Iterative Methods for Optimization

C.T. Kelley

North Carolina State University

Raleigh, North Carolina

Society for Industrial and Applied Mathematics Philadelphia

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Contents**

**Preface xiii How to Get the Software xv**

**I Optimization of Smooth Functions 1**

**1 Basic Concepts 3** 1.1 The Problem .................................... 3 1.2 Notation ...................................... 4 1.3 Necessary Conditions ............................... 5 1.4 Sufficient Conditions ............................... 6 1.5 Quadratic Objective Functions .......................... 6 1.5.1 Positive Definite Hessian ......................... 7

1.5.2 Indefinite Hessian ............................. 9

1.6 Examples ..................................... 9 1.6.1 Discrete Optimal Control ......................... 9

1.6.2 Parameter Identification . . . . . . . . . . . . . . . . . . . . . . . . . 11

1.6.3 Convex Quadratics . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12 1.7 Exercises on Basic Concepts . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**2 Local Convergence of Newton’s Method 13** 2.1 Types of Convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13 2.2 The Standard Assumptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 2.3 Newton’s Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3.1 Errors in Functions, Gradients, and Hessians . . . . . . . . . . . . . . 17 2.3.2 Termination of the Iteration . . . . . . . . . . . . . . . . . . . . . . . 21 2.4 Nonlinear Least Squares . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22 2.4.1 Gauss–Newton Iteration . . . . . . . . . . . . . . . . . . . . . . . . . 23 2.4.2 Overdetermined Problems . . . . . . . . . . . . . . . . . . . . . . . . 24 2.4.3 Underdetermined Problems . . . . . . . . . . . . . . . . . . . . . . . 25 2.5 Inexact Newton Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28 2.5.1 Convergence Rates . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29 2.5.2 Implementation of Newton–CG . . . . . . . . . . . . . . . . . . . . . 30 2.6 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33 2.6.1 Parameter Identification . . . . . . . . . . . . . . . . . . . . . . . . . 33 2.6.2 Discrete Control Problem . . . . . . . . . . . . . . . . . . . . . . . . 34 2.7 Exercises on Local Convergence . . . . . . . . . . . . . . . . . . . . . . . . . 35

ix

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

x CONTENTS

**3 Global Convergence 39** 3.1 The Method of Steepest Descent . . . . . . . . . . . . . . . . . . . . . . . . . 39 3.2 Line Search Methods and the Armijo Rule . . . . . . . . . . . . . . . . . . . . 40

3.2.1 Stepsize Control with Polynomial Models . . . . . . . . . . . . . . . . 43 3.2.2 Slow Convergence of Steepest Descent . . . . . . . . . . . . . . . . . 45 3.2.3 Damped Gauss–Newton Iteration . . . . . . . . . . . . . . . . . . . . 47 3.2.4 Nonlinear Conjugate Gradient Methods . . . . . . . . . . . . . . . . . 48

3.3 Trust Region Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50 3.3.1 Changing the Trust Region and the Step . . . . . . . . . . . . . . . . . 51 3.3.2 Global Convergence of Trust Region Algorithms . . . . . . . . . . . . 52 3.3.3 A Unidirectional Trust Region Algorithm . . . . . . . . . . . . . . . . 54 3.3.4 The Exact Solution of the Trust Region Problem . . . . . . . . . . . . 55 3.3.5 The Levenberg–Marquardt Parameter . . . . . . . . . . . . . . . . . . 56 3.3.6 Superlinear Convergence: The Dogleg . . . . . . . . . . . . . . . . . . 58 3.3.7 A Trust Region Method for Newton–CG . . . . . . . . . . . . . . . . . 63

3.4 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65 3.4.1 Parameter Identification . . . . . . . . . . . . . . . . . . . . . . . . . 67 3.4.2 Discrete Control Problem . . . . . . . . . . . . . . . . . . . . . . . . 68 3.5 Exercises on Global Convergence . . . . . . . . . . . . . . . . . . . . . . . . 68

**4 The BFGS Method 71** 4.1 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72 4.1.1 Local Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72 4.1.2 Global Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77 4.2 Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78 4.2.1 Storage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78 4.2.2 A BFGS–Armijo Algorithm . . . . . . . . . . . . . . . . . . . . . . . 80 4.3 Other Quasi-Newton Methods . . . . . . . . . . . . . . . . . . . . . . . . . . 81 4.4 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83 4.4.1 Parameter ID Problem . . . . . . . . . . . . . . . . . . . . . . . . . . 83 4.4.2 Discrete Control Problem . . . . . . . . . . . . . . . . . . . . . . . . 83 4.5 Exercises on BFGS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

**5 Simple Bound Constraints 87** 5.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87 5.2 Necessary Conditions for Optimality . . . . . . . . . . . . . . . . . . . . . . . 87 5.3 Sufficient Conditions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89 5.4 The Gradient Projection Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 91

5.4.1 Termination of the Iteration . . . . . . . . . . . . . . . . . . . . . . . 91 5.4.2 Convergence Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . 93 5.4.3 Identification of the Active Set . . . . . . . . . . . . . . . . . . . . . . 95 5.4.4 A Proof of Theorem 5.2.4 . . . . . . . . . . . . . . . . . . . . . . . . 96

5.5 Superlinear Convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96 5.5.1 The Scaled Gradient Projection Algorithm . . . . . . . . . . . . . . . . 96 5.5.2 The Projected Newton Method . . . . . . . . . . . . . . . . . . . . . . 100 5.5.3 A Projected BFGS–Armijo Algorithm . . . . . . . . . . . . . . . . . . 102

5.6 Other Approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104 5.6.1 Infinite-Dimensional Problems . . . . . . . . . . . . . . . . . . . . . . 106 5.7 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106 5.7.1 Parameter ID Problem . . . . . . . . . . . . . . . . . . . . . . . . . . 106 5.7.2 Discrete Control Problem . . . . . . . . . . . . . . . . . . . . . . . . 106

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

CONTENTS xi 5.8 Exercises on Bound Constrained Optimization . . . . . . . . . . . . . . . . . . 108

**II Optimization of Noisy Functions 109**

**6 Basic Concepts and Goals 111** 6.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112 6.2 The Simplex Gradient . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112

6.2.1 Forward Difference Simplex Gradient . . . . . . . . . . . . . . . . . . 113 6.2.2 Centered Difference Simplex Gradient . . . . . . . . . . . . . . . . . . 115 6.3 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118 6.3.1 Weber’s Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118 6.3.2 Perturbed Convex Quadratics . . . . . . . . . . . . . . . . . . . . . . 119 6.3.3 Lennard–Jones Problem . . . . . . . . . . . . . . . . . . . . . . . . . 120 6.4 Exercises on Basic Concepts . . . . . . . . . . . . . . . . . . . . . . . . . . . 121

**7 Implicit Filtering 123** 7.1 Description and Analysis of Implicit Filtering . . . . . . . . . . . . . . . . . . 123 7.2 Quasi-Newton Methods and Implicit Filtering . . . . . . . . . . . . . . . . . . 124 7.3 Implementation Considerations . . . . . . . . . . . . . . . . . . . . . . . . . . 125 7.4 Implicit Filtering for Bound Constrained Problems . . . . . . . . . . . . . . . 126 7.5 Restarting and Minima at All Scales . . . . . . . . . . . . . . . . . . . . . . . 127 7.6 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127

7.6.1 Weber’s Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127 7.6.2 Parameter ID . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 129 7.6.3 Convex Quadratics . . . . . . . . . . . . . . . . . . . . . . . . . . . . 129 7.7 Exercises on Implicit Filtering . . . . . . . . . . . . . . . . . . . . . . . . . . 133

**8 Direct Search Algorithms 135** 8.1 The Nelder–Mead Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 135 8.1.1 Description and Implementation . . . . . . . . . . . . . . . . . . . . . 135 8.1.2 Sufficient Decrease and the Simplex Gradient . . . . . . . . . . . . . . 137 8.1.3 McKinnon’s Examples . . . . . . . . . . . . . . . . . . . . . . . . . . 139 8.1.4 Restarting the Nelder–Mead Algorithm . . . . . . . . . . . . . . . . . 141

8.2 Multidirectional Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143 8.2.1 Description and Implementation . . . . . . . . . . . . . . . . . . . . . 143 8.2.2 Convergence and the Simplex Gradient . . . . . . . . . . . . . . . . . 144

8.3 The Hooke–Jeeves Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 145 8.3.1 Description and Implementation . . . . . . . . . . . . . . . . . . . . . 145 8.3.2 Convergence and the Simplex Gradient . . . . . . . . . . . . . . . . . 148

8.4 Other Approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 148 8.4.1 Surrogate Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 148 8.4.2 The DIRECT Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . 149

8.5 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152 8.5.1 Weber’s Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152 8.5.2 Parameter ID . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153 8.5.3 Convex Quadratics . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153 8.6 Exercises on Search Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . 159

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

xii CONTENTS **Bibliography 161**

**Index 177** Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Preface**

This book on unconstrained and bound constrained optimization can be used as a tutorial for self-study or a reference by those who solve such problems in their work. It can also serve as a textbook in an introductory optimization course.

As in my earlier book [154] on linear and nonlinear equations, we treat a small number of methods in depth, giving a less detailed description of only a few (for example, the nonlinear conjugate gradient method and the DIRECT algorithm). We aim for clarity and brevity rather than complete generality and confine our scope to algorithms that are easy to implement (by the reader!) and understand.

One consequence of this approach is that the algorithms in this book are often special cases of more general ones in the literature. For example, in Chapter 3, we provide details only for trust region globalizations of Newton’s method for unconstrained problems and line search globalizations of the BFGS quasi-Newton method for unconstrained and bound constrained problems. We refer the reader to the literature for more general results. Our intention is that both our algorithms and proofs, being special cases, are more concise and simple than others in the literature and illustrate the central issues more clearly than a fully general formulation.

Part II of this book covers some algorithms for noisy or global optimization or both. There are many interesting algorithms in this class, and this book is limited to those deterministic algorithms that can be implemented in a more-or-less straightforward way. We do not, for example, cover simulated annealing, genetic algorithms, response surface methods, or random search procedures.

The reader of this book should be familiar with the material in an elementary graduate level course in numerical analysis, in particular direct and iterative methods for the solution of linear equations and linear least squares problems. The material in texts such as [127] and [264] is sufficient.

A suite of MATLAB*∗* codes has been written to accompany this book. These codes were used to generate the computational examples in the book, but the algorithms do not depend on the MATLAB environment and the reader can easily implement the algorithms in another language, either directly from the algorithmic descriptions or by translating the MATLAB code. The MATLAB environment is an excellent choice for experimentation, doing the exercises, and small-to-medium-scale production work. Large-scale work on high-performance computers is best done in another language. The reader should also be aware that there is a large amount of high-quality software available for optimization. The book [195], for example, provides pointers to several useful packages.

Parts of this book are based upon work supported by the National Science Foundation over several years, most recently under National Science Foundation grants DMS-9321938, DMS 9700569, and DMS-9714811, and by allocations of computing resources from the North Carolina Supercomputing Center. Any opinions, findings, and conclusions or recommendations expressed

*∗*MATLAB is a registered trademark of The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760, USA, (508) 653-1415, info@mathworks.com, http://www.mathworks.com.

xiii

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

xiv PREFACE

in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or of the North Carolina Supercomputing Center.

The list of students and colleagues who have helped me with this project, directly, through collaborations/discussions on issues that I treat in the manuscript, by providing pointers to the literature, or as a source of inspiration, is long. I am particularly indebted to Tom Banks, Jim Banoczi, John Betts, David Bortz, Steve Campbell, Tony Choi, Andy Conn, Douglas Cooper, Joe David, John Dennis, Owen Eslinger, J¨org Gablonsky, Paul Gilmore, Matthias Heinkenschloß, Laura Helfrich, Lea Jenkins,Vickie Kearn, Carl and Betty Kelley, Debbie Lockhart, Casey Miller, Jorge Mor´e, Mary Rose Muccie, John Nelder, Chung-Wei Ng, Deborah Poulson, Ekkehard Sachs, Dave Shanno, Joseph Skudlarek, Dan Sorensen, John Strikwerda, Mike Tocci, Jon Tolle, Virginia Torczon, Floria Tosca, Hien Tran, Margaret Wright, Steve Wright, and Kevin Yoemans.

C. T. Kelley

Raleigh, North Carolina

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**How to Get the Software**

All computations reported in this book were done in MATLAB (version 5.2 on various SUN SPARCstations and on an Apple Macintosh Powerbook 2400). The suite of MATLAB codes that we used for the examples is available by anonymous ftp from ftp.math.ncsu.edu in the directory

FTP/kelley/optimization/matlab

or from SIAM’s World Wide Web server at

http://www.siam.org/books/fr18/

One can obtain MATLAB from

The MathWorks, Inc.

3 Apple Hill Drive

Natick, MA 01760-2098

(508) 647-7000

Fax: (508) 647-7001

E-mail: info@mathworks.com

WWW: http://www.mathworks.com

xv

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Part I**

**Optimization of Smooth Functions** Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Chapter 1**

**Basic Concepts**

**1.1 The Problem**

The unconstrained optimization problem is to minimize a real-valued function *f* of *N* variables. By this we mean to find a *local minimizer*, that is, a point *x∗* such that

*f*(*x∗*) *≤ f*(*x*) for all *x* near *x∗* (1.1) .

It is standard to express this problem as

min*x* (1.2) *f*(*x*)

or to say that we seek to solve the problem min *f*. The understanding is that (1.1) means that we seek a local minimizer. We will refer to *f* as the *objective function* and to *f*(*x∗*) as the *minimum* or *minimum value*. If a local minimizer *x∗* exists, we say a *minimum is attained* at *x∗*.

We say that problem (1.2) is *unconstrained* because we impose no conditions on the inde pendent variables *x* and assume that *f* is defined for all *x*.

The local minimization problem is different from (and much easier than) the *global mini mization problem* in which a *global minimizer*, a point *x∗* such that

*f*(*x∗* (1.3) ) *≤ f*(*x*) for all *x*,

is sought.

The *constrained* optimization problem is to minimize a function *f* over a set *U ⊂ RN* . A local minimizer, therefore, is an *x∗ ∈ U* such that

*f*(*x∗*) *≤ f*(*x*) for all *x ∈ U* near *x∗* (1.4) .

Similar to (1.2) we express this as

min

*x∈U* (1.5) *f*(*x*)

or say that we seek to solve the problem min*U f*. A global minimizer is a point *x∗ ∈ U* such that

*f*(*x∗* (1.6) ) *≤ f*(*x*) for all *x ∈ U*.

We consider only the simplest constrained problems in this book (Chapter 5 and *§*7.4) and refer the reader to [104], [117], [195], and [66] for deeper discussions of constrained optimization and pointers to software.

Having posed an optimization problem one can proceed in the classical way and use methods that require smoothness of *f*. That is the approach we take in this first part of the book. These

3

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

4 ITERATIVE METHODS FOR OPTIMIZATION

methods can fail if the objective function has discontinuities or irregularities. Such nonsmooth effects are common and can be caused, for example, by truncation error in internal calculations for *f*, noise in internal probabilistic modeling in *f*, table lookup within *f*, or use of experimental data in *f*. We address a class of methods for dealing with such problems in Part II.

**1.2 Notation**

In this book, following the convention in [154], vectors are to be understood as column vectors. The vector *x∗* will denote a solution, *x* a potential solution, and *{xk}k≥*0 the sequence of iterates. We will refer to *x*0 as the *initial iterate*. *x*0 is sometimes timidly called the *initial guess*. We will denote the *i*th component of a vector *x* by (*x*)*i* (note the parentheses) and the *i*th component of *xk* by (*xk*)*i*. We will rarely need to refer to individual components of vectors. We will let *∂f /∂xi* denote the partial derivative of *f* with respect to (*x*)*i*. As is standard [154], *e* = *x − x∗* will denote the error, *en* = *xn − x∗* the error in the *n*th iterate, and *B*(*r*) the ball of radius *r* about *x∗*

*B*(*r*) = *{x | e < r}.*

For *x ∈ RN* we let *∇f*(*x*) *∈ RN* denote the *gradient* of *f*,

*∇f*(*x*)=(*∂f /∂x*1*, . . . , ∂f /∂xN* )*,*

when it exists.

We let *∇*2*f* denote the *Hessian* of *f*,

(*∇*2*f*)*ij* = *∂*2*f /∂xi∂xj ,*

when it exists. Note that *∇*2*f* is the Jacobian of *∇f*. However, *∇*2*f* has more structure than a Jacobian for a general nonlinear function. If *f* is twice continuously differentiable, then the Hessian is symmetric ((*∇*2*f*)*ij* = (*∇*2*f*)*ji*) by equality of mixed partial derivatives [229]. In this book we will consistently use the Euclidean norm

*N*

*x* = *i*=1

(*x*)2*i .*

When we refer to a matrix norm we will mean the matrix norm induced by the Euclidean norm *A* = max *x*=0*Axx .*

In optimization definiteness or semidefiniteness of the Hessian plays an important role in the necessary and sufficient conditions for optimality that we discuss in *§*1.3 and 1.4 and in our choice of algorithms throughout this book.

Definition 1.2.1. *An N ×N matrix A is* positive semidefinite *if xT Ax ≥* 0 *for all x ∈ RN . A is* positive definite *if xT Ax >* 0 *for all x ∈ RN , x* = 0*. If A has both positive and negative eigenvalues, we say A is* indefinite*. If A is symmetric and positive definite, we will say A is* spd*.*

We will use two forms of the *fundamental theorem of calculus*, one for the function–gradient pair and one for the gradient–Hessian.

Theorem 1.2.1. *Let f be twice continuously differentiable in a neighborhood of a line segment between points x∗, x* = *x∗* + *e ∈ RN ; then*

1

*f*(*x*) = *f*(*x∗*) + 0

*∇f*(*x∗* + *te*)*T e dt*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. BASIC CONCEPTS 5

*and*

*∇f*(*x*) = *∇f*(*x∗*) +

1 0

*∇*2*f*(*x∗* + *te*)*e dt.*

A direct consequence (see Exercise 1.7.1) of Theorem 1.2.1 is the following form of *Taylor’s theorem* we will use throughout this book.

Theorem 1.2.2. *Let f be twice continuously differentiable in a neighborhood of a point x∗ ∈ RN . Then for e ∈ RN and e sufficiently small*

*f*(*x∗* + *e*) = *f*(*x∗*) + *∇f*(*x∗*)*T e* + *eT ∇*2*f*(*x∗*)*e/*2 + *o*(*e*2 (1.7) )*.*

**1.3 Necessary Conditions**

Let *f* be twice continuously differentiable. We will use Taylor’s theorem in a simple way to show that the gradient of *f* vanishes at a local minimizer and the Hessian is positive semidefinite. These are the *necessary conditions* for optimality.

The necessary conditions relate (1.1) to a nonlinear equation and allow one to use fast al gorithms for nonlinear equations [84], [154], [211] to compute minimizers. Therefore, the necessary conditions for optimality will be used in a critical way in the discussion of local con vergence in Chapter 2. A critical first step in the design of an algorithm for a new optimization problem is the formulation of necessary conditions. Of course, the gradient vanishes at a maxi mum, too, and the utility of the nonlinear equations formulation is restricted to a neighborhood of a minimizer.

Theorem 1.3.1. *Let f be twice continuously differentiable and let x∗ be a local minimizer of f. Then*

*∇f*(*x∗*)=0*.*

*Moreover ∇*2*f*(*x∗*) *is positive semidefinite.*

*Proof*. Let *u ∈ RN* be given. Taylor’s theorem states that for all real *t* sufficiently small *f*(*x∗* + *tu*) = *f*(*x∗*) + *t∇f*(*x∗*)*T u* +*t*22*uT ∇*2*f*(*x∗*)*u* + *o*(*t*2)*.*

Since *x∗* is a local minimizer we must have for *t* sufficiently small 0 *≤ f*(*x∗* + *tu*) *− f*(*x∗*) and hence

*∇f*(*x∗*)*T u* +*t*2*uT ∇*2*f*(*x∗* (1.8) )*u* + *o*(*t*) *≥* 0

for all *t* sufficiently small and all *u ∈ RN* . So if we set *t* = 0 and *u* = *−∇f*(*x∗*) we obtain *∇f*(*x∗*)2 = 0*.*

Setting *∇f*(*x∗*)=0, dividing by *t*, and setting *t* = 0 in (1.8), we obtain

1

2*uT ∇*2*f*(*x∗*)*u ≥* 0

for all *u ∈ RN* . This completes the proof.

The condition that *∇f*(*x∗*)=0 is called the *first-order necessary condition* and a point satisfying that condition is called a *stationary point* or a *critical point*.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

6 ITERATIVE METHODS FOR OPTIMIZATION **1.4 Sufficient Conditions**

A stationary point need not be a minimizer. For example, the function *φ*(*t*) = *−t*4 satisfies the necessary conditions at 0, which is a maximizer of *φ*. To obtain a minimizer we must require that the second derivative be nonnegative. This alone is not sufficient (think of *φ*(*t*) = *t*3) and only if the second derivative is strictly positive can we be completely certain. These are the *sufficient conditions* for optimality.

Theorem 1.4.1. *Let f be twice continuously differentiable in a neighborhood of x∗. Assume that ∇f*(*x∗*)=0 *and that ∇*2*f*(*x∗*) *is positive definite. Then x∗ is a local minimizer of f.*

*Proof*. Let 0 = *u ∈ RN* . For sufficiently small *t* we have

*f*(*x∗* + *tu*) = *f*(*x∗*) + *t∇f*(*x∗*)*T u* +*t*22*uT ∇*2*f*(*x∗*)*u* + *o*(*t*2)

= *f*(*x∗*) + *t*22*uT ∇*2*f*(*x∗*)*u* + *o*(*t*2)*.*

Hence, if *λ >* 0 is the smallest eigenvalue of *∇*2*f*(*x∗*) we have

*f*(*x∗* + *tu*) *− f*(*x∗*) *≥λ*2*tu*2 + *o*(*t*2) *>* 0

for *t* sufficiently small. Hence *x∗* is a local minimizer.

**1.5 Quadratic Objective Functions**

The simplest optimization problems are those with *quadratic objective functions*. Here *f*(*x*) = *−xT b* +12*x* (1.9) *T Hx.*

We may, without loss of generality, assume that *H* is symmetric because

*xT Hx* = *xTH* + *HT*

(1.10) *x.*

2

Quadratic functions form the basis for most of the algorithms in Part I, which approximate an objective function *f* by a *quadratic model* and minimize that model. In this section we discuss some elementary issues in quadratic optimization.

Clearly,

*∇*2*f*(*x*) = *H*

for all *x*. The symmetry of *H* implies that

*∇f*(*x*) = *−b* + *Hx.*

Definition 1.5.1. *The quadratic function f in* (1.9) *is* convex *if H is positive semidefinite.* Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

BASIC CONCEPTS 7

**1.5.1 Positive Definite Hessian**

The necessary conditions for optimality imply that if a quadratic function *f* has a local minimum *x∗*, then *H* is positive semidefinite and

*Hx∗* (1.11) = *b.*

In particular, if *H* is spd (and hence nonsingular), the unique global minimizer is the solution of the linear system (1.11).

If *H* is a dense matrix and *N* is not too large, it is reasonable to solve (1.11) by computing the *Cholesky factorization* [249], [127] of *H*

*H* = *LLT ,*

where *L* is a nonsingular lower triangular matrix with positive diagonal, and then solving (1.11) by two triangular solves. If *H* is indefinite the Cholesky factorization will not exist and the standard implementation [127], [249], [264] will fail because the computation of the diagonal of *L* will require a real square root of a negative number or a division by zero.

If *N* is very large, *H* is sparse, or a matrix representation of *H* is not available, a more efficient approach is the *conjugate gradient* iteration [154], [141]. This iteration requires only matrix–vector products, a feature which we will use in a direct way in *§§*2.5 and 3.3.7. Our formulation of the algorithm uses *x* as both an input and output variable. On input *x* contains *x*0, the initial iterate, and on output the approximate solution. We terminate the iteration if the relative residual is sufficiently small, i.e.,

*b − Hx ≤ b*

or if too many iterations have been taken.

