

Neculai Andrei

# Modern Numerical Nonlinear Optimization

---

# **Springer Optimization and Its Applications**

## **Volume 195**

### **Series Editors**

Panos M. Pardalos , *University of Florida*

My T. Thai , *University of Florida*

### **Honorary Editor**

Ding-Zhu Du, *University of Texas at Dallas*

### **Advisory Editors**

Roman V. Belavkin, *Middlesex University*

John R. Birge, *University of Chicago*

Sergiy Butenko, *Texas A&M University*

Vipin Kumar, *University of Minnesota*

Anna Nagurney, *University of Massachusetts Amherst*

Jun Pei, *Hefei University of Technology*

Oleg Prokopyev, *University of Pittsburgh*

Steffen Rebennack, *Karlsruhe Institute of Technology*

Mauricio Resende, *Amazon (United States)*

Tamás Terlaky, *Lehigh University*

Van Vu, *Yale University*

Michael N. Vrahatis, *University of Patras*

Guoliang Xue, *Arizona State University*

Yinyu Ye, *Stanford University*

## Aims and Scope

Optimization has continued to expand in all directions at an astonishing rate. New algorithmic and theoretical techniques are continually developing and the diffusion into other disciplines is proceeding at a rapid pace, with a spotlight on machine learning, artificial intelligence, and quantum computing. Our knowledge of all aspects of the field has grown even more profound. At the same time, one of the most striking trends in optimization is the constantly increasing emphasis on the interdisciplinary nature of the field. Optimization has been a basic tool in areas not limited to applied mathematics, engineering, medicine, economics, computer science, operations research, and other sciences.

The series **Springer Optimization and Its Applications (SOIA)** aims to publish state-of-the-art expository works (monographs, contributed volumes, textbooks, handbooks) that focus on theory, methods, and applications of optimization. Topics covered include, but are not limited to, nonlinear optimization, combinatorial optimization, continuous optimization, stochastic optimization, Bayesian optimization, optimal control, discrete optimization, multi-objective optimization, and more. New to the series portfolio include Works at the intersection of optimization and machine learning, artificial intelligence, and quantum computing.

*Volumes from this series are indexed by Web of Science, zbMATH, Mathematical Reviews, and SCOPUS.*

---

Neculai Andrei

# Modern Numerical Nonlinear Optimization



Springer

Neculai Andrei  
Academy of Romanian Scientists  
Center for Advanced  
Modeling and Optimization  
Bucharest, Romania

ISSN 1931-6828                   ISSN 1931-6836 (electronic)  
Springer Optimization and Its Applications  
ISBN 978-3-031-08719-6       ISBN 978-3-031-08720-2 (eBook)  
<https://doi.org/10.1007/978-3-031-08720-2>

Mathematics Subject Classification (2010): 90-01, 90-08, 65K05

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

---

## Preface

This book is on nonlinear optimization. Although nowadays excellent books with great mathematical rigor are available, none of them insists on providing the performances of nonlinear optimization algorithms for solving real large-scale applications for a broader audience, ranging from college students to scientists and professional industry bodies.

The purpose of this book is to present the theoretical details and the computational performances of the modern optimization algorithms for solving continuous nonlinear optimization applications from different areas of activity. The idea is to give a comprehensive description of the theoretical aspects and details on the computational numerical linear algebra used in nonlinear optimization algorithms. What distinguishes this book from the others in the field is a rigorous treatment of the optimization methods, emphasizing both the convergence properties and the performances of the algorithms which solve a large class of large-scale unconstrained and constrained optimization problems and applications. The qualifier *modern* in the title refers to those unconstrained and constrained optimization algorithms that combine and integrate the latest optimization techniques and advanced computational linear algebra methods.

We have considered only the local solutions of large-scale, complex, and continuous nonlinear optimization applications. Other types of optimization problems, such as mixed-integer programming, mixed complementarity, equilibrium constraints, network optimization, nonlinear integer programming, nonsmooth optimization, and stochastic programming, and the global solution of nonlinear optimization applications are beyond the goal of this work.

The book has 20 chapters and is structured in two parts. The first part (Chaps. 2, 3, 4, 5, 6, 7, 8, and 9) covers the theoretical and practical aspects of the unconstrained optimization. The second part (Chaps. 12, 13, 14, 15, 16, 17, 18, 19, and 20) is dedicated to detailing the constrained nonlinear optimization. Chapters 10 and 11 present an overview of the constrained nonlinear optimization methods and of the optimality conditions for nonlinear optimization, including the duality. Special attention has been given to the Karush-Kuhn-Tucker (KKT) optimality conditions presented in Chap. 11, which characterize the local optimal solutions of nonlinear optimization problems. In general, a nonlinear optimization algorithm is working under the following strategy: in a current point, a quadratic approximation of

the problem is constructed and solved, thus obtaining a new point in which the original problem is again approximated by a quadratic, the procedure being repeated in this way.

The unconstrained optimization methods are characterized by two strategies: the *line-search* and the *trust-region* paradigms. The line-search strategy computes the *search direction*  $d_k$  in the current point  $x_k$ , which must be a descent direction, and a *stepsize*  $\alpha_k$  taken along the search direction to obtain a new estimation of the minimum point,  $x_{k+1} = x_k + \alpha_k d_k$ . On the other hand, the trust-region methods define a *region* around the current iterate within which we *trust* that the quadratic model is an adequate representation of the minimizing function and then choose the step to be an approximate minimizer of the model in this region. Therefore, the trust-region methods choose the direction and the stepsize simultaneously. Obviously, if a step is not acceptable, then the size of the region is reduced and a new minimizer is computed. Both these strategies are operational in nonlinear optimization. For the unconstrained optimization, the following methods are detailed: the steepest descent including the relaxed steepest and the accelerated steepest; the Newton method and its modified variants including the composite Newton; the conjugate gradient methods and their variants standard and hybrid; the memory-less BFGS preconditioned; the quasi-Newton BFGS, DFP, and SR1; the limited-memory BFGS; the inexact or truncated Newton; the trust-region methods and the direct searching ones which use only the function values.

The modern constrained nonlinear optimization methods considered in this book are based on certain strategies, the most important ones being: the penalty and the augmented Lagrangian, the sequential quadratic programming based on quadratic programming, the generalized reduced gradient, the interior-point, the filter methods, the combination of the interior point with filters, and the direct search methods which do not use the derivative information.

The book emphasizes and illustrates a number of reliable and robust packages for solving large-scale unconstrained nonlinear optimization problems (CG-DESCENT, DESCON, CGOPT, L-BFGS, and TN), as well as constrained nonlinear optimization problems and applications (SPG, L-BFGS-B, TNBC, SPENBAR, MINOS, SNOPT, NLPQLP, KNITRO, CONOPT, filterSQP, and IPOPT). Details on their implementation by presenting the advanced numerical linear algebra are a priority of our work. These packages were tested to solve large-scale optimization problems up to 250,000 variables.

A number of four appendices have been included to facilitate the understanding of the theoretical developments and of the numerical performances of the optimization algorithms. Appendix A reviews the most important mathematical concepts used throughout the chapters of the book, from numerical linear algebra to calculus, topology, and convexity. In order to see the behavior of the unconstrained and constrained optimization algorithms, the book provides many examples and real applications described in the next three appendices. Appendix B includes the collection SMUNO with 16 small-scale continuous unconstrained optimization

applications. The collection LACOP with 18 large-scale continuous nonlinear optimization applications with constraints is presented in Appendix C. Appendix D is for six real unconstrained nonlinear optimization applications from the MINPACK-2 collection of Argonne National Laboratory. The vast majority of algorithms have been tested for solving 800 large-scale optimization problems from the UOP collection (described in Andrei (2020a)) up to 10,000 variables, as well as optimization applications from the LACOP and MINPACK-2 collections with 40,000 or 250,000 variables.

The conclusion of these intensive numerical studies is that there is no algorithm able to solve any nonlinear unconstrained or constrained optimization problem or application. The most modern, powerful nonlinear optimization algorithms presented and tested in this book combine and integrate different optimization methods (modified augmented Lagrangian, sequential linear or quadratic programming, interior-point methods with filter line-search or trust-region) and include advanced computational linear algebra techniques.

The book is an invitation for researchers who work in the nonlinear optimization area to understand, master, and develop new algorithms with better properties. It will be of great interest to all those interested in developing and using new advanced techniques for solving unconstrained and constrained optimization complex problems. Mathematical programming researchers, theoreticians and practitioners in operations research, practitioners in engineering, and industry researchers, as well as graduate students in mathematics, and doctoral and master's students in mathematical programming, can find plenty of information and practical suggestions for solving large-scale optimization problems and applications.

I am grateful to the *Alexander von Humboldt Foundation* for its appreciation and generous fellowship during the 2+ years I spent at different universities in Germany. My thanks also go to Elizabeth Loew and to all the staff at Springer for their encouragement and competent assistance with the preparation of this book. Finally, my deepest thanks go to my wife, Mihaela, for her constant understanding and support throughout the years.

Bucharest and Tohaniță / Bran Resort, Romania  
February 2022

Neculai Andrei

---

# Contents

<b>1</b>	<b>Introduction</b>	1
1.1	Mathematical Modeling: Linguistic Models Versus Mathematical Models . . . . .	2
1.2	Mathematical Modeling and Computational Sciences . . . . .	4
1.3	The Modern Modeling Scheme for Optimization . . . . .	5
1.4	Classification of Optimization Problems . . . . .	8
1.5	Optimization Algorithms . . . . .	12
1.6	Collections of Applications for Numerical Experiments . . . . .	12
1.7	Comparison of Algorithms . . . . .	13
1.8	The Structure of the Book . . . . .	14
<b>2</b>	<b>Fundamentals on Unconstrained Optimization. Stepsize Computation</b>	21
2.1	The Problem . . . . .	21
2.2	Fundamentals on the Convergence of the Line-Search Methods . . . . .	23
2.3	The General Algorithm for Unconstrained Optimization . . . . .	29
2.4	Convergence of the Algorithm with Exact Line-Search . . . . .	31
2.5	Inexact Line-Search Methods . . . . .	37
2.6	Convergence of the Algorithm with Inexact Line-Search . . . . .	54
2.7	Three Fortran Implementations of the Inexact Line-Search . . . . .	68
2.8	Numerical Studies: Stepsize Computation . . . . .	75
<b>3</b>	<b>Steepest Descent Methods</b>	81
3.1	The Steepest Descent . . . . .	81
3.2	The Relaxed Steepest Descent . . . . .	92
3.3	The Accelerated Steepest Descent . . . . .	98
3.4	Comments on the Acceleration Scheme . . . . .	106
<b>4</b>	<b>The Newton Method</b>	109
4.1	The Newton Method for Solving Nonlinear Algebraic Systems . . . . .	109

4.2	The Gauss-Newton Method . . . . .	123
4.3	The Newton Method for Function Minimization . . . . .	125
4.4	The Newton Method with Line-Search . . . . .	129
4.5	Analysis of Complexity . . . . .	133
4.6	The Modified Newton Method . . . . .	140
4.7	The Newton Method with Finite-Differences . . . . .	145
4.8	Errors in Functions, Gradients, and Hessians . . . . .	154
4.9	Negative Curvature Direction Methods . . . . .	157
4.10	The Composite Newton Method . . . . .	163
<b>5</b>	<b>Conjugate Gradient Methods . . . . .</b>	<b>169</b>
5.1	The Concept of Nonlinear Conjugate Gradient . . . . .	169
5.2	The Linear Conjugate Gradient Method . . . . .	172
5.3	General Convergence Results for Nonlinear Conjugate Gradient Methods . . . . .	188
5.4	Standard Conjugate Gradient Methods . . . . .	201
5.5	Hybrid Conjugate Gradient Methods . . . . .	218
5.6	Conjugate Gradient Methods as Modifications of the Standard Schemes . . . . .	231
5.7	Conjugate Gradient Methods Memoryless BFGS Preconditioned . . . . .	244
5.8	Solving Large-Scale Applications . . . . .	257
<b>6</b>	<b>Quasi-Newton Methods . . . . .</b>	<b>261</b>
6.1	DFP and BFGS Methods . . . . .	261
6.2	Modifications of the BFGS Method . . . . .	278
6.3	Quasi-Newton Methods with Diagonal Updating of the Hessian . . . . .	287
6.4	Limited-Memory Quasi-Newton Methods . . . . .	290
6.5	The SR1 Method . . . . .	299
6.6	Sparse Quasi-Newton Updates . . . . .	309
6.7	Quasi-Newton Methods and Separable Functions . . . . .	310
6.8	Solving Large-Scale Applications . . . . .	312
<b>7</b>	<b>Inexact Newton Methods . . . . .</b>	<b>315</b>
7.1	The Inexact Newton Method for Nonlinear Algebraic Systems . . . . .	316
7.2	Inexact Newton Methods for Functions Minimization . . . . .	322
7.3	The Line Search Newton-CG Method . . . . .	324
7.4	Comparison of TN Versus Conjugate Gradient Algorithms . . . . .	325
7.5	Comparison of TN Versus L-BFGS . . . . .	327
7.6	Solving Large-Scale Applications . . . . .	329
<b>8</b>	<b>The Trust-Region Method . . . . .</b>	<b>331</b>
8.1	The Trust-Region . . . . .	331
8.2	Algorithms Based on the Cauchy Point . . . . .	337
8.3	The Trust-Region Newton-CG Method . . . . .	340
8.4	The Global Convergence . . . . .	341

