded above and below by a multiple of the other. To be precise, we have for all $x \in \mathbb{R}^n$ that

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty, \quad \|x\|_\infty \leq \|x\|_1 \leq n \|x\|_\infty, \quad (\text{A.3})$$

and so on. In general, a norm is any mapping $\|\cdot\|$ from \mathbb{R}^n to the nonnegative real numbers that satisfies the following properties:

$$\|x + z\| \leq \|x\| + \|z\|, \quad \text{for all } x, z \in \mathbb{R}^n; \quad (\text{A.4a})$$

$$\|x\| = 0 \Rightarrow x = 0; \quad (\text{A.4b})$$

$$\|\alpha x\| = |\alpha| \|x\|, \quad \text{for all } \alpha \in \mathbb{R} \text{ and } x \in \mathbb{R}^n. \quad (\text{A.4c})$$

Equality holds in (A.4a) if and only if one of the vectors x and z is a nonnegative scalar multiple of the other.

Another interesting property that holds for the Euclidean norm $\|\cdot\| = \|\cdot\|_2$ is the Cauchy–Schwarz inequality, which states that

$$|x^T z| \leq \|x\| \|z\|, \quad (\text{A.5})$$

with equality if and only if one of these vectors is a nonnegative multiple of the other. We can prove this result as follows:

$$0 \leq \|\alpha x + z\|^2 = \alpha^2 \|x\|^2 + 2\alpha x^T z + \|z\|^2.$$

The right-hand-side is a convex function of α , and it satisfies the required nonnegativity property only if there exist fewer than 2 distinct real roots, that is,

$$(2x^T z)^2 \leq 4\|x\|^2 \|z\|^2,$$

proving (A.5). Equality occurs when the quadratic α has exactly one real root (that is, $|x^T z| = \|x\| \|z\|$) and when $\alpha x + z = 0$ for some α , as claimed.

Any norm $\|\cdot\|$ has a *dual norm* $\|\cdot\|_D$ defined by

$$\|x\|_D = \max_{\|y\|=1} x^T y. \quad (\text{A.6})$$

It is easy to show that the norms $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are duals of each other, and that the Euclidean norm is its own dual.

We can derive definitions for certain matrix norms from these vector norm definitions. If we let $\|\cdot\|$ be generic notation for the three norms listed in (A.2), we define the corresponding matrix norm as

$$\|A\| \stackrel{\text{def}}{=} \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}. \quad (\text{A.7})$$

The matrix norms defined in this way are said to be *consistent* with the vector norms (A.2). Explicit formulae for these norms are as follows:

$$\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^m |A_{ij}|, \quad (\text{A.8a})$$

$$\|A\|_2 = \text{largest eigenvalue of } (A^T A)^{1/2}, \quad (\text{A.8b})$$

$$\|A\|_\infty = \max_{i=1,\dots,m} \sum_{j=1}^n |A_{ij}|. \quad (\text{A.8c})$$

The Frobenius norm $\|A\|_F$ of the matrix A is defined by

$$\|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2 \right)^{1/2}. \quad (\text{A.9})$$

This norm is useful for many purposes, but it is not consistent with any vector norm. Once again, these various matrix norms are equivalent with each other in a sense similar to (A.3).

For the Euclidean norm $\|\cdot\| = \|\cdot\|_2$, the following property holds:

$$\|AB\| \leq \|A\| \|B\|, \quad (\text{A.10})$$

for all matrices A and B with consistent dimensions.

The *condition number* of a nonsingular matrix is defined as

$$\kappa(A) = \|A\| \|A^{-1}\|, \quad (\text{A.11})$$

where any matrix norm can be used in the definition. Different norms can be used by the use of a subscript— $\kappa_1(\cdot)$, $\kappa_2(\cdot)$, and $\kappa_\infty(\cdot)$, respectively—with κ denoting κ_2 by default.

Norms also have a meaning for scalar, vector, and matrix-valued functions that are defined on a particular domain. In these cases, we can define Hilbert spaces of functions for which the inner product and norm are defined in terms of an integral over the domain. We omit details, since all the development of this book takes place in the space \mathbb{R}^n , though many of the algorithms can be extended to more general Hilbert spaces. However, we mention for purposes of the analysis of Newton-like methods that the following inequality holds for functions of the type that we consider in this book:

$$\left\| \int_a^b F(t) dt \right\| \leq \int_a^b \|F(t)\| dt, \quad (\text{A.12})$$

where F is a continuous scalar-, vector-, or matrix-valued function on the interval $[a, b]$.

SUBSPACES

Given the Euclidean space \mathbb{R}^n , the subset $\mathcal{S} \subset \mathbb{R}^n$ is a *subspace of \mathbb{R}^n* if the following property holds: If x and y are any two elements of \mathcal{S} , then

$$\alpha x + \beta y \in \mathcal{S}, \quad \text{for all } \alpha, \beta \in \mathbb{R}.$$

For instance, \mathcal{S} is a subspace of \mathbb{R}^2 if it consists of (i) the whole space \mathbb{R}^2 ; (ii) any line passing through the origin; (iii) the origin alone; or (iv) the empty set.

Given any set of vectors $a_i \in \mathbb{R}^n$, $i = 1, 2, \dots, m$, the set

$$\mathcal{S} = \{w \in \mathbb{R}^n \mid a_i^T w = 0, i = 1, 2, \dots, m\} \quad (\text{A.13})$$

is a subspace. However, the set

$$\{w \in \mathbb{R}^n \mid a_i^T w \geq 0, i = 1, 2, \dots, m\} \quad (\text{A.14})$$

is not in general a subspace. For example, if we have $n = 2$, $m = 1$, and $a_1 = (1, 0)^T$, this set would consist of all vectors $(w_1, w_2)^T$ with $w_1 \geq 0$, but then given two vectors $x = (1, 0)^T$ and $y = (2, 3)$ in this set, it is easy to choose multiples α and β such that $\alpha x + \beta y$ has a negative first component, and so lies outside the set.

Sets of the forms (A.13) and (A.14) arise in the discussion of second-order optimality conditions for constrained optimization.

A set of vectors $\{s_1, s_2, \dots, s_m\}$ in \mathbb{R}^n is called a *linearly independent set* if there are no real numbers $\alpha_1, \alpha_2, \dots, \alpha_m$ such that

$$\alpha_1 s_1 + \alpha_2 s_2 + \dots + \alpha_m s_m = 0,$$

unless we make the trivial choice $\alpha_1 = \alpha_2 = \cdots = \alpha_m = 0$. Another way to define linear independence is to say that none of the vectors s_1, s_2, \dots, s_m can be written as a linear combination of the other vectors in this set. If in fact we have $s_i \in \mathcal{S}$ for all $i = 1, 2, \dots, m$, we say that $\{s_1, s_2, \dots, s_m\}$ is a *spanning set* for \mathcal{S} if *any* vector $s \in \mathcal{S}$ can be written as

$$s = \alpha_1 s_1 + \alpha_2 s_2 + \cdots + \alpha_m s_m,$$

for some particular choice of the coefficients $\alpha_1, \alpha_2, \dots, \alpha_m$.

If the vectors s_1, s_2, \dots, s_m are both linearly independent and a spanning set for \mathcal{S} , we call them a *basis* of \mathcal{S} . In this case, m (the number of elements in the basis) is referred to as the *dimension* of \mathcal{S} , and denoted by $\dim(\mathcal{S})$. Note that there are many ways to choose a basis of \mathcal{S} in general, but that all bases contain the same number of vectors.

If A is any real matrix, the *null space* is the subspace

$$\text{Null}(A) = \{w \mid Aw = 0\},$$

while the *range space* is

$$\text{Range}(A) = \{w \mid w = Av \text{ for some vector } v\}.$$

The *fundamental theorem of linear algebra* states that

$$\text{Null}(A) \oplus \text{Range}(A^T) = \mathbb{R}^n,$$

where n is the number of columns in A . (Here, “ \oplus ” denotes the direct sum of two sets: $\mathcal{A} \oplus \mathcal{B} = \{x + y \mid x \in \mathcal{A}, y \in \mathcal{B}\}$.)

When A is square ($n \times n$) and nonsingular, we have $\text{Null}A = \text{Null}A^T = \{0\}$ and $\text{Range}A = \text{Range}A^T = \mathbb{R}^n$. In this case, the columns of A form a basis of \mathbb{R}^n , as do the columns of A^T .

EIGENVALUES, EIGENVECTORS, AND THE SINGULAR-VALUE DECOMPOSITION

A scalar value λ is an *eigenvalue* of the $n \times n$ matrix A if there is a nonzero vector q such that

$$Aq = \lambda q.$$

The vector q is called an *eigenvector* of A . The matrix A is nonsingular if none of its eigenvalues are zero. The eigenvalues of symmetric matrices are all real numbers, while nonsymmetric matrices may have imaginary eigenvalues. If the matrix is positive definite as well as symmetric, its eigenvalues are all *positive* real numbers.

All matrices A (not necessarily square) can be decomposed as a product of three matrices with special properties. When $A \in \mathbb{R}^{m \times n}$ with $m > n$, (that is, A has more rows than columns), this *singular-value decomposition* (SVD) has the form

$$A = U \begin{bmatrix} S \\ 0 \end{bmatrix} V^T, \quad (\text{A.15})$$

where U and V are orthogonal matrices of dimension $m \times m$ and $n \times n$, respectively, and S is an $n \times n$ diagonal matrix with diagonal elements σ_i , $i = 1, 2, \dots, n$, that satisfy

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0.$$

These diagonal values are called the singular values of A . We can define the condition number (A.11) of the $m \times n$ (possibly nonsquare) matrix A to be σ_1/σ_n . (This definition is identical to $\kappa_2(A)$ when A happens to be square and nonsingular.)

When $m \leq n$ (the number of columns is at least equal to the number of rows), the SVD has the form

$$A = U \begin{bmatrix} S & 0 \end{bmatrix} V^T,$$

where again U and V are orthogonal of dimension $m \times m$ and $n \times n$, respectively, while S is $m \times m$ diagonal with nonnegative diagonal elements $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_m$.

When A is symmetric, its n real eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$ and their associated eigenvectors q_1, q_2, \dots, q_n can be used to write a *spectral decomposition* of A as follows:

$$A = \sum_{i=1}^n \lambda_i q_i q_i^T.$$

This decomposition can be restated in matrix form by defining

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n), \quad Q = [q_1 \mid q_2 \mid \dots \mid q_n],$$

and writing

$$A = Q \Lambda Q^T. \quad (\text{A.16})$$

In fact, when A is positive definite as well as symmetric, this decomposition is identical to the singular-value decomposition (A.15), where we define $U = V = Q$ and $S = \Lambda$. Note that the singular values σ_i and the eigenvalues λ_i coincide in this case.

In the case of the Euclidean norm (A.8b), we have for symmetric positive definite matrices A that the singular values and eigenvalues of A coincide, and that

$$\begin{aligned}\|A\| &= \sigma_1(A) = \text{largest eigenvalue of } A, \\ \|A^{-1}\| &= \sigma_n(A)^{-1} = \text{inverse of smallest eigenvalue of } A.\end{aligned}$$

Hence, we have for all $x \in \mathbb{R}^n$ that

$$\sigma_n(A)\|x\|^2 = \|x\|^2 / \|A^{-1}\| \leq x^T A x \leq \|A\|\|x\|^2 = \sigma_1(A)\|x\|^2.$$

For an orthogonal matrix Q , we have for the Euclidean norm that

$$\|Qx\| = \|x\|,$$

and that all the singular values of this matrix are equal to 1.

DETERMINANT AND TRACE

The trace of an $n \times n$ matrix A is defined by

$$\text{trace}(A) = \sum_{i=1}^n A_{ii}. \quad (\text{A.17})$$

If the eigenvalues of A are denoted by $\lambda_1, \lambda_2, \dots, \lambda_n$, it can be shown that

$$\text{trace}(A) = \sum_{i=1}^n \lambda_i, \quad (\text{A.18})$$

that is, the trace of the matrix is the sum of its eigenvalues.

The determinant of an $n \times n$ matrix A , denoted by $\det A$, is the product of its eigenvalues; that is,

$$\det A = \prod_{i=1}^n \lambda_i. \quad (\text{A.19})$$

The determinant has several appealing (and revealing) properties. For instance,

$\det A = 0$ if and only if A is singular;

$\det AB = (\det A)(\det B)$;

$\det A^{-1} = 1/\det A$.

Recall that any orthogonal matrix A has the property that $QQ^T = Q^TQ = I$, so that $Q^{-1} = Q^T$. It follows from the property of the determinant that $\det Q = \det Q^T = \pm 1$.

The properties above are used in the analysis of Chapter 6.

MATRIX FACTORIZATIONS: CHOLESKY, LU, QR

Matrix factorizations are important both in the design of algorithms and in their analysis. One such factorization is the singular-value decomposition defined above in (A.15). Here we define the other important factorizations.