Algorithm 1.5.1. cg(*x, b, H, , kmax*)

1. *r* = *b − Hx*, *ρ*0 = *r*2, *k* = 1.

2. *Do While √~~ρ~~k−*1 *> b and k < kmax*

(a) *if k* = 1 *then p* = *r*

*else*

*β* = *ρk−*1*/ρk−*2 *and p* = *r* + *βp*

(b) *w* = *Hp*

(c) *α* = *ρk−*1*/pT w*

(d) *x* = *x* + *αp*

(e) *r* = *r − αw*

(f) *ρk* = *r*2

(g) *k* = *k* + 1

Note that if *H* is not spd, the denominator in *α* = *ρk−*1*/pT w* may vanish, resulting in *breakdown* of the iteration.

The conjugate gradient iteration minimizes *f* over an increasing sequence of nested subspaces of *RN* [127], [154]. We have that

*f*(*xk*) *≤ f*(*x*) for all *x ∈ x*0 + *Kk,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

8 ITERATIVE METHODS FOR OPTIMIZATION

where *Kk* is the *Krylov subspace*

*Kk* = span(*r*0*,Hr*0*,...,Hk−*1*r*0)

for *k ≥* 1.

While in principle the iteration must converge after *N* iterations and conjugate gradient can be regarded as a direct solver, *N* is, in practice, far too many iterations for the large problems to which conjugate gradient is applied. As an iterative method, the performance of the conjugate gradient algorithm depends both on *b* and on the spectrum of *H* (see [154] and the references cited therein). A general convergence estimate [68], [60], which will suffice for the discussion here, is

*k*

*xk − x∗H ≤* 2*x*0 *− x∗Hκ*(*H*) *−* 1

(1.12) *. κ*(*H*)+1

In (1.12), the *H*-norm of a vector is defined by

*u*2*H* = *uT Hu*

for an spd matrix *H*. *κ*(*H*) is the *l*2 condition number

*κ*(*H*) = *HH−*1*.*

For spd *H*

*κ*(*H*) = *λlλ−*1

*s ,*

where *λl* and *λs* are the largest and smallest eigenvalues of *H*. Geometrically, *κ*(*H*) is large if the ellipsoidal level surfaces of *f* are very far from spherical.

The conjugate gradient iteration will perform well if *κ*(*H*) is near 1 and may perform very poorly if *κ*(*H*)is large. The performance can be improved by *preconditioning*, which transforms (1.11) into one with a coefficient matrix having eigenvalues near 1. Suppose that *M* is spd and a sufficiently good approximation to *H−*1 so that

*κ*(*M*1*/*2*HM*1*/*2)

is much smaller that *κ*(*H*). In that case, (1.12) would indicate that far fewer conjugate gradient iterations might be needed to solve

*M*1*/*2*HM*1*/*2*z* = *M*1*/*2 (1.13) *b*

than to solve (1.11). Moreover, the solution *x∗* of (1.11) could be recovered from the solution *z∗* of (1.13) by

*x* = *M*1*/*2 (1.14) *z.*

In practice, the square root of the preconditioning matrix *M* need not be computed. The algo rithm, using the same conventions that we used for cg, is

Algorithm 1.5.2. pcg(*x, b, H, M, , kmax*)

1. *r* = *b − Hx*, *ρ*0 = *r*2, *k* = 1

2. *Do While √~~ρ~~k−*1 *> b and k < kmax*

(a) *z* = *Mr*

(b) *τk−*1 = *zT r*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

BASIC CONCEPTS 9

(c) *if k* = 1 *then β* = 0 *and p* = *z*

*else*

*β* = *τk−*1*/τk−*2, *p* = *z* + *βp*

(d) *w* = *Hp*

(e) *α* = *τk−*1*/pT w*

(f) *x* = *x* + *αp*

(g) *r* = *r − αw*

(h) *ρk* = *rT r*

(i) *k* = *k* + 1

Note that only products of *M* with vectors in *RN* are needed and that a matrix representation of *M* need not be stored. We refer the reader to [11], [15], [127], and [154] for more discussion of preconditioners and their construction.

**1.5.2 Indefinite Hessian**

If *H* is indefinite, the necessary conditions, Theorem 1.3.1, imply that there will be no local minimum. Even so, it will be important to understand some properties of quadratic problems with indefinite Hessians when we design algorithms with initial iterates far from local minimizers and we discuss some of the issues here.

If

*uT Hu <* 0*,*

we say that *u* is a *direction of negative curvature*. If *u* is a direction of negative curvature, then *f*(*x* + *tu*) will decrease to *−∞* as *t → ∞*.

**1.6 Examples**

It will be useful to have some example problems to solve as we develop the algorithms. The examples here are included to encourage the reader to experiment with the algorithms and play with the MATLAB codes. The codes for the problems themselves are included with the set of MATLAB codes. The author of this book does not encourage the reader to regard the examples as anything more than examples. In particular, they are not real-world problems, and should not be used as an exhaustive test suite for a code. While there are documented collections of test problems (for example, [10] and [26]), the reader should always evaluate and compare algorithms in the context of his/her own problems.

Some of the problems are directly related to applications. When that is the case we will cite some of the relevant literature. Other examples are included because they are small, simple, and illustrate important effects that can be hidden by the complexity of more serious problems.

**1.6.1 Discrete Optimal Control**

This is a classic example of a problem in which gradient evaluations cost little more than function evaluations.

We begin with the continuous optimal control problems and discuss how gradients are com puted and then move to the discretizations. We will not dwell on the functional analytic issues surrounding the rigorous definition of gradients of maps on function spaces, but the reader should be aware that control problems require careful attention to this. The most important results can

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

10 ITERATIVE METHODS FOR OPTIMIZATION

be found in [151]. The function space setting for the particular control problems of interest in this section can be found in [170], [158], and [159], as can a discussion of more general problems. The infinite-dimensional problem is

*u* (1.15) *f,*

min

where

*f*(*u*) =

*T*

(1.16) *L*(*y*(*t*)*, u*(*t*)*, t*) *dt,*

0

and we seek an optimal point *u ∈ L∞*[0*, T*]. *u* is called the *control variable* or simply the *control*. The function *L* is given and *y*, the *state variable*, satisfies the initial value problem (with *y*˙ = *dy/dt*)

(1.17) *y*˙(*t*) = *φ*(*y*(*t*)*, u*(*t*)*, t*)*, y*(0) = *y*0*.*

One could view the problem (1.15)–(1.17) as a constrained optimization problem or, as we do here, think of the evaluation of *f* as requiring the solution of (1.17) before the integral on the right side of (1.16) can be evaluated. This means that evaluation of *f* requires the solution of (1.17), which is called the *state equation*.

*∇f*(*u*), the gradient of *f* at *u* with respect to the *L*2 inner product, is uniquely determined, if it exists, by

*T*

*f*(*u* + *w*) *− f*(*u*) *−*

(1.18) (*∇f*(*u*))(*t*)*w*(*t*) *dt* = *o*(*w*) 0

as *w →* 0, uniformly in *w*. If *∇f*(*u*) exists then

*T* 0

(*∇f*(*u*))(*t*)*w*(*t*) *dt* = *df*(*u* + *ξw*)

*dξ*

*.*

*ξ*=0

If *L* and *φ* are continuously differentiable, then *∇f*(*u*), as a function of *t*, is given by (1.19) *∇f*(*u*)(*t*) = *p*(*t*)*φu*(*y*(*t*)*, u*(*t*)*, t*) + *Lu*(*y*(*t*)*, u*(*t*)*, t*)*.* In (1.19) *p*, the *adjoint variable*, satisfies the final-value problem on [0*, T*] (1.20) *−p*˙(*t*) = *p*(*t*)*φy*(*y*(*t*)*, u*(*t*)*, t*) + *Ly*(*y*(*t*)*, u*(*t*)*, t*)*, p*(*T*)=0*.*

So computing the gradient requires *u* and *y*, hence a solution of the state equation, and *p*, which requires a solution of (1.20), a final-value problem for the *adjoint equation*. In the general case, (1.17) is nonlinear, but (1.20) is a linear problem for *p*, which should be expected to be easier to solve. This is the motivation for our claim that a gradient evaluation costs little more than a function evaluation.

The discrete problems of interest here are constructed by solving (1.17) by numerical in tegration. After doing that, one can derive an adjoint variable and compute gradients using a discrete form of (1.19). However, in [139] the equation for the adjoint variable of the discrete problem is usually not a discretization of (1.20). For the forward Euler method, however, the discretization of the adjoint equation is the adjoint equation for the discrete problem and we use that discretization here for that reason.

The fully discrete problem is min*u f*, where *u ∈ RN* and

*N*

*f*(*u*) =

*j*=1

*L*((*y*)*j ,*(*u*)*j , j*)*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

BASIC CONCEPTS 11

and the states *{xj}* are given by the Euler recursion

*yj*+1 = *yj* + *hφ*((*y*)*j ,*(*u*)*j , j*) for *j* = 0*,...,N −* 1*,*

where *h* = *T /*(*N −* 1) and *x*0 is given. Then

(*∇f*(*u*))*j* = (*p*)*jφu*((*y*)*j ,*(*u*)*j , j*) + *Lu*((*y*)*j ,*(*u*)*j , j*)*,*

where (*p*)*N* = 0 and

(*p*)*j−*1 = (*p*)*j* + *h*

(*p*)*jφy*((*y*)*j ,*(*u*)*j , j*) + *Ly*((*y*)*j ,*(*u*)*j , j*)

for *j* = *N,...,* 1*.*

**1.6.2 Parameter Identification**

This example, taken from [13], will appear throughout the book. The problem is small with *N* = 2. The goal is to identify the damping *c* and spring constant *k* of a linear spring by minimizing the difference of a numerical prediction and measured data. The experimental scenario is that the spring-mass system will be set into motion by an initial displacement from equilibrium and measurements of displacements will be taken at equally spaced increments in time.

The motion of an unforced harmonic oscillator satisfies the initial value problem *u*+ *cu*+ *ku* = 0; *u*(0) = *u*0*, u*(1.21) (0) = 0*,*

on the interval [0*, T*]. We let *x* = (*c, k*)*T* be the vector of unknown parameters and, when the dependence on the parameters needs to be explicit, we will write *u*(*t* : *x*) instead of *u*(*t*) for the solution of (1.21). If the displacement is sampled at *{tj}M*

*j*=1, where *tj* = (*j −* 1)*T /*(*M −* 1),

and the observations for *u* are *{uj}M*

*j*=1, then the objective function is

*f*(*x*) = 12*M*

*|u*(*tj* : *x*) *− uj |*2 (1.22) *.*

*j*=1

This is an example of a *nonlinear least squares* problem.

*u* is differentiable with respect to *x* when *c*2 *−* 4*k* = 0. In that case, the gradient of *f* is

*M j*=1

*∂u*(*tj* :*x*)

*∂c* (*u*(*tj* : *x*) *− uj* )

(1.23) *.*

*∇f*(*x*) =

*M*

*j*=1

*∂u*(*tj* :*x*)

*∂k* (*u*(*tj* : *x*) *− uj* )

We can compute the derivatives of *u*(*t* : *x*) with respect to the parameters by solving the *sensitivity equations*. Differentiating (1.21) with respect to *c* and *k* and setting *w*1 = *∂u/∂c* and *w*2 = *∂u/∂k* we obtain

(1.24)

*w*1 + *u*+ *cw*1 + *kw*1 = 0; *w*1(0) = *w*1(0) = 0*, w*2 + *cw*2 + *u* + *kw*2 = 0; *w*2(0) = *w*2(0) = 0*.*

If *c* is large, the initial value problems (1.21) and (1.24) will be *stiff* and one should use a good variable step stiff integrator. We refer the reader to [110], [8], [235] for details on this issue. In the numerical examples in this book we used the MATLAB code ode15s from [236]. Stiffness can also arise in the optimal control problem from *§*1.6.1 but does not in the specific examples we use in this book. We caution the reader that when one uses an ODE code the results may only be expected to be accurate to the tolerances input to the code. This limitation on the accuracy must be taken into account, for example, when approximating the Hessian by differences.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

12 ITERATIVE METHODS FOR OPTIMIZATION

**1.6.3 Convex Quadratics**

While convex quadratic problems are, in a sense, the easiest of optimization problems, they present surprising challenges to the sampling algorithms presented in Part II and can illustrate fundamental problems with classical gradient-based methods like the steepest descent algorithm from *§*3.1. Our examples will all take *N* = 2, *b* = 0, and

*H* =

*λs* 0 0 *λl*

*,*

where 0 *< λs ≤ λl*. The function to be minimized is

*f*(*x*) = *xT Hx*

and the minimizer is *x∗* = (0*,* 0)*T* .

As *λl/λs* becomes large, the level curves of *f* become elongated. When *λs* = *λl* = 1, min*x f* is the easiest problem in optimization.

**1.7 Exercises on Basic Concepts**

1.7.1. Prove Theorem 1.2.2.

1.7.2. Consider the parameter identification problem for *x* = (*c, k, ω, φ*)*T ∈ R*4 associated with the initial value problem

*u*+ *cu*+ *ku* = sin(*ωt* + *φ*); *u*(0) = 10*, u*(0) = 0*.*

For what values of *x* is *u* differentiable? Derive the sensitivity equations for those values of *x* for which *u* is differentiable.

1.7.3. Solve the system of sensitivity equations from exercise 1.7.2 numerically for *c* = 10, *k* = 1, *ω* = *π*, and *φ* = 0 using the integrator of your choice. What happens if you use a nonstiff integrator?

1.7.4. Let *N* = 2, *d* = (1*,* 1)*T* , and let *f*(*x*) = *xT d* + *xT x*. Compute, by hand, the minimizer using conjugate gradient iteration.

1.7.5. For the same *f* as in exercise 1.7.4 solve the constrained optimization problem min

*x∈U f*(*x*)*,*

where *U* is the circle centered at (0*,* 0)*T* of radius 1*/*3. You can solve this by inspection; no computer and very little mathematics is needed.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Chapter 2**

**Local Convergence of Newton’s**

**Method**

By a local convergence method we mean one that requires that the initial iterate *x*0 is close to a local minimizer *x∗* at which the sufficient conditions hold.

**2.1 Types of Convergence**

We begin with the standard taxonomy of convergence rates [84], [154], [211].

Definition 2.1.1. *Let {xn} ⊂ RN and x∗ ∈ RN . Then*

*• xn → x∗* q-quadratically *if xn → x∗ and there is K >* 0 *such that*

*xn*+1 *− x∗ ≤ Kxn − x∗*2*.*

*• xn → x∗* q-superlinearly with q-order *α >* 1 *if xn → x∗ and there is K >* 0 *such that xn*+1 *− x∗ ≤ Kxn − x∗α.*

*• xn → x∗* q-superlinearly *if*

lim*n→∞xn*+1 *− x∗*

*xn − x∗*= 0*.*

*• xn → x∗* q-linearly *with* q-factor *σ ∈* (0*,* 1) *if*

*xn*+1 *− x∗ ≤ σxn − x∗*

*for n sufficiently large.*

Definition2.1.2. *An iterative method for computing x∗ is said to be* locally *(q-quadratically, q-superlinearly, q-linearly, etc.) convergent if the iterates converge to x∗ (q-quadratically, q superlinearly, q-linearly, etc.) given that the initial data for the iteration is sufficiently good.*

We remind the reader that a q-superlinearly convergent sequence is also q-linearly conver gent with q-factor *σ* for any *σ >* 0. A q-quadratically convergent sequence is q-superlinearly convergent with q-order of 2.

13

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

14 ITERATIVE METHODS FOR OPTIMIZATION

In some cases the accuracy of the iteration can be improved by means that are external to the algorithm, say, by evaluation of the objective function and its gradient with increasing accuracy as the iteration progresses. In such cases, one has no guarantee that the accuracy of the iteration is monotonically increasing but only that the accuracy of the results is improving at a rate determined by the improving accuracy in the function–gradient evaluations. The concept of *r-type* convergence captures this effect.

Definition 2.1.3. *Let {xn} ⊂ RN and x∗ ∈ RN . Then {xn} converges to x∗* r-( quadrat ically, superlinearly, linearly) *if there is a sequence {ξn} ⊂ R converging q-(quadratically, superlinearly, linearly) to* 0 *such that*

*xn − x∗ ≤ ξn.*

*We say that {xn} converges* r-superlinearly *with* r-order *α >* 1 *if ξn →* 0 *q-superlinearly with q-order α.*

**2.2 The Standard Assumptions**

We will assume that local minimizers satisfy the *standard assumptions* which, like the standard assumptions for nonlinear equations in [154], will guarantee that Newton’s method converges q-quadratically to *x∗*. We will assume throughout this book that *f* and *x∗* satisfy Assumption 2.2.1.

Assumption 2.2.1.

1. *f is twice differentiable and*

*∇*2*f*(*x*) *− ∇*2 (2.1) *f*(*y*) *≤ γx − y.*

2. *∇f*(*x∗*)=0*.*

3. *∇*2*f*(*x∗*) *is positive definite.*

We sometimes say that *f* is twice *Lipschitz continuously* differentiable with *Lipschitz constant γ* to mean that part 1 of the standard assumptions holds.

If the standard assumptions hold then Theorem 1.4.1 implies that *x∗* is a local minimizer of *f*. Moreover, the standard assumptions for nonlinear equations [154] hold for the system *∇f*(*x*)=0. This means that all of the local convergence results for nonlinear equations can be applied to unconstrained optimization problems. In this chapter we will quote those results from nonlinear equations as they apply to unconstrained optimization. However, these statements must be understood in the context of optimization. We will use, for example, the fact that the Hessian (the Jacobian of *∇f*) is positive definite at *x∗* in our solution of the linear equation for the Newton step. We will also use this in our interpretation of the Newton iteration.

**2.3 Newton’s Method**

As in [154] we will define iterative methods in terms of the transition from a current iteration *xc* to a new one *x*+. In the case of a system of nonlinear equations, for example, *x*+ is the root of the *local linear model* of *F* about *xc*

*Mc*(*x*) = *F*(*xc*) + *F*(*xc*)(*x − xc*)*.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 15

Solving *Mc*(*x*+)=0 leads to the standard formula for the Newton iteration

*x*+ = *xc − F*(*xc*)*−*1 (2.2) *F*(*xc*)*.*

One could say that Newton’s method for unconstrained optimization is simply the method for nonlinear equations applied to *∇f*(*x*)=0. While this is technically correct if *xc* is near a minimizer, it is utterly wrong if *xc* is near a maximum. A more precise way of expressing the idea is to say that *x*+ is a minimizer of the *local quadratic model* of *f* about *xc*.

*mc*(*x*) = *f*(*xc*) + *∇f*(*xc*)*T* (*x − xc*) + 12(*x − xc*)*T ∇*2*f*(*xc*)(*x − xc*)*.*

If *∇*2*f*(*xc*)is positive definite, then the minimizer *x*+ of *mc* is the unique solution of *∇mc*(*x*) = 0. Hence,

0 = *∇mc*(*x*+) = *∇f*(*xc*) + *∇*2*f*(*xc*)(*x*+ *− xc*)*.*

Therefore,

*x*+ = *xc −* (*∇*2*f*(*xc*)) (2.3) *−*1*∇f*(*xc*)*,*

which is the same as (2.2) with *F* replaced by *∇f* and *F*by *∇*2*f*. Of course, *x*+ is not computed by forming an inverse matrix. Rather, given *xc*, *∇f*(*xc*) is computed and the linear equation

*∇*2 (2.4) *f*(*xc*)*s* = *−∇f*(*xc*)

is solved for the *step s*. Then (2.3) simply says that *x*+ = *xc* + *s*.

However, if *uc* is far from a minimizer, *∇*2*f*(*uc*) could have negative eigenvalues and the quadratic model will not have local minimizers (see exercise 2.7.4), and *Mc*, the local linear model of *∇f* about *uc*, could have roots which correspond to local maxima or saddle points of *mc*. Hence, we must take care when far from a minimizer in making a correspondence between Newton’s method for minimization and Newton’s method for nonlinear equations. In this chapter, however, we will assume that we are sufficiently near a local minimizer for the standard assumptions for local optimality to imply those for nonlinear equations (as applied to *∇f*). Most of the proofs in this chapter are very similar to the corresponding results, [154], for nonlinear equations. We include them in the interest of completeness.

We begin with a lemma from [154], which we state without proof.

Lemma 2.3.1. *Assume that the standard assumptions hold. Then there is δ >* 0 *so that for all x ∈ B*(*δ*)

*∇*2*f*(*x*) *≤* 2*∇*2*f*(*x∗* (2.5) )*,*

(*∇*2*f*(*x*))*−*1 *≤* 2(*∇*2*f*(*x∗*))*−*1 (2.6) *,*

*and*

(*∇*2*f*(*x∗*))*−*1*−*1*e/*2 *≤ ∇f*(*x*) *≤* 2*∇*2*f*(*x∗* (2.7) )*e.*

As a first example, we prove the local convergence for Newton’s method.

Theorem 2.3.2. *Let the standard assumptions hold. Then there are K >* 0 *and δ >* 0 *such that if xc ∈ B*(*δ*)*, the Newton iterate from xc given by* (2.3) *satisfies*

*e*+ *≤ Kec*2 (2.8) *.*

*Proof*. Let *δ* be small enough so that the conclusions of Lemma 2.3.1 hold. By Theorem 1.2.1 1

*e*+ = *ec − ∇*2*f*(*xc*)*−*1*∇f*(*xc*)=(*∇*2*f*(*xc*))*−*1 0

(*∇*2*f*(*xc*) *− ∇*2*f*(*x∗* + *tec*))*ec dt.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

16 ITERATIVE METHODS FOR OPTIMIZATION

By Lemma 2.3.1 and the Lipschitz continuity of *∇*2*f*,

*e*+ *≤* (2(*∇*2*f*(*x∗*))*−*1)*γec*2*/*2*.*

This completes the proof of (2.8) with *K* = *γ*(*∇*2*f*(*x∗*))*−*1.

As in the nonlinear equations setting, Theorem 2.3.2 implies that the complete iteration is locally quadratically convergent.

Theorem 2.3.3. *Let the standard assumptions hold. Then there is δ >* 0 *such that if x*0 *∈ B*(*δ*)*, the Newton iteration*

*xn*+1 = *xn −* (*∇*2*f*(*xn*))*−*1*∇f*(*xn*)

*converges q-quadratically to x∗.*

*Proof*. Let *δ* be small enough so that the conclusions of Theorem 2.3.2 hold. Reduce *δ* if needed so that *Kδ* = *η <* 1. Then if *n ≥* 0 and *xn ∈ B*(*δ*), Theorem 2.3.2 implies that

*en*+1 *≤ Ken*2 (2.9) *≤ ηen < en*

and hence *xn*+1 *∈ B*(*ηδ*) *⊂ B*(*δ*). Therefore, if *xn ∈ B*(*δ*) we may continue the iteration. Since *x*0 *∈ B*(*δ*) by assumption, the entire sequence *{xn}⊂B*(*δ*). (2.9) then implies that *xn → x∗* q-quadratically.

Newton’s method, from the local convergence point of view, is exactly the same as that for nonlinear equations applied to the problem of finding a root of *∇f*. We exploit the extra structure of positive definiteness of *∇*2*f* with an implementation of Newton’s method based on the *Cholesky factorization* [127], [249], [264]

*∇*2*f*(*u*) = *LLT* (2.10) *,*

where *L* is lower triangular and has a positive diagonal.

We terminate the iteration when *∇f* is sufficiently small (see [154]). A natural criterion is to demand a relative decrease in *∇f* and terminate when

(2.11) *∇f*(*xn*) *≤ τr∇f*(*x*0)*,*

where *τr ∈* (0*,* 1) is the desired reduction in the gradient norm. However, if *∇f*(*x*0) is very small, it may not be possible to satisfy (2.11) in floating point arithmetic and an algorithm based entirely on (2.11) might never terminate. A standard remedy is to augment the relative error criterion and terminate the iteration using a combination of relative and absolute measures of *∇f*, i.e., when

(2.12) *∇f*(*xn*) *≤ τr∇f*(*x*0) + *τa.*

In (2.12) *τa* is an absolute error tolerance. Hence, the termination criterion input to many of the algorithms presented in this book will be in the form of a vector *τ* = (*τr, τa*) of relative and absolute residuals.