---

8.5	Iterative Solution of the Subproblem . . . . .	344
8.6	The Scaled Trust-Region . . . . .	351
<b>9</b>	<b>Direct Methods for Unconstrained Optimization . . . . .</b>	<b>355</b>
9.1	The NELMED Algorithm . . . . .	355
9.2	The NEWUOA Algorithm . . . . .	357
9.3	The DEEPS Algorithm . . . . .	359
9.4	Numerical Study: NELMED, NEWUOA, and DEEPS . . . . .	366
<b>10</b>	<b>Constrained Nonlinear Optimization Methods: An Overview . . . . .</b>	<b>371</b>
10.1	Convergence Tests . . . . .	372
10.2	Infeasible Points . . . . .	372
10.3	Approximate Subproblem: Local Models and Their Solving . . . . .	373
10.4	Globalization Strategy: Convergence from Remote Starting Points . . . . .	377
10.5	The Refining the Local Model . . . . .	380
<b>11</b>	<b>Optimality Conditions for Nonlinear Optimization . . . . .</b>	<b>383</b>
11.1	General Concepts in Nonlinear Optimization . . . . .	384
11.2	Optimality Conditions for Unconstrained Optimization . . . . .	386
11.3	Optimality Conditions for Problems with Inequality Constraints . . . . .	389
11.4	Optimality Conditions for Problems with Equality Constraints . . . . .	393
11.5	Optimality Conditions for General Nonlinear Optimization Problems . . . . .	402
11.6	Duality . . . . .	406
<b>12</b>	<b>Simple Bound Constrained Optimization . . . . .</b>	<b>411</b>
12.1	Necessary Conditions for Optimality . . . . .	412
12.2	Sufficient Conditions for Optimality . . . . .	414
12.3	Methods for Solving Simple Bound Optimization Problems . . . . .	414
12.4	The Spectral Projected Gradient Method (SPG) . . . . .	417
12.5	L-BFGS with Simple Bounds (L-BFGS-B) . . . . .	421
12.6	Truncated Newton with Simple Bounds (TNBC) . . . . .	428
12.7	Applications . . . . .	430
<b>13</b>	<b>Quadratic Programming . . . . .</b>	<b>439</b>
13.1	Equality Constrained Quadratic Programming . . . . .	439
13.2	Inequality Constrained Quadratic Programming . . . . .	449
13.3	Interior Point Methods . . . . .	463
13.4	Methods for Convex QP Problems with Equality Constraints . . . . .	467
13.5	Quadratic Programming with Simple Bounds: The Gradient Projection Method . . . . .	468
13.6	Elimination of Variables . . . . .	471

<b>14 Penalty and Augmented Lagrangian Methods . . . . .</b>	475
14.1 The Quadratic Penalty Method . . . . .	475
14.2 The Nonsmooth Penalty Method . . . . .	479
14.3 The Augmented Lagrangian Method . . . . .	482
14.4 Criticism of the Penalty and Augmented Lagrangian Methods . . . . .	486
14.5 A Penalty-Barrier Algorithm (SPENBAR) . . . . .	488
14.6 The Linearly Constrained Augmented Lagrangian (MINOS) . . . . .	503
<b>15 Sequential Quadratic Programming . . . . .</b>	521
15.1 A Simple Approach to SQP . . . . .	526
15.2 Reduced-Hessian Quasi-Newton Approximations . . . . .	531
15.3 Merit Functions . . . . .	532
15.4 Second-Order Correction (Maratos Effect) . . . . .	534
15.5 The Line-Search SQP Algorithm . . . . .	537
15.6 The Trust-Region SQP Algorithm . . . . .	538
15.7 Sequential Linear-Quadratic Programming (SLQP) . . . . .	541
15.8 A SQP Algorithm for Large-Scale-Constrained Optimization (SNOPT) . . . . .	542
15.9 A SQP Algorithm with Successive Error Restoration (NLPQLP) . . . . .	553
15.10 Active-Set Sequential Linear-Quadratic Programming (KNITRO/ACTIVE) . . . . .	557
<b>16 Primal Methods: The Generalized Reduced Gradient with Sequential Linearization . . . . .</b>	569
16.1 Feasible Direction Methods . . . . .	569
16.2 Active Set Methods . . . . .	571
16.3 The Gradient Projection Method . . . . .	573
16.4 The Reduced Gradient Method . . . . .	576
16.5 The Convex Simplex Method . . . . .	578
16.6 The Generalized Reduced Gradient Method (GRG) . . . . .	579
16.7 GRG with Sequential Linear or Sequential Quadratic Programming (CONOPT) . . . . .	587
<b>17 Interior-Point Methods . . . . .</b>	599
17.1 Prototype of the Interior-Point Algorithm . . . . .	601
17.2 Aspects of the Algorithmic Developments . . . . .	603
17.3 Line-Search Interior-Point Algorithm . . . . .	610
17.4 A Variant of the Line-Search Interior-Point Algorithm . . . . .	611
17.5 Trust-Region Interior-Point Algorithm . . . . .	627
17.6 Interior-Point Sequential Linear-Quadratic Programming (KNITRO/INTERIOR) . . . . .	631
<b>18 Filter Methods . . . . .</b>	647
18.1 Sequential Linear Programming Filter Algorithm . . . . .	649
18.2 Sequential Quadratic Programming Filter Algorithm . . . . .	653

<b>19</b>	<b>Interior-Point Filter Line-Search</b>	661
19.1	Basic Algorithm IPOPT	662
19.2	Implementation Details	669
<b>20</b>	<b>Direct Methods for Constrained Optimization</b>	679
20.1	COBYLA Algorithm	680
20.2	DFL Algorithm	684
	<b>Appendix A: Mathematical Review</b>	691
	<b>Appendix B: The SMUNO Collection</b>	721
	<b>Appendix C: The LACOP Collection</b>	727
	<b>Appendix D: The MINPACK-2 Collection</b>	757
	<b>References</b>	767
	<b>Author Index</b>	793
	<b>Subject Index</b>	801

---

## List of Algorithms

Algorithm 2.1	General unconstrained optimization algorithm . . . . .	30
Algorithm 2.2	Hager and Zhang line-search algorithm . . . . .	45
Algorithm 2.3	The update procedure . . . . .	45
Algorithm 2.4	Procedure secant <sup>2</sup> . . . . .	45
Algorithm 2.5	Backtracking . . . . .	46
Algorithm 2.6	Backtracking – variant . . . . .	47
Algorithm 2.7	Zhang and Hager nonmonotone line-search . . . . .	49
Algorithm 2.8	Nonmonotone line-search Huang, Wan and Chen . . . . .	50
Algorithm 2.9	Ou and Liu nonmonotone line-search . . . . .	51
Algorithm 2.10	Weak Wolfe line-search with simple bisection . . . . .	52
Algorithm 3.1	Steepest descent . . . . .	82
Algorithm 3.2	Accelerated steepest descent (with Wolfe line search) . . . . .	100
Algorithm 4.1	Newton for $F(x) = 0$ . . . . .	110
Algorithm 4.2	Newton for $\min f(x)$ . . . . .	126
Algorithm 4.3	Newton with line-search for $\min f(x)$ . . . . .	130
Algorithm 4.4	Modified Cholesky factorization – Gill and Murray . . . . .	145
Algorithm 4.5	Newton with finite-difference for $F(x) = 0$ . . . . .	149
Algorithm 4.6	Negative curvature direction algorithm . . . . .	158
Algorithm 4.7	Stable Newton algorithm – Gill and Murray . . . . .	159
Algorithm 4.8	Composite Newton of order $m$ . . . . .	164
Algorithm 4.9	Composite Newton of order 1 . . . . .	164
Algorithm 5.1	Line search . . . . .	172
Algorithm 5.2	Linear Conjugate Gradient . . . . .	177
Algorithm 5.3	Preconditioned conjugate gradient . . . . .	186
Algorithm 5.4	General nonlinear conjugate gradient . . . . .	202
Algorithm 5.5	General hybrid conjugate gradient algorithm by using the convex combination of standard schemes . . . . .	226
Algorithm 5.6	Guaranteed descent and conjugacy conditions with a modified Wolfe line-search: DESCON/DESCONa . . . . .	240

Algorithm 5.7	Conjugate gradient memoryless BFGS preconditioned: CONMIN . . . . .	249
Algorithm 6.1	DFP and BFGS methods . . . . .	265
Algorithm 6.2	L-BFGS two-loop recursion . . . . .	292
Algorithm 6.3	L-BFGS . . . . .	292
Algorithm 6.4	Accelerated MM-SR1gen and MM-BFGS . . . . .	304
Algorithm 7.1	Line search Newton-CG (truncated Newton) . . . . .	325
Algorithm 8.1	Trust-region . . . . .	333
Algorithm 8.2	Cauchy point computation. . . . .	337
Algorithm 8.3	Newton conjugate gradient (Steihaug) . . . . .	341
Algorithm 8.4	Trust-region subproblem . . . . .	346
Algorithm 8.5	Generalized Cauchy point . . . . .	352
Algorithm 9.1	Nelder-Mead algorithm (NELMED) (one iteration) . . . . .	357
Algorithm 9.2	Powell algorithm (NEWUOA) . . . . .	358
Algorithm 9.3	Andrei algorithm (DEEPS) . . . . .	361
Algorithm 10.1	Prototype for the nonlinear optimization algorithm . . . . .	371
Algorithm 10.2	Prototype of the interior point algorithm . . . . .	376
Algorithm 10.3	Prototype of the line-search method for the nonlinear optimization . . . . .	380
Algorithm 10.4	Prototype of the trust-region method for the nonlinear optimization . . . . .	381
Algorithm 12.1	Gradient projected for simple bounds . . . . .	417
Algorithm 12.2	Spectral projected gradient—SPG . . . . .	418
Algorithm 12.3	Computation of the generalized Cauchy point . . . . .	426
Algorithm 12.4	L-BFGS-B . . . . .	427
Algorithm 13.1	Reduced gradient for linear equality constraints . . . . .	445
Algorithm 13.2	Preconditioned conjugate gradient for reduced systems . . . . .	447
Algorithm 13.3	Projected conjugate gradient . . . . .	448
Algorithm 13.4	Active-set method for convex quadratic programming . . . . .	452
Algorithm 13.5	Active-set method with positive definite Hessian . . . . .	454
Algorithm 13.6	Reduced gradient for linear inequality constraints . . . . .	454
Algorithm 13.7	Reduced gradient for simple bounds constraints . . . . .	455
Algorithm 13.8	Dual algorithm for quadratic programming . . . . .	457
Algorithm 13.9	Predictor-Corrector algorithm for quadratic programming . . . . .	466
Algorithm 13.10	Gradient projection for quadratic programming with box constraints . . . . .	471

---

Algorithm 14.1	Quadratic penalty method . . . . .	476
Algorithm 14.2	$l_1$ penalty method . . . . .	481
Algorithm 14.3	Augmented Lagrangian method: Equality constraints . . . . .	483
Algorithm 14.4	General barrier . . . . .	489
Algorithm 14.5	Penalty-Barrier—SPENBAR—Andrei . . . . .	495
Algorithm 14.6	MINOS—linear constraints—Murtagh and Saunders . . . . .	508
Algorithm 14.7	MINOS—nonlinear constraints—Murtagh and Saunders . . . . .	514
Algorithm 15.1	Sequential quadratic programming—equality constraints . . . . .	523
Algorithm 15.2	Sequential quadratic programming—inequality constraints . . . . .	525
Algorithm 15.3	Sequential quadratic programming—inequality constraints. Variant . . . . .	528
Algorithm 15.4	Line-search SQP algorithm . . . . .	537
Algorithm 15.5	Trust-region SQP algorithm . . . . .	539
Algorithm 15.6	SNOPT—Gill, Murray and Saunders . . . . .	551
Algorithm 15.7	Linear search in NLPQLP . . . . .	555
Algorithm 15.8	NLPQLP—Schittkowski . . . . .	556
Algorithm 15.9	KNITRO/ACTIVE—Byrd, Gould, Nocedal, and Waltz . . . . .	560
Algorithm 15.10	Penalty update algorithm . . . . .	561
Algorithm 15.11	Preconditioned projected conjugate gradient algorithm . . . . .	562
Algorithm 16.1	Generalized reduced gradient . . . . .	586
Algorithm 16.2	CONOPT—Drud . . . . .	589
Algorithm 17.1	Prototype of the interior-point algorithm . . . . .	602
Algorithm 17.2	Inertia correction and regularization . . . . .	606
Algorithm 17.3	Line-search interior-point algorithm . . . . .	610
Algorithm 17.4	Primal-dual interior-point algorithm—PDIP . . . . .	620
Algorithm 17.5	Trust-region algorithm for barrier problems . . . . .	628
Algorithm 17.6	Trust-region interior-point algorithm . . . . .	630
Algorithm 17.7	KNITRO/INTERIOR-DIRECT—Byrd, Hribar, and Waltz . . . . .	634
Algorithm 17.8	KNITRO/INTERIOR-CG—Byrd, Hribar, and Waltz . . . . .	636
Algorithm 17.9	KNITRO crossover algorithm . . . . .	639
Algorithm 18.1	Sequential linear programming filter—filterSD . . . . .	650
Algorithm 18.2	Basic sequential quadratic programming filter—filterSQP . . . . .	654
Algorithm 18.3	Feasibility restoration algorithm . . . . .	657
Algorithm 18.4	Filter sequential quadratic programming—filterSQP . . . . .	658

Algorithm 19.1	Line-search filter barrier algorithm—Wächter and Biegler . . . . .	668
Algorithm 19.2	Inertia correction algorithm . . . . .	671
Algorithm 19.3	KKT error reduction algorithm . . . . .	675
Algorithm 20.1	DFL algorithm . . . . .	687
Algorithm 20.2	Procedure $PE(\bar{\alpha}, \hat{\alpha}, y, p, \gamma, \alpha)$ . . . . .	687

---

## List of Applications

Application N1	Solid fuel ignition. Bratu's problem . . . . .	120
Application N2	Flow in a drive cavity . . . . .	121
Application Q1	Scheduling of three generators to meet the demand for power over a given period of time . . . . .	462
Application P1	ETA-MACRO simulates a market economy through a dynamic nonlinear optimization process . . . . .	557
Application S1	Weber (1) Locate a central facility . . . . .	721
Application S2	Weber (2) Locate a central facility . . . . .	721
Application S3	Weber (3) Locate a central facility . . . . .	721
Application S4	Analysis of enzymes reaction . . . . .	721
Application S5	Stationary solution of a chemical reactor . . . . .	722
Application S6	Robot kinematics problem . . . . .	722
Application S7	Solar Spectroscopy . . . . .	722
Application S8	Estimation of parameters . . . . .	723
Application S9	Propan combustion in air – reduced variant . . . . .	723
Application S10	Gear train of minimum inertia . . . . .	724
Application S11	Human Heart Dipole . . . . .	724
Application S12	Neurophysiology . . . . .	724
Application S13	Combustion application . . . . .	724
Application S14	Circuit design. . . . .	725
Application S15	Thermistor . . . . .	725
Application S16	Optimal design of a gear train . . . . .	726
Application L1	Chemical equilibrium (ELCH) . . . . .	727
Application L2	Optimization of an alkylation process (ALKI) . . .	727
Application L3	Optimal design of a reactor as a geometric programming problem (PREC) . . . . .	728
Application L4	Cost minimization of a transformer design (TRAFO) . . . . .	729
Application L5	Optimization of a multi-spindle automatic lathe (LATHE) . . . . .	730
Application L6	Static power scheduling (PPSE) . . . . .	731
Application L7	Optimization of a separation process in a membrane with three stages (MSP3) . . . . .	731
Application L8	Optimization of a separation process in a membrane with five stages (MSP5) . . . . .	732