All the factorization algorithms described below make use of *permutation matrices*. Suppose that we wish to exchange the first and fourth rows of a matrix A . We can perform this operation by premultiplying A by a permutation matrix P , which is constructed by interchanging the first and fourth rows of an identity matrix that contains the same number of rows as A . Suppose, for example, that A is a 5×5 matrix. The appropriate choice of P would be

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

A similar technique is used to find a permutation matrix P that exchanges columns of a matrix.

The LU factorization of a matrix $A \in \mathbb{R}^{n \times n}$ is defined as

$$PA = LU, \tag{A.20}$$

where

P is an $n \times n$ permutation matrix (that is, it is obtained by rearranging the rows of the $n \times n$ identity matrix),

L is unit lower triangular (that is, lower triangular with diagonal elements equal to 1, and

U is upper triangular.

This factorization can be used to solve a linear system of the form $Ax = b$ efficiently by the following three-step process:

form $\tilde{b} = Pb$ by permuting the elements of b ;

solve $Lz = \tilde{b}$ by performing triangular forward-substitution, to obtain the vector z ;

solve $Ux = z$ by performing triangular back-substitution, to obtain the solution vector x .

The factorization (A.20) can be found by using Gaussian elimination with row partial pivoting, an algorithm that requires approximately $2n^3/3$ floating-point operations when A is dense. Standard software that implements this algorithm (notably, LAPACK [7]) is readily available. The method can be stated as follows.

Algorithm A.1 (Gaussian Elimination with Row Partial Pivoting).

```

Given  $A \in \mathbb{R}^{n \times n}$ ;
Set  $P \leftarrow I, L \leftarrow 0$ ;
for  $i = 1, 2, \dots, n$ 
    find the index  $j \in \{i, i+1, \dots, n\}$  such that  $|A_{ji}| = \max_{k=i, i+1, \dots, n} |A_{ki}|$ ;
    if  $A_{ij} = 0$ 
        stop; (* matrix  $A$  is singular *)
    if  $i \neq j$ 
        swap rows  $i$  and  $j$  of matrices  $A$  and  $L$ ;
    (* elimination step *)
     $L_{ii} \leftarrow 1$ ;
    for  $k = i+1, i+2, \dots, n$ 
         $L_{ki} \leftarrow A_{ki}/A_{ii}$ ;
        for  $l = i+1, i+2, \dots, n$ 
             $A_{kl} \leftarrow A_{kl} - L_{ki}A_{il}$ ;
        end (for)
    end (if)
end (for)
 $U \leftarrow$  upper triangular part of  $A$ .

```

Variants of the basic algorithm allow for rearrangement of the columns as well as the rows during the factorization, but these do not add to the practical stability properties of the algorithm. Column pivoting may, however, improve the performance of Gaussian elimination when the matrix A is sparse, by ensuring that the factors L and U are also reasonably sparse.

Gaussian elimination can be applied also to the case in which A is not square. When A is $m \times n$, with $m > n$, the standard row pivoting algorithm produces a factorization of the form (A.20), where $L \in \mathbb{R}^{m \times n}$ is unit lower triangular and $U \in \mathbb{R}^{n \times n}$ is upper triangular. When $m < n$, we can find an LU factorization of A^T rather than A , that is, we obtain

$$PA^T = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} U, \quad (\text{A.21})$$

where L_1 is $m \times m$ (square) unit lower triangular, U is $m \times m$ upper triangular, and L_2 is a general $(n - m) \times m$ matrix. If A has full row rank, we can use this factorization to calculate

its null space explicitly as the space spanned by the columns of the matrix

$$M = P^T \begin{bmatrix} L_1^{-T} L_2^T \\ -I \end{bmatrix} U^{-T}. \quad (\text{A.22})$$

It is easy to check that M has dimensions $n \times (n - m)$ and that $AM = 0$.

When $A \in \mathbb{R}^{n \times n}$ is symmetric positive definite, it is possible to compute a similar but more specialized factorization at about half the cost—about $n^3/3$ operations. This factorization, known as the Cholesky factorization, produces a matrix L such that

$$A = LL^T. \quad (\text{A.23})$$

(If we require L to have positive diagonal elements, it is uniquely defined by this formula.) The algorithm can be specified as follows.

Algorithm A.2 (Cholesky Factorization).

Given $A \in \mathbb{R}^{n \times n}$ symmetric positive definite;

```

for  $i = 1, 2, \dots, n$ ;
     $L_{ii} \leftarrow \sqrt{A_{ii}}$ ;
    for  $j = i + 1, i + 2, \dots, n$ 
         $L_{ji} \leftarrow A_{ji} / L_{ii}$ ;
        for  $k = i + 1, i + 2, \dots, j$ 
             $A_{jk} \leftarrow A_{jk} - L_{ji} L_{ki}$ ;
        end (for)
    end (for)
end (for)

```

Note that this algorithm references only the lower triangular elements of A ; in fact, it is only necessary to store these elements in any case, since by symmetry they are simply duplicated in the upper triangular positions.

Unlike the case of Gaussian elimination, the Cholesky algorithm can produce a valid factorization of a symmetric positive definite matrix without swapping any rows or columns. However, symmetric permutation (that is, reordering the rows and columns in the same way) can be used to improve the sparsity of the factor L . In this case, the algorithm produces a permutation of the form

$$P^T A P = LL^T$$

for some permutation matrix P .

The Cholesky factorization can be used to compute solutions of the system $Ax = b$ by performing triangular forward- and back-substitutions with L and L^T , respectively, as in the case of L and U factors produced by Gaussian elimination.

The Cholesky factorization can also be used to verify positive definiteness of a symmetric matrix A . If Algorithm A.2 runs to completion with all L_{ii} values well defined and positive, then A is positive definite.

Another useful factorization of rectangular matrices $A \in \mathbb{R}^{m \times n}$ has the form

$$AP = QR, \quad (\text{A.24})$$

where

P is an $n \times n$ permutation matrix,

A is $m \times m$ orthogonal, and

R is $m \times n$ upper triangular.

In the case of a square matrix $m = n$, this factorization can be used to compute solutions of linear systems of the form $Ax = b$ via the following procedure:

set $\tilde{b} = Q^T b$;

solve $Rz = \tilde{b}$ for z by performing back-substitution;

set $x = P^T z$ by rearranging the elements of x .

For a dense matrix A , the cost of computing the QR factorization is about $4m^2n/3$ operations. In the case of a square matrix, the operation count is about twice as high as for an LU factorization via Gaussian elimination. Moreover, it is more difficult to maintain sparsity in a QR factorization than in an LU factorization.

Algorithms to perform QR factorization are almost as simple as algorithms for Gaussian elimination and for Cholesky factorization. The most widely used algorithms work by applying a sequence of special orthogonal matrices to A , known either as Householder transformations or Givens rotations, depending on the algorithm. We omit the details, and refer instead to Golub and Van Loan [136, Chapter 5] for a complete description.

In the case of a rectangular matrix A with $m < n$, we can use the QR factorization of A^T to find a matrix whose columns span the null space of A . To be specific, we write

$$A^T P = QR = [Q_1 \quad Q_2] R,$$

where Q_1 consists of the first m columns of Q , and Q_2 contains the last $n - m$ columns. It is easy to show that columns of the matrix Q_2 span the null space of A . This procedure yields a more satisfactory basis matrix for the null space than the Gaussian elimination procedure (A.22), because the columns of Q_2 are orthogonal to each other and have unit length. It may be more expensive to compute, however, particularly in the case in which A is sparse.

When A has full column rank, we can make an identification between the R factor in (A.24) and the Cholesky factorization. By multiplying the formula (A.24) by its transpose,

we obtain

$$P^T A^T A P = R^T Q^T Q R = R^T R,$$

and by comparison with (A.23), we see that R^T is simply the Cholesky factor of the symmetric positive definite matrix $P^T A^T A P$. Recalling that L is uniquely defined when we restrict its diagonal elements to be positive, this observation implies that R is also uniquely defined for a given choice of permutation matrix P , provided that we enforce positiveness of the diagonals of R . Note, too, that since we can rearrange (A.24) to read $A P R^{-1} = Q$, we can conclude that Q is also uniquely defined under these conditions.

Note that by definition of the Euclidean norm and the property (A.10), and the fact that the Euclidean norms of the matrices P and Q in (A.24) are both 1, we have that

$$\|A\| = \|Q R P^T\| \leq \|Q\| \|R\| \|P^T\| = \|R\|,$$

while

$$\|R\| = \|Q^T A P\| \leq \|Q^T\| \|A\| \|P\| = \|A\|.$$

We conclude from these two inequalities that $\|A\| = \|R\|$. When A is square, we have by a similar argument that $\|A^{-1}\| = \|R^{-1}\|$. Hence the Euclidean-norm condition number of A can be estimated by substituting R for A in the expression (A.11). This observation is significant because various techniques are available for estimating the condition number of triangular matrices R ; see Golub and Van Loan [136, pp. 128–130] for a discussion.

SYMMETRIC INDEFINITE FACTORIZATION

When matrix A is symmetric but indefinite, Algorithm A.2 will break down by trying to take the square root of a negative number. We can however produce a factorization, similar to the Cholesky factorization, of the form

$$P A P^T = L B L^T, \tag{A.25}$$

where L is unit lower triangular, B is a block diagonal matrix with blocks of dimension 1 or 2, and P is a permutation matrix. The first step of this symmetric indefinite factorization proceeds as follows. We identify a submatrix E of A that is suitable to be used as a pivot block. The precise criteria that can be used to choose E are described below, but we note here that E is either a single diagonal element of A (a 1×1 pivot block), or else the 2×2 block consisting of two diagonal elements of A (say, a_{ii} and a_{jj}) along with the corresponding off-diagonal elements (that is, a_{ij} and a_{ji}). In either case, E must be nonsingular. We then

find a permutation matrix P_1 that makes E a leading principal submatrix of A , that is,

$$P_1 A P_1 = \begin{bmatrix} E & C^T \\ C & H \end{bmatrix}, \quad (\text{A.26})$$

and then perform a block factorization on this rearranged matrix, using E as the pivot block, to obtain

$$P_1 A P_1^T = \begin{bmatrix} I & 0 \\ C E^{-1} & I \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & H - C E^{-1} C^T \end{bmatrix} \begin{bmatrix} I & E^{-1} C^T \\ 0 & I \end{bmatrix}.$$

The next step of the factorization consists in applying exactly the same process to $H - C E^{-1} C^T$, known as the *remaining matrix* or the *Schur complement*, which has dimension either $(n-1) \times (n-1)$ or $(n-2) \times (n-2)$. We now apply the same procedure recursively, terminating with the factorization (A.25). Here P is defined as a product of the permutation matrices from each step of the factorization, and B contains the pivot blocks E on its diagonal.

The symmetric indefinite factorization requires approximately $n^3/3$ floating-point operations—the same as the cost of the Cholesky factorization of a positive definite matrix—but to this count we must add the cost of identifying suitable pivot blocks E and of performing the permutations, which can be considerable. There are various strategies for determining the pivot blocks, which have an important effect on both the cost of the factorization and its numerical properties. Ideally, our strategy for choosing E at each step of the factorization procedure should be inexpensive, should lead to at most modest growth in the elements of the remaining matrix at each step of the factorization, and should avoid excessive fill-in (that is, L should not be too much more dense than A).

A well-known strategy, due to Bunch and Parlett [43], searches the whole remaining matrix and identifies the largest-magnitude diagonal and largest-magnitude off-diagonal elements, denoting their respective magnitudes by ξ_{dia} and ξ_{off} . If the diagonal element whose magnitude is ξ_{dia} is selected to be a 1×1 pivot block, the element growth in the remaining matrix is bounded by the ratio $\xi_{\text{dia}}/\xi_{\text{off}}$. If this growth rate is acceptable, we choose this diagonal element to be the pivot block. Otherwise, we select the off-diagonal element whose magnitude is ξ_{off} (a_{ij} , say), and choose E to be the 2×2 submatrix that includes this element, that is,

$$E = \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{bmatrix}.$$

This pivoting strategy of Bunch and Parlett is numerically stable and guarantees to yield a matrix L whose maximum element is bounded by 2.781. Its drawback is that the evaluation of ξ_{dia} and ξ_{off} at each iteration requires many comparisons between floating-point numbers

to be performed: $O(n^3)$ in total during the overall factorization. Since each comparison costs roughly the same as an arithmetic operation, this overhead is not insignificant.

The more economical pivoting strategy of Bunch and Kaufman [42] searches at most two columns of the working matrix at each stage and requires just $O(n^2)$ comparisons in total. Its rationale and details are somewhat tricky, and we refer the interested reader to the original paper [42] or to Golub and Van Loan [136, Section 4.4] for details. Unfortunately, this algorithm can give rise to arbitrarily large elements in the lower triangular factor L , making it unsuitable for use with a modified Cholesky strategy.

The bounded Bunch–Kaufman strategy is essentially a compromise between the Bunch–Parlett and Bunch–Kaufman strategies. It monitors the sizes of elements in L , accepting the (inexpensive) Bunch–Kaufman choice of pivot block when it yields only modest element growth, but searching further for an acceptable pivot when this growth is excessive. Its total cost is usually similar to that of Bunch–Kaufman, but in the worst case it can approach the cost of Bunch–Parlett.