Algorithm 2.3.1. newton(*x, f, τ* )

1. *r*0 = *∇f*(*x*)

2. *Do while ∇f*(*x*) *> τrr*0 + *τa*

(a) *Compute ∇*2*f*(*x*)

(b) *F actor ∇*2*f*(*x*) = *LLT*

(c) *Solve LLT s* = *−∇f*(*x*)

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 17

(d) *x* = *x* + *s*

(e) *Compute ∇f*(*x*).

Algorithm newton, as formulated above, is not completely satisfactory. The value of the objective function *f* is never used and step 2b will fail if *∇*2*f* is not positive definite. This failure, in fact, could serve as a signal that one is too far from a minimizer for Newton’s method to be directly applicable. However, if we are near enough (see Exercise 2.7.8) to a local minimizer, as we assume in this chapter, all will be well and we may apply all the results from nonlinear equations.

**2.3.1 Errors in Functions, Gradients, and Hessians**

In the presence of errors in functions and gradients, however, the problem of unconstrained optimization becomes more difficult than that of root finding. We discuss this difference only briefly here and for the remainder of this chapter assume that gradients are computed exactly, or at least as accurately as *f*, say, either analytically or with automatic differentiation [129], [130]. However, we must carefully study the effects of errors in the evaluation of the Hessian just as we did those of errors in the Jacobian in [154].

A significant difference from the nonlinear equations case arises if only functions are available and gradients and Hessians must be computed with differences. A simple one-dimensional analysis will suffice to explain this. Assume that we can only compute *f* approximately. If we compute ˆ*f* = *f* + *f* rather than *f*, then a forward difference gradient with difference increment

*h*

*Dhf*(*x*) = ˆ*f*(*x* + *h*) *−* ˆ*f*(*x*)

*h*

differs from *f*by *O*(*h*+*f /h*) and this error is minimized if *h* = *O*(*√f* ). In that case the error in the gradient is *g* = *O*(*h*) = *O*(*√f* ). If a forward difference Hessian is computed using *Dh* as an approximation to the gradient, then the error in the Hessian will be

∆ = *O*(*√g*) = *O*(1*/*4

*f* (2.13) )

and the accuracy in *∇*2*f* will be much less than that of a Jacobian in the nonlinear equations case.

If *f* is significantly larger than machine roundoff, (2.13) indicates that using numerical Hessians based on a second numerical differentiation of the objective function will not be very accurate. Even in the best possible case, where *f* is the same size as machine roundoff, finite difference Hessians will not be very accurate and will be very expensive to compute if the Hessian is dense. If, as on most computers today, machine roundoff is (roughly) 10*−*16, (2.13) indicates that a forward difference Hessian will be accurate to roughly four decimal digits.

One can obtain better results with centered differences, but at a cost of twice the number of function evaluations. A centered difference approximation to *∇f* is

*Dhf*(*x*) = ˆ*f*(*x* + *h*) *−* ˆ*f*(*x − h*)

2*h*

and the error is *O*(*h*2 + *f /h*), which is minimized if *h* = *O*(1*/*3

*f* ) leading to an error in the

gradient of *g* = *O*(2*/*3

*f* ). Therefore, a central difference Hessian will have an error of

∆ = *O*((*g*)2*/*3) = *O*(4*/*9

*f* )*,*

which is substantially better. We will find that accurate gradients are much more important than accurate Hessians and one option is to compute gradients with central differences and Hessians

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

18 ITERATIVE METHODS FOR OPTIMIZATION with forward differences. If one does that the centered difference gradient error is *O*(2*/*3

therefore the forward difference Hessian error will be ∆ = *O √g*= *O*(1*/*3

*f* )*.*

*f* ) and

More elaborate schemes [22] compute a difference gradient and then reuse the function evalua tions in the Hessian computation.

In many optimization problems, however, accurate gradients are available. When that is the case, numerical differentiation to compute Hessians is, like numerical computation of Jacobians for nonlinear equations [154], a reasonable idea for many problems and the less expensive forward differences work well.

Clever implementations of difference computation can exploit sparsity in the Hessian [67], [59] to evaluate a forward difference approximation with far fewer than *N* evaluations of *∇f*. In the sparse case it is also possible [22], [23] to reuse the points from a centered difference approximation to the gradient to create a second-order accurate Hessian.

Unless *g*(*xn*) *→* 0 as the iteration progresses, one cannot expect convergence. For this reason estimates like (2.14) are sometimes called *local improvement* [88] results. Theorem 2.3.4 is a typical example.

Theorem 2.3.4. *Let the standard assumptions hold. Then there are K >* ¯ 0*, δ >* 0*, and δ*1 *>* 0 *such that if xc ∈ B*(*δ*) *and* ∆(*xc*) *< δ*1 *then*

*x*+ = *xc −* (*∇*2*f*(*xc*) + ∆(*xc*))*−*1(*∇f*(*xc*) + *g*(*xc*))

*is defined (i.e., ∇*2*f*(*xc*) + ∆(*xc*) *is nonsingular) and satisfies*

*e*+ *≤ K*¯ (*ec*2 (2.14) + ∆(*xc*)*ec* + *g*(*xc*))*.*

*Proof*. Let *δ* be small enough so that the conclusions of Lemma 2.3.1 and Theorem 2.3.2 hold. Let

*xN*+ = *xc −* (*∇*2*f*(*xc*))*−*1*∇f*(*xc*)

and note that

*x*+ = *xN*+ + ((*∇*2*f*(*xc*))*−*1 *−* (*∇*2*f*(*xc*) + ∆(*xc*))*−*1)*∇f*(*xc*) *−* (*∇*2*f*(*xc*) + ∆(*xc*))*−*1*g*(*xc*)*.* Lemma 2.3.1 and Theorem 2.3.2 imply

*e*+ *≤ Kec*2 + 2(*∇*2*f*(*xc*))*−*1 *−* (*∇*2*f*(*xc*) + ∆(*xc*))*−*1*∇*2*f*(*x∗*)*ec* (2.15)

+(*∇*2*f*(*xc*) + ∆(*xc*))*−*1*g*(*xc*)*.*

If

∆(*xc*)*≤*(*∇*2*f*(*x∗*))*−*1*−*1*/*4*,*

then Lemma 2.3.1 implies that

∆(*xc*)*≤*(*∇*2*f*(*xc*))*−*1*−*1*/*2

and the Banach Lemma [12], [154] states that *∇*2*f*(*xc*) + ∆(*xc*) is nonsingular and (*∇*2*f*(*xc*) + ∆(*xc*))*−*1 *≤* 2(*∇*2*f*(*xc*))*−*1 *≤* 4(*∇*2*f*(*x∗*))*−*1*.*

Hence,

(*∇*2*f*(*xc*))*−*1 *−* (*∇*2*f*(*xc*) + ∆(*xc*))*−*1 *≤* 8(*∇*2*f*(*x∗*))*−*12∆(*xc*)*.* Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 19

We use these estimates and (2.15) to obtain

*e*+ *≤ Kec*2 + 16(*∇*2*f*(*x∗*))*−*12*∇*2*f*(*x∗*)∆(*xc*)*ec*+ 4(*∇*2*f*(*x∗*))*−*1*g*(*xc*)*.*

Setting

*K*¯ = *K* + 16(*∇*2*f*(*x∗*))*−*12*∇*2*f*(*x∗*) + 4(*∇*2*f*(*x∗*))*−*1

completes the proof.

As is the case with equations, (2.14) implies that one cannot hope to find a minimizer with more accuracy that one can evaluate *∇f*. In most cases the iteration will *stagnate* once *e* is (roughly) the same size as *g*. The speed of convergence will be governed by the accuracy in the Hessian.

The result for the *chord method* illustrates this latter point. In the chord method we form and compute the Cholesky factorization of *∇*2*f*(*x*0) and use that factorization to compute all subsequent Newton steps. Hence,

*x*+ = *xc −* (*∇*2*f*(*x*0))*−*1*∇f*(*xc*)

and

(2.16) ∆(*xc*) *≤ γx*0 *− xc ≤ γ*(*e*0 + *ec*)*.*

Algorithmically the chord iteration differs from the Newton iteration only in that the computation and factorization of the Hessian is moved outside of the main loop.

Algorithm 2.3.2. chord(*x, f, τ* )

1. *r*0 = *∇f*(*x*)

2. *Compute ∇*2*f*(*x*)

3. *F actor ∇*2*f*(*x*) = *LLT*

4. *Do while ∇f*(*x*) *> τrr*0 + *τa*

(a) *Solve LLT s* = *−∇f*(*x*)

(b) *x* = *x* + *s*

(c) *Compute ∇f*(*x*).

The convergence theory follows from Theorem 2.3.4 with *g* = 0 and ∆ = *O*(*e*0).

Theorem 2.3.5. *Let the standard assumptions hold. Then there are KC >* 0 *and δ >* 0 *such that if x*0 *∈ B*(*δ*) *the chord iterates converge q-linearly to x∗ and*

(2.17) *en*+1 *≤ KC e*0*en.*

*Proof*. Let *δ* be small enough so that the conclusions of Theorem 2.3.4 hold. Assume that *en, e*0 *∈ B*(*δ*). Combining (2.16) and (2.14) implies

*en*+1 *≤ K*¯ (*en*(1 + *γ*) + *γe*0)*en ≤ K*¯ (1 + 2*γ*)*δen.*

Hence, if *δ* is small enough so that

*K*¯ (1 + 2*γ*)*δ* = *η <* 1*,*

then the chord iterates converge q-linearly to *x∗*. Q-linear convergence implies that *en < e*0 and hence (2.17) holds with *KC* = *K*¯ (1 + 2*γ*).

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

20 ITERATIVE METHODS FOR OPTIMIZATION

The Shamanskii method [233], [154], [211] is a generalization of the chord method that updates Hessians after every *m* + 1 nonlinear iterations. Newton’s method corresponds to *m* = 1 and the chord method to *m* = *∞*. The convergence result is a direct consequence of Theorems 2.3.3 and 2.3.5.

Theorem 2.3.6. *Let the standard assumptions hold and let m ≥* 1 *be given. Then there are KS >* 0 *and δ >* 0 *such that if x*0 *∈ B*(*δ*) *the Shamanskii iterates converge q-superlinearly to x∗ with q-order m and*

*en*+1 *≤ KSenm*+1 (2.18) *.*

As one more application of Theorem 2.3.4, we analyze the effects of a difference approxima tion of the Hessian. We follow the notation of [154] where possible. For example, to construct a Hessian matrix, whose columns are *∇*2*f*(*x*)*ej* , where *ej* is the unit vector with *j*th compo nent 1 and other components 0, we could approximate the matrix–vector products *∇*2*f*(*x*)*ej* by forward differences and then symmetrize the resulting matrix. We define

*∇*2*hf*(*x*)=(*Ah* + *ATh* (2.19) )*/*2*,*

where *Ah* is the matrix whose *j*th column is *D*2*hf*(*x* : *ej* ). *D*2*hf*(*x* : *w*), the difference approxi mation of the action of the Hessian *∇*2*f*(*x*) on a vector *w*, is defined to be the quotient

(2.20)

*D*2*hf*(*x* : *w*) =



0*, w* = 0*,*

*∇f*(*x* + *hw/w*) *− ∇f*(*x*)



*h/w , w* = 0*.*

Note that we may also write

*D*2*hf*(*x* : *w*) = *Dh*(*∇f*)(*x* : *w*)*,*

where the notation *Dh*, taken from [154], denotes numerical directional derivative. If *x* is very large, then the error in computing the sum *x* + *hw/w* will have to be taken into consideration in the choice of *h*.

We warn the reader, as we did in [154], that *D*2*f*(*x* : *w*) is not a linear map and that *D*2*f*(*x* : *w*) is not, in general, the same as *∇*2*hf*(*x*)*w*.

If we compute *∇f*(*x*) + *g*(*x*) and the gradient errors satisfy an estimate of the form *g*(*x*) *≤* ¯

for all *x*, then the computed difference Hessian is *∇h*(*∇f* + *g*) and satisfies *∇*2 (2.21) *f*(*x*) *− ∇h*(*∇f* + *g*)(*x*) = *O*(*h* + ¯*/h*)*.*

So, as in [154], the choice *h ≈ √*¯ is optimal in the sense that it minimizes the quantity in the *O*-term in (2.21).

The local convergence theorem in this case is [88], [154], [278], as follows.

Theorem 2.3.7. *Let the standard assumptions hold. Then there are δ,* ¯*, and KD >* 0 *such that if xc ∈ B*(*δ*)*, g*(*x*) *≤* ¯*c for all x ∈ B*(*δ*)*, and*

*g*(*xc*)

*then*

*h ≥ M*

*x*+ = *xc −* (*∇hc* (*∇f*(*xc*) + *g*(*xc*)))*−*1(*∇f*(*xc*) + *g*(*xc*))

*satisfies*

*e*+ *≤ KD*(¯*c* + (¯*c* + *h*)*ec*)*.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 21

**2.3.2 Termination of the Iteration**

It is not safe to terminate the iteration when *f*(*xc*) *− f*(*x*+) is small, and no conclusions can safely be drawn by examination of the differences of the objective function values at successive iterations. While some of the algorithms for difficult problems in Part II of this book do indeed terminate when successive function values are close, this is an act of desperation. For example,

if

*f*(*xn*) = *−*

*n*

*j*=1

*j−*1*,*

then *f*(*xn*) *→ −∞* but *f*(*xn*+1) *− f*(*xn*) = *−*1*/*(*n* + 1) *→* 0. The reader has been warned. If the standard assumptions hold, then one may terminate the iteration when the norm of *∇f* is sufficiently small relative to *∇f*(*x*0) (see [154]). We will summarize the key points here and refer the reader to [154] for the details. The idea is that if *∇*2*f*(*x∗*) is well conditioned, then a small gradient norm implies a small error norm. Hence, for any gradient-based iterative method, termination on small gradients is reasonable. In the special case of Newton’s method, the norm of the step is a very good indicator of the error and if one is willing to incur the added cost of an extra iteration, a very sharp bound on the error can be obtained, as we will see below.

Lemma 2.3.8. *Assume that the standard assumptions hold. Let δ >* 0 *be small enough so that the conclusions of Lemma* 2.3.1 *hold for x ∈ B*(*δ*)*. Then for all x ∈ B*(*δ*)

*∇f*(*x*0) *≤*4*κ*(*∇*2*f*(*x∗*))*e*

*e*

4*e*0*κ*(*∇*2*f*(*x∗*)) *≤ ∇f*(*x*)

*e*0 (2.22) *.*

The meaning of (2.22) is that, up to a constant multiplier, the norm of the relative gradient is the same as the norm of the relative error. This partially motivates the termination condition (2.12).

In the special case of Newton’s method, one can use the steplength as an accurate estimate of the error because Theorem 2.3.2 implies that

*ec* = *s* + *O*(*ec*2 (2.23) )*.*

Hence, near the solution *s* and *ec* are essentially the same size. The cost of using (2.23) is that all the information needed to compute *x*+ = *xc* + *s* has been computed. If we terminate the iteration when *s* is smaller than our desired tolerance and then take *x*+ as the final result, we have attained more accuracy than we asked for. One possibility is to terminate the iteration when

*s* = *O*(*√~~τ~~s*~~)~~ for some *τs >* 0. This, together with (2.23), will imply that *ec* = *O*(*√~~τ~~s*) and hence, using Theorem 2.3.2, that

*e*+ = *O*(*ec*2 (2.24) ) = *O*(*τs*)*.*

For a superlinearly convergent method, termination on small steps is equally valid but one cannot use (2.24). For a superlinearly convergent method we have

(2.25) *ec* = *s* + *o*(*ec*) and *e*+ = *o*(*ec*)*.*

Hence, we can conclude that *e*+ *< τs* if *s < τs*. This is a weaker, but still very useful, result.

For a q-linearly convergent method, such as the chord method, making termination decisions based on the norms of the steps is much riskier. The relative error in estimating *ec* by *s* is

*|ec−s|*

*ec ≤ ec* + *s*

*ec* = *e*+

*ec .*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

22 ITERATIVE METHODS FOR OPTIMIZATION

Hence, estimation of errors by steps is worthwhile only if convergence is fast. One can go further [156] if one has an estimate *ρ* of the q-factor that satisfies

*e*+ *≤ ρec.*

In that case,

(1 *− ρ*)*ec≤ec−e*+*≤ec − e*+ = *s.*

Hence

*e*+ *≤ ρec ≤ ρ*

(2.26) *s.*

1 *− ρ*

So, if the q-factor can be estimated from above by *ρ* and

*s <* (1 *− ρ*)*τs/ρ,*

then *e*+ *< τs*. This approach is used in ODE and DAE codes [32], [234], [228], [213], but requires good estimates of the q-factor and we do not advocate it for q-linearly convergent methods for optimization. The danger is that if the convergence is slow, the approximate q-factor can be a gross underestimate and cause premature termination of the iteration.

It is not uncommon for evaluations of *f* and *∇f* to be very expensive and optimizations are, therefore, usually allocated a fixed maximum number of iterations. Some algorithms, such as the DIRECT, [150], algorithm we discuss in *§*8.4.2, assign a limit to the number of function evaluations and terminate the iteration in only this way.

**2.4 Nonlinear Least Squares**

*Nonlinear least squares* problems have objective functions of the form

*f*(*x*) = 12*M*

*ri*(*x*)22 = 12*R*(*x*)*T* (2.27) *R*(*x*)*.*

*i*=1

The vector *R* = (*r*1*,...,rM*) is called the *residual*. These problems arise in data fitting, for example. In that case *M* is the number of observations and *N* is the number of parameters; for these problems *M>N* and we say the problem is *overdetermined*. If *M* = *N* we have a nonlinear equation and the theory and methods from [154] are applicable. If *M<N* the problem is *underdetermined*. Overdetermined least squares problems arise most often in data fitting applications like the parameter identification example in *§*1.6.2. Underdetermined problems are less common, but are, for example, important in the solution of high-index differential algebraic equations [48], [50].

The local convergence theory for underdetermined problems has the additional complexity that the limit of the Gauss–Newton iteration is not uniquely determined by the distance of the initial iterate to the set of points where *R*(*x∗*)=0. In *§*2.4.3 we describe the difficulties and state a simple convergence result.

If *x∗* is a local minimizer of *f* and *f*(*x∗*)=0, the problem min *f* is called a *zero residual problem* (a remarkable and suspicious event in the data fitting scenario). If *f*(*x∗*) is small, the expected result in data fitting if the model (i.e., *R*) is good, the problem is called a *small residual problem*. Otherwise one has a *large residual problem*.

Nonlinear least squares problems are an intermediate stage between nonlinear equations and optimization problems and the methods for their solution reflect this. We define the *M × N* Jacobian *R*of *R* by

(*R*(2.28) (*x*))*ij* = *∂ri/∂xj ,* 1 *≤ i ≤ M,* 1 *≤ j ≤ N.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 23

With this notation it is easy to show that

*∇f*(*x*) = *R*(*x*)*T R*(*x*) *∈ RN* (2.29) *.*

The necessary conditions for optimality imply that at a minimizer *x∗*

*R*(*x∗*)*T R*(*x∗* (2.30) )=0*.*

In the underdetermined case, if *R*(*x∗*) has full row rank, (2.30) implies that *R*(*x∗*)=0; this is not the case for overdetermined problems.

The cost of a gradient is roughly that of a Jacobian evaluation, which, as is the case with nonlinear equations, is the most one is willing to accept. Computation of the Hessian (an *N ×N* matrix)

*∇*2*f*(*x*) = *R*(*x*)*T R*(*x*) +*M*

*j*=1*ri*(*x*)*T ∇*2 (2.31) *ri*(*x*)

requires computation of the *M* Hessians *{∇*2*ri}* for the second-order term

*M*

*j*=1*ri*(*x*)*T ∇*2*ri*(*x*)

and is too costly to be practical.

We will also express the second-order term as

*M*

*j*=1*ri*(*x*)*T ∇*2*ri*(*x*) = *R*(*x*)*T R*(*x*)*,*

where the second derivative *R*is a tensor. The notation is to be interpreted in the following way. For *v ∈ RM*, *R*(*x*)*T v* is the *N × N* matrix

*R*(*x*)*T v* = *∇*2(*R*(*x*)*T v*) = *M*

*i*=1(*v*)*i∇*2*ri*(*x*)*.*

We will use the tensor notation when expanding *R* about *x∗* in some of the analysis to follow.

**2.4.1 Gauss–Newton Iteration**

The Gauss–Newton algorithm simply discards the second-order term in *∇*2*f* and computes a step

*s* = *−*(*R*(*xc*)*T R*(*xc*))*−*1*∇f*(*xc*)

(2.32)

= *−*(*R*(*xc*)*T R*(*xc*))*−*1*R*(*xc*)*T R*(*xc*)*.*

The Gauss–Newton iterate is *x*+ = *xc*+*s*. One motivation for this approach is that *R*(*x*)*T R*(*x*) vanishes for zero residual problems and therefore might be negligible for small residual problems. Implicit in (2.32) is the assumption that *R*(*xc*)*T R*(*xc*) is nonsingular, which implies that *M ≥ N*. Another interpretation, which also covers underdetermined problems, is to say that the Gauss–Newton iterate is the *minimum norm* solution of the local linear model of our nonlinear least squares problem

min12*R*(*xc*) + *R*(*xc*)(*x − xc*)2 (2.33) *.*

Using (2.33) and linear least squares methods is a more accurate way to compute the step than using (2.32), [115], [116], [127]. In the underdetermined case, the Gauss–Newton step can be computed with the singular value decomposition [49], [127], [249]. (2.33) is an overde termined, square, or underdetermined linear least squares problem if the nonlinear problem is overdetermined, square, or underdetermined.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

24 ITERATIVE METHODS FOR OPTIMIZATION

The standard assumptions for nonlinear least squares problems follow in Assumption 2.4.1.

Assumption 2.4.1. *x∗ is a minimizer of R*22*, R is Lipschitz continuously differentiable near x∗, and R*(*x∗*)*T R*(*x∗*) *has maximal rank. The rank assumption may also be stated as*

*• R*(*x∗*) *is nonsingular* (*M* = *N*),

*• R*(*x∗*) *has full column rank* (*M>N*),

*• R*(*x∗*) *has full row rank* (*M<N*).

**2.4.2 Overdetermined Problems**

Theorem 2.4.1. *Let M>N. Let Assumption* 2.4.1 *hold. Then there are K >* 0 *and δ >* 0 *such that if xc ∈ B*(*δ*) *then the error in the Gauss–Newton iteration satisfies*

*e*+ *≤ K*(*ec*2 + *R*(*x∗* (2.34) )*ec*)*.*

*Proof*. Let *δ* be small enough so that *x−x∗ < δ* implies that *R*(*x*)*T R*(*x*) is nonsingular. Let *γ* be the Lipschitz constant for *R*.

By (2.32)

*e*+ = *ec −* (*R*(*xc*)*T R*(*xc*))*−*1*R*(*xc*)*T R*(*xc*)

(2.35)

= (*R*(*xc*)*T R*(*xc*))*−*1*R*(*xc*)*T* (*R*(*xc*)*ec − R*(*xc*))*.*

Note that

*R*(*xc*)*ec − R*(*xc*) = *R*(*xc*)*ec − R*(*x∗*) + *R*(*x∗*) *− R*(*xc*)

= *−R*(*x∗*)+(*R*(*xc*)*ec* + *R*(*x∗*) *− R*(*xc*))*.*

Now,

*R*(*xc*)*ec* + *R*(*x∗*) *− R*(*xc*) *≤ γec*2*/*2

and, since *R*(*x∗*)*T R*(*x∗*)=0,

*−R*(*xc*)*T R*(*x∗*)=(*R*(*x∗*) *− R*(*xc*))*T R*(*x∗*)*.*

Hence,

*e*+ *≤*(*R*(*xc*)*T R*(*xc*))*−*1(*R*(*x∗*) *− R*(*xc*))*T R*(*x∗*)

+(*R*(*xc*)*T R*(*xc*))*−*1*R*(*xc*)*T γec*2

(2.36)

Setting

2

*≤* (*R*(*xc*)*T R*(*xc*))*−*1*γec*

*R*(*x∗*) + *R*(*xc*)*T ec* 2

*.*

*x∈B*(*δ*)((*R*(*x*)*T R*(*x*))*−*1 *K* = *γ* max

1 + *R*(*x*)*T* 2

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 25

completes the proof.