Application L9	Blending/pooling with five feeds and two products (POOL) . . . . .	734
Application L10	Distribution of electrons on a sphere (DES) . . . . .	735
Application L11	Hanging chain (HANG) . . . . .	736
Application L12	Determine the optimal mixing policy of two catalysts along the length of a tubular plug flow reactor involving several reactions (CAT) . . . . .	737
Application L13	Optimal control of a continuous stirred-tank chemical reactor (CSTC) . . . . .	738
Application L14	Optimal temperature field in a rectangular area (DIFF) . . . . .	740
Application L15	Stationary flow of an incompressible fluid in a rectangular area (FLOW/FLOWO) . . . . .	744
Application L16	Fed-batch fermenter for penicillin production (PENICI) . . . . .	749
Application L17	A standard linear lumped parameter system (CONT) . . . . .	754
Application L18	Van der Pol oscillator (POL) . . . . .	755
Application A1	Elastic-Plastic Torsion . . . . .	757
Application A2	Pressure Distribution in a Journal Bearing . . . . .	759
Application A3	Optimal Design with Composite Materials . . . . .	759
Application A4	Steady-State Combustion . . . . .	761
Application A5	Minimal Surfaces with Enneper Boundary Conditions . . . . .	762
Application A6	Inhomogeneous Superconductors: 1-D Ginzburg-Landau . . . . .	764

---

## List of Figures

Fig. 1.1	The process of modeling and solving (optimizing) a problem .....	4
Fig. 1.2	The modern modeling scheme of nonlinear optimization based on algebraic oriented languages .....	7
Fig. 2.1	Subroutine <i>back</i> which implements backtracking (Armijo) .....	69
Fig. 2.2	Subroutine <i>LSbis</i> which generate stepsizes satisfying the weak Wolfe line-search with simple bisection .....	71
Fig. 2.3	Subroutine <i>wolfeLS</i> which generate safeguarded stepsizes satisfying the inexact weak Wolfe line-search with cubic interpolation .....	72
Fig. 2.4	Performances of the Hestenes-Stiefel conjugate gradient algorithm with the inexact weak Wolfe line-search versus the inexact strong Wolfe line-search .....	78
Fig. 2.5	Performance profiles of the Hager-Zhang algorithm with different line-search procedures .....	79
Fig. 3.1	Representation of function $f_1(x)$ .....	83
Fig. 3.2	(a) Evolution of the error $ f_1(x_k) - f_1^* $ . (b) Evolution of $\ g_k\ _2$ .....	83
Fig. 3.3	Evolution of the error $ f_2(x_k) - f_2(x^*) $ , $n = 500$ .....	86
Fig. 3.4	(a) Evolution of the error $ f_2(x_k) - f_2(x^*) $ , $n = 2$ . (b) Zigzag evolution .....	86
Fig. 3.5	Steepest descent with weak Wolfe (SDW) versus steepest descent with backtracking (SDB) .....	88
Fig. 3.6	Evolution of the error $ f(x_k) - f(x^*) $ for steepest descent versus relaxed steepest descent .....	93
Fig. 3.7	Relaxed steepest descent with backtracking (RSDB) versus steepest descent with backtracking (SDB) .....	98
Fig. 3.8	Accelerated steepest descent with backtracking, $\rho = 0.0001$ . Error $ f(x_k) - f^* $ and contribution $(a_k - b_k)^2/2b_k + \rho a_k$ .....	104
Fig. 3.9	Accelerated steepest descent with backtracking, $\rho = 0.01$ . Error $ f(x_k) - f^* $ and contribution $(a_k - b_k)^2/2b_k + \rho a_k$ ..	104

---

Fig. 3.10	Accelerated steepest descent with backtracking (ASDB) versus steepest descent with backtracking (SDB) .....	105
Fig. 3.11	Accelerated steepest descent with backtracking (ASDB) versus relaxed steepest descent with backtracking (RSDB) and versus steepest descent with backtracking (SDB) .....	105
Fig. 4.1	The number of iterations for solving the system from Example 4.1 when the Newton method is initialized in points from the domain $[1, 3] \times [1, 3]$ .....	115
Fig. 4.2	Evolution of $\ F(x_k)\ $ for applications S5, S6, S9, and S14	118
Fig. 4.3	Solution of Bratu's problem .....	121
Fig. 4.4	(a) Flow in a drive cavity ( $Re = 200$ ). (b) Flow in a drive cavity ( $Re = 400$ ). (c) Flow in a drive cavity ( $Re = 1000$ )	123
Fig. 4.5	Representation of function $f$ .....	126
Fig. 4.6	Number of iterations of the Newton method for different initial points from the domain $[-1, 1] \times [0, 2]$ .....	127
Fig. 5.1	Performance of the linear conjugate gradient algorithm for solving the linear system $Ax = b$ , where (a) $A = diag(1, 2, \dots, 1000)$ , (b) the diagonal elements of $A$ are uniformly distributed in $[0,1)$ , (c) the eigenvalues of $A$ are distributed in 10 intervals, (d) the eigenvalues of $A$ are distributed in 5 intervals .....	182
Fig. 5.2	Performance of the linear conjugate gradient algorithm for solving the linear system $Ax = b$ , where the matrix $A$ has a large eigenvalue separated from the others, which are uniformly distributed in $[0,1)$ .....	182
Fig. 5.3	Evolution of the error $\ b - Ax_k\ $ .....	183
Fig. 5.4	Evolution of the error $\ b - Ax_k\ $ of the linear conjugate gradient algorithm for different numbers ( $n_2$ ) of blocks on the main diagonal of matrix $A$ .....	185
Fig. 5.5	Performance profiles of the standard conjugate gradient methods .....	215
Fig. 5.6	Performance profiles of the standard conjugate gradient methods .....	216
Fig. 5.7	Performance profiles of seven standard conjugate gradient methods .....	217
Fig. 5.8	Performance profiles of some hybrid conjugate gradient methods based on the projection concept .....	223
Fig. 5.9	Global performance profiles of six hybrid conjugate gradient methods .....	224
Fig. 5.10	Performance profiles of NDLSDY versus the standard conjugate gradient methods LS, DY, PRP, CD, FR, and HS .....	230
Fig. 5.11	Performance profiles of CG-DESCENT versus HS, PRP, DY, and LS .....	235

---

Fig. 5.12	Performance profiles of CG-DESCENTaw (CG-DESCENT with approximate Wolfe conditions) versus HS, PRP, DY, and LS .....	236
Fig. 5.13	Performance profiles of CG-DESCENT and CG-DESCENTaw (CG-DESCENT with approximate Wolfe conditions) versus DL ( $t = 1$ ) and DL+ ( $t = 1$ ) ..	237
Fig. 5.14	Performance profile of DESCONa versus HS and versus PRP .....	243
Fig. 5.15	Performance profile of DESCONa versus DL ( $t = 1$ ) and versus CG-DESCENT .....	243
Fig. 5.16	Performances of DESCONa versus CG-DESCENTaw ..	244
Fig. 5.17	Performance profiles of CONMIN versus HS, PRP, DY, and LS .....	250
Fig. 5.18	Performance profiles of CONMIN versus DL ( $t = 1$ ), DL+ ( $t = 1$ ), CG-DESCENT and DESCONa .....	251
Fig. 5.19	Performance profiles of DK+w versus CG-DESCENT and DESCONa .....	256
Fig. 5.20	Performance profiles of DK+aw versus CONMIN and CG-DESCENTaw .....	256
Fig. 5.21	Performance profiles of DK+iw versus DK+w and DK+aw .....	257
Fig. 6.1	Performance profiles of BFGS versus CONMIN, DESCON, CG-DESCENT, and CG-DESCENTaw .....	268
Fig. 6.2	Performance profiles of BFGSFI versus BFGS, BFGSB, BFGSC, BFGSD, BFGSE, and BFGSY. CPU time metric	284
Fig. 6.3	Performance profiles of YONS versus QND, WQND, QCD, and DNRTR. CPU time metric, $n = 500$ .....	290
Fig. 6.4	Performance profiles of L-BFGS versus CONMIN, DESCONa, CG-DESCENT, and CG-DESCENTaw .....	293
Fig. 6.5	Solution of application A1. Elastic-plastic torsion. $nx =$ $200$ , $ny = 200$ .....	294
Fig. 6.6	Solution of application A2. Pressure distribution in a journal bearing. $nx = 200$ , $ny = 200$ .....	295
Fig. 6.7	Solution of application A3. Optimal design with com- posite materials. $nx = 200$ , $ny = 200$ .....	295
Fig. 6.8	Solution of application A4. Steady-state combustion. $nx$ $= 200$ , $ny = 200$ .....	296
Fig. 6.9	Solution of application A5. Minimal surfaces with Enneper boundary conditions. $nx = 200$ , $ny = 200$ .....	297
Fig. 6.10	Solution of application A6. Inhomogeneous superconductors: 1-D Ginzburg-Landau. $n = 1000$ .....	297
Fig. 6.11	Accelerated MM-SR1gen versus MM-BFGS, range [1000, 10000] .....	307
Fig. 6.12	Accelerated MM-SR1gen versus BFGS from CONMIN, range [100, 1000] .....	308

---

Fig. 7.1	Performance of TN (Nash) versus conjugate gradient algorithms CONMIN, DESCON, CG-DESCENT, and CG-DESCENTaw .....	326
Fig. 7.2	Performance profiles of L-BFGS ( $m = 5$ ) versus TN (truncated Newton) based on iterations calls, function calls, and CPU time, respectively .....	328
Fig. 12.1	SPG: Quadratic interpolation versus cubic interpolation .	420
Fig. 12.2	L-BFGS-B versus SPG with quadratic interpolation (SPGp) .....	428
Fig. 12.3	L-BFGS-B versus SPG with cubic interpolation (SPGc)	428
Fig. 12.4	TNBC versus L-BFGS-B ( $m = 5$ ) and versus SPGp.....	429
Fig. 12.5	TNBC versus L-BFGS-B ( $m = 5$ ) and versus SPGc .....	430
Fig. 12.6	Solution of the application A1 without simple bounds. $nx = 200$ , $ny = 200$ .....	431
Fig. 12.7	Solution of the application A1 with simple bounds $0 \leq v \leq 0.01$ . $nx = 200$ , $ny = 200$ .....	431
Fig. 12.8	Solution of application A2 without simple bounds. $nx = 200$ , $ny = 200$ .....	432
Fig. 12.9	Solution of application A2 with simple bound $0 \leq v \leq 1$ . $nx = 200$ , $ny = 200$ .....	433
Fig. 12.10	Solution of application A3 without simple bounds. $nx = 200$ , $ny = 200$ .....	434
Fig. 12.11	Solution of application A3 with simple bounds $-0.02 \leq v \leq 0$ . $nx = 200$ , $ny = 200$ .....	434
Fig. 12.12	Solution of application A4 without simple bounds. $nx = 200$ , $ny = 200$ .....	435
Fig. 12.13	Solution of application A4 with simple bounds $0 \leq v \leq 0.2$ . $nx = 200$ , $ny = 200$ .....	435
Fig. 12.14	Solution of application A6 without simple bounds. $n = 1000$ .....	436
Fig. 12.15	Solution of application A6 with simple bounds $0 \leq v \leq 0.001$ . $n = 1000$ .....	436
Fig. 16.1	Computation of $X^1 = [x_1 \ y_1]^T \in V$ . The point $\tilde{X}^1 \in L$ corresponds to $\alpha = \alpha_1$ , while point $\bar{X}$ corresponds to $\alpha = \bar{\alpha}$ .....	585
Fig. 16.2	The iterations for solving the system (16.50) starting from the point $\tilde{X}^1$ lead to the point $\hat{X}^2 \notin V$ .....	585
Fig. 16.3	The iterations for solving the system (16.50) starting from the point $\tilde{X}^1$ lead to the points $\hat{X}^2$ , $\hat{X}^3$ , and then to $X^1$ ...	585
Fig. L1	Hanging chain of minimal potential energy of length $L = 4$ or $L = 6$ .....	737
Fig. L2	Evolution of $u(t)$ given by CONOPT and MINOS .....	738
Fig. L3	Evolution of $x_1(t)$ and $x_2(t)$ given by CONOPT .....	738
Fig. L4	Evolution of $x_1(t)$ and $x_2(t)$ .....	739
Fig. L5	Evolution of $u(t)$ .....	739