So far, we have ignored the effect of the choice of pivot block E on the sparsity of the final L factor. This consideration is important when the matrix to be factored is large and sparse, since it greatly affects both the CPU time and the amount of storage required by the algorithm. Algorithms that modify the strategies above to take account of sparsity have been proposed by Duff et al. [97], Duff and Reid [95], and Fourer and Mehrotra [113].

SHERMAN–MORRISON–WOODBURY FORMULA

If the square nonsingular matrix A undergoes a rank-one update to become

$$\bar{A} = A + ab^T,$$

where $a, b \in \mathbb{R}^n$, then if \bar{A} is nonsingular, we have

$$\bar{A}^{-1} = A^{-1} - \frac{A^{-1}ab^T A^{-1}}{1 + b^T A^{-1}a}. \quad (\text{A.27})$$

It is easy to verify this formula: Simply multiply the definitions of \bar{A} and \bar{A}^{-1} together and check that they produce the identity.

This formula can be extended to higher-rank updates. Let U and V be matrices in $\mathbb{R}^{n \times p}$ for some p between 1 and n . If we define

$$\hat{A} = A + UV^T,$$

then \hat{A} is nonsingular if and only if $(I + V^T A^{-1}U)$ is nonsingular, and in this case we have

$$\hat{A}^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}. \quad (\text{A.28})$$

CONDITIONING AND STABILITY

Conditioning and *stability* are two terms that are used frequently in connection with numerical computations. Unfortunately, their meaning sometimes varies from author to author, but the general definitions below are widely accepted, and we adhere to them in this book.

Conditioning is a property of the numerical problem at hand (whether it is a linear algebra problem, an optimization problem, a differential equations problem, or whatever). A problem is said to be *well conditioned* if its solution is not affected greatly by small perturbations to the data that define the problem. Otherwise, it is said to be *ill conditioned*.

A simple example is given by the following 2×2 system of linear equations:

$$\begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

By computing the inverse of the coefficient matrix, we find that the solution is simply

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

If we replace the first right-hand-side element by 3.00001, the solution becomes $(x_1, x_2)^T = (0.99999, 1.00001)^T$, which is only slightly different from its exact value $(1, 1)^T$. We would note similar insensitivity if we were to perturb the other elements of the right-hand-side or elements of the coefficient matrix. We conclude that this problem is well conditioned. On the other hand, the problem

$$\begin{bmatrix} 1.00001 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.00001 \\ 2 \end{bmatrix}$$

is ill conditioned. Its exact solution is $x = (1, 1)^T$, but if we change the first element of the right-hand-side from 2.00001 to 2, the solution would change drastically to $x = (0, 2)^T$.

For general square linear systems $Ax = b$ where $A \in \mathbb{R}^{n \times n}$, the condition number of the matrix (defined in (A.11)) can be used to quantify the conditioning. Specifically, if we perturb A to \tilde{A} and b to \tilde{b} and take \tilde{x} to be the solution of the perturbed system $\tilde{A}\tilde{x} = \tilde{b}$, it can be shown that

$$\frac{\|x - \tilde{x}\|}{\|x\|} \approx \kappa(A) \left[\frac{\|A - \tilde{A}\|}{\|A\|} + \frac{\|b - \tilde{b}\|}{\|b\|} \right]$$

(see, for instance, Golub and Van Loan [136, Section 2.7]). Hence, a large condition number $\kappa(A)$ indicates that the problem $Ax = b$ is ill conditioned, while a modest value indicates well conditioning.

Note that the concept of conditioning has nothing to do with the particular algorithm that is used to solve the problem, only with the numerical problem itself.

Stability, on the other hand, is a property of the algorithm. An algorithm is stable if it is guaranteed to produce accurate answers to all well-conditioned problems in its class, even when floating-point arithmetic is used.

As an example, consider again the linear equations $Ax = b$. We can show that Algorithm A.1, in combination with triangular substitution, yields a computed solution \tilde{x} whose relative error is approximately

$$\frac{\|x - \tilde{x}\|}{\|x\|} \approx \kappa(A) \frac{\text{growth}(A)}{\|A\|} \mathbf{u}, \quad (\text{A.32})$$

where $\text{growth}(A)$ is the size of the largest element that arises in A during execution of Algorithm A.1. In the worst case, we can show that $\text{growth}(A)/\|A\|$ may be around 2^{n-1} , which indicates that Algorithm A.1 is an unstable algorithm, since even for modest n (say, $n = 200$), the right-hand-side of (A.32) may be large even when $\kappa(A)$ is modest. In practice, however, large growth factors are rarely observed, so we conclude that Algorithm A.1 is stable for all practical purposes.

Gaussian elimination without pivoting, on the other hand, is definitely unstable. If we omit the possible exchange of rows in Algorithm A.1, the algorithm will fail to produce a factorization even of some well-conditioned matrices, such as

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}.$$

For systems $Ax = b$ in which A is symmetric positive definite, the Cholesky factorization in combination with triangular substitution constitutes a stable algorithm for producing a solution x .

A.2 ELEMENTS OF ANALYSIS, GEOMETRY, TOPOLOGY

SEQUENCES

Suppose that $\{x_k\}$ is a sequence of points belonging to \mathbb{R}^n . We say that a sequence $\{x_k\}$ *converges* to some point x , written $\lim_{k \rightarrow \infty} x_k = x$, if for any $\epsilon > 0$, there is an index K such that

$$\|x_k - x\| \leq \epsilon, \quad \text{for all } k \geq K.$$

For example, the sequence $\{x_k\}$ defined by $x_k = (1 - 2^{-k}, 1/k^2)^T$ converges to $(1, 0)^T$.

which is nondecreasing. If $\{v_i\}$ is bounded above, it converges to a point \bar{v} which we call the “lim inf” of $\{t_k\}$, denoted by $\liminf t_k$. As an example, the sequence $1, \frac{1}{2}, 1, \frac{1}{4}, 1, \frac{1}{8}, \dots$ has a lim inf of 0 and a lim sup of 1.

RATES OF CONVERGENCE

One of the key measures of performance of an algorithm is its rate of convergence. Here, we define the terminology associated with different types of convergence.

Let $\{x_k\}$ be a sequence in \mathbb{R}^n that converges to x^* . We say that the convergence is *Q-linear* if there is a constant $r \in (0, 1)$ such that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r, \quad \text{for all } k \text{ sufficiently large.} \quad (\text{A.34})$$

This means that the distance to the solution x^* decreases at each iteration by at least a constant factor bounded away from 1. For example, the sequence $1 + (0.5)^k$ converges Q-linearly to 1, with rate $r = 0.5$. The prefix “Q” stands for “quotient,” because this type of convergence is defined in terms of the quotient of successive errors.

The convergence is said to be *Q-superlinear* if

$$\lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0.$$

For example, the sequence $1 + k^{-k}$ converges superlinearly to 1. (Prove this statement!) *Q-quadratic* convergence, an even more rapid convergence rate, is obtained if

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M, \quad \text{for all } k \text{ sufficiently large,}$$

where M is a positive constant, not necessarily less than 1. An example is the sequence $1 + (0.5)^{2^k}$.

The speed of convergence depends on r and (more weakly) on M , whose values depend not only on the algorithm but also on the properties of the particular problem. Regardless of these values, however, a quadratically convergent sequence will always eventually converge faster than a linearly convergent sequence.

Obviously, any sequence that converges Q-quadratically also converges Q-superlinearly, and any sequence that converges Q-superlinearly also converges Q-linearly. We can also define higher rates of convergence (cubic, quartic, and so on), but these are less interesting in practical terms. In general, we say that the Q-order of convergence is p (with $p > 1$) if there is a positive constant M such that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^p} \leq M, \quad \text{for all } k \text{ sufficiently large.}$$

Quasi-Newton methods for unconstrained optimization typically converge Q-superlinearly, whereas Newton's method converges Q-quadratically under appropriate assumptions. In contrast, steepest descent algorithms converge only at a Q-linear rate, and when the problem is ill-conditioned the convergence constant r in (A.34) is close to 1.

In the book, we omit the letter Q and simply talk about superlinear convergence, quadratic convergence, and so on.

A slightly weaker form of convergence, characterized by the prefix “R” (for “root”), is concerned with the overall rate of decrease in the error, rather than the decrease over each individual step of the algorithm. We say that convergence is *R-linear* if there is a sequence of nonnegative scalars $\{v_k\}$ such that

$$\|x_k - x^*\| \leq v_k \text{ for all } k, \text{ and } \{v_k\} \text{ converges Q-linearly to zero.}$$

The sequence $\{\|x_k - x^*\|\}$ is said to be *dominated* by $\{v_k\}$. For instance, the sequence

$$x_k = \begin{cases} 1 + (0.5)^k, & k \text{ even,} \\ 1, & k \text{ odd,} \end{cases} \quad (\text{A.35})$$

(the first few iterates are 2, 1, 1.25, 1, 1.03125, 1, ...) converges R-linearly to 1, because we have $|1 + (0.5)^k - 1| = (0.5)^k$, and the sequence $\{(0.5)^k\}$ converges Q-linearly to zero. Likewise, we say that $\{x_k\}$ converges R-superlinearly to x^* if $\{\|x_k - x^*\|\}$ is dominated by a sequence of scalars converging Q-superlinearly to zero, and $\{x_k\}$ converges R-quadratically to x^* if $\{\|x_k - x^*\|\}$ is dominated by a sequence converging Q-quadratically to zero.

Note that in the R-linear sequence (A.35), the error actually increases at every second iteration! Such behavior occurs even in sequences whose R-rate of convergence is arbitrarily high, but it cannot occur for Q-linear sequences, which insist on a decrease at every step k , for k sufficiently large.

For an extensive discussion of convergence rates see Ortega and Rheinboldt [230].

TOPOLOGY OF THE EUCLIDEAN SPACE \mathbb{R}^n

The set \mathcal{F} is *bounded* if there is some real number $M > 0$ such that

$$\|x\| \leq M, \quad \text{for all } x \in \mathcal{F}.$$

A subset $\mathcal{F} \subset \mathbb{R}^n$ is *open* if for every $x \in \mathcal{F}$, we can find a positive number $\epsilon > 0$ such that the ball of radius ϵ around x is contained in \mathcal{F} ; that is,

$$\{y \in \mathbb{R}^n \mid \|y - x\| \leq \epsilon\} \subset \mathcal{F}.$$

The set \mathcal{F} is *closed* if for all possible sequences of points $\{x_k\}$ in \mathcal{F} , all limit points of $\{x_k\}$ are elements of \mathcal{F} . For instance, the set $\mathcal{F} = (0, 1) \cup (2, 10)$ is an open subset of \mathbb{R} , while

$\mathcal{F} = [0, 1] \cup [2, 5]$ is a closed subset of \mathbb{R} . The set $\mathcal{F} = (0, 1]$ is a subset of \mathbb{R} that is neither open nor closed.

The *interior* of a set \mathcal{F} , denoted by $\text{int } \mathcal{F}$, is the largest open set contained in \mathcal{F} . The *closure* of \mathcal{F} , denoted by $\text{cl } \mathcal{F}$, is the smallest closed set containing \mathcal{F} . In other words, we have

$$x \in \text{cl } \mathcal{F} \quad \text{if } \lim_{k \rightarrow \infty} x_k = x \text{ for some sequence } \{x_k\} \text{ of points in } \mathcal{F}.$$

If $\mathcal{F} = (-1, 1] \cup [2, 4)$, then

$$\text{cl } \mathcal{F} = [-1, 1] \cup [2, 4], \quad \text{int } \mathcal{F} = (-1, 1) \cup (2, 4).$$

Note that if \mathcal{F} is open, then $\text{int } \mathcal{F} = \mathcal{F}$, while if \mathcal{F} is closed, then $\text{cl } \mathcal{F} = \mathcal{F}$.

We note the following facts about open and closed sets. The union of finitely many closed sets is closed, while any intersection of closed sets is closed. The intersection of finitely many open sets is open, while any union of open sets is open.

The set \mathcal{F} is *compact* if every sequence $\{x^k\}$ of points in \mathcal{F} has at least one limit point, and all such limit points are in \mathcal{F} . (This definition is equivalent to the more formal one involving covers of \mathcal{F} .) The following is a central result in topology:

$\mathcal{F} \subset \mathbb{R}^n$ is closed and bounded $\Rightarrow \mathcal{F}$ is compact.

Given a point $x \in \mathbb{R}^n$, we call $\mathcal{N} \subset \mathbb{R}^n$ a *neighborhood of x* if it is an open set containing x . An especially useful neighborhood is the *open ball of radius ϵ around x* , which is denoted by $\mathbb{B}(x, \epsilon)$; that is,

$$\mathbb{B}(x, \epsilon) = \{y \mid \|y - x\| < \epsilon\}.$$

Given a set $\mathcal{F} \subset \mathbb{R}^n$, we say that \mathcal{N} is a *neighborhood of \mathcal{F}* if there is $\epsilon > 0$ such that

$$\bigcup_{x \in \mathcal{F}} \mathbb{B}(x, \epsilon) \subset \mathcal{N}.$$

CONVEX SETS IN \mathbb{R}^n

A *convex combination* of a finite set of vectors $\{x_1, x_2, \dots, x_m\}$ in \mathbb{R}^m is any vector x of the form

$$x = \sum_{i=1}^m \alpha_i x_i, \quad \text{where } \sum_{i=1}^m \alpha_i = 1, \quad \text{and } \alpha_i \geq 0 \text{ for all } i = 1, 2, \dots, m.$$

The *convex hull* of $\{x_1, x_2, \dots, x_m\}$ is the set of all convex combinations of these vectors.