There are several important consequences of Theorem 2.4.1. The first is that for zero residual problems, the local convergence rate is q-quadratic because the *R*(*x∗*)*ec* term on the right side of (2.34) vanishes. For a problem other than a zero residual one, even q-linear convergence is not guaranteed. In fact, if *xc ∈ B*(*δ*) then (2.35) will imply that *e*+ *≤ rec* for some 0 *<r<* 1 if

*K*(*δ* + *R*(*x∗* (2.37) )) *≤ r*

and therefore the q-factor will be *KR*(*x∗*). Hence, for small residual problems and accurate initial data the convergence of Gauss–Newton will be fast. Gauss–Newton may not converge at all for large residual problems.

Equation (2.36) exposes a more subtle issue when the term

(*R*(*x∗*) *− R*(*xc*))*T R*(*x∗*)

is considered as a whole, rather than estimated by

*γecR*(*x∗*)*.*

Using Taylor’s theorem and the necessary conditions (*R*(*x∗*)*T R*(*x∗*)=0) we have

*R*(*xc*)*T R*(*x∗*) =[*R*(*x∗*) + *R*(*x∗*)*ec* + *O*(*ec*2)]*T R*(*x∗*)

= *eTc R*(*x∗*)*T R*(*x∗*) + *O*(*ec*2)*.*

Recall that

*R*(*x∗*)*T R*(*x∗*) = *∇*2*f*(*x∗*) *− R*(*x∗*)*T R*(*x∗*)

and hence

(*R*(*x∗*)*−R*(*xc*))*T R*(*x∗*)

(2.38)

*≤ ∇*2*f*(*x∗*) *− R*(*x∗*)*T R*(*x∗*)*R*(*x∗*) + *O*(*ec*2)*.*

In a sense (2.38) says that even for a large residual problem, convergence can be fast if the problem is not very nonlinear (small *R*). In the special case of a linear least squares problem (where *R*= 0) Gauss–Newton becomes simply the solution of the normal equations and converges in one iteration.

So, Gauss–Newton can be expected to work well for overdetermined small residual problems and good initial iterates. For large residual problems and/or initial data far from the solution, there is no reason to expect Gauss–Newton to give good results. We address these issues in *§*3.2.3.

**2.4.3 Underdetermined Problems**

We begin with the linear underdetermined least squares problem

min *Ax − b*2 (2.39) *.*

If *A* is *M × N* with *M<N* there will not be a unique minimizer but there will be a unique minimizer with minimum norm. The *minimum norm* solution can be expressed in terms of the *singular value decomposition* [127], [249] of *A*,

*A* = *U*Σ*V T* (2.40) *.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

26 ITERATIVE METHODS FOR OPTIMIZATION

In (2.40), Σ is an *N ×N* diagonal matrix. The diagonal entries of Σ, *{σi}* are called the *singular values*. *σi ≥* 0 and *σi* = 0 if *i>M*. The columns of the *M × N* matrix *U* and the *N × N* matrix *V* are called the left and right *singular vectors*. *U* and *V* have orthonormal columns and hence the minimum norm solution of (2.39) is

*x* = *A†b,*

where *A†* = *V* Σ*†UT* ,

Σ*†* = diag(*σ†*1*,..., σ†N* )*,*

and



*σ†i* =



*σ−*1

*i , σi* = 0*,* 0*, σi* = 0*.*

*A†* is called the *Moore–Penrose inverse* [49], [189], [212]. If *A* is a square nonsingular matrix, then *A†* = *A−*1; if *M>N* then the definition of *A†* using the singular value decomposition is still valid; and, if *A* has full column rank, *A†* = (*AT A*)*−*1*AT* .

Two simple properties of the Moore–Penrose inverse are that *A†A* is a projection onto the range of *A†* and *AA†* is a projection onto the range of *A*. This means that

*A†AA†* = *A†,*(*A†A*)*T* = *A†A, AA†A* = *A,* and (*AA†*)*T* = *AA†* (2.41) *.*

So the minimum norm solution of the local linear model (2.33) of an underdetermined nonlinear least squares problem can be written as [17], [102]

*s* = *−R*(*xc*)*†* (2.42) *R*(*xc*)

and the Gauss–Newton iteration [17] is

*x*+ = *xc − R*(*xc*)*†* (2.43) *R*(*xc*)*.*

The challenge in formulating a local convergence result is that there is not a unique optimal point that attracts the iterates.

In the linear case, where *R*(*x*) = *Ax − b*, one gets

*x*+ = *xc − A†*(*Axc − b*)=(*I − A†A*)*xc − A†b.*

Set *e* = *x − A†b* and note that

*A†AA†b* = *A†b*

by (2.41). Hence

*e*+ = (*I − A†A*)*ec.*

This does not imply that *x*+ = *A†b*, the minimum norm solution, only that *x*+ is a solution of the problem and the iteration converges in one step. The Gauss–Newton iteration cannot correct for errors that are not in the range of *A†*.

Let

*Z* = *{x | R*(*x*)=0*}.*

We show in Theorem 2.4.2, a special case of the result in [92], that if the standard assumptions hold at a point *x∗ ∈ Z*, then the iteration will converge q-quadratically to a point *z∗ ∈ Z*. However, there is no reason to expect that *z∗* = *x∗*. In general *z∗* will depend on *x*0, a very different situation from the overdetermined case. The hypotheses of Theorem 2.4.2, especially that of full column rank in *R*(*x*), are less general than those in [24], [17], [25], [92], and [90].

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 27

Theorem 2.4.2. *Let M ≤ N and let Assumption* 2.4.1 *hold for some x∗ ∈ Z. Then there is δ >* 0 *such that if*

*x*0 *− x∗ ≤ δ,*

*then the Gauss–Newton iterates*

*xn*+1 = *xn − R*(*xn*)*†R*(*xn*)

*exist and converge r-quadratically to a point z∗ ∈ Z.*

*Proof*. Assumption 2.4.1 and results in [49], [126] imply that if *δ* is sufficiently small then there is *ρ*1 such that *R*(*x*)*†* is Lipschitz continuous in the set

*B*1 = *{x | x − x∗ ≤ ρ*1*}*

and the singular values of *R*(*x*) are bounded away from zero in *B*1. We may, reducing *ρ*1 if necessary, apply the Kantorovich theorem [154], [151], [211] to show that if *x ∈ B*1 and *w ∈ Z* is such that

*x − w* = min *z∈Z x − z,*

then there is *ξ* = *ξ*(*x*) *∈ Z* such that

*w − ξ*(*x*) = *O*(*x − w*2) *≤ x − w/*2

and *ξ* is in the range of *R*(*w*)*†*, i.e.,

*R*(*w*)*†R*(*w*)(*x − ξ*(*x*)) = *x − ξ*(*x*)*.*

The method of the proof is to adjust *δ* so that the Gauss–Newton iterates remain in *B*1 and *R*(*xn*) *→* 0 sufficiently rapidly. We begin by requiring that *δ<ρ*1*/*2.

Let *xc ∈ B*1 and let *e* = *x − ξ*(*xc*). Taylor’s theorem, the fundamental theorem of calculus, and (2.41) imply that

*e*+ = *ec − R*(*xc*)*†R*(*xc*)

= *ec −* (*R*(*xc*)*† − R*(*ξ*(*x*))*†*)*R*(*x*) *− R*(*x∗*)*†R*(*x*)

= *e*0 *− R*(*x∗*)*†R*(*x*) + *O*(*ec*2)

= (*I − R*(*x∗*)*†R*(*x∗*))*ec* + *O*(*ec*2) = *O*(*ec*2)*.*

If we define *d*(*x*) = min*z∈Z x − z* then there is *K*1 such that

*d*(*x*+) *≤ x*+ *− ξ*(*xc*) *≤ K*1*xc − ξ*(*xc*)2 *≤ K*1*d*(*xc*)2 (2.44) *.*

Now let

*ρ*2 = min(*ρ*1*,*(2*K*1)*−*1)*.*

So if

*xc ∈ B*2 = *{x | x − x∗ ≤ ρ*2*}*

then

(2.45) *d*(*x*+) *≤ d*(*xc*)*/*2*.*

Finally, there is *K*2 such that

*x*+ *− x∗ ≤xc − x∗* + *x*+ *− xc* = *xc − x∗* + *R*(*xc*)*†R*(*xc*)

*≤ xc − x∗* + *K*2*xc − ξ*(*xc*)*.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. 28 ITERATIVE METHODS FOR OPTIMIZATION

We now require that

*δ ≤ ρ*2

2(1 + *K*2) (2.46) *.*

We complete the proof by induction. If *x*0 *− x∗ ≤ δ* and the Gauss–Newton iterates *{xk}nk*=0 are in *B*2, then *xn*+1 is be defined and, using (2.46) and (2.44),

*n*

*xn*+1 *− x∗≤x*0 *− x∗* + *K*3

+1

*d*(*xk*) *≤ δ* + 2*K*3*d*(*x*0) *≤ ρ*1*.*

*k*=0

Hence, the Gauss–Newton iterates exist, remain in *B*0, and *dn →* 0.

To show that the sequence of Gauss–Newton iterates does in fact converge, we observe that there is *K*3 such that

*x*+ *− xc* = *R*(*xc*)*†R*(*xc*) *≤ K*3*xc − ξ*(*xc*) *≤ K*3*d*(*xc*)*.*

Therefore (2.45) implies that for any *m, n ≥* 0,

*xn*+*m − xn ≤ n*+*m−*1

*l*=*n xl*+1 *− xl*

*l*=*n d*(*xl*) = *d*(*xn*)1 *−* 2*−m*

= *n*+*m*

2

*≤* 2*d*(*xn*) *≤* 2*−n*+1*d*(*x*0)*.*

Hence, *{xk}* is a Cauchy sequence and therefore converges to a point *z∗ ∈ Z*. Since *xn − z∗ ≤* 2*d*(*xn*)*,*

(2.44) implies that the convergence is r-quadratic.

**2.5 Inexact Newton Methods**

An *inexact Newton method* [74] uses an approximate Newton step *s* = *x*+ *− xc*, requiring only that

*∇*2 (2.47) *f*(*xc*)*s* + *∇f*(*xc*) *≤ ηc∇f*(*xc*)*,*

i.e., that the linear residual be small. We will refer to any vector *s* that satisfies (2.47) with *ηc <* 1 as an *inexact Newton step*. We will refer to the parameter *ηc* on the right-hand side of (2.47) as the *forcing term* [99] .

Inexact Newton methods are also called *truncated Newton methods* [75], [198], [199] in the context of optimization. In this book, we consider *Newton–iterative* methods. This is the class of inexact Newton methods in which the linear equation (2.4) for the Newton step is also solved by an iterative method and (2.47) is the termination criterion for that linear iteration. It is standard to refer to the sequence of Newton steps *{xn}* as the *outer iteration* and the sequence of iterates for the linear equation as the *inner iteration*. The naming convention (see [33], [154], [211]) is that Newton–CG, for example, refers to the Newton–iterative method in which the *conjugate gradient* [141] algorithm is used to perform the inner iteration.

Newton–CG is particularly appropriate for optimization, as we expect positive definite Hes sians near a local minimizer. The results for inexact Newton methods from [74] and [154] are sufficient to describe the local convergence behavior of Newton–CG, and we summarize the relevant results from nonlinear equations in *§*2.5.1. We will discuss the implementation of Newton–CG in *§*2.5.2.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 29

**2.5.1 Convergence Rates**

We will prove the simplest of the convergence results for Newton–CG, Theorem 2.5.1, from which Theorem 2.5.2 follows directly. We refer the reader to [74] and [154] for the proof of Theorem 2.5.3.

Theorem 2.5.1. *Let the standard assumptions hold. Then there are δ and KI such that if xc ∈ B*(*δ*)*, s satisfies* (2.47)*, and x*+ = *xc* + *s, then*

(2.48) *e*+ *≤ KI* (*ec* + *ηc*)*ec.*

*Proof*. Let *δ* be small enough so that the conclusions of Lemma 2.3.1 and Theorem 2.3.2 hold. To prove the first assertion (2.48) note that if

*r* = *−∇*2*f*(*xc*)*s − ∇f*(*xc*)

is the linear residual, then

*s* + (*∇*2*f*(*xc*))*−*1*∇f*(*xc*) = *−*(*∇*2*f*(*xc*))*−*1*r*

and

*e*+ = *ec* + *s* = *ec −* (*∇*2*f*(*xc*))*−*1*∇f*(*xc*) *−* (*∇*2*f*(*xc*))*−*1 (2.49) *r.*

Now, (2.47), (2.7), and (2.6) imply that

*s* + (*∇*2*f*(*xc*))*−*1*∇f*(*xc*) *≤*(*∇*2*f*(*xc*))*−*1*ηc∇f*(*xc*)

*≤* 4*κ*(*∇*2*f*(*x∗*))*ηcec.*

Hence, using (2.49) and Theorem 2.3.2, we have that

*e*+ *≤ec − ∇*2*f*(*xc*)*−*1*∇f*(*xc*) + 4*κ*(*F*(*x∗*))*ηcec*

*≤ Kec*2 + 4*κ*(*∇*2*f*(*x∗*))*ηcec,*

where *K* is the constant from (2.8). If we set

*KI* = *K* + 4*κ*(*∇*2*f*(*x∗*))*,*

the proof is complete.

Theorem 2.5.2. *Let the standard assumptions hold. Then there are δ and η*¯ *such that if x*0 *∈ B*(*δ*)*, {ηn} ⊂* [0*, η*¯]*, then the inexact Newton iteration*

*xn*+1 = *xn* + *sn,*

*where*

*∇*2*f*(*xn*)*sn* + *∇f*(*xn*) *≤ ηn∇f*(*xn*)*,*

*converges q-linearly to x∗. Moreover*

*• if ηn →* 0 *the convergence is q-superlinear, and*

*• if ηn ≤ Kη∇f*(*xn*)*p for some Kη >* 0 *the convergence is q-superlinear with q-order* 1 + *p.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

30 ITERATIVE METHODS FOR OPTIMIZATION

The similarity between Theorem 2.5.2 and Theorem 2.3.5, the convergence result for the chord method, should be clear. Rather than require that the approximate Hessian be accurate, we demand that the linear iteration produce a sufficiently small relative residual. Theorem 2.5.3 is the remarkable statement that any reduction in the relative linear residual will suffice for linear convergence in a certain norm. This statement implies [154] that *∇f*(*xn*) will converge to zero q-linearly, or, equivalently, that *xn → x∗* q-linearly with respect to *·∗*, which is defined by

*x∗* = *∇*2*f*(*x∗*)*x.*

Theorem 2.5.3. *Let the standard assumptions hold. Then there is δ such that if xc ∈ B*(*δ*)*, s satisfies* (2.47)*, x*+ = *xc* + *s, and ηc ≤ η < η <*¯ 1*, then*

(2.50) *e*+*∗ ≤ η*¯*ec∗.*

Theorem 2.5.4. *Let the standard assumptions hold. Then there is δ such that if x*0 *∈ B*(*δ*)*, {ηn} ⊂* [0*, η*] *with η < η <*¯ 1*, then the inexact Newton iteration*

*xn*+1 = *xn* + *sn,*

*where*

*∇*2*f*(*xn*)*sn* + *∇f*(*xn*) *≤ ηn∇f*(*xn*)

*converges q-linearly with respect to ·∗ to x∗. Moreover*

*• if ηn →* 0 *the convergence is q-superlinear, and*

*• if ηn ≤ Kη∇f*(*xn*)*p for some Kη >* 0 *the convergence is q-superlinear with q-order* 1 + *p.*

Q-linear convergence of *{xn}* to a local minimizer with respect to *·∗* is equivalent to q-linear convergence of *{∇f*(*xn*)*}* to zero. We will use the rate of convergence of *{∇f*(*xn*)*}* in our computational examples to compare various methods.

**2.5.2 Implementation of Newton–CG**

Our implementation of Newton–CG approximately solves the equation for the Newton step with CG. We make the implicit assumption that *∇f* has been computed sufficiently accurately for *D*2*hf*(*x* : *w*) to be a useful approximate Hessian of the Hessian–vector product *∇*2*f*(*x*)*w*.

**Forward Difference CG**

Algorithm fdcg is an implementation of the solution by CG of the equation for the Newton step (2.4). In this algorithm we take care to identify failure in CG (i.e., detection of a vector *p* for which *pT Hp ≤* 0). This failure either means that *H* is singular (*pT Hp* = 0; see exercise 2.7.3) or that *pT Hp <* 0, i.e., *p* is a *direction of negative curvature*. The algorithms we will discuss in *§*3.3.7 make good use of directions of negative curvature. The initial iterate for forward difference CG iteration should be the zero vector. In this way the first iterate will give a steepest descent step, a fact that is very useful. The inputs to Algorithm fdcg are the current point *x*, the objective *f*, the forcing term *η*, and a limit on the number of iterations *kmax*. The output is the inexact Newton direction *d*. Note that in step 2b *D*2*hf*(*x* : *p*) is used as an approximation to *∇*2*f*(*x*)*p*.

Algorithm 2.5.1. fdcg(*x, f, η, kmax, d*)

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 31

1. *r* = *−∇f*(*x*), *ρ*0 = *r*22, *k* = 1, *d* = 0.

2. *Do While √~~ρ~~k−*1 *> η∇f*(*x*) *and k < kmax*

(a) *if k* = 1 *then p* = *r*

*else*

*β* = *ρk−*1*/ρk−*2 *and p* = *r* + *βp*

(b) *w* = *D*2*hf*(*x* : *p*)

*If pT w* = 0 *signal indefiniteness; stop.*

*If pT w <* 0 *signal negative curvature; stop.*

(c) *α* = *ρk−*1*/pT w*

(d) *d* = *d* + *αp*

(e) *r* = *r − αw*

(f) *ρk* = *r*2

(g) *k* = *k* + 1

Preconditioning can be incorporated into a Newton–CG algorithm by using a forward dif ference formulation, too. Here, as in [154], we denote the preconditioner by *M*. Aside from *M*, the inputs and output of Algorithm fdpcg are the same as that for Algorithm fdcg.

Algorithm 2.5.2. fdpcg(*x, f, M, η, kmax, d*)

1. *r* = *−∇f*(*x*), *ρ*0 = *r*22, *k* = 1, *d* = 0*.*

2. *Do While √~~ρ~~k−*1 *> η∇f*(*x*) *and k < kmax*

(a) *z* = *Mr*

(b) *τk−*1 = *zT r*

(c) *if k* = 1 *then β* = 0 *and p* = *z*

*else*

*β* = *τk−*1*/τk−*2, *p* = *z* + *βp*

(d) *w* = *D*2*hf*(*x* : *p*)

*If pT w* = 0 *signal indefiniteness; stop.*

*If pT w <* 0 *signal negative curvature; stop.*

(e) *α* = *τk−*1*/pT w*

(f) *d* = *d* + *αp*

(g) *r* = *r − αw*

(h) *ρk* = *rT r*

(i) *k* = *k* + 1

In our formulation of Algorithms fdcg and fdpcg, indefiniteness is a signal that we are not sufficiently near a minimum for the theory in this section to hold. In *§*3.3.7 we show how negative curvature can be exploited when far from the solution.

One view of preconditioning is that it is no more than a rescaling of the independent variables. Suppose, rather than (1.2), we seek to solve

min*y*ˆ (2.51) *f*(*y*)*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

32 ITERATIVE METHODS FOR OPTIMIZATION

where ˆ*f*(*y*) = *f*(*M*1*/*2*y*) and *M* is spd. If *y∗* is a local minimizer of ˆ*f*, then *x∗* = *M*1*/*2*y∗* is a local minimizer of *f* and the two problems are equivalent. Moreover, if *x* = *M*1*/*2*y* and *∇x* and *∇y* denote gradients in the *x* and *y* coordinates, then

*∇y* ˆ*f*(*y*) = *M*1*/*2*∇xf*(*x*)

and

*∇*2*y* ˆ*f*(*y*) = *M*1*/*2(*∇*2*xf*(*x*))*M*1*/*2*.*

Hence, the scaling matrix plays the role of the square root of the preconditioner for the precon ditioned conjugate gradient algorithm.

**Newton–CG**

The theory guarantees that if *x*0 is near enough to a local minimizer then *∇*2*f*(*xn*) will be spd for the entire iteration and *xn* will converge rapidly to *x∗*. Hence, Algorithm newtcg will not terminate with failure because of an increase in *f* or an indefinite Hessian. Note that both the forcing term *η* and the preconditioner *M* can change as the iteration progresses.

Algorithm 2.5.3. newtcg(*x, f, τ, η*)

1. *rc* = *r*0 = *∇f*(*x*)

2. *Do while ∇f*(*x*) *> τrr*0 + *τa*

(a) *Select η and a preconditioner M*.

(b) fdpcg(*x, f, M, η, kmax, d*)

*If indefiniteness has been detected, terminate with failure.*

(c) *x* = *x* + *d*.

(d) *Evaluate f and ∇f*(*x*)*.*

*If f has not decreased, terminate with failure.*

(e) *r*+ = *∇f*(*x*)*, σ* = *r*+*/rc, rc* = *r*+*.*

The implementation of Newton–CG is simple, but, as presented in Algorithm newtcg, incomplete. The algorithm requires substantial modification to be able to generate the good initial data that the local theory requires. We return to this issue in *§*3.3.7.

There is a subtle problem with Algorithm fdpcg in that the algorithm is equivalent to the application of the preconditioned conjugate gradient algorithm to the matrix *B* that is determined by

*Bpi* = *wi* = *D*2*hf*(*x* : *pi*)*,* 1 *≤ i ≤ N.*

However, since the map *p → D*2*hf*(*x* : *p*) is not linear in *p*, the quality of *B* as an approximation to *∇*2*f*(*x*) may degrade as the linear iteration progresses. Usually this will not cause problems unless many iterations are needed to satisfy the inexact Newton condition. However, if one does not see the expected rate of convergence in a Newton–CG iteration, this could be a factor [128]. One partial remedy is to use a centered-difference Hessian–vector product [162], which reduces the error in *B*. In exercise 2.7.15 we discuss a more complex and imaginative way to compute accurate Hessians.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 33 **2.6 Examples**

In this section we used the collection of MATLAB codes but disabled the features (see Chapter 3) that assist in convergence when far from a minimizer. We took care to make certain that the initial iterates were near enough to the minimizer so that the observations of local convergence corresponded to the theory. In practical optimization problems, good initial data is usually not available and the globally convergent methods discussed in Chapter 3 must be used to start the iteration.

The plots in this section have the characteristics of local convergence in that both the gradient norms and function values are decreasing. The reader should contrast this with the examples in Chapter 3.

**2.6.1 Parameter Identification**

For this example, *M* = 100, and the observations are those of the exact solution with *c* = *k* = 1, which we computed analytically. We used *T* = 10 and *u*0 = 10. We computed the displacement and solved the sensitivity equations with the stiff solver ode15s. These results could be obtained as well in a FORTRAN environment using, for example, the LSODE code [228]. The relative and absolute error tolerances for the integrator were both set to 10*−*8. In view of the expected accuracy of the gradient, we set the forward difference increment for the approximate Hessian to *h* = 10*−*4. We terminated the iterations when *∇f <* 10*−*4. Our reasons for this are that, for the zero residual problem considered here, the standard assumptions imply that *f*(*x*) = *O*(*∇f*(*x*)) for *x* near the solution. Hence, since we can only resolve *f* to an accuracy of 10*−*8, iteration beyond the point where *∇f <* 10*−*4 cannot be expected to lead to a further decrease in *f*. In fact we observed this in our computations.

The iterations are very sensitive to the initial iterate. We used *x*0 = (1*.*1*,* 1*.*05)*T* ; initial iterates much worse than that caused Newton’s method to fail. The more robust methods from Chapter 3 should be viewed as essential components of even a simple optimization code.