---

Fig. L6	Evolution of $x_1(t)$ and $x_2(t)$ .....	740
Fig. L7	Evolution of $u(t)$ .....	740
Fig. L8	Solution to the heat transportation problem with two fixed boundary conditions .....	742
Fig. L9	Solution to the heat transportation problem with two fixed boundary conditions and one heat source on the cell (I5, J5) .....	742
Fig. L10	Solution to the heat transportation problem with two fixed boundary conditions and two heat sources .....	742
Fig. L11	Solution of the transient heat transportation problem at 6 time periods .....	743
Fig. L12	Flow domain and its discretization .....	746
Fig. L13	Velocity in the direction $x$ .....	746
Fig. L14	Three cross-sections of velocity in the direction $x$ .....	747
Fig. L15	Flow domain with an obstacle .....	747
Fig. L16	Velocity in the direction $x$ .....	748
Fig. L17	Three cross-sections of velocity in the direction $x$ .....	748
Fig. L18	Three cross-sections of velocity in the direction $x$ with inertial effects .....	749
Fig. L19	Velocity in the direction $x$ for two obstacles .....	749
Fig. L20	Three cross-sections of velocity in the direction $x$ .....	749
Fig. L21	Evolution of the control $u(t)$ .....	752
Fig. L22	State variables $y_1(t)$ and $y_2(t)$ .....	752
Fig. L23	State variables $y_3(t)$ and $y_4(t)$ .....	752
Fig. L24	Time evolution of variables (backward Euler method) .....	753
Fig. L25	Time evolution of the state variables .....	754
Fig. L26	Time evolution of the control $u(t)$ .....	755
Fig. L27	Evolution of $y_1(t)$ and $y_2(t)$ .....	755
Fig. L28	Evolution of $y_3(t)$ .....	756
Fig. L29	Control $u(t)$ .....	756
Fig. A1	Solution of Application A1. $nx = 200$ , $ny = 200$ .....	759
Fig. A2	Solution of Application A2. $nx = 200$ , $ny = 200$ .....	760
Fig. A3	Solution of Application A3. $nx = 200$ , $ny = 200$ .....	761
Fig. A4	Solution of Application A4. $nx = 200$ , $ny = 200$ .....	763
Fig. A5	Solution of Application A5. $nx = 200$ , $ny = 200$ .....	764
Fig. A6	Solution of Application A6. $n = 1000$ .....	765

---

## List of Tables

Table 1.1	Algebraic oriented modeling languages (partial list) ...	6
Table 2.1	Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Steepest descent algorithm. Function $f_1(x)$ .....	76
Table 2.2	Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Steepest descent algorithm. Function $f_2(x)$ .....	76
Table 2.3	Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Steepest descent algorithm. Function $f_3(x)$ .....	76
Table 2.4	Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Hestenes- Stiefel conjugate gradient algorithm. Function $f_1(x)$ ....	76
Table 2.5	Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Hestenes- Stiefel conjugate gradient algorithm. Function $f_2(x)$ ....	77
Table 2.6	Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Hestenes- Stiefel conjugate gradient algorithm. Function $f_3(x)$ ....	77
Table 2.7	Global performances of HZ for solving 800 uncon- strained optimization problems from the UOP collec- tion with different line-searches .....	80
Table 3.1	Number of iterations of the steepest descent and of the relaxed steepest descent for the quadratic problem from Example 3.3 .....	94
Table 3.2	Number of iterations and the average stepsize for SDB and RSDB with backtracking for Example 3.4 .....	95
Table 4.1	Iterations generated by the Newton method .....	111
Table 4.2	Evolution of $\ x^* - x_k\ $ and $\ F(x_k)\ $ .....	111
Table 4.3	Stationary solution of a chemical reactor. Initial point, solution, functions values in these points. Newton method. 5 iterations. $\epsilon = 10^{-6}$ .....	117

---

Table 4.4	Robot kinematics problem. Initial point, solution, functions values in these points. Newton method. 3 iterations. $\epsilon = 10^{-6}$ .....	119
Table 4.5	Propan combustion in air – reduced variant. Initial point, solution, functions values in these points. Newton method. 9 iterations. $\epsilon = 10^{-6}$ .....	119
Table 4.6	Circuit design. Initial point, solution, functions values in these points. Newton method. 4 iterations. $\epsilon = 10^{-6}$ .....	120
Table 4.7	Initial points, value of function in initial points, number of iterations. $n = 50$ , $\epsilon = 10^{-8}$ , $\rho = 0.0001$ and $\beta = 0.8$ .....	139
Table 4.8	Initial points, value of function in initial points, number of iterations. $n = 100$ , $\epsilon = 10^{-8}$ , $\rho = 0.0001$ and $\beta = 0.8$ .....	139
Table 4.9	Initial points, value of function in initial points, number of iterations. $n = 300$ , $\epsilon = 10^{-8}$ , $\rho = 0.0001$ and $\beta = 0.8$ .....	139
Table 4.10	Stationary solution of a chemical reactor. Initial point, solution, functions values in these points. Newton with finite-difference method. 5 iterations. $\epsilon = 10^{-6}$ .....	149
Table 4.11	Robot kinematics problem. Initial point, solution, functions values in these points. Newton with finite-difference method. 4 iterations. $\epsilon = 10^{-6}$ .....	150
Table 4.12	Propan combustion in air – reduced variant. Initial point, solution, functions values in these points. Newton with finite-difference method. 10 iterations. $\epsilon = 10^{-6}$ .....	150
Table 4.13	Circuit design. Initial point, solution, functions values in these points. Newton with finite-difference method. 4 iterations. $\epsilon = 10^{-6}$ .....	151
Table 4.14	Stationary solution of a chemical reactor. Initial point, solution, functions values in these points. Composite Newton method of order 1. 4 iterations. $\epsilon = 10^{-6}$ .....	166
Table 4.15	Robot kinematics problem. Initial point, solution, functions values in these points. Composite Newton method of order 1. 2 iterations. $\epsilon = 10^{-6}$ .....	166
Table 4.16	Propan combustion in air – reduced variant. Initial point, solution, functions values in these points. Composite Newton method of order 1. 7 iterations. $\epsilon = 10^{-6}$ .....	167
Table 4.17	Circuit design. Initial point, solution, functions values in these points. Composite Newton method of order 1. 3 iterations. $\epsilon = 10^{-6}$ .....	167
Table 5.1	Choices of $\beta_k$ in standard conjugate gradient methods .....	201

---

Table 5.2	Performances of HS, FR, and PRP for solving five applications from the MINPACK-2 collection .....	217
Table 5.3	Performances of PRP+ and CD for solving five applications from the MINPACK-2 collection .....	217
Table 5.4	Performances of LS and DY for solving five applications from the MINPACK-2 collection .....	218
Table 5.5	Hybrid selection of $\beta_k$ based on the projection concept .....	219
Table 5.6	Performances of TAS, PRP-FR, and GN for solving five applications from the MINPACK-2 collection .....	224
Table 5.7	Performances of HS-DY, hDY, and LS-CD for solving five applications from the MINPACK-2 collection .....	224
Table 5.8	Performances of NDLSDY for solving five applications from the MINPACK-2 collection .....	231
Table 5.9	Performances of CG-DESCENT and CG-DESCENTaw for solving five applications from the MINPACK-2 collection .....	237
Table 5.10	Performances of DESCONa for solving five applications from the MINPACK-2 collection .....	244
Table 5.11	Performances of CONMIN for solving five applications from the MINPACK-2 collection .....	251
Table 5.12	Performances of DK+w and DK+aw for solving five applications from the MINPACK-2 collection .....	257
Table 5.13	Performances of CG-DESCENT and of CG-DESCENTaw for solving five large-scale applications from the MINPACK-2 collection .....	258
Table 5.14	Performances of DESCON and of DESCONa for solving five large-scale applications from the MINPACK-2 collection .....	258
Table 5.15	Performances of DK+w and of DK+iw for solving five large-scale applications from the MINPACK-2 collection .....	258
Table 6.1	Performances of L-BFGS for different values of $m$ ....	293
Table 6.2	Performances of L-BFGS Elastic-plastic torsion. $n = 40,000$ . $\varepsilon = 10^{-6}$ .....	294
Table 6.3	Performances of L-BFGS Pressure distribution in a journal bearing. $n = 40,000$ . $\varepsilon = 10^{-6}$ .....	294
Table 6.4	Performance of L-BFGS. Optimal design with composite materials. $n = 40,000$ . $\varepsilon = 10^{-6}$ .....	295
Table 6.5	Performances of L-BFGS. Steady-state combustion. $n = 40,000$ . $\varepsilon = 10^{-6}$ .....	296
Table 6.6	Performances of L-BFGS. Minimal surfaces with Enneper boundary conditions. $n = 40,000$ . $\varepsilon = 10^{-6}$ .....	296
Table 6.7	Performances of L-BFGS. Inhomogeneous superconductors: 1-D Ginzburg-Landau. $n = 1000$ . $\varepsilon = 10^{-6}$ .....	297

---

Table 6.8	Performances of MM-SR1gen versus MM-BFGS (40,000 variables, <i>cpu</i> seconds) .....	308
Table 6.9	Performances of L-BFGS ( $m = 5$ ) for solving five applications from the MINPACK-2 collection (250,000 variables, <i>cpu</i> seconds) .....	312
Table 6.10	Performances of MM-SR1gen for solving five applications from the MINPACK-2 collection (250,000 variables, <i>cpu</i> seconds) .....	313
Table 7.1	Performances of TN for solving five applications from the MINPACK-2 collection (40,000 variables) .....	329
Table 7.2	Performances of TN for solving five applications from the MINPACK-2 collection (250,000 variables) .....	329
Table 8.1	Performances of TRON for solving five applications from the MINPACK-2 collection .....	351
Table 9.1	Performances of NELMED .....	366
Table 9.2	Performances of NEWUOA .....	366
Table 9.3	Performances of DEEPS .....	366
Table 9.4	Performances of NELMED for solving 16 small-scale applications from the SMUNO collection .....	367
Table 9.5	Performances of NEWUOA for solving 16 small-scale applications from the SMUNO collection .....	367
Table 9.6	Performances of DEEPS for solving 16 small-scale applications from the SMUNO collection .....	368
Table 9.7	Performances of NEWUOA for solving 10 large-scale applications from the SMUNO collection .....	368
Table 9.8	Performances of DEEPS for solving 10 large-scale applications from the SMUNO collection .....	369
Table 12.1	Elastic-plastic torsion. SPG. 40,000 variables. $M = 10$	431
Table 12.2	Elastic-plastic torsion. L-BFGS-B. 40,000 variables. $m = 5$ .....	432
Table 12.3	Elastic-plastic torsion. TNBC. 40,000 variables .....	432
Table 12.4	Pressure distribution in a journal bearing. SPG. 40,000 variables. $M = 10$ .....	433
Table 12.5	Pressure distribution in a journal bearing. L-BFGS-B. 40,000 variables. $m = 5$ .....	433
Table 12.6	Optimal design with composite materials. SPG. 40,000 variables. $M = 10$ .....	433
Table 12.7	Optimal design with composite materials. L-BFGS-B. 40,000 variables. $m = 5$ .....	434
Table 12.8	Steady-state combustion. SPG. 40,000 variables. $M = 10$ .....	435
Table 12.9	Steady-state combustion. L-BFGS-B. 40,000 variables. $m = 5$ .....	435
Table 12.10	1-D Ginzburg-Landau problem. SPG. 1000 variables. $M = 10$ .....	437
Table 12.11	1-D Ginzburg-Landau problem. L-BFGS-B. 1000 variables. $m = 5$ .....	437

---

Table 13.1	Optimization process by QLD .....	461
Table 13.2	Initial point, solution, and bounds on variables .....	462
Table 13.3	Dynamic Optimization process by QLD .....	463
Table 14.1	Augmented Lagrangian method .....	485
Table 14.2	Quadratic penalty method .....	486
Table 14.3	Iterations generated by SPENBAR. $\sigma_0 = 0.1$ , $\beta = 0.9$ , $\tau = 10^{-8}$ .....	500
Table 14.4	Iterations generated by SPENBAR. $\sigma_0 = 0.1$ , $\beta = 0.9$ , $\tau = 10^{-8}$ . Optimization of a heavy body .....	501
Table 14.5	Performances of SPENBAR. $\sigma_0 = 0.1$ , $\beta = 0.9$ , $\tau = 10^{-8}$ . Application DES. Thomson problem .....	502
Table 14.6	Performances of SPENBAR. $\sigma_0 = 0.1$ , $\beta = 0.9$ , $\tau = 10^{-8}$ . Application HANG. Dog curve .....	502
Table 14.7	Performances of SPENBAR for solving nine applications from the LACOP collection. $\sigma_0 = 0.1$ , $\beta = 0.9$ , $\tau = 10^{-8}$ .....	503
Table 14.8	MINOS for solving some large-scale linear programs .....	510
Table 14.9	MINOS with different initializations. Example 14.4 ..	515
Table 14.10	MINOS with different initializations. Example 14.5 ..	516
Table 14.11	Performances of MINOS for solving 12 applications from the LACOP collection. Small-scale nonlinear application .....	517
Table 14.12	Performances of MINOS for solving six applications from the LACOP collection. Large-scale nonlinear applications .....	517
Table 14.13	Performances of MINOS for solving the HANG application from the LACOP collection. Large-scale nonlinear applications .....	518
Table 15.1	Performances of SNOPT for solving 12 applications from the LACOP collection. Small-scale nonlinear optimization applications .....	552
Table 15.2	Performances of SNOPT for solving 6 applications from the LACOP collection. Large-scale nonlinear optimization applications .....	552
Table 15.3	Comparison: MINOS versus SNOPT for solving 15 large-scale applications from the LACOP collection. Large-scale nonlinear optimization applications .....	553
Table 15.4	Performances of NLPQLP for solving 8 applications from the LACOP collection. Small-scale nonlinear optimization applications .....	557
Table 15.5	Performances of NLPQLP for solving the ETA- MACRO application .....	557
Table 15.6	Performances of KNITRO/ACTIVE for solving 12 applications from the LACOP collection. Option 3. Small-scale nonlinear optimization applications .....	564