A *cone* is a set \mathcal{F} with the property that for all $x \in \mathcal{F}$ we have

$$x \in \mathcal{F} \Rightarrow \alpha x \in \mathcal{F}, \quad \text{for all } \alpha > 0. \tag{A.36}$$

For instance, the set $\mathcal{F} \subset \mathbb{R}^2$ defined by

$$\{(x_1, x_2)^T \mid x_1 > 0, x_2 \geq 0\}$$

is a cone in \mathbb{R}^2 . Note that cones are not necessarily convex. For example, the set $\{(x_1, x_2)^T \mid x_1 \geq 0 \text{ or } x_2 \geq 0\}$, which encompasses three quarters of the two-dimensional plane, is a cone.

The *cone generated by* $\{x_1, x_2, \dots, x_m\}$ is the set of all vectors x of the form

$$x = \sum_{i=1}^m \alpha_i x_i, \quad \text{where } \alpha_i \geq 0 \text{ for all } i = 1, 2, \dots, m.$$

Note that all cones of this form are convex.

Finally, we define the affine hull and relative interior of a set. An *affine set* in \mathbb{R}^n is a the set of all vectors $\{x\} \oplus \mathcal{S}$, where $x \in \mathbb{R}^n$ and \mathcal{S} is a subspace of \mathbb{R}^n . Given $\mathcal{F} \subset \mathbb{R}^n$, the *affine hull* of \mathcal{F} (denoted by $\text{aff } \mathcal{F}$) is the smallest affine set containing \mathcal{F} . For instance, when \mathcal{F} is the “ice-cream cone” defined in three dimensions as

$$\Omega = \left\{ x \in \mathbb{R}^3 \mid x_3 \geq 2\sqrt{x_1^2 + x_2^2} \right\} \quad (\text{A.37})$$

(see Figure A.1), we have $\text{aff } \mathcal{F} = \mathbb{R}^3$. If \mathcal{F} is the set of two isolated points $\mathcal{F} = \{(1, 0, 0)^T, (0, 2, 0)^T\}$, we have

$$\text{aff } \mathcal{F} = \{(1, 0, 0)^T + \alpha(-1, 2, 0)^T \mid \text{for all } \alpha \in \mathbb{R}\}.$$

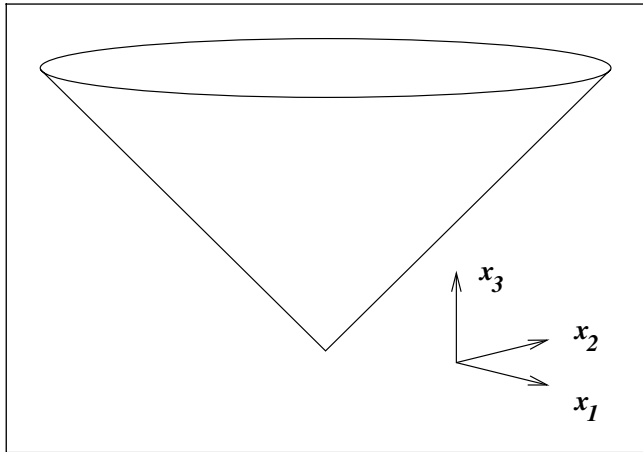


Figure A.1 “Ice-cream cone” set.

The *relative interior* $\text{ri } \mathcal{F}$ of the set \mathcal{F} is its *interior relative to* $\text{aff } \mathcal{F}$. If $x \in \mathcal{F}$, then $x \in \text{ri } \mathcal{F}$ if there is an $\epsilon > 0$ such that

$$(x + \epsilon \mathcal{B}) \cap \text{aff } \mathcal{F} \subset \mathcal{F}.$$

Referring again to the ice-cream cone (A.37), we have that

$$\text{ri } \mathcal{F} = \left\{ x \in \mathbb{R}^3 \mid x_3 > 2\sqrt{x_1^2 + x_2^2} \right\}.$$

For the set of two isolated points $\mathcal{F} = \{(1, 0, 0)^T, (0, 2, 0)^T\}$, we have $\text{ri } \mathcal{F} = \emptyset$. For the set \mathcal{F} defined by

$$\mathcal{F} \stackrel{\text{def}}{=} \{x \in \mathbb{R}^3 \mid x_1 \in [0, 1], x_2 \in [0, 1], x_3 = 0\},$$

we have that

$$\text{aff } \mathcal{F} = \mathbb{R} \times \mathbb{R} \times \{0\}, \quad \text{ri } \mathcal{F} = \{x \in \mathbb{R}^3 \mid x_1 \in (0, 1), x_2 \in (0, 1), x_3 = 0\}.$$

CONTINUITY AND LIMITS

Let f be a function that maps some domain $\mathcal{D} \subset \mathbb{R}^n$ to the space \mathbb{R}^m . For some point $x_0 \in \text{cl } \mathcal{D}$, we write

$$\lim_{x \rightarrow x_0} f(x) = f_0 \tag{A.38}$$

(spoken “the limit of $f(x)$ as x approaches x_0 is f_0 ”) if for all $\epsilon > 0$, there is a value $\delta > 0$ such that

$$\|x - x_0\| < \delta \text{ and } x \in \mathcal{D} \Rightarrow \|f(x) - f_0\| < \epsilon.$$

We say that f is *continuous* at x_0 if $x_0 \in \mathcal{D}$ and the expression (A.38) holds with $f_0 = f(x_0)$. We say that f is continuous on its domain \mathcal{D} if f is continuous for all $x_0 \in \mathcal{D}$.

An example is provided by the function

$$f(x) = \begin{cases} -x & \text{if } x \in [-1, 1], x \neq 0, \\ 5 & \text{for all other } x \in [-10, 10]. \end{cases} \tag{A.39}$$

This function is defined on the domain $[-10, 10]$ and is continuous at all points of the domain except the points $x = 0$, $x = 1$, and $x = -1$. At $x = 0$, the expression (A.38) holds with $f_0 = 0$, but the function is not continuous at this point because $f_0 \neq f(0) = 5$. At

$x = -1$, the limit (A.38) is not defined, because the function values in the neighborhood of this point are close to both 5 and -1 , depending on whether x is slightly smaller or slightly larger than -1 . Hence, the function is certainly not continuous at this point. The same comments apply to the point $x = 1$.

In the special case of $n = 1$ (that is, the argument of f is a real scalar), we can also define the *one-sided limit*. Given $x_0 \in \text{cl}\mathcal{D}$, We write

$$\lim_{x \downarrow x_0} f(x) = f_0 \quad (\text{A.40})$$

(spoken “the limit of $f(x)$ as x approaches x_0 from above is f_0 ”) if for all $\epsilon > 0$, there is a value $\delta > 0$ such that

$$x_0 < x < x_0 + \delta \text{ and } x \in \mathcal{D} \Rightarrow \|f(x) - f_0\| < \epsilon.$$

Similarly, we write

$$\lim_{x \uparrow x_0} f(x) = f_0 \quad (\text{A.41})$$

(spoken “the limit of $f(x)$ as x approaches x_0 from below is f_0 ”) if for all $\epsilon > 0$, there is a value $\delta > 0$ such that

$$x_0 - \delta < x < x_0 \text{ and } x \in \mathcal{D} \Rightarrow \|f(x) - f_0\| < \epsilon.$$

For the function defined in (A.39), we have that

$$\lim_{x \downarrow 1} f(x) = 5, \quad \lim_{x \uparrow 1} f(x) = 1.$$

Considering again the general case of $f : \mathcal{D} \rightarrow \mathbb{R}^m$ where $\mathcal{D} \subset \mathbb{R}^n$ for general m and n . The function f is said to be *Lipschitz continuous* on some set $\mathcal{N} \subset \mathcal{D}$ if there is a constant $L > 0$ such that

$$\|f(x_1) - f(x_0)\| \leq L\|x_1 - x_0\|, \quad \text{for all } x_0, x_1 \in \mathcal{N}. \quad (\text{A.42})$$

(L is called the *Lipschitz constant*.) The function f is *locally Lipschitz continuous* at a point $\bar{x} \in \text{int}\mathcal{D}$ if there is some neighborhood \mathcal{N} of \bar{x} with $\mathcal{N} \subset \mathcal{D}$ such that the property (A.42) holds for some $L > 0$.

If g and h are two functions mapping $\mathcal{D} \subset \mathbb{R}^n$ to \mathbb{R}^m , Lipschitz continuous on a set $\mathcal{N} \subset \mathcal{D}$, their sum $g + h$ is also Lipschitz continuous, with Lipschitz constant equal to the sum of the Lipschitz constants for g and h individually. If g and h are two functions mapping $\mathcal{D} \subset \mathbb{R}^n$ to \mathbb{R} , the product gh is Lipschitz continuous on a set $\mathcal{N} \subset \mathcal{D}$ if both g and h are Lipschitz continuous on \mathcal{N} and both are bounded on \mathcal{N} (that is, there is $M > 0$

such that $|g(x)| \leq M$ and $|h(x)| \leq M$ for all $x \in \mathcal{N}$). We prove this claim via a sequence of elementary inequalities, for arbitrary $x_0, x_1 \in \mathcal{N}$:

$$\begin{aligned}
 & |g(x_0)h(x_0) - g(x_1)h(x_1)| \\
 & \leq |g(x_0)h(x_0) - g(x_1)h(x_0)| + |g(x_1)h(x_0) - g(x_1)h(x_1)| \\
 & = |h(x_0)| |g(x_0) - g(x_1)| + |g(x_1)| |h(x_0) - h(x_1)| \\
 & \leq 2ML \|x_0 - x_1\|,
 \end{aligned} \tag{A.43}$$

where L is an upper bound on the Lipschitz constant for both g and h .

DERIVATIVES

Let $\phi : \mathbb{R} \rightarrow \mathbb{R}$ be a real-valued function of a real variable (sometimes known as a *univariate* function). The first derivative $\phi'(\alpha)$ is defined by

$$\frac{d\phi}{d\alpha} = \phi'(\alpha) \stackrel{\text{def}}{=} \lim_{\epsilon \rightarrow 0} \frac{\phi(\alpha + \epsilon) - \phi(\alpha)}{\epsilon}. \tag{A.44}$$

The second derivative is obtained by substituting ϕ by ϕ' in this same formula; that is,

$$\frac{d^2\phi}{d\alpha^2} = \phi''(\alpha) \stackrel{\text{def}}{=} \lim_{\epsilon \rightarrow 0} \frac{\phi'(\alpha + \epsilon) - \phi'(\alpha)}{\epsilon}. \tag{A.45}$$

Suppose now that α in turn depends on another quantity β (we denote this dependence by writing $\alpha = \alpha(\beta)$). We can use the *chain rule* to calculate the derivative of ϕ with respect to β :

$$\frac{d\phi(\alpha(\beta))}{d\beta} = \frac{d\phi}{d\alpha} \frac{d\alpha}{d\beta} = \phi'(\alpha) \alpha'(\beta). \tag{A.46}$$

Consider now the function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, which is a real-valued function of n independent variables. We typically gather the variables into a vector $x = (x_1, x_2, \dots, x_n)^T$. We say that f is differentiable at x if there exists a vector $g \in \mathbb{R}^n$ such that

$$\lim_{y \rightarrow 0} \frac{f(x + y) - f(x) - g^T y}{\|y\|} = 0, \tag{A.47}$$

where $\|\cdot\|$ is any vector norm of y . (This type of differentiability is known as *Frechet differentiability*.) If g satisfying (A.47) exists, we call it the *gradient* of f at x , and denote it

by $\nabla f(x)$, written componentwise as

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}. \quad (\text{A.48})$$

Here, $\partial f / \partial x_i$ represents the partial derivative of f with respect to x_i . By setting $y = \epsilon e_i$ in (A.47), where e_i is the vector in \mathbb{R}^n consisting all all zeros, except for a 1 in position i , we obtain

$$\begin{aligned} & \frac{\partial f}{\partial x_i} \\ & \stackrel{\text{def}}{=} \lim_{\epsilon \rightarrow 0} \frac{f(x_1, \dots, x_{i-1}, x_i + \epsilon, x_{i+1}, \dots, x_n) - f(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)}{\epsilon} \\ & = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}. \end{aligned}$$

A gradient with respect to only a subset of the unknowns can be expressed by means of a subscript on the symbol ∇ . Thus for the function of two vector variables $f(z, t)$, we use $\nabla_z f(z, t)$ to denote the gradient with respect to z (holding t constant).