In Table 2.1 we tabulate the history of the iteration for both the Newton and Gauss–Newton methods. As expected for a small residual problem, Gauss–Newton performs well and, for this example, even converges in fewer iterations. The real benefit of Gauss–Newton is that com putation of the Hessian can be avoided, saving considerable computational work by exploiting the structure of the problem. In the computation reported here, the MATLAB flops com mand indicates that the Newton iteration took roughly 1.9 million floating point operations and Gauss–Newton roughly 650 thousand. This difference would be much more dramatic if there were more than two parameters or the cost of an evaluation of *f* depended on *N* in a significant way (which it does not in this example).

Table 2.1: *Parameter identification problem, locally convergent iterations.*

Newton Gauss–Newton

n *∇f*(*xn*) *f*(*xn*) *∇f*(*xn*) *f*(*xn*)

0 2.33e+01 7.88e-01 2.33e+01 7.88e-01

1 6.87e+00 9.90e-02 1.77e+00 6.76e-03

2 4.59e-01 6.58e-04 1.01e-02 4.57e-07

3 2.96e-03 3.06e-08 9.84e-07 2.28e-14

4 2.16e-06 4.15e-14

Figure 2.1 is a graphical representation of the convergence history from Table 2.1. We think that the plots are a more effective way to understand iteration statistics and will present mostly

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

34 ITERATIVE METHODS FOR OPTIMIZATION

graphs for the remainder of the book. The concavity of the plots of the gradient norms is the signature of superlinear convergence.

Gradient Norm

102Newton’s Method 100

10−2

10−4

Function Value

100Newton’s Method 10−5

10−10

10−6

0 2 4 Iterations

10−15

0 2 4 Iterations

Gradient Norm

102Gauss−Newton Method 100

10−2

10−4

10−6

Function Value

100Gauss−Newton Method 10−5

10−10

10−8

0 1 2 3 Iterations

10−15

0 1 2 3 Iterations

Figure 2.1: *Local Optimization for the Parameter ID Problem*

We next illustrate the difference between Gauss–Newton and Newton on a nonzero residual problem. We use the same example as before with the observations randomly perturbed. We used the MATLAB rand function for this, perturbing the samples of the analytic solution by *.*5 *×* rand(*M,* 1). The least squares residual is about 3.6 and the plots in Figure 2.2 indicate that Newton’s method is still converging quadratically, but the rate of Gauss–Newton is linear. The linear convergence of Gauss–Newton can be seen clearly from the linear semilog plot of the gradient norms. Even so, the Gauss–Newton iteration was more efficient, in terms of floating point operation, than Newton’s method. The Gauss–Newton iteration took roughly 1 million floating point operations while the Newton iteration took 1.4 million.

**2.6.2 Discrete Control Problem**

We solve the discrete control problem from *§*1.6.1 with *N* = 400, *T* = 1, *y*0 = 0, *L*(*y, u, t*)=(*y −* 3)2 + *.*5*u*2*,* and *φ*(*y, u, t*) = *uy* + *t*2

with Newton–CG and two different choices, *η* = *.*1*, .*0001, of the forcing term. The initial iterate was *u*0 = (10*,* 10*,...,* 10)*T* and the iteration was terminated when *∇f <* 10*−*8. In

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. LOCAL CONVERGENCE 35

Gradient Norm

102Newton’s Method 100

10−2

10−4

Function Value

4

3.9 3.8 3.7

Newton’s Method

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

10−6

0 1 2 3 Iterations

3.6

0 1 2 3 Iterations

Gradient Norm

102Gauss−Newton Method 100

10−2

10−4

Function Value

4

3.9 3.8 3.7

Gauss−Newton Method

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

10−6

0 2 4 6 Iterations

3.6

0 2 4 6 Iterations

Figure 2.2: *Local Optimization for the Parameter ID Problem, Nonzero Residual*

Figure 2.3 one can see that the small forcing term produces an iteration history with the concavity of superlinear convergence. The limiting q-linear behavior of an iteration with constant *η* is not yet visible. The iteration with the larger value of *η* is in the q-linearly convergent stage, as the linear plot of *∇f* against the iteration counter shows.

The cost of the computation is not reflected by the number of nonlinear iterations. When *η* = *.*0001, the high accuracy of the linear solve is not rewarded. The computation with *η* = *.*0001 required 8 nonlinear iterations, a total of 32 CG iterations, roughly 1.25 million floating point operations, and 41 gradient evaluations. The optimization with *η* = *.*1 needed 10 nonlinear iterations, a total of 13 CG iterations, roughly 820 thousand floating point operations, and 24 gradient evaluations.

**2.7 Exercises on Local Convergence**

2.7.1. Apply Newton’s method with (a) analytic first and second derivatives, (b) analytic first derivatives and forward difference second derivatives, and (c) forward difference first and second derivatives to find a local minimum of

1. *f*(*x*) = sin2(*x*),

2. *f*(*x*) = *ex*2, and

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. 36 ITERATIVE METHODS FOR OPTIMIZATION

Gradient Norm

105 eta = .1 100

10−5

Function Value

103 eta = .1 102

101

10−10

0 5 10 Iterations

100

0 5 10 Iterations

Gradient Norm

105 eta = .0001 100

10−5

10−10

Function Value

103 eta = .0001 102

101

10−15

0 2 4 6 8 Iterations

100

0 2 4 6 8 Iterations

Figure 2.3: *Newton–CG for the Discrete Control Problem: η* = *.*1*, .*0001

3. *f*(*x*) = *x*4.

Use difference steps of *h* = 10*−*1*,* 10*−*2*,* 10*−*4, and 10*−*8. Explain your results. 2.7.2. Repeat part (c) of exercise 2.7.1. Experiment with

*f*(*x*) = *ex*2+ 10*−*4rand(*x*) and *f*(*x*) = *x*2 + 10*−*4rand(*x*)*,*

where *rand* denotes the random number generator in your computing environment. Ex plain the differences in the results.

2.7.3. Show that if *A* is symmetric, *p* = 0, and *pT Ap* = 0, then *A* is either singular or indefinite.

2.7.4. Show that if *b ∈ RN* and the *N ×N* matrix *A* is symmetric and has a negative eigenvalue, then the quadratic functional

*m*(*x*) = *xT Ax* + *xT b*

does not have a minimizer. Show that if *u* is an eigenvector corresponding to a negative eigenvalue of the Hessian, then *u* is a direction of negative curvature.

2.7.5. If *N* = 1, the local quadratic model could easily be replaced by a local *quartic* (i.e., fourth-degree) model (what would be wrong with a cubic model?). If a method is based on minimization of the local quartic model, what kind of local convergence would you

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

LOCAL CONVERGENCE 37

expect? How would you extend this method to the case *N >* 1? Look at [30] for some results on this.

2.7.6. Show that if the standard assumptions hold, *h* is sufficiently small, and *x* is sufficiently near *x∗*, the difference Hessian defined by (2.19), *∇*2*hf*(*x*), is spd.

2.7.7. Write a locally convergent Newton method code based on accurate function and gradient information and forward difference Hessians using (2.19). Be sure that your code tests for positivity of the Hessian so that you can avoid convergence to a local maximum. Is the test for positivity expensive? Apply your code to the parameter ID problem from *§*1.6.2. If you use an ODE solver that lets you control the accuracy of the integration, try values of the accuracy from 10*−*8 to 10*−*2 and see how the iteration performs. Be sure that your difference Hessian reflects the accuracy in the gradient.

2.7.8. Let the standard assumptions hold and let *λs >* 0 be the smallest eigenvalue of *∇*2*f*(*x∗*). Give the best (i.e., largest) bound you can for *ρ* such that *∇*2*f*(*x*) is positive definite for all *x ∈ B*(*ρ*).

2.7.9. Use the definition of *A†* to prove (2.41).

2.7.10. Fill in the missing details in the proof of Theorem 2.4.2 by showing how the Kantorovich theorem can be used to prove the existence of *ξ*(*x*).

2.7.11. Let *f*(*x*) = *x*2 and *f* (*x*) = sin(100*x*)*/*10. Using an initial iterate of *x*0 = 1, try to find a local minimum of *f* + *f* using Newton’s method with analytic gradients and Hessians. Repeat the experiment with difference gradients and Hessians (try forward differences with a step size of *h* = *.*2).

2.7.12. Solve the parameter ID problem from *§*2.6 with the observations perturbed randomly (for example, you could use the MATLAB rand function for this). Vary the amplitude of the perturbation and see how the performance of Newton and Gauss–Newton changes.

2.7.13. Derive sensitivity equations for the entries of the Hessian for the parameter ID objective function. In general, if there are *P* parameters, how many sensitivity equations would need to be solved for the gradient? How many for the Hessian?

2.7.14. Solve the discrete control problem from *§*2.6.2 using Newton–CG with forcing terms that depend on *n*. Consider *ηn* = *.*5*/n*, *ηn* = min(*.*1*, ∇f*(*un*)), and some of the choices from [99]. Vary *N* and the termination criteria and compare the performance with the constant *η* choice in *§*2.6.2.

2.7.15. Let *F* : *RN → RM* (where *M* and *N* need not be the same) be sufficiently smooth (how smooth is that?) and be such that *F* can also be computed for complex arguments. Show that [181], [245]

Im(*F*(*x* + *ihu*))*/h* = *F*(*x*)*u* + *O*(*h*2)*,*

where *Im* denotes imaginary part. What happens if there is error in *F*? How can you use this fact to compute better difference gradients and Hessians?

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Chapter 3**

**Global Convergence**

The locally convergent algorithms discussed in Chapter 2 can and do fail when the initial iterate is not near the root. The reasons for this failure, as we explain below, are that the Newton direction may fail to be a direction of descent for *f* and that even when a search direction is a direction of decrease of *f*, as *−∇f* is, the length of the step can be too long. Hence, taking a Newton (or Gauss–Newton, or inexact Newton) step can lead to an increase in the function and divergence of the iteration (see exercise 3.5.14 for two dramatic examples of this). The *globally convergent* algorithms developed in this chapter partially address this problem by either finding a local minimum or failing in one of a small number of easily detectable ways.

These are not algorithms for global optimization. When these algorithms are applied to problems with many local minima, the results of the iteration may depend in complex ways on the initial iterate.

**3.1 The Method of Steepest Descent**

The *steepest descent direction* from *x* is *d* = *−∇f*(*x*). The *method of steepest descent* [52] updates the current iteration *xc* by the formula

(3.1) *x*+ = *xc − λ∇f*(*xc*)*.*

If we take the simple choice *λ* = 1, then *x*+ is not guaranteed to be nearer a solution than *xc*, even if *xc* is very near a solution that satisfies the standard assumptions. The reason for this is that, unlike the Newton direction, the steepest descent direction scales with *f*. The Newton step, on the other hand, is the same for *f* as it is for *cf* for any *c* = 0 but need not be a direction of decrease for *f*.

To make the method of steepest descent succeed, it is important to choose the *steplength λ*. One way to do this, which we analyze in *§*3.2, is to let *λ* = *βm*, where *β ∈* (0*,* 1) and *m ≥* 0 is the smallest nonnegative integer such that there is *sufficient decrease* in *f*. In the context of the steepest descent algorithm, this means that

*f*(*xc − λ∇f*(*xc*)) *− f*(*xc*) *< −αλ∇f*(*xc*)2 (3.2) *.*

This strategy, introduced in [7] and called the *Armijo rule*, is an example of a *line search* in which one searches on a ray from *xc* in a direction in which *f* is locally decreasing. In (3.2), *α ∈* (0*,* 1) is a parameter, which we discuss after we fully specify the algorithm. This strategy of repeatedly testing for sufficient decrease and reducing the stepsize if the test is failed is called *backtracking* for obvious reasons.

39

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

40 ITERATIVE METHODS FOR OPTIMIZATION

The motivation for (3.2) is that if we approximate *f* by the *linear model*

*mc* = *f*(*xc*) + *∇f*(*xc*)(*x − xc*)*,*

then the reduction in the model (i.e., the *predicted reduction* in *f*) is

*pred* = *mc*(*xc*) *− mc*(*x*+) = *λ∇f*(*xc*)2*.*

(3.2) says that the *actual reduction* in *f*

*ared* = *f*(*xc*) *− f*(*x*+)

is at least as much as a fraction of the predicted reduction in the linear model. The parameter *α* is typically set to 10*−*4.

The reason we demand sufficient decrease instead of *simple decrease* (i.e., *f*(*xc*) *< f*(*x*+) or *α* = 0) is largely theoretical; a nonzero value of *α* is required within the proof to insure that the iteration does not stagnate before convergence.

Algorithm 3.1.1. steep(*x, f, kmax*)

1. *For k* = 1*, . . . , kmax*

(a) *Compute f and ∇f; test for termination.*

(b) *Find the least integer m ≥* 0 *such that* (3.2) *holds for λ* = *βm.*

(c) *x* = *x* + *λd*.

2. *If k* = *kmax and the termination test is failed, signal failure.*

The termination criterion could be based on (2.12), for example.

**3.2 Line Search Methods and the Armijo Rule**

We introduce a few new concepts so that our proof of convergence of Algorithm steep will also apply to a significantly more general class of algorithms.

Definition 3.2.1. *A vector d ∈ RN is a descent direction for f at x if*

*df*(*x* + *td*)

*dt*

= *∇f*(*x*)*T d <* 0*. t*=0

Clearly the steepest descent direction *d* = *−∇f*(*x*) is a descent direction. A *line search algorithm* searches for decrease in *f* in a descent direction, using the Armijo rule for stepsize control, unless *∇f*(*x*)=0.

We will consider descent directions based on *quadratic models* of *f* of the form *m*(*x*) = *f*(*xc*) + *∇f*(*xc*)*T* (*x − xc*) + 12(*x − xc*)*T Hc*(*x − xc*)*,*

where *Hc*, which is sometimes called the *model Hessian*, is spd. We let *d* = *x − xc* be such that *m*(*x*) is minimized. Hence,

*∇m*(*x*) = *∇f*(*xc*) + *Hc*(*x − xc*)=0

and hence

*d* = *−H−*1 (3.3) *c ∇f*(*xc*)*.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 41

The steepest descent direction satisfies (3.3) with *Hc* = *I*. However, the Newton direction *d* = *−∇*2*f*(*x*)*−*1*∇f*(*x*) may fail to be a descent direction if *x* is far from a minimizer because *∇*2*f* may not be spd. Hence, unlike the case for nonlinear equations [154], Newton’s method is not a generally good global method, even with a line search, and must be modified (see [113], [117], [231], and [100]) to make sure that the model Hessians are spd.

The algorithm we analyze in this section is an extension of Algorithm steep that allows for descent directions that satisfy (3.3) for spd *H*. We modify (3.2) to account for *H* and the new descent direction *d* = *−H−*1*∇f*(*x*). The general *sufficient decrease* condition is

*f*(*xc* + *λd*) *− f*(*xc*) *< αλ∇f*(*xc*)*T* (3.4) *d.*

Here, as in (3.2), *α ∈* (0*,* 1) is an algorithmic parameter. Typically *α* = 10*−*4.

The stepsize reduction scheme in step 1b of Algorithm steep is crude. If *β* is too large, too many stepsize reductions may be needed before a step is accepted. If *β* is too small, the progress of the entire iteration may be retarded. We will address this problem in two ways. In *§*3.2.1 we will construct polynomial models of *f* along the descent direction to predict an optimal factor by which to reduce the step. In *§*3.3.3 we describe a method which remembers the steplength from the previous iteration.

Our proofs require only the following general line search strategy. If a steplength *λc* has been rejected (i.e., (3.4) fails with *λ* = *λc*), construct

(3.5) *λ*+ *∈* [*βlowλc, βhighλc*]*,*

where 0 *< βlow ≤ βhigh <* 1. The choice *β* = *βlow* = *βhigh* is the simple rule in Algo rithm steep. An *exact line search*, in which *λ* is the exact minimum of *f*(*xc* + *λd*), is not only not worth the extra expense but can degrade the performance of the algorithm.

Algorithm 3.2.1. optarm(*x, f, kmax*)

1. *For k* = 1*, . . . , kmax*

(a) *Compute f and ∇f; test for termination.*

(b) *Construct an spd matrix H and solve* (3.3) *to obtain a descent direction d.*

(c) *Beginning with λ* = 1*, repeatedly reduce λ using any strategy that satisfies* (3.5) *until* (3.4) *holds.*

(d) *x* = *x* + *λd*.

2. *If k* = *kmax and the termination test is failed, signal failure.*

In the remainder of this section we prove that if the sequence of model Hessians remains uniformly bounded and positive definite and the sequence of function values *{f*(*xk*)*}* is bounded from below, then any limit point of the sequence *{xk}* generated by Algorithm optarm con verges to a point *x∗* that satisfies the necessary conditions for optimality. We follow that analysis with a local convergence theory that is much less impressive than that for Newton’s method.

We begin our analysis with a simple estimate that follows directly from the spectral theorem for spd matrices.

Lemma 3.2.1. *Let H be spd with smallest and largest eigenvalues* 0 *< λs < λl. Then for all z ∈ RN ,*

*λ−*1

*l z*2 *≤ zT H−*1*z ≤ λ−*1

*s z*2*.*

The first step in the analysis is to use Lemma 3.2.1 to obtain a lower bound for the steplength. Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

42 ITERATIVE METHODS FOR OPTIMIZATION

Lemma 3.2.2. *Assume that ∇f is Lipschitz continuous with Lipschitz constant L. Let α ∈* (0*,* 1)*, x ∈ RN , and H be an spd matrix. Let λs >* 0 *be the smallest and λl ≥ λs the largest eigenvalues of H. Let d be given by* (3.3)*. Assume that ∇f*(*x*) = 0*. Then* (3.4) *holds for any λ such that*

0 *< λ ≤*2*λs*(1 *− α*)

*Lκ*(*H*) (3.6) *.*

*Proof*. Let *d* = *−H−*1*∇f*(*x*). By the fundamental theorem of calculus

1

*f*(*x* + *λd*) *− f*(*x*) = 0

*∇f*(*x* + *tλd*)*T λd dt.*

Hence*f*(*x* + *λd*) = *f*(*x*) + *λ∇f*(*x*)*T d*

(3.7)

+*λ* 10 (*∇f*(*x* + *tλd*) *− ∇f*(*x*))*T d dt.*

Therefore,

*f*(*x* + *λd*) = *f*(*x − λH−*1*∇f*(*x*)) *≤ f*(*x*) + *λ∇f*(*x*)*T d* +*λ*2*L*

2 *d*2*.*

Positivity of *H*, Lemma 3.2.1, and *κ*(*H*) = *λlλ−*1 *s* imply that

*d*2 = *H−*1*∇f*(*x*)2 *≤ λ−*2 *s ∇f*(*x*)*T ∇f*(*x*)

*≤ −λlλ−*2 *s ∇f*(*x*)*T d* = *−κ*(*H*)*λ−*1 *s ∇f*(*x*)*T d.*

Hence

*f*(*x* + *λd*) *≤ f*(*x*)+(*λ − λ*2*Lλ−*1

*s κ*(*H*)*/*2)*∇f*(*x*)*T d,*

which implies (3.4) if

*α ≤* (1 *− λLλ−*1

*s κ*(*H*)*/*2)*.*

This is equivalent to (3.6).

Lemma 3.2.3. *Let ∇f be Lipschitz continuous with Lipschitz constant L. Let {xk} be the iteration given by Algorithm* optarm *with spd matrices Hk that satisfy*

(3.8) *κ*(*Hk*) *≤ κ*¯

*for all k. Then the steps*

*sk* = *xk*+1 *− xk* = *λkdk* = *−λkH−*1

*k ∇f*(*xk*)

*satisfy*

*λk ≥ λ*¯ = 2*βlowλs*(1 *− α*)

*Lκ*¯ (3.9)

*and at most*

*m* = log 2*λs*(1 *− α*)

(3.10) */* log(*βhigh*)

*Lκ*¯

*stepsize reductions will be required.*

*Proof*. In the context of Algorithm optarm, Lemma 3.2.2 implies that the line search will

terminate when

*λ ≤*2*λs*(1 *− α*)

*Lκ*(*Hk*) *,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 43

if not before. The most that one can overshoot this is by a factor of *βlow*, which proves (3.9). The line search will require at most *m* stepsize reductions, where *m* is the least nonnegative integer such that

2*λs*(1 *− α*)

*Lκ*(*Hk*) *> βmhigh.*

This implies (3.10).

The convergence theorem for Algorithm optarm says that if the condition numbers of the matrices *H* and the norms of the iterates remain bounded, then every limit point of the iteration is a stationary point. Boundedness of the sequence of iterates implies that there will be limit points, but there is no guarantee that there is a unique limit point.

Theorem 3.2.4. *Let ∇f be Lipschitz continuous with Lipschitz constant L. Assume that the matrices Hk are spd and that there are κ*¯ *and λl such that κ*(*Hk*) *≤ κ*¯*, and Hk ≤ λl for all k. Then either f*(*xk*) *is unbounded from below or*

lim

*k→∞* (3.11) *∇f*(*xk*)=0

*and hence any limit point of the sequence of iterates produced by Algorithm* optarm *is a stationary point.*

*In particular, if f*(*xk*) *is bounded from below and xkl → x∗ is any convergent subsequence of {xk}, then ∇f*(*x∗*)=0*.*

*Proof*. By construction, *f*(*xk*) is a decreasing sequence. Therefore, if *f*(*xk*) is bounded from below, then lim*k→∞ f*(*xk*) = *f ∗* exists and

lim

*k→∞* (3.12) *f*(*xk*+1) *− f*(*xk*)=0*.*

By (3.4) and Lemma 3.2.3 we have

*f*(*xk*+1) *− f*(*xk*) *< −αλk∇f*(*xk*)*T H−*1

*k ∇f*(*xk*)

*≤ −αλλ*¯ *−*1

*l ∇f*(*xk*)2 *≤* 0*.*

Hence, by (3.12)

*∇f*(*xk*)2 *≤λl*(*f*(*xk*) *− f*(*xk*+1))

*αλ*~~¯~~ *→* 0

as *k → ∞*. This completes the proof.

The analysis of the Armijo rule is valid for other line search methods [84], [125], [272], [273]. The key points are that the sufficient decrease condition can be satisfied in finitely many steps and that the stepsizes are bounded away from zero.

**3.2.1 Stepsize Control with Polynomial Models**

Having computed a descent direction *d* from *xc*, one must decide on a stepsize reduction scheme for iterations in which (3.4) fails for *λ* = 1. A common approach [73], [84], [114], [197], [117] is to model

*ξ*(*λ*) = *f*(*xc* + *λd*)

by a cubic polynomial. The data on hand initially are

*ξ*(0) = *f*(*xc*)*, ξ*(0) = *∇f*(*xc*)*T d <* 0*,* and *ξ*(1) = *f*(*x* + *d*)*,*

which is enough to form a quadratic model of *ξ*. So, if (3.4) does not hold with *λ* = *λ*0 = 1, i.e.,

*ξ*(1) = *f*(*xc* + *d*) *≥ f*(*xc*) + *α∇f*(*xc*)*T d* = *ξ*(0) + *αξ*(0)*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

44 ITERATIVE METHODS FOR OPTIMIZATION

we approximate *ξ* by the quadratic polynomial

*q*(*λ*) = *ξ*(0) + *ξ*(0)*λ* + (*ξ*(1) *− ξ*(0) *− ξ*(0))*λ*2

and let *λ*1 be the minimum of *q* on the interval [*βlow, βhigh*] *⊂* (0*,* 1). This minimum can be computed directly since *α ∈* (0*,* 1) and failure of (3.4) imply

*q*(*λ*) = 2(*ξ*(1) *− ξ*(0) *− ξ*(0)) *>* 2(*α −* 1)*ξ*(0) *>* 0*.*

Therefore, the global minimum of *q* is

*λt* = *−ξ*(0)

2(*ξ*(1) *− ξ*(0) *− ξ*(0))*.*

So

(3.13)

*λ*+ =

 

*βlow, λt ≤ βlow,*

*λt, βlow < λt < βhigh, βhigh, λt ≥ βhigh.*

If our first reduced value of *λ* does not satisfy (3.4), we base additional reductions on the data

*ξ*(0) = *f*(*xc*)*, ξ*(0) = *∇f*(*xc*)*T d, ξ*(*λ−*)*, ξ*(*λc*)*,*

where *λc < λ−* are the most recent values of *λ* to fail to satisfy (3.4). This is sufficient data to approximate *ξ* with a cubic polynomial

*q*(*λ*) = *ξ*(0) + *ξ*(0)*λ* + *c*2*λ*2 + *c*3*λ*3*,*

where *c*2 and *c*3 can be determined by the equations

*q*(*λc*) = *ξ*(*λc*) = *f*(*xc* + *λcd*)*,*

*q*(*λ−*) = *ξ*(*λ−*) = *f*(*xc* + *λ−d*)*,*

which form the nonsingular linear system for *c*2 and *c*3

*λ*2*c λ*3*c*

*c*2

*ξ*(*λc*) *− ξ*(0) *− ξ*(0)*λc*

(3.14) *.*

*λ*2*− λ*3*−*

*c*3

=

*ξ*(*λ−*) *− ξ*(0) *− ξ*(0)*λ−*

As with the quadratic case, *q* has a local minimum [84] at

*λt* = *−c*2 + *c*~~2~~2 *−* 3*c*3*ξ*(0)

(3.15) *.*

3*c*3

With *λt* in hand, we compute *λ*+ using (3.13). The application of (3.13) is called *safeguarding* and is important for the theory, as one can see from the proof of Theorem 3.2.4. Safeguarding is also important in practice because, if the cubic model is poor, the unsafeguarded model can make steplength reductions that are so small that the iteration can stagnate or so large (i.e., near 1) that too many reductions are needed before (3.4) holds.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 45

**3.2.2 Slow Convergence of Steepest Descent**

Unfortunately, methods based on steepest descent do not enjoy good local convergence prop erties, even for very simple functions. To illustrate this point we consider the special case of *convex quadratic* objective functions

*f*(*x*) = 12*xT Ax − bT x* + *a,*

where *A* is spd, *b ∈ RN* , and *a* is a scalar. We will look at a very simple example, using the method of steepest descent with *Hk* = *I* (so *λl* = *λs* = 1) and show how the speed of convergence depends on conditioning and scaling.