---

Table 15.7	Performances of KNITRO/ACTIVE for solving 6 applications from the LACOP collection. Option 3.	564
	Large-scale nonlinear optimization applications .....	
Table 15.8	Performances of KNITRO/ACTIVE for solving the HANG application from the LACOP collection. Option 3. Large-scale nonlinear optimization applications .....	565
Table 16.1	Performances of CONOPT for solving 12 applications from the LACOP collection. Small-scale nonlinear optimization applications .....	594
Table 16.2	Performances of CONOPT for solving the PENICI application from the LACOP collection .....	594
Table 16.3	Performances of CONOPT for solving 6 applications from the LACOP collection. Large-scale nonlinear optimization applications .....	595
Table 16.4	Comparison between KNITRO/ACTIVE and CONOPT	595
Table 16.5	Performances of CONOPT for solving 6 applications from the LACOP collection with SQP inhibited. Large-scale nonlinear optimization applications .....	596
Table 16.6	CONOPT with SQP versus CONOPT without SQP ....	596
Table 16.7	Performances of CONOPT with SQP for solving the HANG application from the LACOP collection .....	597
Table 16.8	Performances of CONOPT without SQP for solving the HANG application from the LACOP collection .....	597
Table 16.9	Comparison: MINOS, KNITRO/ACTIVE, SNOPT and CONOPT. CPU computing time (seconds) .....	597
Table 17.1	Evolution of some elements of PDIP. Example 17.1 ...	624
Table 17.2	Evolution of parameters of PDIP. Example 17.1 .....	625
Table 17.3	Evolution of some elements of PDIP. Example 17.2 ...	626
Table 17.4	Evolution of parameters of PDIP. Example 17.2 .....	626
Table 17.5	Performances of PDIP for solving 4 applications from the LACOP collection .....	626
Table 17.6	Performances of KNITRO for solving 12 applications from the LACOP collection. Option 0. Small-scale nonlinear optimization applications .....	640
Table 17.7	Performances of KNITRO/INTERIOR-DIRECT for solving 12 applications from the LACOP collection. Option 1. Small-scale nonlinear optimization applications .....	640
Table 17.8	Performances of KNITRO/INTERIOR-CG for solving 12 applications from the LACOP collection. Option 2. Small-scale nonlinear optimization applications .....	641
Table 17.9	Performances of the KNITRO algorithms. Small-scale nonlinear optimization applications .....	641
Table 17.10	Performances of KNITRO for solving 6 applications from the LACOP collection. Option 0. Large-scale nonlinear optimization applications .....	642

<b>Table 17.11</b>	Performances of KNITRO/INTERIOR-DIRECT for solving 6 applications from the LACOP collection. Option 1. Large-scale nonlinear optimization applications .....	642
<b>Table 17.12</b>	Performances of KNITRO/INTERIOR-CG for solving 6 applications from the LACOP collection. Option 2. Large-scale nonlinear optimization applications .....	643
<b>Table 17.13</b>	Performances of the KNITRO algorithms. Large-scale nonlinear optimization applications .....	643
<b>Table 17.14</b>	Performances of KNITRO. Application PENICI ( $n = 707, l = 602, m = 0$ ) .....	643
<b>Table 17.15</b>	Comparisons between KNITRO and CONOPT. Small-scale nonlinear optimization applications .....	644
<b>Table 17.16</b>	Comparisons between KNITRO and CONOPT. Large-scale nonlinear optimization applications .....	644
<b>Table 18.1</b>	Performances of filterSD for solving 8 applications from the LACOP collection .....	652
<b>Table 18.2</b>	Comparison of filterSD versus NLPQLP and KNITRO .....	652
<b>Table 19.1</b>	Performances of IPOPT for solving 12 applications from the LACOP collection. Small-scale nonlinear optimization applications .....	676
<b>Table 19.2</b>	Performances of IPOPT for solving 6 applications from the LACOP collection. Large-scale nonlinear optimization applications .....	676
<b>Table 19.3</b>	Performances of IPOPT for solving the PENICI application .....	677
<b>Table 19.4</b>	Performances of the IPOPT algorithm. Small-scale nonlinear optimization applications .....	677
<b>Table 19.5</b>	Performances of the IPOPT algorithm. Large-scale nonlinear optimization applications .....	677
<b>Table 20.1</b>	Performances of COBYLA. $\rho_{beg} = 1.5d0$ , $\rho_{end} = 1. d - 8$ .....	684
<b>Table 20.2</b>	Performances of DFL .....	689



# Introduction

1

This book is on modern continuous nonlinear optimization. Continuous nonlinear optimization problems have a simple mathematical model and always refer to a real physical system, the running of which we want to optimize. Firstly, a nonlinear optimization problem contains an *objective function* which measures the performances or requirements of the system. Often, this function represents a profit, a time interval, a level, a sort of energy or combination of different quantities which have a physical significance for the modeler. The objective function depends on some characteristics of the system, called *variables* or *unknowns*. The purpose of any optimization problem is to find the values of these variables that minimize (or maximize) the objective function subject to some *constraints* which the variables must satisfy. Constraints of an optimization problem may have different algebraic expressions. There are static and dynamic constraints called *functional constraints*. The difference between these types of constraints comes from the structure of their Jacobian. The dynamic constraints always involve a Jacobian with a block-diagonal structure. Besides, these functional constraints are nonlinear functions which may be equalities or inequalities, or even range constraints. Another very important type of constraints is the *simple bounds on variables*. The real applications of optimization include simple bounds on variables which express the constructive engineering conditions and limits of the system to be optimized. The constraints of an optimization problem define the so-called *feasible domain* of the problem. Both the objective function and the constraints may depend on some *parameters* with known values which characterize the system under optimization. The process of identifying the variables, parameters, simple bounds on variables, the objective functions, and constraints is known as *modeling*, one of the finest intellectual activities. In this book we assume that the variables can take real values and the objective function and the constraints are smooth enough (at least twice continuously differentiable) with known first-order derivatives. When the number of variables and the number of constraints is large, then the optimization problem is quite challenging.

If the objective function and the constraints are all linear, then the problem is known as *linear programming*, (see Dantzig (1963), Vanderbei (2001), and Andrei (2011d)) which represents a special chapter in the mathematical programming area and is not material to this book. If the objective function is a quadratic function and the constraints are all linear, we have a *quadratic programming* problem with plenty of theoretical and computational results and applications (see Chap. 13 of this book). There are optimization problems with only simple bounds on variables (see Chap. 12 of this book). The *general nonlinear optimization* problems include both nonlinear functional constraints

and simple bounds on variables. The mathematical formulation of a general nonlinear optimization problem is as follows:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x), \\ & \text{subject to} \\ & e_i(x) = 0, \quad i \in E, \quad c_j(x) \geq 0, \quad j \in I, \quad l \leq x \leq u, \end{aligned} \tag{1.1}$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $E$  and  $I$  are the set of indices for *equality and inequality constraints*, respectively. The equality constraints can be assembled as  $e(x) = 0$ , where  $e: \mathbb{R}^n \rightarrow \mathbb{R}^m$ , and  $m = \text{card}(E)$ . Similarly, the inequality constraints can be assembled as  $c(x) \geq 0$ , where  $c: \mathbb{R}^n \rightarrow \mathbb{R}^p$ , and  $p = \text{card}(I)$ . The bounds  $l, u \in \mathbb{R}^n$  satisfy the natural conditions  $-\infty < l_i \leq u_i < +\infty$  for all  $i = 1, \dots, n$ .  $l_i$  is called the *lower bound* of variable  $x_i$ , while  $u_i$  is called the *upper bound* of  $x_i$ . If  $l_i = u_i$ , then the variable  $x_i$  is *fixed* and can be eliminated from the problem.

Suppose there is an algorithm or a collection of algorithms able to solve general nonlinear optimization problems. For recognizing whether the algorithm has found the optimal solution of the problem, one needs some mathematical tools. The most elegant and effective one is the so-called *optimality conditions*. These are expressed as a nonlinear algebraic system of equations which may be used both to design nonlinear optimization algorithms and to check whether the current set of variables is indeed the solution of the problem. There are two types of optimality conditions. The *necessary optimality conditions* are conditions that must be satisfied by any solution point. The *sufficient optimality conditions* are those that, if satisfied at a certain point, guarantee that this point is in fact a solution. Observe the nuances of these two concepts (see Chap. 11 of this book). The necessary conditions are also called the *first-order conditions* because they involve the properties of both the gradients of the objective function and the constraints and they also show how these gradients are related to each other at a solution. Actually, these conditions are called KKT (Karush-Kuhn-Tucker) conditions and are the foundation of many algorithms for nonlinear optimization. The *second-order conditions*, both necessary and sufficient, use the second derivatives of the functions defining the problem and examine these derivatives to see whether this extra information can resolve the issue of increase or decrease of the objective function values in undecided directions.

If the optimality conditions are not satisfied at a current point, then they must be used to see how the current solution estimation can be improved. This is the subject of *sensitivity analysis*, which shows the sensitivity of the current solution to changes in the structure and in the data of the model of the optimization problem.

---

## 1.1 Mathematical Modeling: Linguistic Models Versus Mathematical Models

Normally, everyone is familiar with different representations of the surrounding world, the so-called *mental models*, used in each moment of our existence. The decisions made in different situations are not based on the real world, but on our mental images of the real world, on our mental images of the relations among the components of the real world. The mental models represent our understanding of the part of the creation that we want to know. Having in view that the support of our thinking is the word, it follows that mental models are actually *linguistic models*. Therefore, in one way or another, the human beings set forth their understanding of the real world as a *linguistic description* expressed as a corpus of assertions (theorems) of the general form: *if ... then ...*.

The mental models have some advantages which recommend them to be used in our efforts to understand the world. A mental model is *flexible* in the sense that it can take into consideration a

domain of information sensibly larger than the numerical one. Besides, a linguistic model may be quickly *adapted* to some new situations and can be *modified* as soon as some new information is available. The mental models are *filters* through which we can explain our experiences and evaluate and select different actions. In a way, the greatest philosophical systems, political and economical doctrines, theories of physics, or the literature itself are linguistic models.

However, the mental models have some disadvantages as well. They are not so easily understood by the others. Their interpretation is *dependent* on their creator. Besides, the hypothesis used to generate them is difficult to examine and even to accept. The ambiguities and contradictions contained in these types of models can remain undetected, unexplained, and therefore unsolved. It is quite natural to have difficulties in understanding the linguistic models suggested by the others. Surprisingly, we are not so good at developing and understanding our own mental models or at using them during the process of decision-making. Psychologists have shown that we are able to consider only a very small number of factors in decision-making. In other words, the mental models we use in decision-making are *extremely simple*. These are often imperfect because we frequently persist in error when deducing the consequences from the suppositions on which they are based. These models often express what we would like to happen and not what actually happens in reality.

But the greatest *imperfection* of the mental models lies in the fact that these intellectual developments do not satisfy the criteria of *completeness*, *minimality*, and *non-contradiction*. It is very likely that some very important assertions which change their significance may be omitted in a mental (linguistic) model. Clearly, some new assertions may be introduced, which are in contradiction with those we have previously considered in our reasoning. At the same time, in using the linguistic models we are often faced with the very difficult problem of *word rectification*. This problem dramatically limits the usage of linguistic models with different groups of people. Finally, we must emphasize that during the linguistic models analysis and solving process, we are very likely to be confronted with the *circularity danger*. The problem of the circularity of formal systems was solved by Gödel (1931), who showed that expressing knowledge into a formal system and into logical formal systems like those of Russel or of Zermelo-Fraenkel-Newmann is an illusion. There are relatively simple assertions (theorems) which are impossible to be solved (decidable) in this kind of formal systems.

A *mathematical model* is a representation in mathematical symbols of the relations between the variables and the parameters belonging to the part of the creation we are interested in. The relations describing the mathematical model often include variables and their derivatives, thus expressing the *local character* of the model and that of *predictability* as well. The mathematical models have a number of advantages versus the linguistic models. The most important is that mathematical models do not have any imperfections while the linguistic models do have. They are *explicit* in the sense that the hypothesis and the assumptions used in their development are public and subject to any criticism. Besides, the (logical) consequences after solving them are very well *justified mathematically*. Finally, the mathematical models are more *comprehensive*, being able to simultaneously consider an appreciable multitude of factors. But the most important characteristic of mathematical models is that they are *written on the basis of the conservation laws*. In this respect the Noether theorem shows that the conservation laws are direct consequences of different symmetries (Andrei, 2008h). For example, the conservation of energy is a consequence of the temporal symmetry, while the conservation of momentum is a consequence of the symmetry of space. The entire physical world is depicted as being governed according to mathematical laws.

## 1.2 Mathematical Modeling and Computational Sciences

The theory and practice of mathematical modeling is based on computational sciences. Broadly, computational sciences consist in the usage of computer systems for analyzing and solving scientific problems. There is a sharp distinction between computer science and computational science. Computer science focuses on the building and the analysis of computers as well as on computability. On the other side, computational sciences emphasize the development of methods, techniques, and algorithms for solving mathematical models and center on the convergence and complexity of algorithms.

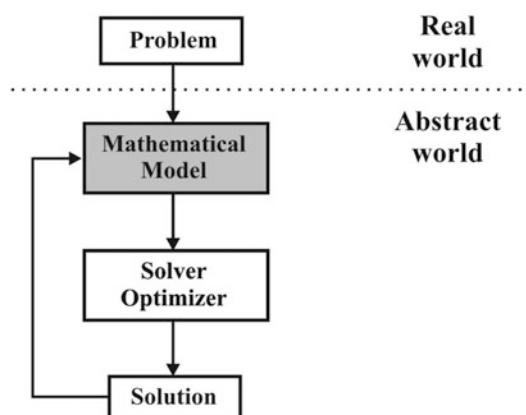
The purpose of computational sciences is to understand the evolution of a real-world system by analyzing and solving the corresponding mathematical models associated to it and by using high performance computers. By making use of mathematical symbols as well as of mathematical accuracy and rigor, mathematical modeling has a central place in computational sciences. Mathematical modeling leads us to the maturation of the domain the mathematical model belongs to, and also to the generation and the development of new mathematical methods and techniques able to solve new problems. The place of the mathematical model within the general scheme for modeling and solving a problem is shown in Fig. 1.1.

Observe that a mathematical model is synthesized from the real world, thus arriving into an abstract world where mathematical rigor is present through mathematical concepts. The model is solved by using mathematical theories irrespective of the physical significance of the elements of the problem. The interpretation of the solution may lead us to some modifications of the model, which again determines our addressing to the real world.

Notice that in the abstract world of the modeler we operate with concepts belonging to the mathematical domain which the modeling process refers to. This is the question of the phenomenology and the conservation laws that characterize the domain of interest. At the same time, certain concepts from the theory of languages, of translators, and of compilers are used in order to develop informatics technologies able to elaborate, update, and maintain the mathematical models.

On the other hand, in the abstract world of algorithms, developers operate with advanced mathematical concepts in order to generate optimization algorithms and to study their convergence and complexity. Additionally, some advanced languages like Fortran, C++, or Matlab are being used for implementing algorithms in computer programs.