The matrix of second partial derivatives of f is known as the *Hessian*, and is defined as

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

We say that f is *differentiable* on a domain \mathcal{D} if $\nabla f(x)$ exists for all $x \in \mathcal{D}$, and *continuously differentiable* if $\nabla f(x)$ is a continuous functions of x . Similarly, f is *twice differentiable* on \mathcal{D} if $\nabla^2 f(x)$ exists for all $x \in \mathcal{D}$ and *twice continuously differentiable* if $\nabla^2 f(x)$ is continuous on \mathcal{D} . Note that when f is twice continuously differentiable, the Hessian is a symmetric matrix, since

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}, \quad \text{for all } i, j = 1, 2, \dots, n.$$

When f is a vector valued function that is $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ (See Chapters 10 and 11), we define $\nabla f(x)$ to be the $n \times m$ matrix whose i th column is $\nabla f_i(x)$, that is, the gradient of

f_i with respect to x . Often, for notational convenience, we prefer to work with the transpose of his matrix, which has dimensions $m \times n$. This matrix is called the *Jacobian* and is often denoted by $J(x)$. Specifically, the (i, j) element of $J(x)$ is $\partial f_i(x)/\partial x_j$.

When the vector x in turn depends on another vector t (that is, $x = x(t)$), we can extend the chain rule (A.46) for the univariate function. Defining

$$h(t) = f(x(t)), \quad (\text{A.49})$$

we have

$$\nabla h(t) = \sum_{i=1}^n \frac{\partial f}{\partial x_i} \nabla x_i(t) = \nabla x(t) \nabla f(x(t)). \quad (\text{A.50})$$

□ **EXAMPLE A.1**

Let $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ be defined by $f(x_1, x_2) = x_1^2 + x_1 x_2$, where $x_1 = \sin t_1 + t_2^2$ and $x_2 = (t_1 + t_2)^2$. Defining $h(t)$ as in (A.49), the chain rule (A.50) yields

$$\begin{aligned} \nabla h(t) &= \sum_{i=1}^n \frac{\partial f}{\partial x_i} \nabla x_i(t) \\ &= (2x_1 + x_2) \begin{bmatrix} \cos t_1 \\ 2t_2 \end{bmatrix} + x_1 \begin{bmatrix} 2(t_1 + t_2) \\ 2(t_1 + t_2) \end{bmatrix} \\ &= (2(\sin t_1 + t_2^2) + (t_1 + t_2)^2) \begin{bmatrix} \cos t_1 \\ 2t_2 \end{bmatrix} + (\sin t_1 + t_2^2) \begin{bmatrix} 2(t_1 + t_2) \\ 2(t_1 + t_2) \end{bmatrix}. \end{aligned}$$

If, on the other hand, we substitute directly for x into the definition of f , we obtain

$$h(t) = f(x(t)) = (\sin t_1 + t_2^2)^2 + (\sin t_1 + t_2^2)(t_1 + t_2)^2.$$

The reader should verify that the gradient of this expression is identical to the one obtained above by applying the chain rule. □

Special cases of the chain rule can be derived when $x(t)$ in (A.50) is a linear function of t , say $x(t) = Ct$. We then have $\nabla x(t) = C^T$, so that

$$\nabla h(t) = C^T \nabla f(Ct).$$

In the case in which f is a scalar function, we can differentiate twice using the chain rule to obtain

$$\nabla^2 h(t) = C^T \nabla^2 f(Ct) C.$$

(The proof of this statement is left as an exercise.)

DIRECTIONAL DERIVATIVES

The *directional derivative* of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ in the direction p is given by

$$D(f(x); p) \stackrel{\text{def}}{=} \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon p) - f(x)}{\epsilon}. \quad (\text{A.51})$$

The directional derivative may be well defined even when f is not continuously differentiable; in fact, it is most useful in such situations. Consider for instance the ℓ_1 norm function $f(x) = \|x\|_1$. We have from the definition (A.51) that

$$D(\|x\|_1; p) = \lim_{\epsilon \rightarrow 0} \frac{\|x + \epsilon p\|_1 - \|x\|_1}{\epsilon} = \lim_{\epsilon \rightarrow 0} \frac{\sum_{i=1}^n |x_i + \epsilon p_i| - \sum_{i=1}^n |x_i|}{\epsilon}.$$

If $x_i > 0$, we have $|x_i + \epsilon p_i| = |x_i| + \epsilon p_i$ for all ϵ sufficiently small. If $x_i < 0$, we have $|x_i + \epsilon p_i| = |x_i| - \epsilon p_i$, while if $x_i = 0$, we have $|x_i + \epsilon p_i| = \epsilon |p_i|$. Therefore, we have

$$D(\|x\|_1; p) = \sum_{i|x_i < 0} -p_i + \sum_{i|x_i > 0} p_i + \sum_{i|x_i = 0} |p_i|,$$

so the directional derivative of this function exists for any x and p . The first derivative $\nabla f(x)$ does *not* exist, however, whenever any of the components of x are zero.

When f is in fact continuously differentiable in a neighborhood of x , we have

$$D(f(x); p) = \nabla f(x)^T p.$$

To verify this formula, we define the function

$$\phi(\alpha) = f(x + \alpha p) = f(y(\alpha)), \quad (\text{A.52})$$

where $y(\alpha) = x + \alpha p$. Note that

$$\lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon p) - f(x)}{\epsilon} = \lim_{\epsilon \rightarrow 0} \frac{\phi(\epsilon) - \phi(0)}{\epsilon} = \phi'(0).$$

By applying the chain rule (A.50) to $f(y(\alpha))$, we obtain

$$\begin{aligned}\phi'(\alpha) &= \sum_{i=1}^n \frac{\partial f(y(\alpha))}{\partial y_i} \nabla y_i(\alpha) \\ &= \sum_{i=1}^n \frac{\partial f(y(\alpha))}{\partial y_i} p_i = \nabla f(y(\alpha))^T p = \nabla f(x + \alpha p)^T p.\end{aligned}\tag{A.53}$$

We obtain (A.51) by setting $\alpha = 0$ and comparing the last two expressions.

MEAN VALUE THEOREM

We now recall the mean value theorem for univariate functions. Given a continuously differentiable function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ and two real numbers α_0 and α_1 that satisfy $\alpha_1 > \alpha_0$, we have that

$$\phi(\alpha_1) = \phi(\alpha_0) + \phi'(\xi)(\alpha_1 - \alpha_0)\tag{A.54}$$

for some $\xi \in (\alpha_0, \alpha_1)$. An extension of this result to a multivariate function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is that for any vector p we have

$$f(x + p) = f(x) + \nabla f(x + \alpha p)^T p,\tag{A.55}$$

for some $\alpha \in (0, 1)$. (This result can be proved by defining $\phi(\alpha) = f(x + \alpha p)$, $\alpha_0 = 0$, and $\alpha_1 = 1$ and applying the chain rule, as above.)

□ EXAMPLE A.2

Consider $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ defined by $f(x) = x_1^3 + 3x_1x_2^2$, and let $x = (0, 0)^T$ and $p = (1, 2)^T$. It is easy to verify that $f(x) = 0$ and $f(x + p) = 13$. Since

$$\nabla f(x + \alpha p) = \begin{bmatrix} 3(x_1 + \alpha p_1)^2 + 3(x_2 + \alpha p_2)^2 \\ 6(x_1 + \alpha p_1)(x_2 + \alpha p_2) \end{bmatrix} = \begin{bmatrix} 15\alpha^2 \\ 12\alpha^2 \end{bmatrix},$$

we have that $\nabla f(x + \alpha p)^T p = 39\alpha^2$. Hence the relation (A.55) holds when we set $\alpha = 1/\sqrt{13}$, which lies in the open interval $(0, 1)$, as claimed. □

An alternative expression to (A.55) can be stated for twice differentiable functions: We have

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + \alpha p)^T p, \quad (\text{A.56})$$

for some $\alpha \in (0, 1)$. In fact, this expression is one form of Taylor's theorem, Theorem 2.1 in Chapter 2, to which we refer throughout the book.

The extension of (A.55) to a vector-valued function $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ for $m > 1$ is not immediate. There is in general no scalar α such that the natural extension of (A.55) is satisfied. However, the following result is often a useful analog. As in (10.3), we denote the Jacobian of $r(x)$, by $J(x)$, where $J(x)$ is the $m \times n$ matrix whose (j, i) entry is $\partial r_j / \partial x_i$, for $j = 1, 2, \dots, m$ and $i = 1, 2, \dots, n$, and assume that $J(x)$ is defined and continuous on the domain of interest. Given x and p , we then have

$$r(x + p) - r(x) = \int_0^1 J(x + \alpha p) p \, d\alpha. \quad (\text{A.57})$$

When p is sufficiently small in norm, we can approximate the right-hand side of this expression adequately by $J(x)p$, that is,

$$r(x + p) - r(x) \approx J(x)p.$$

If J is Lipschitz continuous in the vicinity of x and $x + p$ with Lipschitz constant L , we can use (A.12) to estimate the error in this approximation as follows:

$$\begin{aligned} \|r(x + p) - r(x) - J(x)p\| &= \left\| \int_0^1 [J(x + \alpha p) - J(x)] p \, d\alpha \right\| \\ &\leq \int_0^1 \|J(x + \alpha p) - J(x)\| \|p\| \, d\alpha \\ &\leq \int_0^1 L\alpha \|p\|^2 \, d\alpha = \frac{1}{2} L \|p\|^2. \end{aligned}$$

IMPLICIT FUNCTION THEOREM

The implicit function theorem lies behind a number of important results in local convergence theory of optimization algorithms and in the characterization of optimality (see Chapter 12). Our statement of this result is based on Lang [187, p. 131] and Bertsekas [19, Proposition A.25].

Theorem A.2 (Implicit Function Theorem).

Let $h : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ be a function such that

- (i) $h(z^*, 0) = 0$ for some $z^* \in \mathbb{R}^n$,
- (ii) the function $h(\cdot, \cdot)$ is continuously differentiable in some neighborhood of $(z^*, 0)$, and
- (iii) $\nabla_z h(z, t)$ is nonsingular at the point $(z, t) = (z^*, 0)$.

Then there exist open sets $\mathcal{N}_z \subset \mathbb{R}^n$ and $\mathcal{N}_t \subset \mathbb{R}^m$ containing z^* and 0, respectively, and a continuous function $z : \mathcal{N}_t \rightarrow \mathcal{N}_z$ such that $z^* = z(0)$ and $h(z(t), t) = 0$ for all $t \in \mathcal{N}_t$. Further, $z(t)$ is uniquely defined. Finally, if h is p times continuously differentiable with respect to both its arguments for some $p > 0$, then $z(t)$ is also p times continuously differentiable with respect to t , and we have

$$\nabla z(t) = -\nabla_t h(z(t), t) [\nabla_z h(z(t), t)]^{-1}$$

for all $t \in \mathcal{N}_t$.

This theorem is frequently applied to parametrized systems of linear equations, in which z is obtained as the solution of

$$M(t)z = g(t),$$

where $M(\cdot) \in \mathbb{R}^{n \times n}$ has $M(0)$ nonsingular, and $g(\cdot) \in \mathbb{R}^n$. To apply the theorem, we define

$$h(z, t) = M(t)z - g(t).$$

If $M(\cdot)$ and $g(\cdot)$ are continuously differentiable in some neighborhood of 0, the theorem implies that $z(t) = M(t)^{-1}g(t)$ is a continuous function of t in some neighborhood of 0.

ORDER NOTATION

In much of our analysis we are concerned with how the members of a sequence behave *eventually*, that is, when we get far enough along in the sequence. For instance, we might ask whether the elements of the sequence are bounded, or whether they are similar in size to the elements of a corresponding sequence, or whether they are decreasing and, if so, how rapidly. *Order notation* is useful shorthand to use when questions like these are being examined. It saves us defining many constants that clutter up the argument and the analysis.

We will use three varieties of order notation: $O(\cdot)$, $o(\cdot)$, and $\Omega(\cdot)$. Given two nonnegative infinite sequences of scalars $\{\eta_k\}$ and $\{\nu_k\}$, we write

$$\eta_k = O(\nu_k)$$

if there is a positive constant C such that

$$|\eta_k| \leq C|v_k|$$

for all k sufficiently large. We write

$$\eta_k = o(v_k)$$

if the sequence of ratios $\{\eta_k/v_k\}$ approaches zero, that is,

$$\lim_{k \rightarrow \infty} \frac{\eta_k}{v_k} = 0.$$

Finally, we write

$$\eta_k = \Omega(v_k)$$

if there are two constants C_0 and C_1 with $0 < C_0 \leq C_1 < \infty$ such that

$$C_0|v_k| \leq |\eta_k| \leq C_1|v_k|,$$

that is, the corresponding elements of both sequences stay in the same ballpark for all k . This definition is equivalent to saying that $\eta_k = O(v_k)$ and $v_k = O(\eta_k)$.