Lemma 3.2.5. *Let f be a convex quadratic and let Hk* = *I for all k. Then the sequence {xk} generated by Algorithm* optarm *converges to the unique minimizer of f.*

*Proof*. In exercise 3.5.4 you are asked to show that *f* is bounded from below and that *∇f*(*x*) = *Ax − b*. Hence *∇f*(*x∗*) vanishes only at *x∗* = *A−*1*b*. Since *∇*2*f*(*x*) = *A* is spd, the second-order sufficient conditions hold and *x∗* is the unique minimizer of *f*.

Theorem 3.2.4 implies that

*k→∞ ∇f*(*xk*) = *Axk − b* = *A*(*xk − x∗*)=0*,*

lim

and hence *xk → x∗*.

Since the steepest descent iteration converges to the unique minimizer of a convex quadratic, we can investigate the rate of convergence without concern about the initial iterate. We do this in terms of the *A*-norm. The problems can be illustrated with the simplest case *a* = 0 and *b* = 0.

Proposition 3.2.6. *Let f*(*x*) = *xT Ax/*2 *and let {xk} be given by Algorithm* optarm *with Hk* = *I for all k. Then the sequence {xk} satisfies*

*xk*+1*A* = (1 *− O*(*κ*(*A*)*−*2 (3.16) ))*xkA.*

*Proof*. The sufficient decrease condition, (3.4), implies that for all *k*

*xTk*+1*Axk*+1 *− xTk Axk* = 2(*f*(*xk*+1) *− f*(*xk*))

(3.17)

*≤* 2*α∇f*(*xk*)*T* (*xk*+1 *− xk*)

= 2*αλk∇f*(*xk*)*T d* = *−*2*αλk*(*Axk*)*T* (*Axk*)*.*

The Lipschitz constant of *∇f* is simply *λl* = *A*; hence we may write (3.9) as *λk ≥ λ*¯ = 2*β*(1 *− α*)

*λlκ*(*A*) (3.18) *.*

In terms of the *A*-norm, (3.17) can be written as

*xk*+12*A − xk*2*A ≤ −*2*αλλ*¯ *sxk*2*A,*

where we use the fact that

*Az*2 = (*Az*)*T* (*Az*) *≥ λszT Az* = *λsz*2*A.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

46 ITERATIVE METHODS FOR OPTIMIZATION

Hence,

*xk*+12*A ≤* (1 *−* 2*αλλ*¯ *s*)*xk*2*A ≤* (1 *−* 4*α*(1 *− α*)*βκ*(*A*)*−*2)*xk*2*A.*

This completes the proof.

Now we consider two specific examples. Let *N* = 1 and define

*f*(*x*) = *ωx*2*/*2*,*

where

(3.19) *ω <* 2(1 *− α*)*.*

In this case *x∗* = 0. We have *∇f*(*x*) = *f*(*x*) = *ωx* and hence for all *x ∈ R*,

*f*(*x − ∇f*(*x*)) *− f*(*x*) = *ωx*2

2 ((1 *− ω*)2 *−* 1)

= *ω*2*x*2

2 (*ω −* 2)

*< −α|f*(*x*)*|*2 = *−αω*2*x*2

because (3.19) implies that

*ω −* 2 *< −*2*α.*

Hence (3.4) holds with *d* = *∇f*(*x*) and *λ* = 1 for all *x ∈ R*. The rate of convergence can be computed directly since

*x*+ = (1 *− ω*)*xc*

for all *xc*. The convergence is q-linear with q-factor 1*−ω*. So if *ω* is very small, the convergence will be extremely slow.

Similarly, if *ω* is large, we see that

*f*(*x − λ∇f*(*x*)) *− f*(*x*) = *ω*2*x*2

2 (*λω −* 2) *< −αλω*2*x*2

only if

*λ <* 2(1 *− α*)

*ω .*

So

*β* 2(1 *− α*)

*ω< βm* = *λ <* 2(1 *− α*) *ω .*

If *ω* is very large, many steplength reductions will be required with each iteration and the line search will be very inefficient.

These are examples of *poor scaling*, where a change in *f* by a multiplicative factor can dramatically improve the efficiency of the line search or the convergence speed. In fact, if *ω* = 1, steepest descent and Newton’s method are the same and only one iteration is required.

The case for a general convex quadratic is similar. Let *λl* and *λs* be the largest and smallest eigenvalues of the spd matrix *A*. We assume that *b* = 0 and *a* = 0 for this example. We let *ul* and *us* be unit eigenvectors corresponding to the eigenvalues *λl* and *λs*. If

*λs <* 2(1 *− α*)

is small and the initial iterate is in the direction of *us*, convergence will require a very large number of iterations (slow). If *λl* is large and the initial iterate is in the direction of *ul*, the line search will be inefficient (many stepsize reductions at each iteration).

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 47

Newton’s method does not suffer from poor scaling of *f* and converges rapidly with no need for a line search when the initial iterate is near the solution. However, when far away from the solution, the Newton direction may not be a descent direction at all and the line search may fail. Making the transition from steepest descent, which is a good algorithm when far from the solution, to Newton’s or some other superlinearly convergent method as the iteration moves toward the solution, is the central problem in the design of line search algorithms. The scaling problems discussed above must also be addressed, even when far from the solution.

**3.2.3 Damped Gauss–Newton Iteration**

As we showed in *§*2.4, the steepest descent direction for the overdetermined least squares objec tive

*ri*(*x*)22 = 12*R*(*x*)*T R*(*x*)

is

*f*(*x*) = 12*M i*=1

*−∇f*(*x*) = *−R*(*x*)*T R*(*x*)*.*

The steepest descent algorithm could be applied to nonlinear least squares problems with the good global performance and poor local convergence that we expect.

The Gauss–Newton direction at *x*

*dGS* = *−*(*R*(*x*)*T R*(*x*))*−*1*R*(*x*)*T R*(*x*)

is not defined if *R*fails to have full column rank. If *R*does have full column rank, then (*dGS*)*T ∇f*(*x*) = *−*(*R*(*x*)*T R*(*x*))*T* (*R*(*x*)*T R*(*x*))*−*1*R*(*x*)*T R*(*x*) *<* 0*,*

and the Gauss–Newton direction is a descent direction. The combination of the Armijo rule with the Gauss–Newton direction is called *damped Gauss–Newton* iteration.

A problem with the damped Gauss–Newton algorithm is that, in order for Theorem 3.2.4 to be applicable, the matrices *{R*(*xk*)*T R*(*xk*)*}* must not only have full column rank but also must be uniformly bounded and well conditioned, which are very strong assumptions (but if they are satisfied, damped Gauss–Newton is a very effective algorithm).

The *Levenberg–Marquardt* method [172], [183] addresses these issues by adding a regular ization parameter *ν >* 0 to *R*(*xc*)*T R*(*xc*) to obtain *x*+ = *xc* + *s* where

*s* = *−*(*νcI* + *R*(*xc*)*T R*(*xc*))*−*1*R*(*xc*)*T* (3.20) *R*(*xc*)*,*

where *I* is the *N × N* identity matrix. The matrix *νcI* + *R*(*xc*)*T R*(*xc*) is positive definite. The parameter *ν* is called the *Levenberg–Marquardt parameter*.

It is not necessary to compute *R*(*xc*)*T R*(*xc*) to compute a Levenberg–Marquardt step [76]. One can also solve the full-rank (*M* + *N*) *× N* linear least squares problem

2

(3.21)

min12*R*(*xc*) *√~~ν~~c~~I~~*

*s* +

*R*(*xc*)

0

to compute *s* (see exercise 3.5.6). Compare this with computing the undamped Gauss–Newton step by solving (2.33).

If one couples the Levenberg–Marquardt method with the Armijo rule, then Theorem 3.2.4 is applicable far from a minimizer and Theorem 2.4.1 nearby. We ask the reader to provide the details of the proof in exercise 3.5.7.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

48 ITERATIVE METHODS FOR OPTIMIZATION

Theorem3.2.7. *LetRbe Lipschitz continuous. Let xk be the Levenberg–Marquardt–Armijo iterates. Assume that R*(*xk*) *is uniformly bounded and that the sequence of Levenberg– Marquardt parameters {νk} is such that*

*κ*(*νkI* + *R*(*xk*)*T R*(*xk*))

*is bounded. Then*

*k→∞ R*(*xk*)*T R*(*xk*)=0*.*

lim

*Moreover, if x∗ is any limit point of {xk} at which R*(*x∗*)=0*, Assumption* 2.4.1 *holds, and νk →* 0*, then xk → x∗ q-superlinearly. If, moreover,*

*νk* = *O*(*R*(*xk*))

*as k → ∞ then the convergence is q-quadratic.*

For example, if *κ*(*R*(*xk*)*T R*(*xk*)) and *R*(*xk*) are bounded then *νk* = min(1*, R*(*xk*)) would satisfy the assumptions of Theorem 3.2.7. For a zero residual problem, this addresses the potential conditioning problems of the damped Gauss–Newton method and still gives quadratic convergence in the terminal phase of the iteration. The Levenberg–Marquardt–Armijo iteration will also converge, albeit slowly, for a large residual problem.

We will not discuss globally convergent methods for underdetermined least squares problems in this book. We refer the reader to [24], [252], and [253] for discussion of underdetermined problems.

**3.2.4 Nonlinear Conjugate Gradient Methods**

Operationally, the conjugate gradient iteration for a quadratic problem updates the current iter ation with a linear combination of the current residual *r* and a search direction *p*. The search direction is itself a linear combination of previous residuals. Only *r* and *p* need be stored to continue the iteration. The methods discussed in this section seek to continue this idea to more nonlinear problems.

Nonlinear conjugate gradient algorithms have the significant advantage of low storage over most of the other algorithms covered in this book, the method of steepest descent being the exception. For problems so large that the Newton or quasi–Newton methods cannot be imple mented using the available storage, these methods are among the few options (see [177] and [5] for examples).

Linear conjugate gradient seeks to minimize *f*(*x*) = *xT Hx/*2 *− xT b*. The residual *r* = *b − Hx* is simply *−∇f*(*x*), leading to a natural extension to nonlinear problems in which *r*0 = *p*0 = *∇f*(*x*0) and, for *k ≥* 1,

(3.22) *rk* = *∇f*(*xk*) and *pk* = *rk* + *βkpk−*1*.*

The update of *x*

*xk*+1 = *xk* + *αkpk*

can be done with a simple analytic minimization in the quadratic case, but a line search will be needed in the nonlinear case. The missing pieces, therefore, are the choice of *βk*, the way the line search will be done, and convergence theory. Theory is needed, for example, to decide if *pk* is a descent direction for all *k*.

The general form of the algorithm is very simple. The inputs are an initial iterate, which will be overwritten by the solution, the function to be minimized, and a termination vector *τ* = (*τr, τa*) of relative and absolute residuals.

Algorithm 3.2.2. nlcg(*x, f, τ* )

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 49

1. *r*0 = *∇f*(*x*); *k* = 0

2. *Do while ∇f*(*x*) *> τrr*0 + *τa*

(a) *If k* = 0 *then p* = *−∇f*(*x*) *else*

*p* = *−∇f*(*x*) + *βp*

(b) *x* = *x* + *αp*

The two most common choices for *β*, both of which are equivalent to the linear CG formula in the quadratic case, are the Fletcher–Reeves [106]

*k* = *∇f*(*xk*)2

*βF R*

*∇f*(*xk−*1)2 (3.23)

and Polak–Ribi`ere [215], [216]

*βk* = *∇f*(*xk*)*T* (*∇f*(*xk*) *− ∇f*(*xk−*1))

*∇f*(*xk−*1)2 (3.24)

formulas. The Fletcher–Reeves method has been observed to take long sequences of very small steps and virtually stagnate [112], [207], [208], [226]. The Polak–Ribi`ere formula performs much better and is more commonly used but has a less satisfactory convergence theory.

The line search has more stringent requirements, at least for proofs of convergence, than are satisfied by the Armijo method that we advocate for steepest descent. We require that the steplength parameter satisfies the *Wolfe conditions* [272], [273]

*f*(*xk* + *αkpk*) *≤ f*(*xk*) + *σααk∇f*(*xk*)*T* (3.25) *pk*

and

*∇f*(*xk* + *αkpk*)*T pk ≥ σβ∇f*(*xk*)*T* (3.26) *pk,*

where 0 *< σα < σβ <* 1. The first of the Wolfe conditions (3.25) is the sufficient decrease condition, (3.4), that all line search algorithms must satisfy. The second (3.26) is often, but not always, implied by the Armijo backtracking scheme of alternating a test for sufficient decrease and reduction of the steplength. One can design a line search method that will, under modest assumptions, find a steplength satisfying the Wolfe conditions [104], [171], [193].

The convergence result [3] for the Fletcher–Reeves formula requires a bit more. The proof that *pk* is descent direction requires the *strong Wolfe conditions*, which replace (3.26) by

*|∇f*(*xk* + *αkpk*)*T pk|≤−σβ∇f*(*xk*)*T* (3.27) *pk*

and demand that 0 *< σα < σβ <* 1*/*2. The algorithm from [193], for example, will find a point satisfying the strong Wolfe conditions.

Theorem 3.2.8. *Assume that the set*

*N* = *{x | f*(*x*) *≤ f*(*x*0)*}*

*is bounded and that f is Lipschitz continuously differentiable in a neighborhood of N . Let Algorithm* nlcg*be implemented with the Fletcher–Reeves formula and a line search that satisfies the strong Wolfe conditions. Then*

lim *∇f*(*xk*)=0*.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

50 ITERATIVE METHODS FOR OPTIMIZATION

This result has been generalized to allow for any choice of *βk* such that *|βk| ≤ βF R*

*k* [112].

A similar result for the Polak–Ribi`ere method, but with more complex conditions on the line search, has been proved in [134]. This complexity in the line search is probably necessary, as there are examples where reasonable line searches lead to failure in the Polak–Ribi`ere method, [222]. One can also prove convergence if *βP R*

*k* is replaced by max(*βP R*

*k ,* 0) [112].

There is continuing research on these methods and we point to [112], [134], [205], and [202] as good sources.

**3.3 Trust Region Methods**

*Trust region* methods overcome the problems that line search methods encounter with non-spd approximate Hessians. In particular, a Newton trust region strategy allows the use of complete Hessian information even in regions where the Hessian has negative curvature. The specific trust region methods we will present effect a smooth transition from the steepest descent direction to the Newton direction in a way that gives the global convergence properties of steepest descent and the fast local convergence of Newton’s method.

The idea is very simple. We let ∆ be the radius of the ball about *xc* in which the quadratic model

*mc*(*x*) = *f*(*xc*) + *∇f*(*xc*)*T* (*x − xc*)+(*x − xc*)*T Hc*(*x − xc*)*/*2

can be trusted to accurately represent the function. ∆ is called the *trust region radius* and the ball

*T* (∆) = *{x | x − xc ≤* ∆*}*

is called the *trust region*.

We compute the new point *x*+ by (approximately) minimizing *mc* over *T* (∆). The *trust region problem* for doing that is usually posed in terms of the difference *st* between *xc* and the minimizer of *mc* in the trust region

(3.28) *mc*(*xc* + *s*)*.*

min

*s≤*∆

We will refer to either the trial step *st* or the trial solution *xt* = *xc* + *st* as the solution to the trust region problem.

Having solved the trust region problem, one must decide whether to accept the step and/or to change the trust region radius. The trust region methods that we will discuss in detail approximate the solution of the trust region problem with the minimizer of the quadratic model along a piecewise linear path contained in the trust region. Before discussing these specific methods, we present a special case of a result from [223] on global convergence.

A prototype trust region algorithm, upon which we base the specific instances that follow, is Algorithm 3.3.1.

Algorithm 3.3.1. trbasic(*x, f*)

1. *Initialize the trust region radius* ∆*.*

2. *Do until termination criteria are satisfied*

(a) *Approximately solve the trust region problem to obtain xt.*

(b) *Test both the trial point and the trust region radius and decide whether or not to accept the step, the trust region radius, or both. At least one of x or* ∆ *will change*

*in this phase.*

Most trust region algorithms differ only in how step 2a in Algorithm trbasic is done. There are also different ways to implement step 2b, but these differ only in minor details and the approach we present next in *§*3.3.1 is very representative.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 51

**3.3.1 Changing the Trust Region and the Step**

The trust region radius and the new point are usually tested simultaneously. While a notion of sufficient decrease is important, the test is centered on how well the quadratic model approximates the function inside the trust region. We measure this by comparing the *actual reduction* in *f*

*ared* = *f*(*xc*) *− f*(*xt*)

with the *predicted reduction*, i.e., the decrease in the quadratic model

*pred* = *mc*(*xc*) *− mc*(*xt*) = *−∇f*(*xc*)*T st − sTt Hcst/*2*.*

*pred >* 0 for all the trust region algorithms we discuss in this book unless *∇f*(*xc*)=0. We will introduce three control parameters

*µ*0 *≤ µlow < µhigh,*

which are used to determine if the trial step should be rejected (*ared/pred < µ*0) and/or the trust region radius should be decreased (*ared/pred < µlow*), increased (*ared/pred > µhigh*), or left unchanged. Typical values are *.*25 for *µlow* and *.*75 for *µhigh*. Both *µ*0 = 10*−*4 or *µ*0 = *µlow* are used. One can also use the sufficient decrease condition (3.4) to determine if the trial step should be accepted [84].

We will contract and expand the trust region radius by simply multiplying ∆ by constants 0 *< ωdown <* 1 *< ωup.*

Typical values are *ωdown* = 1*/*2 and *ωup* = 2. There are many other ways to implement a trust region adjustment algorithm that also give global convergence. For example, the relative error *|pred − ared|/∇f* can be used [84] rather than the ratio *ared/pred*. Finally we limit the number of times the trust region radius can be expanded by requiring

(3.29) ∆ *≤ CT ∇f*(*xc*)*,*

for some *CT >* 1, which may depend on *xc*. This only serves to eliminate the possibility of infinite expansion and is used in the proofs. Many of the dogleg methods which we consider later automatically impose (3.29).

The possibility of expansion is important for efficiency in the case of poor scaling of *f*. The convergence theory presented here [162] also uses the expansion phase in the proof of convergence, but that is not essential. We will present the algorithm to test the trust region in a manner, somewhat different from much of the literature, that only returns once a new iterate has been accepted.

Algorithm 3.3.2. trtest(*xc, xt, x*+*,f,* ∆)

1. *z* = *xc*

2. *Do while z* = *xc*

(a) *ared* = *f*(*xc*) *− f*(*xt*), *st* = *xt − xc*, *pred* = *−∇f*(*xc*)*T st − sTt Hcst/*2

(b) *If ared/pred < µ*0 *then set z* = *xc,* ∆ = *ωdown*∆*, and solve the trust region problem with the new radius to obtain a new trial point. If the trust region radius*

*was just expanded, set z* = *xold*

*t .*

(c) *If µ*0 *≤ ared/pred < µlow, then set z* = *xt and* ∆ = *ωdown*∆*.*

(d) *If µlow ≤ ared/pred ≤ µhigh, set z* = *xt.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

52 ITERATIVE METHODS FOR OPTIMIZATION

(e) *If µhigh < ared/pred and st* = ∆ *≤ CT ∇f*(*xc*)*, then set z* = *xc,* ∆ = *ωup*∆*, and solve the trust region problem with the new radius to obtain a new trial point.*

*Store the old trial point as xold*

*t in case the expansion fails.*

3. *x*+ = *z*.

The loop inAlgorithm trtest serves the same purpose as the loop in a line search algorithm such as Algorithm steep. One must design the solution to the trust region problem in such a way that that loop will terminate after finitely many iterations and a general way to do that is the subject of the next section.

We incorporate Algorithm trtest into a general trust region algorithm paradigm that we will use for the remainder of this section.

Algorithm 3.3.3. trgen(*x, f*)

1. *Initialize* ∆

2. *Do forever*

(a) *Let xc* = *x. Compute ∇f*(*xc*) *and an approximate Hessian Hc.*

*(b) Solve the trust region problem to obtain a trial point xt.*

(c) *Call* trtest(*xc, xt, x, f,* ∆)

Hessians and gradients are computed only in step 2a of Algorithm trgen.

**3.3.2 Global Convergence of Trust Region Algorithms**

While one can, in principal, solve the trust region problem exactly (see *§*3.3.4), it is simpler and more efficient to solve the problem approximately. It is amazing that one need not do a very good job with the trust region problem in order to get global (and even locally superlinear) convergence.

Our demands of our solutions of the trust region problem and our local quadratic models are modest and readily verifiable. The parameter *σ* in part 1 of Assumption 3.3.1, like the parameter *CT* in (3.29), is used in the analysis but plays no role in implementation. In the specific algorithms that we discuss in this book, *σ* can be computed. Part 2 follows from well conditioned and bounded model Hessians if Algorithm trtest is used to manage the trust region.

Assumption 3.3.1.

1. *There is σ >* 0 *such that*

(3.30) *pred* = *f*(*xc*) *− mc*(*xt*) *≥ σ∇f*(*xc*) min(*st, ∇f*(*xc*))*.*

2. *There is M >* 0 *such that either st ≥ ∇f*(*xc*)*/M or st* = ∆*c.*

The global convergence theorem based on this assumption should be compared with the similar result on line search methods—Theorem 3.2.4.

Theorem 3.3.1. *Let ∇f be Lipschitz continuous with Lipschitz constant L. Let {xk} be generated by Algorithm* trgen *and let the solutions for the trust region problems satisfy Assumption* 3.3.1*. Assume that the matrices {Hk} are bounded. Then either f is unbounded from below, ∇f*(*xk*)=0 *for some finite k, or*

*k→∞* (3.31) *∇f*(*xk*)=0*.*

lim

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 53

*Proof*. Assume that *∇f*(*xk*) = 0 for all *k* and that *f* is bounded from below. We will show that there is *MT ∈* (0*,* 1] such that once an iterate is taken (i.e., the step is accepted and the trust region radius is no longer a candidate for expansion), then

(3.32) *sk ≥ MT ∇f*(*xk*)*.*

Assume (3.32) for the present. Since *sk* is an acceptable step, Algorithm trtest and part 1 of Assumption 3.3.1 imply that

*aredk ≥ µ*0*predk ≥ µ*0*∇f*(*xk*)*σ* min(*sk, ∇f*(*xk*))*.*

We may then use (3.32) to obtain

*aredk ≥ µ*0*σMT ∇f*(*xk*)2 (3.33) *.*

Now since *f*(*xk*) is a decreasing sequence and *f* is bounded from below, lim*k→∞ aredk* = 0. Hence (3.33) implies (3.31).