**Fig. 1.1** The process of modeling and solving (optimizing) a problem



Therefore, we are faced with the following dilemma. *Which is the better: to consider an approximate (simplified) mathematical model of the problem and then try to get an exact solution of it as much as possible or to use a mathematical model as accurately as possible and then determine an approximate solution of it?* The practical experience recommends that the second alternative gives better results. Indeed, the idea is that a mathematical model which follows to be optimized should be developed by keeping a balance between the aims of improved accuracy in the model which involves added complexity in its formulation and the increased ease of the optimization. This might be achieved by optimizing successive more complicated versions of the model. In this way, the effects of each refinement of the model on the optimization process can be monitored. Thus the difficulty which the optimization algorithms have to face can be discovered much more quickly than if no optimization were applied until the model was complete. In other words, it is better to start with a simplified model and use it in an iterative scheme in which the model is alternatively optimized and refined until an acceptable version of it is obtained both from its mathematical representation of reality and its optimization viewpoints.

---

### 1.3 The Modern Modeling Scheme for Optimization

Formulating and developing a mathematical model of a given reality is one of the finest creative intellectual activities. A short glimpse at the optimization domain shows a substantial progress of the development of algorithms and computer codes to solve real large-scale nonlinear optimization applications. From the application viewpoint, one can see that a large part of the time required for developing, improving, and maintaining an optimization model is spent with data preparation and transformation, as well as with report preparation and documentation. A nonlinear optimization model involves time and intellectual effort to organize the data and write computer programs that transform the data into a form required by the optimization codes. A typical example is the MPS (IBM LP product) representation of a linear programming problem. In nonlinear optimization, the situation is more complex because the user must prepare the functions defining the problem and their derivatives in a form admitted by a particular optimizer. Of course, the user must also transmit to the optimizer the simple bounds on variables, the initial point, and some other parameters associated to a particular optimizer. Nonlinear optimization problems may be transmitted to the optimization codes by using the so-called SIF – Standard Input Format (Bongartz, Conn, Gould, & Toint, 1995). It is worth mentioning that such an approach of using an optimizer is only accessible to the specialist who wrote the optimizer and not to the wide audience of analysts in charge of some projects involving the optimizer. It should be emphasized that the optimizers have very different (and difficult) communication facilities subject to transmitting the functions of the problem and their derivatives, simple bounds on variables, the initial point, some other parameters, optimization conditions, etc.

Currently, the advanced modern industrial usage of optimization algorithms involves the *algebraic oriented languages* that offer advanced capabilities for expressing an optimization model in a form immediately accepted by an optimizer. The most important modeling technologies with optimization facilities are GAMS and AMPL (see Table 1.1 below).

In the last two decades significant research activities have taken place in the area of local and global optimization, including many theoretical, computational, and software contributions. The access to this advanced optimization software needs more and more sophisticated modeling tools. Algebraic oriented optimization modeling languages represent an important class of tools that facilitate the communication of optimization models towards decision-making systems based on the optimization paradigm. In a wider context, the algebraic oriented modeling tools evolve towards

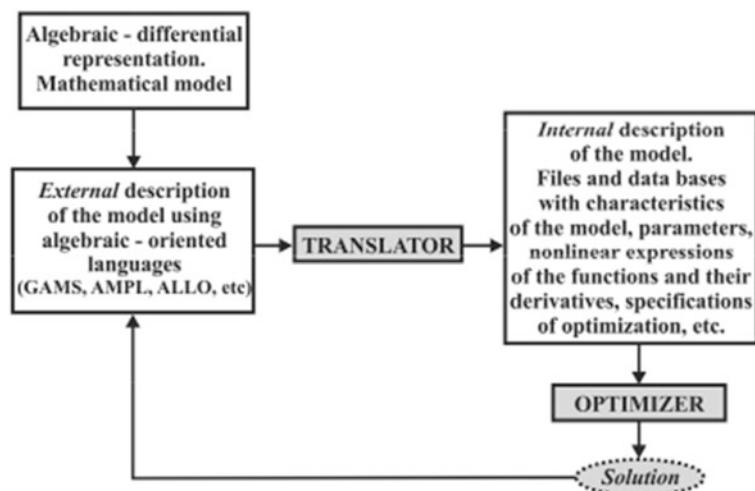
**Table 1.1** Algebraic oriented modeling languages (partial list)

<b>GAMS</b>	<b>General Algebraic Modeling System</b> ( <a href="http://www.gams.com">www.gams.com</a> ) Development Research Center. The World Bank, 1818 H. Street, Washington D.C., USA. Brooke, A., Kendrick, D., Meeraus, A., Raman, R., Rosenthal, R.E., <i>GAMS A user's guide</i> . GAMS Development Corporation, December 1998.
<b>AMPL</b>	<b>A Mathematical Programming Language</b> ( <a href="http://www.ampl.com">www.ampl.com</a> ) Department of Industrial Engineering and Management Sciences. Northwestern University, Evanstone, Illinois 60201, USA. Fourer, R., Gay, M., Kernighan, B.W., <i>AMPL: A modeling language for mathematical programming</i> . Second edition. Duxbury Press/Brooks/Cole Publishing Company, 2002.
<b>ALLO</b>	<b>A Language for Linear Optimization</b> ( <a href="http://camo.ici.ro/projects/allo/allo.htm">http://camo.ici.ro/projects/allo/allo.htm</a> ) Research Institute for Informatics – Bucharest. 8-10 Bdl. Mareșal Alexandru Averescu, sector 1, 011455 Bucharest – Romania. Andrei, N., <i>The ALLO language for linear programming</i> . ICI Technical Report No.1/2004, Bucharest, Romania. Andrei, N., <i>Criticism of the linear programming algorithms reasoning</i> . Academy Publishing House, Bucharest 2011. (Chap. 17, Annex A6)
<b>LPL</b>	<b>A Structured Language for Modeling Linear Programs</b> ( <a href="http://diuflx71.unifr.ch/lpl/mainmodel.html">http://diuflx71.unifr.ch/lpl/mainmodel.html</a> ) Institut of Informatics, University of Fribourg, Regina Mundi, rue de Fauchigny 2, CH-1700 Fribourg, Switzerland. Hürlimann, T., <i>The LPL modeling language: Highlights</i> . University of Fribourg, Switzerland, December 8, 2011.
<b>AIMMS</b>	<b>The Modeling System</b> ( <a href="http://www.aimms.com">www.aimms.com</a> ) Paragon Decision Technology B.V. P.O. Box 3277, 2001 DG Haarlem, The Netherlands Bisschop, J., Roelofs, M., <i>AIMMS – The user's guide</i> . Paragon Decision Technology, 1999.
<b>MPL</b>	<b>Modeling System</b> ( <a href="http://www.maximalsoftware.com">www.maximalsoftware.com</a> ) Maximal Software, Inc., 2111 Wilson Blvd. Suite 700, Arlington, VA, USA. Kristjansson, B., <i>MPL User manual</i> . Maximal Software Inc., Iceland, 1993.
<b>LINDO</b>	<b>Powerful Library of Optimization Solvers and Mathematical Programming Tools</b> ( <a href="http://www.lindo.com">http://www.lindo.com</a> ) Lindo Systems Inc., 1415 North Dayton Street, Chicago, IL, 60642, USA. Schrage, L., <i>Optimization Modeling with LINDO. 5<sup>th</sup> edition</i> . Duxbury Press, 1997.
<b>MOSEL</b>	<b>An Extensible Environment for Modeling and Programming Solutions</b> ( <a href="http://www.fico.com/en/Company">www.fico.com/en/Company</a> ) FICO® Xpress Optimization Suite. 901 Marquette Avenue, Suite 3200, Minneapolis, MN 55402, USA. (formerly DASH) Colombani, Y., Heipcke, S., <i>Mosel: An Overview</i> , Xpress Team, FICO, Leamington Spa CV32 5YN, UK. ( <a href="http://www.fico.com/xpress">http://www.fico.com/xpress</a> )
<b>TOMLAB</b>	<b>TOMLAB Optimization Environment</b> ( <a href="http://tomopt.com/tomlab">tomopt.com/tomlab</a> ) 113 Cherry St Ste 95594. Seattle, WA 98104-2205, USA

fully integrated modeling and optimization management systems with access to databases, spreadsheets, and graphical user interfaces.

As already known, at the fundamentals of every mathematical model lie the *conservation laws* which are active in the domain to which the mathematical model belongs. Therefore, knowing these conservation laws is essential, and their usage leads to models that satisfy the *adequacy to real principle*. However, besides the conservation laws, additional empirical knowledge, principles, different rules, or previous experience need to be considered in order to develop a mathematical model of a given reality. In this chapter, certain aspects concerning the mathematical modeling

**Fig. 1.2** The modern modeling scheme of nonlinear optimization based on algebraic oriented languages



process in the context of mathematical modeling technologies based on the algebraic oriented languages are discussed.

Suppose that we have written the optimization mathematical model, i.e., we have specified the objective function, the constraints, and the simple bounds on variables. The problem now is how to transmit this model to the solver (optimizer) in order to get a solution. Since we consider mathematical models with thousands of variables and constraints, this problem appears to be quite challenging. An elegant and very efficient solution is to use algebraic oriented languages for achieving an external description of the model which can automatically be translated into the internal description of it by means of a translator associated to the language used in this respect. For example, in case of linear programming problems, the internal representation of the model is given by the MPS form. Therefore, the translator associated to an algebraic oriented language generates the MPS form of the model from its external description. In case of nonlinear optimization problems, the mechanism is more complex, but mainly the same.

A deep analysis of the modeling and solving process (see Fig. 1.2) shows that mathematical modeling involves more effort and time for data analysis, verification, and documentation, as well as for updating and improving different variants of the model. The difficulties of the data management for mathematical modeling are coming from the fact that nowadays we do not have a clear methodology for mathematical modeling expressed as an algorithm or as a group of algorithms to be used in a general context. However, understanding the linguistic representation of the process that we need to represent through a mathematical model is crucial.

*The development of a mathematical model is a complex process consisting in a mapping between the objects (together with the relations among them) from the real world and the symbolic and mathematical objects.* This process involves both a very good knowledge of the reality we want to represent in mathematical terms and a multitude of methods and techniques for mathematical modeling. The correctness of a model is usually established after its solving. As a result of intensive computational experiments, it is often necessary to reformulate the model by introducing some new algebraic or differential equations (which were initially ignored) and also the corresponding data. The mathematical modeling process is closely linked to the solving process of the model, supporting each other. These two processes interact in order to build up a mathematical object placed in the perspective of the infinite similarity with reality.

Let us now try to detail the modeling process. At the very beginning, for the process we want to represent into mathematical terms we have a *linguistic description*, actually a text expressed in the natural language which describes the process. It should be pointed out that this linguistic description is a *scientific text*, not a literary one. Obviously, this description includes references to the important conservation laws which are active in the process domain under consideration. As a result of an abstracting creation activity from this linguistic description and by using the conservation laws, we get an *algebraic-differential representation* of the process, a *mathematical model*. This activity mainly consists in identifying the variables and parameters of the process, together with the algebraic and differential relations among these entities by using principles, empirical knowledge, different rules, and some other additional knowledge of the process. Once having this algebraic-differential representation of the process, the next step is to develop an *internal representation* of it, a representation directly admitted by any professional solver, in our case an optimizer. This is done through the *external representation* of the model by making use of the so-called *algebraic oriented mathematical modeling languages*. Therefore, the mathematical model has three forms of representation: an algebraic-differential one with mathematical symbols, an external one by using an algebraic oriented mathematical modeling language (like GAMS, AMPL, ALLO, etc.), and an internal representation which is mainly an informatics description including different files and data bases directly admitted by any solver (optimizer). Figure 1.2 presents the modern modeling scheme for nonlinear optimization based on algebraic oriented languages.

Algebraic modeling languages represent a problem in a purely declarative way. Most of them include some computational facilities to manipulate the data as well as certain control structures. Many optimization models can be well represented declaratively in a very concise way, which also favors the insight of the model. Besides, a mathematical model is stated in a declarative way by using mathematical equalities and inequalities. This gives a clear documentation of the model which can be parsed and interpreted by a compiler or a translator.

The algebraic languages are more declarative rather than imperative. A declarative language emphasizes “*what is being computed*” more than “*how it is being computed*.” Furthermore, the reasons for which we use a declarative presentation of models are *conciseness*, *insight*, and *documentation*. The most important algebraic oriented languages used for solving nonlinear optimization problems are presented in Table 1.1.

As already mentioned, the algebraic oriented modeling languages use essentially declarative statements as opposed to programming languages (Fortran, C++, etc.) in which the use of procedural statements dominates over the use of the declarative ones. For instance, the few procedural statements used in algebraic oriented languages are **read/write data** and **solve** commands. In order to deal with more complicated nonlinear optimization models, these languages include **if-then-else**, **for**, **while** commands. Such commands enable the modeler to write certain solution algorithms directly in the modeling language (Bisschop, & Meeraus, 1982; Fourer, 1983; Hürlimann, 1999; Kallrath, & Wilson, 1997; Conejo, Castillo, Minguez, & García-Bertrand, 2006; Castillo, Conejo, Pedregal, García, & Alguacil, 2001).

---

## 1.4 Classification of Optimization Problems

Plenty of problems of nonlinear optimization are known. The general nonlinear optimization problems can be classified according to the presence or absence of the constraints (unconstrained or constrained), the number of variables (small- or large-scale), the smoothness of the functions defining the problem (differentiable or nondifferentiable), the type of variables (integer or real), the knowledge of functions and parameters (deterministic or stochastic), the type of functions (linear,

quadratic, nonlinear), the convexity of functions (convex or nonconvex), etc. Observe that there are a lot of optimization problems and consequently specialized optimization algorithms and software have been developed. *In this work we will consider only the continuous nonlinear optimization.*

### Linear and Nonlinear Optimization

A general optimization problem may have all the constraints as linear functions, in which case the problem is called *nonlinear optimization with linear constraints*. If additionally the objective function is also linear, then the problem is called *linear programming*. Linear programming is a special domain of optimization with special algorithms: the simplex algorithm (Dantzig, 1963; Vanderbei, 1996; Andrei, 2011d; etc.), interior point algorithms (Karmarkar, 1984), affine scaling interior point algorithms (Dikin, 1967, 1974), the affine scaling in null space (Nazareth, 1987; Kim, 1991; Kim, & Nazareth, 1992, 1994), the primal-dual following the central trajectory (Lustig, Marsten, & Shanno, 1990, 1991), the predictor-corrector following the central trajectory (Lustig, Marsten, & Shanno 1992, 1994), homogeneous, self-dual methods (Mizuno, Todd, & Ye, 1993; Ye, Todd, & Mizuno, 1994), etc.