The same notation is often used in the context of quantities that depend continuously on each other as well. For instance, if $\eta(\cdot)$ is a function that maps \mathbb{R} to \mathbb{R} , we write

$$\eta(v) = O(v)$$

if there is a constant C such that $|\eta(v)| \leq C|v|$ for all $v \in \mathbb{R}$. (Typically, we are interested only in values of v that are either very large or very close to zero; this should be clear from the context. Similarly, we use

$$\eta(v) = o(v) \tag{A.58}$$

to indicate that the ratio $\eta(v)/v$ approaches zero either as $v \rightarrow 0$ or $v \rightarrow \infty$. (Again, the precise meaning should be clear from the context.)

As a slight variant on the definitions above, we write

$$\eta_k = O(1)$$

to indicate that there is a constant C such that $|\eta_k| \leq C$ for all k , while

$$\eta_k = o(1)$$

indicates that $\lim_{k \rightarrow \infty} \eta_k = 0$. We sometimes use vector and matrix quantities as arguments, and in these cases the definitions above are intended to apply to the norms of these quantities. For instance, if $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, we write $f(x) = O(\|x\|)$ if there is a constant $C > 0$ such that $\|f(x)\| \leq C\|x\|$ for all x in the domain of f . Typically, as above, we are interested only in some subdomain of f , usually a small neighborhood of 0. As before, the precise meaning should be clear from the context.

ROOT-FINDING FOR SCALAR EQUATIONS

In Chapter 11 we discussed methods for finding solutions of nonlinear systems of equations $F(x) = 0$, where $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Here we discuss briefly the case of scalar equations ($n = 1$), for which the algorithm is easy to illustrate. Scalar root-finding is needed in the trust-region algorithms of Chapter 4, for instance. Of course, the general theorems of Chapter 11 can be applied to derive rigorous convergence results for this special case.

The basic step of Newton's method (Algorithm Newton of Chapter 11) in the scalar case is simply

$$p_k = -F(x_k)/F'(x_k), \quad x_{k+1} \leftarrow x_k + p_k \quad (\text{A.59})$$

(cf. (11.6)). Graphically, such a step involves taking the tangent to the graph of F at the point x_k and taking the next iterate to be the intersection of this tangent with the x axis (see Figure A.2). Clearly, if the function F is nearly linear, the tangent will be quite a good approximation to F itself, so the Newton iterate will be quite close to the true root of F .

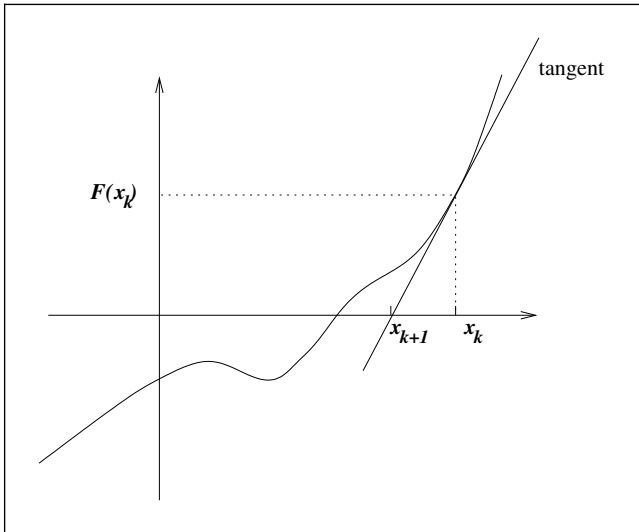


Figure A.2 One step of Newton's method for a scalar equation.

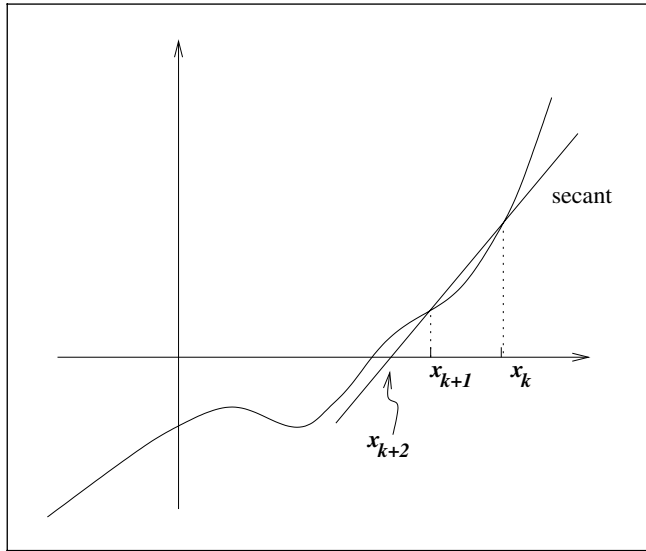


Figure A.3 One step of the secant method for a scalar equation.

The secant method for scalar equations can be viewed as the specialization of Broyden's method to the case of $n = 1$. The issues are simpler in this case, however, since the secant equation (11.27) completely determines the value of the 1×1 approximate Hessian B_k . That is, we do not need to apply extra conditions to ensure that B_k is fully determined. By combining (11.24) with (11.27), we find that the secant method for the case of $n = 1$ is defined by

$$B_k = (F(x_k) - F(x_{k-1})) / (x_k - x_{k-1}), \quad (\text{A.60a})$$

$$p_k = -F(x_k) / B_k, \quad x_{k+1} = x_k + p_k. \quad (\text{A.60b})$$

By illustrating this algorithm, we see the origin of the term “secant.” B_k approximates the slope of the function at x_k by taking the secant through the points $(x_{k-1}, F(x_{k-1}))$ and $(x_k, F(x_k))$, and x_{k+1} is obtained by finding the intersection of this secant with the x axis. The method is illustrated in Figure A.3.

Index

- Accumulation point, *see* Limit point
- Active set, 308, 323, 336, 342
- Affine scaling
 - direction, 395, 398, 414
 - method, 417
- Alternating variables method, *see also*
 - Coordinate search method, 104, 230
- Angle test, 41
- Applications
 - design optimization, 1
 - finance, 7
 - portfolio optimization, 1, 449–450, 492
 - transportation, 4
- Armijo line search, *see* Line search, Armijo
- Augmented Lagrangian function, 423
 - as merit function, 436
 - definition, 514
 - exactness of, 517–518
 - example, 516
- Augmented Lagrangian method, 422, 498, 514–526
 - convergence, 518–519
 - framework for, 515
 - implementation, 519–523
 - LANCELOT, 175, 519–522
 - motivation, 514–515
- Automatic differentiation, 170, 194
 - adjoint variables, 208, 209
 - and graph-coloring algorithms, 212, 216–218
 - checkpointing, 210
 - common expressions, 211
 - computational graph, 205–206, 208, 210, 211, 213, 215

- Automatic (*cont.*)
 - computational requirements, 206–207, 210, 214, 216, 219
 - forward mode, 206–207, 278
 - forward sweep, 206, 208, 210, 213–215, 219
 - foundations in elementary arithmetic, 194, 204
 - Hessian calculation
 - forward mode, 213–215
 - interpolation formulae, 214–215
 - reverse mode, 215–216
 - intermediate variables, 205–209, 211, 212, 218
 - Jacobian calculation, 210–213
 - forward mode, 212
 - reverse mode, 212–213
 - limitations of, 216–217
 - reverse mode, 207–210
 - reverse sweep, 208–210, 218
 - seed vectors, 206, 207, 212, 213, 216
 - software, 194, 210, 217
- Backtracking, 37, 240
- Barrier functions, 566, 583
- Barrier method, 563–566
 - primal, 583
- Basic variables, 429
- Basis matrix, 429–431
- BFGS method, 24, 29, 136–143
 - damping, 537
 - implementation, 142–143
 - properties, 141–142, 161
 - self-correction, 142
 - skipping, 143, 537
- Bound-constrained optimization, 97, 485–490
- BQPD, 490
- Broyden class, *see* Quasi-Newton method, Broyden class
- Broyden's method, 273, 274, 284, 285, 302, 634
 - derivation of, 279–281
 - limited-memory variants, 283
 - rate of convergence, 281–283
 - statement of algorithm, 281
- Byrd–Omojokun method, 547, 579
- Calculus of variations, 9
- Cancellation error, *see* Floating-point arithmetic, cancellation
- Cauchy point, 71–73, 76, 77, 93, 100, 170, 172, 262, 486
 - calculation of, 71–72, 96
 - for nonlinear equations, 291–292
 - role in global convergence, 77–79
- Cauchy sequence, 618
- Cauchy–Schwarz inequality, 75, 99, 151, 600
- Central path, 397–399, 417
 - for nonlinear problems, 565, 584, 594
 - neighborhoods of, 399–401, 403, 406, 413
- Chain rule, 29, 194, 204, 206–208, 213, 625, 627, 629
- Cholesky factorization, 87, 141, 143, 161, 251, 259, 289, 292, 454, 599, 608–609, 617
 - incomplete, 174
 - modified, 48, 51–54, 63, 64, 76
 - bounded modified factorization property, 48
 - sparse, 412–413
 - stability of, 53, 617
- Classification of algorithms, 422
- Combinatorial difficulty, 424
- Complementarity condition, 70, 313, 321, 333, 397
 - strict, 321, 337, 342, 533, 565, 591
- Complementarity problems
 - linear (LCP), 415
 - nonlinear (NCP), 417
- Complexity of algorithms, 388–389, 393, 406, 415, 417
- Conditioning, *see also* Matrix, condition number, 426, 430–432, 616–617
 - ill conditioned, 29, 502, 514, 586, 616
 - well conditioned, 616
- Cone, 621
- Cone of feasible directions, *see* Tangent cone
- Conjugacy, 25, 102
- Conjugate direction method, 103

- expanding subspace minimization, 106, 172, 173
- termination of, 103
- Conjugate gradient method, 71, 101–132, 166, 170–173, 253, 278
 - n -step quadratic convergence, 133
 - clustering of eigenvalues, 116
 - effect of condition number, 117
 - expanding subspace minimization, 112
 - Fletcher–Reeves, *see* Fletcher–Reeves method
 - for reduced system, 459–461
 - global convergence, 40
 - Hestenes–Stiefel, 123
 - Krylov subspace, 113
 - modified for indefiniteness, 169–170
 - nonlinear, 25, 121–131
 - numerical performance, 131
 - optimal polynomial, 113
 - optimal process, 112
 - Polak–Ribière, *see* Polak–Ribière method
 - practical version, 111
 - preconditioned, 118–119, 170, 460
 - projected, 461–463, 548, 571, 581, 593
 - rate of convergence, 112
 - relation to limited-memory, 180
 - restarts, 124
 - superlinear convergence, 132
 - superquadratic, 133
 - termination, 115, 124
- Constrained optimization, 6
 - nonlinear, 4, 6, 211, 293, 356, 421, 498, 500
- Constraint qualifications, 315–320, 333, 338–340, 350
 - linear independence (LICQ), 320, 321, 323, 339, 341, 358, 464, 503, 517, 533, 557, 565, 591
 - Mangasarian–Fromovitz (MFCQ), 339–340
- Constraints, 2, 307
 - bounds, 434, 519, 520
 - equality, 305
 - inequality, 305
- Continuation methods for nonlinear equations, 274, 303
 - application to KKT conditions for nonlinear optimization, 565
 - convergence of, 300–301
 - formulation as initial-value ODE, 297–299
 - motivation, 296–297
 - predictor–corrector method, 299–300
 - zero path, 296–301, 303
 - divergence of, 300–301
 - tangent, 297–300
 - turning point, 296, 297, 300
- Convergence, rate of, 619–620
 - n -step quadratic, 133
 - linear, 262, 619, 620
 - quadratic, 23, 29, 49, 168, 257, 619, 620
 - sublinear, 29
 - superlinear, 23, 29, 73, 132, 140, 142, 160, 161, 168, 262–265, 414, 619, 620
 - superquadratic, 133
- Convex combination, 621
- Convex hull, 621
- Convex programming, 7, 8, 335
- Convexity, 7–8
 - of functions, 8, 16–17, 28, 250
 - of sets, 8, 28, 352
 - strict, 8
- Coordinate descent method, *see* Alternating variables method, 233
- Coordinate relaxation step, 431
- Coordinate search method, 135, 230–231
- CPLEX, 490
- Critical cone, 330
- Data-fitting problems, 11–12, 248**
- Degeneracy, 465
 - of basis, 366, 369, 372, 382
 - of linear program, 366
- Dennis and Moré characterization, 47
- Descent direction, 21, 29, 30
- DFP method, 139
- Differential equations
 - ordinary, 299
 - partial, 216, 302
- Direct sum, 603
- Directional derivative, 206, 207, 437, 628–629
- Discrete optimization, 5–6