It remains to prove (3.32). To begin note that if *sk <* ∆*k* then by part 2 ofAssumption 3.3.1 *sk ≥ ∇f*(*xk*)*/M.*

Hence, we need only consider the case in which

(3.34) *sk* = ∆*k* and *sk < ∇f*(*xk*)*,*

since if (3.34) does not hold then (3.32) holds with *MT* = min(1*,* 1*/M*).

We will complete the proof by showing that if (3.34) holds and *sk* is accepted, then

*sk* = ∆*k ≥*2*σ* min(1 *− µhigh,*(1 *− µ*0)*ω−*2 *up* )

*M* + *L* (3.35) *∇f*(*xk*)*.*

This will complete the proof with

*MT* = min 1*,* 1*/M,*2*σ* min(1 *− µhigh,*(1 *− µ*0)*ω−*2 *up* )

*.*

*M* + *L*

Now increase the constant *M >* 0 in part 1 of Assumption 3.3.1 if needed so that (3.36) *Hk ≤ M* for all *k*.

We prove (3.35) by showing that if (3.34) holds and (3.35) does not hold for a trial step *st*, then the trust region radius will be expanded and the step corresponding to the larger radius will be acceptable. Let *st* be a trial step such that *st < ∇f*(*xk*) and

*st* = ∆*t <*2*σ* min(1 *− µhigh,*(1 *− µ*0)*ω−*2 *up* )

*M* + *L* (3.37) *∇f*(*xk*)*.* We use the Lipschitz continuity of *∇f* and (3.36) to obtain

1

*aredt* = *−∇f*(*xk*)*T st −* 0

(*∇f*(*xk* + *tst*) *− ∇f*(*xk*))*T st dt* 1

= *predt* + *sTt Hkst/*2 *−* 0

(*∇f*(*xk* + *tst*) *− ∇f*(*xk*))*T st dt*

*≥ predt −* (*M* + *L*)*st*2*/*2*.*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

54 ITERATIVE METHODS FOR OPTIMIZATION

Therefore, using (3.30) from Assumption 3.3.1, we have

*predt≥* 1 *−* (*M* + *L*)*st*2

*aredt*

(3.38)

2*predt*

*≥* 1 *−* (*M* + *L*)*st*2

2*σ∇f*(*xk*) min(*∇f*(*xk*)*, st*)*.*

Now since *st < ∇f*(*xk*) by (3.34) we have

min(*∇f*(*xk*)*, st*) = *st*

and hence*aredk*

*predk≥* 1 *−* (*M* + *L*)*st*

2*∇f*(*xk*)*σ* (3.39) *> µhigh*

by (3.37). Hence, an expansion step will be taken by replacing ∆*t* by ∆+*t* = *ωup*∆*t* and *st* by *s*+*t* , the minimum of the quadratic model in the new trust region.

Now (3.38) still holds and, after the expansion,

*s*+*t  ≤ ωupst < ωup∇f*(*xk*)*.*

So

min(*∇f*(*xk*)*, s*+*t* ) *> s*+*t /ωup.*

Hence,

*pred*+*t≥* 1 *−* (*M* + *L*)*s*+*t* 2

*ared*+*t*

2*σ∇f*(*xk*) min(*∇f*(*xk*)*, s*+*t* )

2*∇f*(*xk*)*σ ≥* 1 *−* (*M* + *L*)*ω*2*upst*

*≥* 1 *−* (*M* + *L*)*ωups*+*t*

2*∇f*(*xk*)*σ ≥ µ*0

by (3.37). Hence, the expansion will produce an acceptable step. This means that if the final accepted step satisfies (3.34), it must also satisfy (3.35). This completes the proof.

**3.3.3 A Unidirectional Trust Region Algorithm**

The most direct way to compute a trial point that satisfies Assumption 3.3.1 is to mimic the line search and simply minimize the quadratic model in the steepest descent direction subject to the trust region bound constraints.

In this algorithm, given a current point *xc* and trust region radius ∆*c*, our trial point is the minimizer of

*ψc*(*λ*) = *mc*(*xc − λ∇f*(*xc*))

subject to the constraint that

*x*(*λ*) = *xc − λ∇f*(*xc*) *∈ T* (∆*c*)*.*

Clearly the solution is *x*(*λ*ˆ) where

(3.40)

*λ*ˆ =



∆*c*

*∇f*(*xc*) if *∇f*(*xc*)*T Hc∇f*(*xc*) *≤* 0 *,* 

min

*∇f*(*xc*)2

*∇f*(*xc*)*~~T~~ Hc∇f*(*xc*) *,* ∆*c ∇f*(*xc*)

if *∇f*(*xc*)*T Hc∇f*(*xc*) *>* 0.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 55

*x*(*λ*ˆ), the minimizer of the quadratic model in the steepest descent direction, subject to the trust region bounds, is called the *Cauchy point*. We will denote the Cauchy point by *xCP*

*c* .1

Then with *xCP* as trial point, one can use Theorem 3.3.1 to derive a global convergence theorem for the unidirectional trust region.

Theorem 3.3.2. *Let ∇f be Lipschitz continuous with Lipschitz constant L. Let {xk} be generated by Algorithm* trgen *with xt* = *xCP and* (3.40)*. Assume that the matrices {Hk} are bounded. Then either f*(*xk*) *is unbounded from below, ∇f*(*xk*)=0 *for some finite k, or*

*k→∞ ∇f*(*xk*)=0*.*

lim

*Proof*. We show that *xt* satisfies part 2 of Assumption 3.3.1. If *st* = ∆*c* then the assertion holds trivially. If *st <* ∆*c* then, by definition of *xCP*

*c* ,

*st* = *− ∇f*(*xc*)2*∇f*(*xc*)

*∇f*(*xc*)*T Hc∇f*(*xc*)*.*

Hence, if *Hc ≤ M*,

*st ≥ ∇f*(*xc*)*/M*

as asserted.

We leave the proof that *xt* satisfies part 1 for the reader (exercise 3.5.8).

The assumptions we used are stronger that those in, for example, [104] and [223], where lim inf *∇f*(*xk*) = 0

rather than *∇f*(*xk*) *→* 0 is proved.

**3.3.4 The Exact Solution of the Trust Region Problem**

The theory of constrained optimization [117], [104] leads to a characterization of the solutions of the trust region problem. In this section we derive that characterization via an elementary argument (see also [84], [242], and [109]). This book focuses on approximate solutions, but the reader should be aware that the exact solution can be computed accurately [192], [243].

Theorem 3.3.3. *Let g ∈ RN and let A be a symmetric N × N matrix. Let*

*m*(*s*) = *gT s* + *sT As/*2*.*

*A vector s is a solution to*

(3.41) *m*(*s*)

min

*s≤*∆

*if and only if there is ν ≥* 0 *such that*

(*A* + *νI*)*s* = *−g*

*and either ν* = 0 *or s* = ∆*.*

*Proof*. If *s <* ∆ then *∇m*(*s*) = *g* + *As* = 0, and the conclusion follows with *ν* = 0. To consider the case where *s* = ∆, let *λ*1 *≤ λ*2 *≤··· λN* be the eigenvalues of *A*.

1In some of the literature, [84], for example, *Hc* is assumed to be positive definite and the Cauchy point is taken to be the global minimizer of the quadratic model.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

56 ITERATIVE METHODS FOR OPTIMIZATION

Clearly, for any *ν*,

*m*(*s*) = *gT s* + *sT As/*2

= *gT s* + *sT* (*A* + *νI*)*s/*2 *− ν*∆2*/*2*.*

Consider the function, defined for *ν>ν*0 = max(0*, −λ*1)*,*

*s*(*ν*) = *−*(*A* + *νI*)*−*1*g.*

Since

lim*ν→∞ s*(*ν*)=0

and *s*(*ν*) is a continuous decreasing function of *ν ∈* (*ν*0*,∞*) we see that if

lim*ν→ν*0*s*(*ν*) *>* ∆

then there is a unique *ν* such that *s*(*ν*) = ∆. Since *ν ≥ ν*0, *A* + *νI* is positive semidefinite; therefore, *s*(*ν*) is a global minimizer of

*gT s* + *sT* (*A* + *νI*)*s/*2*.*

Hence, we must have

*m*(*s*) *≥ m*(*s*(*ν*))

for all *s* such that *s* = ∆. Hence, *s*(*ν*) is a solution of (3.41).

The remaining case is

lim*ν→ν*0*s*(*ν*) *≤* ∆*.*

This implies that *g* is orthogonal to the nontrivial space *S*0 of eigenfunctions corresponding to *−ν*0 (for otherwise the limit would be infinite). If we let *s* = *s*1 + *s*2, where *s*2 is the projection of *s* onto *S*0, we have

*m*(*s*) = *sT*1 *g* + *sT*1 (*A* + *ν*0)*s*1*/*2 + *sT*2 (*A* + *ν*0)*s*2*/*2 *− ν*0∆2*/*2

= *sT*1 *g* + *sT*1 (*A* + *λ*0)*s*1*/*2 *− ν*0∆2*/*2*.*

Hence, *m*(*s*)is minimized by setting *s*1 equal to the minimum norm solution of(*A*+*ν*0)*x* = *−g* (which exists by orthogonality of *g* to *S*0) and letting *s*2 be any element of *S*0 such that

*s*22 = ∆2 *− s*12*.*

This completes the proof.

**3.3.5 The Levenberg–Marquardt Parameter**

The solution of the trust region problem presented in *§*3.3.4 suggests that, rather than controlling ∆, one could set

*st* = *−*(*νI* + *Hc*)*−*1*g,*

adjust *ν* in response to *ared/pred* instead of ∆, and still maintain global convergence. A natural application of this idea is control of the Levenberg–Marquardt parameter. This results in a much simpler algorithm than Levenberg–Marquardt–Armijo in that the stepsize control can be eliminated. We need only vary the Levenberg–Marquardt parameter as the iteration progresses.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 57

We present the algorithm from [190] to illustrate this point. For an inexact formulation, see [276].

The Levenberg–Marquardt quadratic model of least squares objective

*f*(*x*) = 12*M*

*i*=1

with parameter *νc* at the point *xc* is

*ri*(*x*)22 = 12*R*(*x*)*T R*(*x*)

(3.42)

*mc*(*x*) = *f*(*xc*)+(*x − xc*)*T R*(*xc*)*T R*(*xc*) +12 (*x − xc*)*T* (*R*(*xc*)*T R*(*xc*) + *νcI*)(*x − xc*)*.*

The minimizer of the quadratic model is the trial point

*xt* = *xc −* (*R*(*xc*)*T R*(*xc*) + *νcI*)*−*1*R*(*xc*)*T* (3.43) *R*(*xc*)*,* the step is *s* = *xt − xc*, and the predicted reduction is

*pred* = *m*(*xc*) *− m*(*xt*) = *−sT R*(*xc*)*T R*(*xc*) *−* 12 *sT* (*R*(*xc*)*T R*(*xc*) + *νcI*)*s* = *−sT R*(*xc*)*T R*(*xc*) + 12 *sT R*(*xc*)*T R*(*xc*) = *−*12 *sT ∇f*(*xc*)*.*

The algorithm we present below follows the trust region paradigm and decides on accepting the trial point and on adjustments in the Levenberg–Marquardt parameter by examinaing the ratio*ared*

*pred* = *f*(*xc*) *− f*(*xt*)

*m*(*xc*) *− m*(*xt*)

= *−*2*f*(*xc*) *− f*(*xt*)

*sT ∇f*(*xc*) *.*

In addition to the trust region parameters 0 *< ωdown <* 1 *< ωup* and *µ*0 *≤ µlow < µhigh* we require a default value *ν*0 of the Levenberg–Marquardt parameter.

The algorithm for testing the trial point differs from Algorithm trtest in that we decrease (increase) *ν* rather that increasing (decreasing) a trust region radius if *ared/pred* is large (small). We also attempt to set the Levenberg–Marquardt parameter to zero when possible in order to recover the Gauss–Newton iteration’s fast convergence for small residual problems.

Algorithm 3.3.4. trtestlm(*xc, xt, x*+*,f,ν*)

1. *z* = *xc*

2. *Do while z* = *xc*

(a) *ared* = *f*(*xc*) *− f*(*xt*), *st* = *xt − xc*, *pred* = *−∇f*(*xc*)*T st/*2*.*

(b) *If ared/pred < µ*0 *then set z* = *xc, ν* = max(*ωupν, ν*0)*, and recompute the trial point with the new value of ν.*

(c) *If µ*0 *≤ ared/pred < µlow, then set z* = *xt and ν* = max(*ωupν, ν*0)*.*

(d) *If µlow ≤ ared/pred, then set z* = *xt.*

*If µhigh < ared/pred, then set ν* = *ωdownν.*

*If ν<ν*0*, then set ν* = 0*.*

3. *x*+ = *z*.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

58 ITERATIVE METHODS FOR OPTIMIZATION

The Levenberg–Marquardt version ofAlgorithmtrgen is simple to describe and implement. Algorithm 3.3.5. levmar(*x, R, kmax*)

1. *Set ν* = *ν*0*.*

2. *For k* = 1*, . . . , kmax*

(a) *Let xc* = *x.*

(b) *Compute R, f, R, and ∇f; test for termination.*

(c) *Compute xt using* (3.43)*.*

(d) *Call* trtestlm(*xc, xt, x, f, ν*)

We state a convergence result [190], [276] without proof.

Theorem 3.3.4. *Let R be Lipschitz continuously differentiable. Let {xk} and {νk} be the sequence of iterates and Levenberg–Marquardt parameters generated by Algorithm* levmar *with kmax* = *∞. Assume that {νk} is bounded from above. Then either R*(*xk*)*T R*(*xk*)=0 *for some finite k or*

*k→∞ R*(*xk*)*T R*(*xk*)=0*.*

lim

*Moreover, if x∗ is a limit point of {xk} for which R*(*x∗*)=0 *and R*(*x∗*) *has full rank, then xk → x∗ q-quadratically and νk* = 0 *for k sufficiently large.*

**3.3.6 Superlinear Convergence: The Dogleg**

The convergence of the unidirectional trust region iteration can be as slow as that for steepest descent. To improve the convergence speed in the terminal phase we must allow for approx imations to the Newton direction. The power of trust region methods is the ease with which the transition from steepest descent, with its good global properties, to Newton’s method can be managed.

We define the *Newton point* at *xc* as

*xNc* = *xc − H−*1 *c ∇f*(*xc*)*.*

If *Hc* is spd, the Newton point is the global minimizer of the local quadratic model. On the other hand, if *Hc* has directions of negative curvature the local quadratic model will not have a finite minimizer, but the Newton point is still useful. Note that if *H* = *I* the Newton point and the Cauchy point are the same if the Newton point is inside the trust region.

We will restrict our attention to a special class of algorithms that approximate the solution of the trust region problem by minimizing *mc* along a piecewise linear path *S⊂T* (∆). These paths are sometimes called *doglegs* because of the shapes of the early examples [84], [80], [218], [217], [220]. In the case where *∇*2*f*(*x*) is spd, one may think of the dogleg path as a piecewise linear approximation to the path with parametric representation

*{x −* (*λI* + *∇*2*f*(*x*))*−*1*∇f*(*x*)*|* 0 *≤ λ}.*

This is the path on which the exact solution of the trust region problem lies.

The next step up from the unidirectional path, the *classical dogleg* path [220], has as many as three nodes, *xc*, *xCP ∗ c* , and *xNc* . Here *xCP ∗ c* is the global minimizer of the quadratic model in the steepest descent direction, which will exist if and only if *∇f*(*xc*)*T Hc∇f*(*xc*) *>* 0. If *xCP ∗ c* exists and

(*xNc − xCP ∗ c* )*T* (*xCP ∗* (3.44) *c − xc*) *>* 0*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 59

we will let *xNc* be the terminal node. If (3.44) holds, as it always will if *Hc* is spd, then the path can be parameterized by the distance from *xc* and, moreover, *mc* decreases along the path. If (3.44) does not hold, we do not use *xNc* as a node and revert to the unidirectional path in the steepest descent direction.

Note that (3.44) implies

*∇f*(*xc*)*T* (*xN* (3.45) *c − xc*) *<* 0*.*

We can express the conditions for using the three node path rather than the unidirectional path very simply. If *xCP*

*c* is on the boundary of the trust region then we accept *xCP*

*c* as the trial point.

*c* = *xCP ∗ c* is in the interior of the trust region, then we test (3.44) to decide what to do.

If *xCP*

With this in mind our trial point for the classical dogleg algorithm will be

(3.46)

*xD*(∆) =



*xCP* if *xc − xCP*

*c* = ∆

or *xCP ∗* exists and (3.44) fails,

*xN* if *xc − xCP*

*c  < xc − xNc  ≤* ∆



and (3.44) holds,

*yD*(∆) otherwise*.*

Here *yD*(∆) is the unique point between *xCP*

*c* and *xNc* such that *xD − xc* = ∆.

The important properties of dogleg methods are as follows:

*•* No two points on the path have the same distance from *xc*; hence the path may be param eterized as *x*(*s*), where *s* = *x*(*s*) *− xc*.

*• mc*(*x*(*s*)) is a strictly decreasing function of *s*.

This enables us to show that the dogleg approximate solution of the trust region problem sat isfies Assumption 3.3.1 and apply Theorem 3.3.1 to conclude global convergence. Superlinear convergence will follow if *Hk* is a sufficiently good approximation to *∇*2*f*(*xk*).

Lemma 3.3.5. *Let xc, Hc, and* ∆ *be given. Let Hc be nonsingular,*

*sN* = *−H−*1 *c ∇f*(*xc*)*, and xN* = *xc* + *sN .*

*Assume that ∇f*(*xc*)*T Hc∇f*(*xc*) *>* 0 *and let*

*sCP ∗* = *xCP ∗ − xc* = *− ∇f*(*xc*)2

*∇f*(*xc*)*T Hc∇f*(*xc*)*∇f*(*xc*)*.*

*Let S be the piecewise linear path from xc to xCP ∗ to xN . Then if*

(*sN − sCP ∗*)*T sCP ∗* (3.47) *>* 0*,*

*for any δ ≤ sN  there is a unique point x*(*δ*) *on S such that*

*x*(*δ*) *− xc* = *δ.*

*Proof*. Clearly the statement of the result holds on the segment of the path from *x* to *xCP ∗*. To prove the result on the segment from *xCP ∗* to *xN* we must show that

*φ*(*λ*) = 12(1 *− λ*)*sCP ∗* + *λsN* 2

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

60 ITERATIVE METHODS FOR OPTIMIZATION

is strictly monotone increasing for *λ ∈* (0*,* 1).

Since (3.47) implies that

*sN sCP ∗ ≥* (*sN* )*T sCP ∗ > sCP ∗*2

and therefore that *sN  > sCP ∗*, we have

*φ*(*λ*) =(*sN − sCP ∗*)*T* ((1 *− λ*)*sCP ∗* + *λsN* )

= *−*(1 *− λ*)*sCP ∗*2 + (1 *− λ*)(*sN* )*T sCP ∗* + *λsN* 2 *− λ*(*sN* )*T sCP ∗*

*> λ*(*sN* 2 *−* (*sN* )*T sCP ∗*) *≥ λ*(*sN −sCP ∗*)*sN  >* 0*.*

Hence, *φ* is an increasing function and the proof is complete.

The next stage is to show that the local quadratic model decreases on the dogleg path *S*. Lemma 3.3.6. *Let the assumptions of Lemma* 3.3.5 *hold. Then the local quadratic model mc*(*x*) = *f*(*xc*) + *∇f*(*xc*)*T* (*x − xc*) + 12(*x − xc*)*T Hc*(*x − xc*)

*is strictly monotone decreasing on S.*

*Proof*. Since *xCP ∗ c* is the minimum of the local quadratic model in the steepest descent direction, we need only show that *mc* is strictly decreasing on the segment of the path between *xCP ∗ c* and *xN* . Set

*ψ*(*λ*) = *mc*(*xc* + (1 *− λ*)*sCP ∗* + *λsN* )

= *f*(*xc*) + *∇f*(*xc*)*T* ((1 *− λ*)*sCP ∗* + *λsN* )

+12 ((1 *− λ*)*sCP ∗* + *λsN* )*T Hc*((1 *− λ*)*sCP ∗* + *λsN* )*.*

Noting that *HcsN* = *−∇f*(*xc*) and *sCP ∗* = *−λ*ˆ*∇f*(*xc*), we obtain

*ψ*(*λ*) = *f*(*xc*) *− λ*ˆ(1 *− λ*)2*∇f*(*xc*)2

+*λ*(1 *− λ/*2)*∇f*(*xc*)*T sN*

+12 (1 *− λ*)2*λ*ˆ2*∇f*(*xc*)*T Hc∇f*(*xc*)*.*

Therefore,

*ψ*(*λ*)=2*λ*ˆ(1 *− λ*)*∇f*(*xc*)2

+(1 *− λ*)*∇f*(*xc*)*T sN −* (1 *− λ*)*λ*ˆ2*∇f*(*xc*)*T Hc∇f*(*cc*)*.*

Since

we have, using (3.44),

*λ*ˆ*∇f*(*xc*)*T Hc∇f*(*cc*) = *∇f*(*xc*)2

*ψ*(*λ*) = (1 *− λ*)(*λ*ˆ*∇f*(*xc*)2 *− ∇f*(*xc*)*T H−*1 *c ∇f*(*xc*)) = (1 *− λ*)*∇f*(*xc*)*T* (*λ*ˆ*∇f*(*xc*) *− H−*1 *c ∇f*(*xc*))

= 1 *− λ*

*λ*ˆ (*xc − xCP ∗ c* )*T* (*xNc − xc*) *<* 0*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 61

completing the proof.

At this point we have shown that the approximate trust region problem has a unique solution. To prove global convergence we need only verify that the approximate solution of the trust region problem *xD* satisfies Assumption 3.3.1.

Theorem 3.3.7. *Let ∇f be Lipschitz continuous with Lipschitz constant L. Let {xk} be generated by Algorithm* trgen *and the solutions for the trust region problem be given by* (3.46)*. Assume that the matrices {Hk} are bounded. Then either f*(*xk*) *is unbounded from below, ∇f*(*xk*)=0 *for some finite k, or*

*k→∞* (3.48) *∇f*(*xk*)=0*.*

lim

*Proof*. We need to check that the solutions of the trust region problem satisfy Assump tion 3.3.1. Part 2 of the assumption follows from the definition, (3.46), of *xD* and the bounded ness of the approximate Hessians. Let

*Hk ≤ M*

for all *k*. If *sk <* ∆, then (3.46) implies that (3.44) must hold and so *xt* = *xNk* is the Newton point. Hence,

*sk* = *xk − xNk* = *H−*1

*k ∇f*(*xk*) *≥ ∇f*(*xk*)*/M,*

which proves part 2.

Verification of part 1 will complete the proof. There are several cases to consider depending on how *xD* is computed.

If *xD* = *xCP* then either *sCP* = ∆*c* or (3.44) fails. We first consider the case where *∇f*(*xc*)*T Hc∇f*(*xc*) *≤* 0. In that case *sCP* = ∆*c* and *λ*ˆ = ∆*c/∇f*(*xc*). Therefore,

*pred* = *λ*ˆ*∇f*(*xc*)2 *− λ*ˆ22 *∇f*(*xc*)*T Hc∇f*(*xc*)

= ∆*c∇f*(*xc*) *−* ∆2*c∇f*(*xc*)*T Hc∇f*(*xc*)

2*∇f*(*xc*)2

*≥* ∆*c∇f*(*xc*) = *s∇f*(*xc*)*.*

Hence (3.30) holds with *σ* = 1.