A special nonlinear optimization is the *quadratic optimization* also known as the *quadratic programming*, in which the objective function is a quadratic function and all the constraints are linear (see Chap. 13). If at least the minimizing function or some of the constraints are nonlinear functions, then the problem is called *nonlinear optimization* or *nonlinear programming*.

### Unconstrained and Constrained Optimization

By *unconstrained optimization* we understand the optimization problems for which  $E = I = \emptyset$  in (1.1). The unconstrained optimization arises in natural situations, but it is often the result of reformulations of the constrained optimization problems in which the constraints are introduced by penalization terms into the minimizing function in order to discourage constraints violations. Plenty of unconstrained optimization algorithms are known: direct methods or the zero-th order methods like the Nelder-Mead method (1965), the steepest descent (Cauchy, 1847), the Newton method (Nocedal, & Wright, 2006; Sun, & Yuan, 2006), the conjugate gradient methods (Hestenes, 1980; Andrei, 2020a), the quasi-Newton and limited-memory quasi-Newton methods (Dennis, & Moré, 1977; Dennis, & Schnabel, 1983; Nocedal, & Wright, 2006; Liu, & Nocedal, 1989), the inexact or truncated Newton methods (Nash, 1984a, 1984b; Schlick, & Fogelson, 1992a, 1992b), the trust-region methods (Nocedal, & Wright, 2006; Conn, Gould, & Toint, 2000), etc.

The *constrained optimization* is an optimization problem in which the variables have to satisfy at least a functional constraint, or simple bounds. A special type of constrained optimization is the so-called *simple bounds optimization*, in which the only constraints are simple bounds on variables. These optimization problems are formulated and solved in a large range of activity domains, particularly in physical sciences and engineering, industry and economics, management, finances, agriculture, forestry, oceanography and environment, biology and medicine applications, atmospheric sciences and weather forecasting, etc. Plenty of papers and books deal with nonlinear optimization methods: (Luenberger, 1973, 1984; Gill, Murray, & Wright, 1981; Vavasis, 1991; Bazaraa, Sherali, & Shetty, 1993; Mangasarian, 1969, 1995; Bertsekas, 1982b, 1999; Nocedal, & Wright, 2006; Sun, & Yuan, 2006; Bartholomew-Biggs, 2008; Griva, Nash, & Sofer, 2009; Andrei, 2015a, 2017c), etc.

### Continuous and Discrete Optimization

By *continuous optimization* we understand those optimization problems in which the variables can take any real value. The feasible set for continuous optimization problems – the class of problems considered in this book – is usually uncountable infinite. In contrast, in *discrete optimization*

problems, the variables are restricted to take only integer values, that is,  $x_i \in \mathbb{Z}$ , where  $\mathbb{Z}$  is the set of integers or binary values which have the form  $x_i \in \{0, 1\}$ . The problems with integer variables are called *integer programming*. If some of the variables in the problem are not restricted to be integer or binary variables, then they are called *mixed integer programming*. The characteristic of discrete optimization problems is that the unknown  $x$  is drawn from a finite, but often very large, set. In other words, the feasible set for the discrete optimization is a finite set. Normally, the continuous optimization problems are easier to solve because the smoothness of the functions defining the problem makes it possible to use the information on the minimizing function and constraints at a particular current point  $x$  to obtain information about these functions at all points in a vicinity of the current point  $x$ . In discrete optimization problems the situation is completely different. The behavior of the minimizing function and of the constraints may change significantly when moving from a feasible point to another one, even if these feasible points are close enough. Discrete optimization problems are not considered in this book. The interested reader may consult the texts by Papadimitriou and Steiglitz (1982), Nemhauser and Wolsey (1988), and Wolsey (1998).

### Global and Local Optimization

Usually, the algorithms for nonlinear optimization seek only a *local solution* that is a point at which the value of the minimizing function is smaller than the values of function at all other nearby points. Some algorithms seek the *global solution*, that is, a point where the value of the minimizing function is smaller than the values of function at all feasible points. Any optimization algorithm needs an *initial point* from where the computations start. Therefore, a local solution is a point near the initial point where the function's value is smaller than its value in all the other points near the initial point, including the initial point. Although there are many practical optimization applications that require global solutions, these are difficult to recognize and even difficult to compute. Global optimization algorithms can be found in Pardalos and Rosen (1987), Floudas and Pardalos (1992), Horst, Pardalos, and Thoai (2000), etc. On the other hand, the vast majority of books and papers on optimization describe algorithms for determining local solutions.

There is a fundamental result in optimization: for linear programming and for convex programming problems local solutions are also global solutions. Observe that general unconstrained or constrained nonlinear optimization problems may have local solutions that are not global.

### Convex Programming

The concept of *convexity* is crucial in optimization, and many practical optimization problems have this property. The optimization problems with this property always have a very nice theory behind them and are easier to solve. The term “convex” can be associated both with the sets and the functions. A set  $C \subset \mathbb{R}^n$  is a *convex set* if a straight line segment connecting any two points in  $C$  lies entirely inside  $C$ . In other words,  $C$  is a convex set if for any two points  $x, y \in C$ , it follows that  $\alpha x + (1 - \alpha)y \in C$ , for all  $\alpha \in [0, 1]$ . A function  $f: C \rightarrow \mathbb{R}$ , where  $C \subset \mathbb{R}^n$ , is a *convex function* if its domain  $C$  is a convex set and for any two points  $x, y \in C$ , the following property holds:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \quad \text{for all } \alpha \in [0, 1]. \quad (1.2)$$

Examples of convex sets include polyhedrons defined by a set of linear equalities and inequalities, that is:  $\{x \in \mathbb{R}^n : Ax = b, Dx \leq d\}$ , where  $A$  and  $D$  are matrices of appropriate dimensions and  $b$  and  $d$  are vectors, is a convex set. The unit ball  $\{x \in \mathbb{R}^n : \|x\|_2 \leq 1\}$  is a convex set. Examples of convex functions are the linear functions  $f(x) = c^T x + d$ , for any constant vector  $c$  and scalar  $d$ . Also the quadratic functions  $f(x) = x^T H x$ , where  $H$  is a symmetric positive semidefinite matrix, are a convex function.

A function  $f$  is a *strictly convex function* if the inequality (1.2) is strict whenever  $x \neq y$  and  $\alpha \in (0, 1)$ . A function  $f$  is *concave* if  $-f$  is convex.

*Convex programming* – a branch of nonlinear optimization – defines a special case of the general constrained optimization (1.1) in which the objective function is a convex function, the equality constraints  $e_i(x) = 0, i \in E$ , are all linear functions, and the inequality constraints functions  $c_j(x) \geq 0, j \in I$ , are all concave.

As it has been mentioned, if the objective function of the problem (1.1) and the feasible domain defined by the constraints of (1.1) are both convex, then any local solution of the problem is also a global solution.

A classical, comprehensive text on convex analysis is Rockafellar's book (1970).

### Deterministic and Stochastic Optimization

*Deterministic optimization* is characterized by the fact that the mathematical model describing the problem depends on some parameters that are known with a given degree of accuracy. In other words, the mathematical description of the problem is completely known as regards both the structure of the problem and to the values of parameters defining the problem. In contrast, *stochastic optimization* refers to those optimization problems with uncertain data in the model, problems that have an incomplete description concerning both their structure and the values of their parameters. Stochastic optimization methods are optimization methods that generate and use random variables. For stochastic optimization problems, the random variables appear in the formulation of the problem itself, which involves random objective functions or random constraints. The stochastic optimization is described by Birge and Louveaux (1997), Kall and Wallace (1994), Spall (2003), etc.

Methods of optimization with uncertain data in the model of the optimization problem are *chance-constrained optimization* and *robust optimization*. The chance-constrained optimization is a formulation of an optimization problem that ensures that the variable  $x$  satisfies the constrained of the problem with a probability above of certain level. Classical applications of the chance-constrained optimization include water reservoir management, financial risk management, unmanned autonomous vehicle navigation (optimal navigation and reliable obstacle avoidance), as well as optimal renewable energy generation. By robust optimization we understand that certain constraints are required to be satisfied for all the possible values of the uncertain data in the model of the optimization problem (Ben-Tal, El Ghaoui, & Nemirovski, 2009).

### Small-Scale Versus Large-Scale Optimization

A measure of complexity and of course of difficulty of a nonlinear optimization problem is its size, defined in terms of the number of variables and/or the number of constraints. Roughly speaking, we distinguish two classes of problems: *small-scale optimization problems* with 5–10 unknowns and constraints and *large-scale optimization problems* with hundreds or thousands or even millions of variables and constraints. The theoretical developments of the nonlinear optimization, mainly the necessary and sufficient conditions satisfied by a solution point including the Karush-Kuhn-Tucker (KKT) conditions and their extensions, are irrespective of the size of the problem. However, when the optimization algorithms turn to be implemented in computing programs, this classification as small-scale or large-scale problems is crucial. Two aspects have to be taken into account for solving large-scale optimization problems: structure and sparsity. For example, the unconstrained optimization problems may be presented in generalized or extended forms given by the structure of their Hessian matrix. The Hessian of the minimizing functions in generalized form has a block-diagonal structure, while the Hessian of the minimizing functions in extended form has a band structure. Efficient optimization algorithms for solving constrained optimization problems take into account the structure

and the sparsity of the KKT conditions. It is well known that an important component of the vast majority of optimization algorithms is to solve an algebraic system of linear equations. Advanced techniques for solving linear systems are based on sparse matrix techniques.

## 1.5 Optimization Algorithms

By optimization we understand the design and analysis of algorithms (convergence and complexity) for solving optimization problems and applications. All the optimization algorithms are iterative. For solving an optimization problem, they start the computation from an initial point and generate a hopefully convergent sequence of improved estimates of the solution until a specified termination criterion has been satisfied. The optimization algorithms are differentiated by the strategy used to move from a given iteration to the next one. These strategies use the values of the objective function and of constraints and possibly the values of the first and of the second derivatives of these functions. Some algorithms are based only on the local information of the functions at the current point, while others accumulate information gathered from the previous iterations.

Good optimization algorithms should fulfill the following requirements: *efficiency* (they should not require excessive computing time and storage), *robustness* (they should solve a wide variety of optimization problems from different starting points), and *accuracy* (they should be able to generate a solution with a specified precision without being sensitive to errors in data or to arithmetic rounding errors). The most efficient and robust nonlinear optimization algorithms use combinations of some techniques like the active-set methods, or the interior point methods with globalization strategies and refining mechanisms, all of them in a frame of the advanced computational linear algebra.

The theory of iterative algorithms involves three aspects. The first one is the *creation* of the algorithms themselves. The second aspect is the so-called *global convergence analysis* that addresses the question of whether the algorithm, when initialized far away from the solution point, will eventually converge to it. The third aspect is the *local convergence analysis*, which refers to the *rate* at which the generated sequence of points  $\{x_k\}$  converges to the solution. Two aspects of the convergence-rate theory are important. The first is known as *complexity analysis*, which focuses on how fast the algorithms converge to the solution. They differentiate themselves as polynomial-time algorithms and non-polynomial-time algorithms. The second aspect includes a detailed analysis of *how fast the algorithms converge in their final stages*.

Advanced optimization algorithms include some ingredients which make them reliable and increase their numerical performances. The most important ingredients are as follows: automatic differentiation (Griewank, 2000), finite-difference derivative estimates (Gill, Murray, & Wright, 1981), inertia correcting and singularity (Fletcher, 1987), sparse technology for solving linear algebraic system of equations, quasi-Newton approximation of Hessian, stepsize computation by line-search, tests for stopping the iterations, etc.

## 1.6 Collections of Applications for Numerical Experiments

In order to see the numerical performances of the nonlinear optimization algorithm described in this book, a number of four collections of unconstrained and constrained optimization test problems and applications are used. The unconstrained optimization algorithms are tested on the collection *SMUNO* which includes a number of 16 applications with the maximum number of 10 variables (see Appendix B). Also, the collection *UOP* that includes 80 large-scale unconstrained optimization

problems in generalized or extended form, presented in Andrei (2013e, 2020a), is used to see the performances of unconstrained optimization algorithms. For each test function from this collection, we have considered 10 numerical experiments with the number of variables increasing as  $n = 1000, 2000, \dots, 10000$ . Therefore, a set of 800 unconstrained optimization problems are solved in our numerical experiments with unconstrained optimization algorithms. The constrained optimization algorithms use the collection *LACOP*. This collection includes a number of 18 real constrained nonlinear optimization applications (see Appendix C). Additionally, in Appendix D, we presented six unconstrained optimization applications from the collection *MINPACK-2* (Averick, Carter, & Moré, 1991; Averick, Carter, Moré, & Xue, 1992), each of them with 40,000 variables, or 250,000 variables, which are used in our numerical experiments.

## 1.7 Comparison of Algorithms

Plenty of algorithms for unconstrained and constrained optimization are shown in this book. Some of them accumulate information gathered at previous iterations, while others use only the local information available at the current iteration. As it has already been mentioned, good algorithms must have the following three properties: *efficiency*, *robustness*, and *accuracy*. To be reliable and to have a guarantee that they find a solution, some properties and their convergence results have to be discussed. However, this is not enough. The irony is that in optimization, algorithms for which there are very well-established theoretical properties (descent, convergence, and complexity) prove to have modest numerical performances. Having in view that the final test of a theory is its capacity to solve the problems and the applications which originated it, our efforts have been directed to see the numerical performances of the algorithms for solving the optimization problems and applications from the abovementioned collections. In this book the algorithms have been compared in the following environment.