- Dual slack variables, 359
- Dual variables, *see also* Lagrange multipliers, 359
- Duality, 350
 - in linear programming, 359–362
 - in nonlinear programming, 343–349
 - weak, 345, 361
- Eigenvalues, 84, 252, 337, 599, 603, 613
 - negative, 77, 92
 - of symmetric matrix, 604
- Eigenvectors, 84, 252, 603
- Element function, 186
- Elimination of variables, 424
 - linear equality constraints, 428–433
 - nonlinear, 426–428
 - when inequality constraints are present, 434
- Ellipsoid algorithm, 389, 393, 417
- Error
 - absolute, 614
 - relative, 196, 251, 252, 614, 617
 - truncation, 216
- Errors-in-variables models, 265
- Feasibility restoration, 439–440
- Feasible sequences, 316–325, 332–333, 336
 - limiting directions of, 316–325, 329, 333
- Feasible set, 3, 305, 306, 338
 - geometric properties of, 340–341
 - primal, 358
 - primal-dual, 397, 399, 405, 414
- Filter method, 437–440
- Filters, 424, 437–440, 575, 589
 - for interior-point methods, 575
- Finite differencing, 170, 193–204, 216, 268, 278
 - and graph-coloring algorithms, 202–204
 - and noise, 221
 - central-difference formula, 194, 196–197, 202, 217
 - forward-difference formula, 195, 196, 202, 217
 - gradient approximation, 195–197
 - graph-coloring algorithms and, 200–201
 - Hessian approximation, 201–204
 - Jacobian approximation, 197–201, 283
- First-order feasible descent direction, 310–315
- First-order optimality conditions, *see also* Karush–Kuhn–Tucker (KKT) conditions, 90, 275, 307–329, 340, 352
 - derivation of, 315–329
 - examples, 308–315, 317–319, 321–322
 - fundamental principle of, 325–326
 - unconstrained optimization, 14–15, 513
- Fixed-regressor model, 248
- Fletcher–Reeves method, 102, 121–131
 - convergence of, 125
 - numerical performance, 131
- Floating-point arithmetic, 216, 614–615, 617
 - cancellation, 431, 615
 - double-precision, 614
 - roundoff error, 195, 217, 251, 615
 - unit roundoff, 196, 217, 614
- Floating-point numbers, 614
 - exponent, 614
 - fractional part, 614
- Forcing sequence, *see* Newton’s method, inexact, forcing sequence
- Function
 - continuous, 623–624
 - continuously differentiable, 626, 631
 - derivatives of, 625–630
 - differentiable, 626
 - Lipschitz continuous, 624, 630
 - locally Lipschitz continuous, 624
 - one-sided limit, 624
 - univariate, 625
- Functions
 - smooth, 10, 14, 306–307, 330
- Fundamental theorem of algebra, 603
- Gauss–Newton method, 254–258, 263, 266, 275
 - connection to linear least squares, 255
 - line search in, 254
 - performance on large-residual problems, 262
- Gaussian elimination, 51, 430, 455, 609
 - sparse, 430, 433
 - stability of, 617
 - with row partial pivoting, 607, 617

- Global convergence, 77–92, 261, 274
- Global minimizer, 12–13, 16, 17, 502, 503
- Global optimization, 6–8, 422
- Global solution, *see also* Global minimizer, 6, 69–70, 89–91, 305, 335, 352
- GMRES, 278, 459, 492, 571
- Goldstein condition, 36, 48
- Gradient, 625
 - generalized, 18
- Gradient projection method, 464, 485–490, 492, 521
- Group partial separability, *see* Partially separable function, group partially separable
- Hessian, 14, 19, 20, 23, 26, 626
 - average, 138, 140
- Homotopy map, 296
- Homotopy methods, *see* Continuation methods for nonlinear equations
- Implicit filtering, 240–242
- Implicit function theorem, 324, 630–631
- Inexact Newton method, *see* Newton's method, inexact
- Infeasibility measure, 437
- Inner product, 599
- Integer programming, 5, 416
 - branch-and-bound algorithm, 6
- Integral equations, 302
- Interior-point methods, *see* Primal-dual interior-point methods
 - nonlinear, *see* Nonlinear interior-point method
- Interlacing eigenvalue theorem, 613
- Interpolation conditions, 223
- Invariant subspace, *see* Partially separable optimization, invariant subspace
- Iterative refinement, 463
- Jacobian, 246, 254, 256, 269, 274, 324, 395, 504, 627, 630
- Karmarkar's algorithm, 389, 393, 417
- Karush–Kuhn–Tucker (KKT) conditions, 330, 332, 333, 335–337, 339, 350, 354, 503, 517, 520, 528
 - for general constrained problem, 321
 - for linear programming, 358–360, 367, 368, 394–415
 - for linear programming, 394
- KNITRO, 490, 525, 583, 592
- Krylov subspace, 108
 - method, 459
- L-BFGS algorithm, 177–180, 183
- Lagrange multipliers, 310, 330, 333, 337, 339, 341–343, 353, 358, 360, 419, 422
 - estimates of, 503, 514, 515, 518, 521, 522, 584
- Lagrangian function, 90, 310, 313, 320, 329, 330, 336
 - for linear program, 358, 360
 - Hessian of, 330, 332, 333, 335, 337, 358
- LANCELOT, 520, 525, 592
- Lanczos method, 77, 166, 175–176
- LAPACK, 607
- Least-squares multipliers, 581
- Least-squares problems, linear, 250–254
 - normal equations, 250–251, 255, 259, 412
 - sensitivity of solutions, 252
 - solution via QR factorization, 251–252
 - solution via SVD, 252–253
- Least-squares problems, nonlinear, 12, 210
 - applications of, 246–248
 - Dennis–Gay–Welsch algorithm, 263–265
 - Fletcher–Xu algorithm, 263
 - large-residual problems, 262–265
 - large-scale problems, 257
 - scaling of, 260–261
 - software for, 263, 268
 - statistical justification of, 249–250
 - structure, 247, 254
- Least-squares problems, total, 265
- Level set, 92, 261
- Levenberg–Marquardt method, 258–262, 266, 289
 - as trust-region method, 258–259, 292
 - for nonlinear equations, 292
 - implementation via orthogonal transformations, 259–260
 - inexact, 268

- Levenberg–Marquardt (*cont.*)
 - local convergence of, 262
 - performance on large-residual problems, 262
- lim inf, lim sup, 618–619
- Limit point, 28, 79, 92, 99, 502, 503, 618, 620
- Limited-memory method, 25, 176–185, 190
 - compact representation, 181–184
 - for interior-point method, 575, 597
 - L-BFGS, 176–180, 538
 - memoryless BFGS method, 180
 - performance of, 179
 - relation to CG, 180
 - scaling, 178
 - SR1, 183
 - two-loop recursion, 178
- Line search, *see also* Step length selection
 - Armijo, 33, 48, 240
 - backtracking, 37
 - curvature condition, 33
 - Goldstein, 36
 - inexact, 31
 - Newton’s method with, 22–23
 - quasi-Newton methods with, 23–25
 - search directions, 20–25
 - strong Wolfe conditions, *see* Wolfe conditions, strong
 - sufficient decrease, 33
 - Wolfe conditions, *see* Wolfe conditions
- Line search method, 19–20, 30–48, 66, 67, 71, 230–231, 247
 - for nonlinear equations, 271, 285, 287–290
 - global convergence of, 287–288
 - poor performance of, 288–289
- Linear programming, 4, 6, 7, 9, 293
 - artificial variables, 362, 378–380
 - basic feasible points, 362–366
 - basis B , 362–368, 378
 - basis matrix, 363
 - dual problem, 359–362
 - feasible polytope, 356
 - vertices of, 365–366
 - fundamental theorem of, 363–364
 - infeasible, 356, 357
 - nonbasic matrix, 367
 - primal solution set, 356
 - slack and surplus variables, 356, 357, 362, 379, 380
 - splitting variables, 357
 - standard form, 356–357
 - unbounded, 356, 357, 369
 - warm start, 410, 416
- Linearly constrained Lagrangian methods, 522–523, 527
 - MINOS, 523, 527
- Linearly dependent, 337
- Linearly independent, 339, 503, 504, 517, 519, 602
- Lipschitz continuity, *see also* Function, Lipschitz continuous, 80, 93, 256, 257, 261, 269, 276–278, 287, 294
- Local minimizer, 12, 14, 273
 - isolated, 13, 28
 - strict, 13, 14, 16, 28, 517
 - weak, 12
- Local solution, *see also* Local minimizer, 6, 305–306, 316, 325, 329, 332, 340, 342, 352, 513
 - isolated, 306
 - strict, 306, 333, 335, 336
 - strong, 306
- Log-barrier function, 417, 597
 - definition, 583–584
 - difficulty of minimizing, 584–585
 - example, 586
 - ill conditioned Hessian of, 586
- Log-barrier method, 498, 584
- LOQO, 490, 592
- LSQR method, 254, 268, 459, 492, 571
- LU factorization, 606–608
- Maratos effect, 440–446, 543, 550
 - example of, 440, 543
 - remedies, 442
- Matlab, 416
- Matrix
 - condition number, 251, 601–602, 604, 610, 616
 - determinant, 154, 605–606
 - diagonal, 252, 412, 429, 599
 - full-rank, 298, 300, 504, 609
 - identity, 599

- indefinite, 76
- inertia, 55, 454
- lower triangular, 599, 606, 607
- modification, 574
- nonsingular, 325, 337, 601, 612
- null space, 298, 324, 337, 430, 432, 603, 608, 609
- orthogonal, 251, 252, 337, 432, 599, 604, 609
- permutation, 251, 429, 606
- positive definite, 15, 16, 23, 28, 68, 76, 337, 599, 603, 609
- positive semidefinite, 8, 15, 70, 415, 599
- projection, 462
- range space, 430, 603
- rank-deficient, 253
- rank-one, 24
- rank-two, 24
- singular, 337
- sparse, 411, 413, 607
 - Cholesky factorization, 413
- symmetric, 24, 68, 412, 599, 603
- symmetric indefinite, 413
- symmetric positive definite, 608
- trace, 154, 605
- transpose, 599
- upper triangular, 251, 337, 599, 606, 607, 609
- Maximum likelihood estimate, 249
- Mean value theorem, 629–630
- Merit function, *see also* Penalty function, 435–437, 446
 - ℓ_1 , 293, 435–436, 513, 540–543, 550
 - choice of parameter, 543
- exact, 435–436
 - definition of, 435
 - nonsmoothness of, 513
- Fletcher's augmented Lagrangian, 436, 540
- for feasible methods, 435
- for nonlinear equations, 273, 285–287, 289, 290, 293, 296, 301–303, 505
- for SQP, 540–543
- Merit functions, 424, 575
- Method of multipliers, *see* Augmented Lagrangian method
- MINOS, *see also* Linearly constrained Lagrangian methods, 523, 525, 592
- Model-based methods for derivative-free optimization, 223–229
 - minimum Frobenius change, 228
- Modeling, 2, 9, 11, 247–249
- Monomial basis, 227
- MOSEK, 490
- Multiobjective optimization, 437
- Negative curvature direction, 49, 50, 63, 76, 169–172, 175, 489, 491
- Neighborhood, 13, 14, 28, 256, 621
- Network optimization, 358
- Newton's method, 25, 247, 254, 257, 263
 - for log-barrier function, 585
 - for nonlinear equations, 271, 274–277, 281, 283, 285, 287–290, 294, 296, 299, 302
 - cycling, 285
 - inexact, 277–279, 288
 - for quadratic penalty function, 501, 506
 - global convergence, 40
 - Hessian-free, 165, 170
 - in one variable, 84–87, 91, 633
 - inexact, 165–168, 171, 213
 - forcing sequence, 166–169, 171, 277
- large scale
 - LANCELOT, 175
 - line search method, 49
 - TRON, 175
- modified, 48–49
 - adding a multiple of I, 51
 - eigenvalue modification, 49–51
- Newton–CG, 202
 - line search, 168–170
 - preconditioned, 174–175
 - trust-region, 170–175
- Newton–Lanczos, 175–176, 190
- rate of convergence, 44, 76, 92, 166–168, 275–277, 281–282, 620
- scale invariance, 27
- Noise in function evaluation, 221–222
- Nondifferentiable optimization, 511
- Nonlinear equations, 197, 210, 213, 633
 - degenerate solution, 274, 275, 283, 302
 - examples of, 271–272, 288–289, 300–301