Now assume that *∇f*(*xc*)*T Hc∇f*(*xc*) *>* 0 and *sCP* = ∆*c*. In this case

*∇f*(*xc*)2

*∇f*(*xc*)*T Hc∇f*(*xc*) *≥*∆*c*

*∇f*(*xc*)

and so

*pred* = *λ*ˆ*∇f*(*xc*)2 *− λ*ˆ22 *∇f*(*xc*)*T Hc∇f*(*xc*) = ∆*c∇f*(*xc*) *−* ∆2*c∇f*(*xc*)*T Hc∇f*(*xc*) 2*∇f*(*xc*)2

*≥* ∆*c∇f*(*xc*)*/*2*,*

which is (3.30) with *σ* = 1*/*2.

If (3.44) fails, *∇f*(*xc*)*T Hc∇f*(*xc*) *>* 0, and *sCP  <* ∆*c*, then

*λ*ˆ = *∇f*(*xc*)2

*∇f*(*xc*)*T Hc∇f*(*xc*)*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. 62 ITERATIVE METHODS FOR OPTIMIZATION

and

*pred* = *λ*ˆ*∇f*(*xc*)2 *− λ*ˆ22 *∇f*(*xc*)*T Hc∇f*(*xc*) 2*∇f*(*xc*)*T Hc∇f*(*xc*) = *λ*ˆ*∇f*(*xc*)2

= *∇f*(*xc*)4

2

= *s∇f*(*xc*)

2 *,*

which is (3.30) with *σ* = 1*/*2.

The final case is if (3.44) holds and *xD* = *xCP* . In that case the predicted reduction is more than *Cauchy decrease*, i.e., the decrease obtained by taking the Cauchy point, and hence

*pred ≥ ∇f*(*xc*)4

2*∇f*(*xc*)*T Hc∇f*(*xc*)

*≥ ∇f*(*xc*)2

2*M ,*

which is (3.30) with *σ* = 1*/*(2*M*). This completes the proof.

The last part of the proof of this theorem is very important, asserting that any solution of the trust region problem for which *pred* is at least a fixed fraction of Cauchy decrease will give global convergence. We refer the reader to [232] and [104] for a more general and detailed treatment using this point of view.

Corollary 3.3.8. *Any algorithm for solving the trust region problem that satisfies for some τ >* 0

*pred ≥ τ* (*mc*(*xc*) *− mc*(*xCP*

*c* ))

*satisfies* (3.30) *for σ* = *τ /*2*.*

The trust region CG algorithm we present in *§*3.3.7 can be analyzed with this corollary. If *Hk* = *∇*2*f*(*xk*) or a sufficiently good approximation, then the classical dogleg will become Newton’s method (or a superlinearly convergent method) as the iterations approach a minimizer that satisfies the standard assumptions. Hence, the algorithm makes a smooth and automatic transition into the superlinearly convergent stage.

Theorem 3.3.9. *Let ∇f be Lipschitz continuous with Lipschitz constant L. Let {xk} be generated by Algorithm* trgen *and the solutions for the trust region problem are given by* (3.46)*. Assume that Hk* = *∇*2*f*(*xk*) *and that the matrices {Hk} are bounded. Let f be bounded from below. Let x∗ be a minimizer of f at which the standard assumptions hold. Then if x∗ is a limit point of xk, then xk → x∗ and the convergence is locally q-quadratic.*

*Proof*. Since *x∗* is a limit point of *{xk}*, there is, for any *ρ >* 0, a *k* sufficiently large so that *ek < ρ, Hk ≤* 2*∇*2*f*(*x∗*)*, H−*1

*k  ≤* 2(*∇*2*f*(*x∗*))*−*1*,*

and *xk* is near enough for the assumptions of Theorem 2.3.2 to hold. If *Hk* is spd, so is *H−*1

*k*

and for such *k*, (3.44) holds. Hence, the dogleg path has the nodes *xk*, *xCP*

if *ρ* is sufficiently small, then

*H−*1

*k ∇f*(*xk*) *≤* 2*ek ≤* 2*ρ.*

*k* , and *xNk* . Moreover,

We complete the proof by showing that if *ρ* is sufficiently small, the trust region radius will be expanded if necessary until the Newton step is in the trust region. Once we do this, the proof is complete as then the local quadratic convergence of Newton’s method will take over.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 63

Now

*predk ≥ sk∇f*(*xk*)*/*2

by the proof of Theorem 3.3.7. Using *Hk* = *∇*2*f*(*xk*) we have

1

*aredk* = *−∇f*(*xk*)*T stk −* 0

(*∇f*(*xk* + *tstk*) *− ∇f*(*xk*))*T stk dt* 1

= *predk* + *sTtk∇*2*f*(*xk*)*stk/*2 *−* 0

= *predk* + *O*(*sk∇f*(*xk*)*ρ*)

(*∇f*(*xk* + *tstk*) *− ∇f*(*xk*))*T stk dt*

and therefore *ared/pred* = 1*−O*(*ρ*). Hence, for *ρ* sufficiently small, the trust region radius will be increased, if necessary, until the Newton point is inside the trust region and then a Newton step will be taken. This completes the proof.

The classical dogleg algorithm is implemented in Algorithm ntrust, which uses the trust radius adjustment scheme from Algorithm trtest. It is to be understood that trtest is implemented so that *xt* is given by (3.46) and hence trtest only samples points on the piecewise linear search path determined by the Cauchy point, the Newton point, and (3.44).

Algorithm 3.3.6. ntrust(*x, f, τ* )

1. *Compute f*(*x*) *and ∇f*(*x*)

2. *τ* = *τa* + *τr∇f*(*x*)

3. *Do while ∇f*(*x*) *> τ*

(a) *Compute and factor ∇*2*f*(*x*)

(b) *Compute the Cauchy and Newton points and test* (3.44)

(c) *Call* trtest(*x, xt, x*+*,f,* ∆)

(d) *Compute f*(*x*+) *and ∇f*(*x*+)*; x* = *x*+

We implement Algorithm ntrust in the collection of MATLAB codes.

**3.3.7 A Trust Region Method for Newton–CG**

In this section we present a brief account of an algorithm from [247] (see also [257]) that combines the trust region paradigm of *§*3.3.6 with the inexact Newton ideas of *§*2.5.2. We follow *§*2.5.2 and denote the preconditioner by *M* and let *C* = *M−*1. We solve the scaled trust region problem

*dC ≤*∆ *φ*(*d*)*,*

min

where the quadratic model is still

*φ*(*d*) = *∇f*(*x*)*T d* +12*dT ∇*2*f*(*x*)*d.*

Here the *C*-norm is

*dC* = (*dT Cd*)1*/*2*.*

The algorithmic description of the trust region problem solver from the TR–CG method given below is from [162]. In [247] the algorithm is expressed in terms of *C* rather than *M*.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

64 ITERATIVE METHODS FOR OPTIMIZATION

This is a dogleg method in that the approximate solution of the trust region problem lies on a piecewise linear path with the CG iterations as nodes. As long as CG is performing properly (i.e., *pT w >* 0) nodes are added to the path until the path intersects the trust region boundary. If a direction of indefiniteness is found (*pT w ≤* 0), then that direction is followed to the boundary. In this way a negative curvature direction, if found in the course of the CG iteration, can be exploited.

The inputs to Algorithm trcg are the current point *x*, the objective *f*, the forcing term *η*, and the current trust region radius ∆. The output is the approximate solution of the trust region problem *d*. This algorithm is not the whole story, as once the trust region problem is solved approximately, one must use *f*(*xc* + *d*) to compute *ared* and then make a decision on how the trust region radius should be changed. Our formulation differs from that in [247] in that the termination criterion measures relative residuals in the *l*2-norm rather than in the *C*-norm. This change in the norm has no effect on the analysis in [247], and, therefore, we can apply the results in *§*2.5 directly to draw conclusions about local convergence.

Algorithm 3.3.7. trcg(*d, x, f, M, η,* ∆*, kmax*)

1. *r* = *−∇f*(*x*), *ρ*0 = *r*22, *k* = 1, *d* = 0

2. *Do While √~~ρ~~k−*1 *> η∇f*(*x*)2 *and k < kmax*

(a) *z* = *Mr*

(b) *τk−*1 = *zT r*

(c) *if k* = 1 *then β* = 0 *and p* = *z*

*else*

*β* = *τk−*1*/τk−*2*, p* = *z* + *βp*

(d) *w* = *∇*2*f*(*x*)*p*

*If pT w ≤* 0 *then*

*Find τ such that d* + *τ pC* = ∆

*d* = *d* + *τ p; return*

(e) *α* = *τk−*1*/pT w*

(f) *r* = *r − αw*

(g) *ρk* = *rT r*

(h) ˆ*d* = *d* + *αp*

(i) *If* ˆ*dC >* ∆ *then*

*Find τ such that d* + *τ pC* = ∆

*d* = *d* + *τ p; return*

(j) *d* = ˆ*d*; *k* = *k* + 1

Algorithm trcg does what we would expect a dogleg algorithm to do in that the piecewise linear path determined by the iteration moves monotonically away from *x* (in the *·C* -norm!) and the quadratic model decreases on that path [247]. Algorithm trcg will, therefore, compute the same Newton step as Algorithm fdpcg. One might think that it may be difficult to compute the *C*-norm if one has, for example, a way to compute the action of *M* on a vector that does not require computation of the matrix *C*. However, at the cost of storing two additional vectors we can update *Cp* and *Cd* as the iteration progresses. So, when *p* is updated to *z* + *βp* then *Cp* = *r* + *βCp* can be updated at the same time without computing the product of *C* with *p*. Then *pC* = *pT Cp*. Similarly *d* = *d* + *τ p* implies that *Cd* = *Cd* + *τCp*.

Algorithm cgtrust combines the solution of the trust region problem from trcg, the trust region radius adjustment scheme from trtest, and (indirectly) the locally convergent algorithm newtcg. The result fits nicely into our paradigm algorithm trgen.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 65

Algorithm 3.3.8. cgtrust(*x, f, τ* )

1. *Initialize* ∆*, M, η, kmax.*

2. *Do forever*

(a) *Let xc* = *x. Compute ∇f*(*xc*)*.*

(b) *Call* trcg(*d, x, f, M, η,* ∆*, kmax*) *to solve the trust region subproblem.*

*Set xt* = *x* + *d.*

(c) *Call* trtest(*xc, xt, x, f,* ∆)*,*

*solving the trust region subproblem with Algorithm* trcg*.*

(d) *Update η.*

Theorem 3.3.10 combines several results from [247].

Theorem 3.3.10. *Let f be twice Lipschitz continuously differentiable. Let M be a given positive definite matrix and let {ηn} satisfy* 0 *< ηn <* 1 *for all n. Let {xn} be the sequence generated by Algorithm* cgtrust *and assume that {∇*2*f*(*xn*)*} is bounded. Then*

lim*n→∞* (3.49) *∇f*(*xn*)=0*.*

*Moreover, if x∗ is a local minimizer for which the standard assumptions hold and xn → x∗, then*

*• if ηn →* 0 *the convergence is q-superlinear, and*

*• if ηn ≤ Kη∇f*(*xn*)*p for some Kη >* 0 *the convergence is q-superlinear with q-order* 1 + *p.*

*Finally, there are δ and* ∆ *such that if x*0 *− x∗ ≤ δ and* ∆0 *≤* ∆ *then xn → x∗.*

One can, as we do in the MATLAB code cgtrust, replace the Hessian–vector product with a difference Hessian. The accuracy of the difference Hessian and the loss of symmetry present the potential problem that was mentioned in *§*2.5. Another, very different, approach is to approximate the exact solution of the trust region subproblem with an iterative method [243].

**3.4 Examples**

The results we report here used the MATLAB implementations of steepest descent, steep.m, damped Gauss–Newton, gaussn.m, the dogleg trust region algorithm for Newton’s method, ntrust.m, and the PCG–dogleg algorithms, cgtrust.m, from the software collection.

Our MATLAB implementation of Algorithm steep guards against extremely poor scaling and very long steps by setting *λ* to

(3.50) *λ*0 = min(1*,* 100*/*(1 + *∇f*(*x*)))

at the beginning of the line search. We invite the reader in Exercise 3.5.3 to attempt the control example with *λ*0 = 1.

We not only present plots, which are an efficient way to understand convergence rates, but we also report counts of function, gradient, and Hessian evaluations and the results of the MATLAB flops command.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. 66 ITERATIVE METHODS FOR OPTIMIZATION

Gradient Norm

105 Dogleg 100

105

Function Value

105 Dogleg 100

105

1010

1010

0 10 20 30 Iterations

1015

0 10 20 30 Iterations

Gradient Norm

104 Steepest Descent 102

100

102

Function Value

105 Steepest Descent 100

105

104

0 20 40 60 Iterations

1010

0 20 40 60 Iterations

Figure 3.1: *Steepest Descent and Newton–Dogleg for Parameter ID Problem*

Gradient Norm

102 Damped Gauss–Newton 100

102

104

Function Value

105 Damped Gauss–Newton 100

105

1010

106

0 2 4 6 Iterations

1015

0 2 4 6 Iterations

Gradient Norm

102 Levenberg–Marquardt 100

102

104

Function Value

105 Levenberg–Marquardt 100

105

1010

106

0 5 10 15 Iterations

1015

0 5 10 15 Iterations

Figure 3.2: *Gauss–Newton and Levenberg–Marquardt for Parameter ID Problem* Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. GLOBAL CONVERGENCE 67

Gradient Norm

105 Dogleg–CG 100

105

Function Value

107 Dogleg–CG 106

105

104

1010

0 2 4 6 Iterations

103

0 2 4 6 Iterations

Gradient Norm

105 Steepest Descent 100

105

Function Value

107 Steepest Descent 106

105

104

1010

0 20 40 60 Iterations

103

0 20 40 60 Iterations

Figure 3.3: *Steepest Descent and Dogleg–CG for Discrete Control Problem*

**3.4.1 Parameter Identification**

We consider the problem from*§*2.6.1 except we use the initial data *x*0 = (5*,* 5)*T* . Both the Gauss– Newton and Newton methods will fail to converge with this initial data without globalization (see Exercise 3.5.14). Newton’s method has particular trouble with this problem because the Newton direction is not a descent direction in the early phases of the iteration. The termination criterion and difference increment for the finite difference Hessian was the same as for the computation in *§*2.6.1.

In Figure 3.1 we compare the performance of the Newton dogleg algorithm with the steepest descent algorithm. Our implementation of the classical dogleg in ntrust uses the standard values

(3.51) *ωdown* = *.*5*, ωup* = 2*, µ*0 = *µlow* = *.*25*,* and *µhigh* = *.*75*.*

The plots clearly show the locally superlinear convergence of Newton’s method and the linear convergence of steepest descent. However, the graphs do not completely show the difference in computational costs. In terms of gradient evaluations, steepest descent was marginally better than the Newton dogleg algorithm, requiring 50 gradients as opposed to 55 (which includes those needed for the 18 difference Hessian evaluations) for the Newton dogleg algorithm. However, the steepest descent algorithm required 224 function evaluations, while the Newton dogleg needed only 79. As a result, the Newton dogleg code was much more efficient, needing roughly 5 million floating point operations instead of the 10 million needed by the steepest descent code.

In Figure 3.2 we plot the performance of the damped Gauss–Newton and Levenberg– Marquardt algorithms. These exploit the least squares structure of the problem and are locally superlinearly convergent because this is a zero residual problem. They also show that algo rithms that effectively exploit the structure of the least squares problem are much more efficient. Gauss–Newton required 6 gradient evaluations, 14 function evaluations, and 750 thousand float ing point operations, and Levenberg–Marquardt required 12 gradients, 23 functions, and 1.3

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

68 ITERATIVE METHODS FOR OPTIMIZATION million floating point operations.

**3.4.2 Discrete Control Problem**

We consider the discrete control problem from *§*1.6.1 with *N* = 400, *T* = 1, *y*0 = 0,

*L*(*y, u, t*)=(*y −* 3)2 + *.*5 *∗ u*2*,* and *φ*(*y, u, t*) = *uy* + *t*2*.*

We chose the poor initial iterate

*u*0(*t*) = 5 + 300 sin(20*πt*)*.*

This problem can be solved very efficiently with Algorithm cgtrust. In our implementa tion we use the same parameters from (3.51). In Figure 3.3 we compare the dogleg–CG iteration with steepest descent. We terminated both iterations when *∇f <* 10*−*8. For the dogleg–CG code we used *η* = *.*01 throughout the entire iteration and an initial trust region radius of *u*0. The steepest descent computation required 48 gradient evaluations, 95 function evaluations, and roughly 1 million floating point operations, and dogleg–CG needed 17 gradient evaluations, 21 function evaluations, and roughly 530 thousand floating point operations. Note that the steepest descent algorithm performed very well in the terminal phase of the iteration. The reason for this is that, in this example, the Hessian is near the identity.

**3.5 Exercises on Global Convergence**

3.5.1. Let *F* be a nonlinear function from *RN → RN* . Let

*f*(*x*) = *F*(*x*)2*/*2*.*

What is *∇f*? When is the Newton step for the nonlinear equation *F*(*x*)=0,

*d* = *−F*(*x*)*−*1*F*(*x*)*,*

a descent direction for *f* at *x*?

3.5.2. Prove Lemma 3.2.1.

3.5.3. Implement Algorithm steep without the scaling fixup in (3.50). Apply this crippled algorithm to the control problem example from *§*3.4.2. What happens and why?

3.5.4. Show that if *f* is a convex quadratic then *f* is bounded from below.

3.5.5. Verify (3.40).

3.5.6. Show that the Levenberg–Marquardt steps computed by (3.20) and (3.21) are the same. 3.5.7. Prove Theorem 3.2.7.

3.5.8. Complete the proof of Theorem 3.3.2.

3.5.9. Prove Theorem 3.3.4.

3.5.10. Look at the trust region algorithm for nonlinear equations from [218] or [84]. What are the costs of that algorithm that are not present in a line search? When might this trust region approach have advantages for solving nonlinear equations? Could it be implemented inexactly?

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

GLOBAL CONVERGENCE 69

3.5.11. The double dogleg method [80], [84] puts a new node on the dogleg path in the Newton direction, thereby trying more aggressively for superlinear convergence. Implement this method, perhaps by modifying the MATLAB code ntrust.m, and compare the results with the examples in *§*3.4. Prove convergence results like Theorems 3.3.7 and 3.3.9 for this method.

3.5.12. In [51] a trust region algorithm was proposed that permitted inaccurate gradient computa tions, with the relative accuracy being tightened as the iteration progresses. Look at [51] and try to design a similar algorithm based on the line search paradigm. What problems do you encounter? How do you solve them?

3.5.13. Suppose one modifies Algorithm trtest by not resolving the trust region problem if the trial point is rejected, but instead performing a line search from *xt*, and setting ∆ = *x*+ *− xc*, where *x*+ is the accepted point from the line search. Discuss the merits of this modification and any potential problems. See [209] for the development of this idea.

3.5.14. Write programs for optimization that take full Gauss–Newton or Newton steps (you can cripple the MATLAB codes gaussn.m and ntrust.m for this). Apply these codes to the parameter identification problem from *§*3.4.1. What happens?

3.5.15. Write a nonlinear CG code and apply it to the problems in *§*3.4. Try at least two ways to manage the line search. How important are the (strong) Wolfe conditions?

3.5.16. Discuss the impact of using a difference Hessian in Algorithm trcg. How will the global convergence of Algorithm cgtrust be affected? How about the local convergence? Consider the accuracy in the evaluation of *∇f* in your results.

3.5.17. Without looking at [247] describe in general terms how the proof of Theorem 3.3.1 should be modified to prove Theorem 3.3.10. Then examine the proof in [247] to see if you left anything out.

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed. Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

**Chapter 4**

**The BFGS Method**

*Quasi-Newton* methods update an approximation of *∇*2*f*(*x∗*) as the iteration progresses. In general the transition from current approximations *xc* and *Hc* of *x∗* and *∇*2*f*(*x∗*) to new ap proximations *x*+ and *H*+ is given (using a line search paradigm) by the following steps:

1. Compute a search direction *d* = *−H−*1 *c ∇f*(*xc*).

2. Find *x*+ = *xc* + *λd* using a line search to insure sufficient decrease.

3. Use *xc*, *x*+, and *Hc* to *update Hc* and obtain *H*+.

The way in which *H*+ is computed determines the method.

The BFGS (Broyden, Fletcher, Goldfarb, Shanno) [36], [103], [124], [237] method, which is the focus of this chapter, and the other methods we will mention in *§*4.3 are also called *secant methods* because they satisfy the *secant equation*

(4.1) *H*+*s* = *y.*

In (4.1)

*s* = *x*+ *− xc* and *y* = *∇f*(*x*+) *− ∇f*(*xc*)*.*

If *N* = 1, all secant methods reduce to the classical *secant method* for the single nonlinear equation *f*(*x*)=0, i.e.,

*x*+ = *xc − f*(*xc*)(*xc − x−*)

*f*(*xc*) *− f*(*x−*) (4.2) *,*

where *x−* is the iterate previous to *xc*.

The standard quasi-Newton update for nonlinear equations is Broyden’s [34] method, a rank-one update,

*H*+ = *Hc* + (*y − Hcs*)*sT*

(4.3) *.*

*sT s*

Broyden’s method does not preserve the structural properties needed for line search methods in optimization, namely, symmetry and positive definiteness, and could, in fact, encourage con vergence to a local maximum. For that reason quasi-Newton methods in optimization are more complex than those used for nonlinear equations. The methods of analysis and implementation are more complex as well.

In this chapter we will concentrate on the BFGS method [36], [103], [124], [237], which is the rank-two update

*H*+ = *Hc* + *yyT*

*yT s −* (*Hcs*)(*Hcs*)*T*

(4.4) *.*

*sT Hcs*

We will briefly discuss other updates and variations that exploit problem structure in *§*4.3.

71

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.

Copyright ©1999 by the Society for Industrial and Applied Mathematics. This electronic version is for personal use and may not be duplicated or distributed.

72 ITERATIVE METHODS FOR OPTIMIZATION **4.1 Analysis**

This section begins with some simple observations on nonsingularity and positivity of the update. It is very useful for both theory and practice to express (4.4) in terms of the inverse matrices. The formula we use in this book is Lemma 4.1.1.

Lemma 4.1.1. *Let Hc be spd, yT s* = 0*, and H*+ *given by* (4.4)*. Then H−*1

*and*

*H−*1

*I − syT*

*H−*1 *c*

*I − ysT*

+ *is nonsingular*

+*ssT*

(4.5) *.*

+ =

*Proof*. See exercise 4.5.2.

*yT s*

*yT s*

*yT s*

Lemma 4.1.2. *Let Hc be spd, yT s >* 0*, and H*+ *given by* (4.4)*. Then H*+ *is spd. Proof*. Positivity of *Hc* and *yT s* = 0 imply that for all *z* = 0,

*zT H*+*z* = (*zT y*)2

*yT s*+ *zT Hcz −* (*zT Hcs*)2

*sT Hcs .*

Using the symmetry and positivity of *Hc*, we have

(*zT Hcs*)2 *≤* (*sT Hcs*)(*zT Hcz*)*,*

with equality only if *z* = 0 or *s* = 0, and, therefore, since *z, s* = 0 and *yT s >* 0, *zT H*+*z >* (*zT y*)2

*yT s≥* 0*,*

as asserted.

If *yT s ≤* 0 the update is considered a failure.

**4.1.1 Local Theory**

The local theory [37] requires accurate initial approximations to both *x∗* and *∇*2*f*(*x∗*). The statement of the convergence result is easy to understand.

Theorem 4.1.3. *Let the standard assumptions hold. Then there is δ such that if x*0 *− x∗ ≤ δ and H*0 *− ∇*2*f*(*x∗*) *≤ δ,*

*then the BFGS iterates are defined and converge q-superlinearly to x∗.*

**Technical Details**

The proof of Theorem 4.1.3 is technical and we subdivide it into several lemmas. Our proof is a hybrid of ideas from [37], [135], and [154]. Similar to other treatments of this topic [45] we begin with the observation (see *§*2.5.2) that one may assume *∇*2*f*(*x∗*) = *I* for the convergence analysis.

Lemma 4.1.4. *Let the standard assumptions hold and let*

ˆ*f*(*y*) = *f*(*Ay*)*,*

Buy this book from SIAM at http://www.ec-securehost.com/SIAM/FR18.html.