- Nonlinear (*cont.*)
 - merit function, *see* Merit function, for
 - nonlinear equations
 - multiple solutions, 273–274
 - nondegenerate solution, 274
 - quasi-Newton methods, *see* Broyden's method
 - relationship to least squares, 271–272, 275, 292–293, 302
 - relationship to optimization, 271
 - relationship to primal-dual interior-point methods, 395
 - solution, 271
 - statement of problem, 270–271
- Nonlinear interior-point method, 423, 563–593
 - barrier formulation, 565
 - feasible version, 576
 - global convergence, 589
 - homotopy formulation, 565
 - superlinear convergence, 591
 - trust-region approach, 578
- Nonlinear least-squares, *see* Least-squares problems, nonlinear
- Nonlinear programming, *see* Constrained optimization, nonlinear
- Nonmonotone strategy, 18, 444–446
 - relaxed steps, 444
- Nonnegative orthant, 97
- Nonsmooth functions, 6, 17–18, 306, 307, 352
- Nonsmooth penalty function, *see* Penalty function, nonsmooth
- Norm
 - dual, 601
 - Euclidean, 25, 51, 251, 280, 302, 600, 601, 605, 610
 - Frobenius, 50, 138, 140, 601
 - matrix, 601–602
 - vector, 600–601
- Normal cone, 340–341
- Normal distribution, 249
- Normal subproblem, 580
- Null space, *see* Matrix, null space
- Numerical analysis, 355
- Objective function, 2, 10, 304
- One-dimensional minimization, 19, 56
- OOPS, 490
- OOQP, 490
- Optimality conditions, *see also* First-order optimality conditions, Second-order optimality conditions, 2, 9, 305
 - for unconstrained local minimizer, 14–17
- Order notation, 631–633
- Orthogonal distance regression, 265–267
 - contrast with least squares, 265–266
 - structure, 266–267
- Orthogonal transformations, 251, 259–260
 - Givens, 259, 609
 - Householder, 259, 609
- Partially separable function, 25, 186–189, 211
 - automatic detection, 211
 - definition, 211
- Partially separable optimization, 165
 - BFGS, 189
 - compactifying matrix, 188
 - element variables, 187
 - quasi-Newton method, 188
 - SR1, 189
- Penalty function, *see also* Merit function, 498
 - ℓ_1 , 507–513
 - exact, 422–423, 507–513
 - nonsmooth, 497, 507–513
 - quadratic, *see also* Quadratic penalty method, 422, 498–507, 525–527, 586
 - difficulty of minimizing, 501–502
 - Hessian of, 505–506
 - relationship to augmented Lagrangian, 514
 - unbounded, 500
- Penalty parameter, 435, 436, 498, 500, 501, 507, 514, 521, 525
 - update, 511, 512
- PENNON, 526
- Pivoting, 251, 617
- Polak–Ribière method, 122
 - convergence of, 130
- Polak–Ribière method
 - numerical performance, 131

- Polynomial bases, 226
 - monomials, 227
- Portfolio optimization, *see* Applications, portfolio optimization
- Preconditioners, 118–120
 - banded, 120
 - constraint, 463
 - for constrained problems, 462
 - for primal-dual system, 571
 - for reduced system, 460
 - incomplete Cholesky, 120
 - SSOR, 120
- Preprocessing, *see* Presolving
- Presolving, 385–388
- Primal interior-point method, 570
- Primal-dual interior-point methods, 389, 597
 - centering parameter, 396, 398, 401, 413
 - complexity of, 393, 406, 415
 - contrasts with simplex method, 356, 393
 - convex quadratic programs, 415
 - corrector step, 414
 - duality measure, 395, 398
 - infeasibility detection, 411
 - linear algebra issues, 411–413
 - Mehrotra's predictor-corrector algorithm, 393, 407–411
 - path-following algorithms, 399–414
 - long-step, 399–406
 - predictor-corrector (Mizuno–Todd–Ye) algorithm, 413
 - short-step, 413
 - potential function, 414
 - Tanabe–Todd–Ye, 414
 - potential-reduction algorithms, 414
 - predictor step, 413
 - quadratic programming, 480–485
 - relationship to Newton's method, 394, 395
 - starting point, 410–411
- Primal-dual system, 567
- Probability density function, 249
- Projected conjugate gradient method, *see* Conjugate gradient method, projected
- Projected Hessian, 558
 - two-sided, 559
- Proximal point method, 523
- QMR method, 459, 492, 571
- QPA, 490
- QPOPT, 490
- QR factorization, 251, 259, 290, 292, 298, 337, 432, 433, 609–610
 - cost of, 609
 - relationship to Cholesky factorization, 610
- Quadratic penalty method, *see also* Penalty function, quadratic, 497, 501–502, 514
 - convergence of, 502–507
- Quadratic programming, 422, 448–492
 - active-set methods, 467–480
 - big M method, 473
 - blocking constraint, 469
 - convex, 449
 - cycling, 477
 - duality, 349, 490
 - indefinite, 449, 467, 491–492
 - inertia controlling methods, 491, 492
 - initial working set, 476
 - interior-point method, 480–485
 - nonconvex, *see* Quadratic programming, indefinite
 - null-space method, 457–459
 - optimal active set, 467
 - optimality conditions, 464
 - phase I, 473
 - Schur-complement method, 455–456
 - software, 490
 - strictly convex, 349, 449, 472, 477–478
 - termination, 477–478
 - updating factorizations, 478
 - working set, 468–478
- Quasi-Newton approximate Hessian, 23, 24, 73, 242, 634
- Quasi-Newton method, 25, 165, 247, 263, 501, 585
 - BFGS, *see* BFGS method, 263
 - bounded deterioration, 161
 - Broyden class, 149–152
 - curvature condition, 137

- Quasi-Newton (*cont.*)
 - DFP, *see* DFP method, 190, 264
 - for interior-point method, 575
 - for nonlinear equations, *see* Broyden's method
 - for partially separable functions, 25
 - global convergence, 40
 - large-scale, 165–189
 - limited memory, *see* Limited memory method
 - rate of convergence, 46, 620
 - secant equation, 24, 137, 139, 263–264, 280, 634
 - sparse, *see* Sparse quasi-Newton method
- Range space, *see* Matrix, range space
- Regularization, 574
- Residuals, 11, 245, 262–265, 269
 - preconditioned, 462
 - vector of, 18, 197, 246
- Restoration phase, 439
- Robust optimization, 7
- Root, *see* Nonlinear equations, solution
- Rootfinding algorithm, *see also* Newton's method, in one variable, 259, 260, 633
 - for trust-region subproblem, 84–87
- Rosenbrock function
 - extended, 191
- Roundoff error, *see* Floating-point arithmetic, roundoff error
- Row echelon form, 430
- $S\ell_1$ QP method, 293, 549
- Saddle point, 28, 92
- Scale invariance, 27, 138, 141
 - of Newton's method, *see* Newton's method, scale invariance
- Scaling, 26–27, 95–97, 342–343, 585
 - example of poor scaling, 26–27
 - matrix, 96
- Schur complement, 456, 611
- Secant method, *see also* Quasi-Newton method, 280, 633, 634
- Second-order correction, 442–444, 550
- Second-order optimality conditions, 330–337, 342, 602
 - for unconstrained optimization, 15–16
 - necessary, 92, 331
 - sufficient, 333–336, 517, 557
- Semidefinite programming, 415
- Sensitivity, 252, 616
- Sensitivity analysis, 2, 194, 341–343, 350, 361
- Separable function, 186
- Separating hyperplane, 327
- Sequential linear-quadratic programming (SLQP), 293, 423, 534
- Sequential quadratic programming, 423, 512, 523, 529–560
 - Byrd–Omojokun method, 547
 - derivation, 530–533
 - full quasi-Newton Hessian, 536
 - identification of optimal active set, 533
 - IQP vs. EQP, 533
 - KKT system, 275
 - least-squares multipliers, 539
 - line search algorithm, 545
 - local algorithm, 532
 - Newton–KKT system, 531
 - null-space, 538
 - QP multipliers, 538
 - rate of convergence, 557–560
 - reduced-Hessian approximation, 538–540
 - relaxation constraints, 547
 - $S\ell_1$ QP method, *see* $S\ell_1$ QP method
 - step computation, 545
 - trust-region method, 546–549
 - warm start, 545
- Set
 - affine, 622
 - affine hull of, 622
 - bounded, 620
 - closed, 620
 - closure of, 621
 - compact, 621
 - interior of, 621
 - open, 620
 - relative interior of, 622, 623
- Sherman–Morrison–Woodbury formula, 139, 140, 144, 162, 283, 377, 612–613
- Simplex method
 - as active-set method, 388

- basis \mathcal{B} , 365
- complexity of, 388–389
- cycling, 381–382
 - lexicographic strategy, 382
 - perturbation strategy, 381–382
- degenerate steps, 372, 381
- description of single iteration, 366–372
- discovery of, 355
- dual simplex, 366, 382–385
- entering index, 368, 370, 372, 375–378
- finite termination of, 368–370
- initialization, 378–380
- leaving index, 368, 370
- linear algebra issues, 372–375
- Phase I/Phase II, 378–380
- pivoting, 368
- pricing, 368, 370, 375–376
 - multiple, 376
 - partial, 376
- reduced costs, 368
- revised, 366
- steepest-edge rule, 376–378
- Simulated annealing, 221
- Singular values, 255, 604
- Singular-value decomposition (SVD), 252, 269, 303, 603–604
- Slack variables, *see also* Linear programming, slack/surplus variables, 424, 519
- SNOPT, 536, 592
- Software
 - BQPD, 490
 - CPLEX, 490
 - for quadratic programming, 490
 - IPOPT, 183, 592
 - KNITRO, 183, 490, 525, 592
 - L-BFGS-B, 183
 - LANCELOT, 520, 525, 592
 - LOQO, 490, 592
 - MINOS, 523, 525, 592
 - MOSEK, 490
 - OOPS, 490
 - OOQP, 490
 - PENNON, 526
 - QPA, 490
 - QPOPT, 490
 - SNOPT, 592
 - TRON, 175
 - VE09, 490
 - XPRESS-MP, 490
- Sparse quasi-Newton method, 185–186, 190
- SR1 method, 24, 144, 161
 - algorithm, 146
 - for constrained problems, 538, 540
 - limited-memory version, 177, 181, 183
 - properties, 147
 - safeguarding, 145
 - skipping, 145, 160
- Stability, 616–617
- Starting point, 18
- Stationary point, 15, 28, 289, 436, 505
- Steepest descent direction, 20, 21, 71, 74
- Steepest descent method, 21, 25–27, 31, 73, 95, 585
 - rate of convergence, 42, 44, 620
- Step length, 19, 30
 - unit, 23, 29
- Step length selection, *see also* Line search, 56–62
 - bracketing phase, 57
 - cubic interpolation, 59
 - for Wolfe conditions, 60
 - initial step length, 59
 - interpolation in, 57
 - selection phase, 57
- Stochastic optimization, 7
- Stochastic simulation, 221
- Strict complementarity, *see* Complementarity condition, strict
- Subgradient, 17
- Subspace, 602
 - basis, 430, 603
 - orthonormal, 432
 - dimension, 603
 - spanning set, 603
- Sufficient reduction, 71, 73, 79
- Sum of absolute values, 249
- Sum of squares, *see* Least-squares problems, nonlinear
- Symbolic differentiation, 194
- Symmetric indefinite factorization, 455, 570, 610–612
- Bunch–Kaufman, 612

- Symmetric (*cont.*)
 - Bunch–Parlett, 611
 - modified, 54–56, 63
 - sparse, 612
- Symmetric rank-one update, *see* SR1 method
- Tangent, 315–325
- Tangent cone, 319, 340–341
- Taylor series, 15, 22, 28, 29, 67, 274, 309, 330, 332, 334, 502
- Taylor’s theorem, 15, 21–23, 80, 123, 138, 167, 193–195, 197, 198, 202, 274, 280, 294, 323, 325, 332, 334, 341, 630
 - statement of, 14
- Tensor methods, 274
 - derivation, 283–284
- Termination criterion, 92
- Triangular substitution, 433, 606, 609, 617
- Truncated Newton method, *see* Newton’s method, Newton-CG, line-search
- Trust region
 - boundary, 69, 75, 95, 171–173
 - box-shaped, 19, 293
 - choice of size for, 67, 81
 - elliptical, 19, 67, 95, 96, 100
 - radius, 20, 26, 68, 69, 73, 258, 294
 - spherical, 95, 258
- Trust-region method, 19–20, 69, 77, 79, 80, 82, 87, 91, 247, 258, 633
 - contrast with line search method, 20, 66–67
 - dogleg method, 71, 73–77, 79, 84, 91, 95, 99, 173, 291–293, 548
 - double-dogleg method, 99
 - for derivative-free optimization, 225
 - for nonlinear equations, 271, 273, 285, 290–296
 - global convergence of, 292–293
 - local convergence of, 293–296
 - global convergence, 71, 73, 76–92, 172
 - local convergence, 92–95
 - Newton variant, 26, 68, 92
 - software, 98
 - Steihaug’s approach, 77, 170–173, 489
 - strategy for adjusting radius, 69
 - subproblem, 19, 25–26, 68, 69, 72, 73, 76, 77, 91, 95–97, 258
 - approximate solution of, 68, 71
 - exact solution of, 71, 77, 79, 83–92
 - hard case, 87–88
 - nearly exact solution of, 95, 292–293
 - two-dimensional subspace
 - minimization, 71, 76–77, 79, 84, 95, 98, 100
- Unconstrained optimization, 6, 352, 427, 432, 499, 501
 - of barrier function, 584
- Unit ball, 91
- Unit roundoff, *see* Floating-point arithmetic, unit roundoff
- Variable metric method, *see* Quasi-Newton method
- Variable storage method, *see* Limited memory method
- VE09, 490
- Watchdog technique, 444–446
- Weakly active constraints, 342
- Wolfe conditions, 33–36, 48, 78, 131, 137, 138, 140–143, 146, 160, 179, 255, 287, 290
 - scale invariance of, 36
 - strong, 34, 35, 122, 125, 126, 128, 131, 138, 142, 162, 179
- XPRESS-MP, 490
- Zoutendijk condition, 38–41, 128, 156, 287