**Remark 1.1** All algorithms have been coded in double precision Fortran, compiled with f77 (default compiler settings) and run on an Intel Pentium 4, 1.8 GHz workstation. For each problem from the UOP collection, 10 numerical experiments with an increasing number of variables as  $n = 1000, 2000, \dots, 10000$  have been performed. Hence, 800 problems have been solved in this set of numerical experiments. Some optimization algorithms were tested for solving the applications from the MINPACK-2 collection with 40,000 or 250,000 variables.

The algorithms compared in these numerical experiments find local solutions. Therefore, the comparisons of the algorithms are given in the following context. Let  $f_i^{ALG1}$  and  $f_i^{ALG2}$  be the optimal value found by ALG1 and ALG2 for problem  $i = 1, \dots, 800$ , respectively. We say that, in the particular problem  $i$ , the performance of ALG1 was better than the performance of ALG2 if:

$$|f_i^{ALG1} - f_i^{ALG2}| < 10^{-3} \quad (1.3)$$

and if the number of iterations (#iter), or the number of function-gradient evaluations (#fg), or *cpu* the CPU time of ALG1 was less than the number of iterations, or the number of function-gradient evaluations, or the CPU time corresponding to ALG2, respectively.

The iterations are stopped if the inequality  $\|g_k\|_\infty \leq 10^{-6}$  is satisfied, where  $\|\cdot\|_\infty$  is the maximum absolute component of a vector. The maximum number of iterations was limited to 2000.

To compare the performances of algorithms, the Dolan and Moré (2002) performance profiles are used. For  $n_p$  problems and  $n_s$  solvers, the performance profile  $P : \mathbb{R} \rightarrow [0, 1]$  is defined as follows. Let  $P$  and  $S$  be the set of problems and the set of solvers, respectively. For each problem  $p \in P$  and for

each solver  $s \in S$ , define  $t_{p,s}$  = computing time (similarly for the number of iterations or the number of function and its gradient evaluations) required to solve problem  $p$  by solver  $s$ . The idea is to compare the performance of solver  $s$  on problem  $p$  with the best performance by any solver on this problem. Therefore, the *performance ratio* is defined by:

$$r_{p,s} = \frac{t_{p,s}}{\min_{s \in S} \{t_{p,s}\}}. \quad (1.4)$$

With this, the *performance profile* may be defined by:

$$P_s(\tau) = \frac{1}{n_p} \text{size}\{p \in P : r_{p,s} \leq \tau\}, \quad (1.5)$$

for  $\tau > 0$ , where for any set  $A$ ,  $\text{size}(A)$  stands for the number of elements in  $A$ . The performance profile  $P_s : \mathbb{R} \rightarrow [0, 1]$  for a solver  $s$  is a nondecreasing, piecewise constant function, continuous from the right at each breakpoint.  $P_s(\tau)$  is the probability for solver  $s \in S$  so that the performance ratio  $r_{p,s}$  is within a factor  $\tau > 0$  of the best possible ratio. The function  $P_s$  is the cumulative distribution function for the performance ratio. Observe that  $1 - P_s(\tau)$  is the fraction of problems that the solver cannot solve within a factor  $\tau$  of the best solver. In our numerical experiments, in each figure, we used  $n_p = 800$  and  $\tau = 16$ .

The performance profile plot of solvers may be computed as follows. Suppose that two solvers ( $n_s = 2$ ) have to be compared subject to a given metric (which can be the number of iterations, the number of function and its gradient evaluations or the CPU computing time) for solving  $n_p$  problems. Consider an integer value for parameter  $\tau$ . Firstly, out of  $n_p$  problems, only the problems for which the criterion (1.3) is satisfied are retained. Let  $\bar{n}_p$  be the number of problems satisfying the criterion (1.3). For each solver and for each problem, compute the performance ratio  $r_{p,s}$ , for  $p = 1, \dots, \bar{n}_p$  and  $s = 1, 2$ , corresponding to the metric selected. For each solver  $s = 1, 2$  and for  $i = 1, \dots, \tau$ , compute the performance profile  $P_s(\tau)$ .

The percentage of problems for which an algorithm is the best is given on the left side of the plot. On the other hand, the right side of the plot gives the percentage of the problems that are successfully solved. In other words, for a given algorithm, the plot for  $\tau = 1$ , represents the fraction of problems for which the algorithm was the most efficient over all algorithms. The plot for  $\tau = \infty$  represents the fraction of problems solved by the algorithm irrespective of the required effort. Therefore, the plot for  $\tau = 1$  is associated to the *efficiency* of the algorithm, while the plot for  $\tau = \infty$  is associated to the *robustness* of the algorithm. ♦

---

## 1.8 The Structure of the Book

This book has two parts. The first one includes Chaps. 2, 3, 4, 5, 6, 7, 8, and 9 that cover the theory and practice of the unconstrained optimization. The second part contains Chaps. 12, 13, 14, 15, 16, 17, 18, 19, and 20 that are dedicated to the constrained optimization. Chapters 10 and 11 present an overview on the constrained nonlinear optimization methods and the optimality conditions for nonlinear optimization, respectively. Besides, the book has four appendices with a mathematical review containing the main mathematical concepts and results used along the chapters, as well as three collections with unconstrained and constrained applications on nonlinear optimization with which the optimization algorithms presented in this book were tested.

We emphasize that the book presents the theoretical developments and the computational performances of the *modern unconstrained and constrained optimization algorithms* for solving

large-scale optimization problems and applications. Some other algorithms have not been considered in this book: the conic model and colinear scaling (Davidon, 1980), tensor methods for solving system of nonlinear equations (Schnabel, & Frank, 1984) and for unconstrained optimization (Schnabel, & Chow, 1991), gradient flow methods for unconstrained optimization (Brown, & Bartholomew-Biggs, 1987, 1989), methods based on KKT optimality conditions (Abadie, & Guerrero, 1984), Frank-Wolfe linearization (Frank, & Wolfe, 1956), successive linear programming (Griffith, & Stewart, 1961), the convex simplex (Zangwill, 1967), feasible direction methods (Zoutendijk, 1960), the gradient projection method (Rosen, 1960, 1961), the reduced gradient method (Wolfe, 1967), the generalized reduced gradient (Abadie, 1978, 1979), generalized elimination methods, ellipsoid methods for constrained optimization (Ecker, & Kupferschmid, 1983, 1985), etc.

Every chapter includes the description of a *particular method* for continuous nonlinear optimization, the most important theoretical results on convergence, comparisons with other methods, comments on the linear algebra used in implementations in computer programs, as well as the numerical performances of the *representative algorithms* for solving optimization problems and applications from appendices. The algorithms are tested for solving large-scale optimization problems from the UOP collection up to 10,000 variables, as well as optimization applications from the LACOP and MINPACK-2 collections with 40,000 or 250,000 variables. Finally, every chapter terminates with Notes and References where some details, references, perspectives, and historical comments are given.

As we know, for minimizing a sufficiently smooth function, at every iteration  $x_k$  an optimization algorithm based on the *line-search strategy* computes the *search direction*  $d_k$  in the current point  $x_k$ , which must be a descent one, and a *stepsize*  $\alpha_k$  taken along the search direction to obtain a new estimation of the minimum point,  $x_{k+1} = x_k + \alpha_k d_k$ . The purpose of Chap. 2 is to present the most important methods as well as their convergence properties for computing the stepsize  $\alpha_k$ , a crucial aspect in any unconstrained optimization algorithm. Chapter 3 presents the *steepest descent algorithm* for unconstrained optimization as well as its *relaxed* and *accelerated* variants. In Chap. 4 the *Newton method* is detailed for solving both the nonlinear algebraic system of equations and the function minimization. It is proved that if well initialized, the Newton method is quadratic convergent to a minimum point of the minimizing function. Some modifications of the Newton method and the *composite Newton method* are also shown. Chapter 5 is dedicated to the *conjugate gradient methods* for unconstrained optimization. These methods are very well suited for solving large-scale unconstrained optimization problems and applications. The linear conjugate gradient methods are first discussed, followed by the nonlinear conjugate gradient methods for unconstrained optimization, insisting on standard, hybrid, modifications of the standard conjugate gradient schemes, and memoryless BFGS preconditioned. The chapter illustrates the performances of the modern conjugate gradient algorithms CG-DESCENT (Hager, & Zhang, 2005, 2006a), DESCN (Andrei, 2013c), and CGOPT/DK (Dai, & Kou, 2013) for solving five applications from MINPACK2 collection, each of them up to 250,000 variables. In Chap. 6 the *quasi-Newton methods* DFP, BFGS, and SR1 as well as the *limited memory BFGS method* (L-BFGS) are discussed. Some modifications of the BFGS method with one or two parameters are also presented. The new quasi-Newton method *memoryless SR1 with generalized secant equation* is introduced and tested for solving five applications from the MINPACK2 collection of applications, each of them up to 250,000 variables. Finally, the performances of L-BFGS ( $m = 5$ ) for solving five applications from the MINPACK-2 collection, each of them with 250,000 variables, illustrate that L-BFGS ( $m = 5$ ) is faster versus the modern conjugate gradient algorithms CG-DESCENT, DESCN, and DK. The *inexact or truncated Newton method* (TN) (Nash, 1984b, 1985) is considered in Chap. 7. Intensive numerical experiments show that subject to the number of iterations and to the function calls TN is better than L-BFGS, but subject to the CPU computing time metric, both L-BFGS and TN have similar performances, L-BFGS being

slightly faster. Comparing the performances of TN subject to the CPU computing time versus the modern conjugate gradient algorithms CG-DESCENT and DESCON, we see that the conjugate gradient algorithms are top performers. Chapter 8 introduces another paradigm for solving unconstrained optimization problems – *the trust-region method*. The line-search methods presented in the previous chapters generate a descent search direction  $d$  and then determine a suitable stepsize  $\alpha$  along this direction, hoping that the function values will reduce. On the other hand, the trust-region methods define a *region* around the current iterate within which they *trust* the quadratic model to be an adequate representation of the minimizing function and then choose the step to be the approximate minimizer of the model in this region. Therefore, trust-region methods choose the direction and the stepsize simultaneously. Of course, if a step is not acceptable, they reduce the size of the region and find a new minimizer. For solving the applications from the MINPACK2 collection with 40,000 variables, we used the trust-region method implemented in TRON (Lin, & Moré, 1999) without simple bounds. For solving these five applications, when comparing the performances of TRON versus TN, we see that TN is top performer. In Chap. 9 the *direct methods* known as the *zero-th order* methods for unconstrained optimization are described. These methods consider the case in which the derivatives of the minimizing function are unavailable, impractical to obtain, or unreliable. From the multitude of derivative-free optimization algorithms, we selected in this chapter only the algorithm of Nelder-Mead (NELMED) (Nelder, & Mead, 1965), algorithm of Powell (NEWUOA) (Powell, 2003, 2004, 2006), and of Andrei (DEEPS) (Andrei, 2021a). The numerical experiments with these algorithms for solving 16 real unconstrained optimization applications, included in the collection SMUNO from Appendix B, show that NEWUOA is way more efficient than NELMED and than DEEPS. In Chap. 10 an overview on constrained nonlinear optimization methods is given. Some aspects concerning convergence tests, infeasible points, approximate sub-problems – local models and their solving, globalization strategy – convergence from remote starting points, and refining the local model are also discussed. It is shown that the most efficient and robust nonlinear optimization algorithms use combinations of the active-set methods (local models based on sequential linear or quadratic programming), or interior point methods with globalization strategies (augmented Lagrangian, penalty and merit function, or filters methods) and refining mechanisms (line-search or trust-region methods), all of them in a frame where advanced computational linear algebra techniques are used. Chapter 11 is dedicated to the presentation of the optimality conditions for nonlinear optimization. The key to understanding the nonlinear optimization are the Karush-Kuhn-Tucker (KKT) optimality conditions. This is a major result which identifies an algebraic system of equalities and inequalities that corresponds to the solution of any nonlinear optimization problem. For problems with inequality constraints, the KKT approach generalizes the method of Lagrange multipliers, which allows only equality constraints. For the development of the KKT optimality conditions, three possible approaches can be used. One is based on the *separation and support theorems* from the convex set theory. Another one uses the *penalty functions* while the third one comes from the *theory of Lagrange multipliers*. Each of these approaches has its own virtues and provides its own insights into the KKT Theorem. In this book we consider the optimality conditions for the continuous nonlinear optimization (the mathematical programming), using the formalism of Lagrange. The simple bounded optimization is the subject of Chap. 12. From the multitude of algorithms dedicated to this problem, only the following are presented: the *spectral projected gradient method* (SPG) by Birgin, Martínez, and Raydan (2000, 2001); the *limited memory BFGS algorithm with gradient projection* (L-BFGS-B) by Byrd, Lu, Nocedal, and Zhu (1994, 1995); and the *truncated Newton with simple bounds* (TNBC) by Nash (1984a, 1984b, 1985). Intensive numerical experiments showed that L-BFGS-B ( $m = 5$ ) is more robust than SPG and than TNBC. Both SPG and L-BFGS-B are able to solve large-scale simple bound optimization problems, L-BFGS-B being slightly faster. TNBC is very sensitive to the interval in which the bounds on variable are defined. In Chap. 13, one of the most

important nonlinear optimization problems is discussed, namely, the *quadratic programming*, in which a quadratic objective function is minimized with respect to linear equality and inequality constraints. Both the equality constrained and inequality constrained quadratic programming are presented. For these problems, the interior point and the elimination of constraints are also discussed. For the equality constrained quadratic programming, the following methods are presented: factorization of the full KKT system, the Schur-complement method, the null-space method, the reduced gradient for linear equality constraints, the preconditioned conjugate gradient for reduced systems, and the projected conjugate gradient method. On the other hand, for the inequality constrained quadratic programming, the primal active-set method, the active-set method with positive definite Hessian, the reduced-gradient for inequality constraints, the reduced-gradient for simple bounds, and the primal-dual active-set method are detailed. Penalty and augmented Lagrangian methods are the subject of Chap. 14. The idea of these methods is to replace the original problem by a sequence of sub-problems in which the constraints are expressed by terms added to the objective function. The penalty concept is implemented in two different methods: the *quadratic penalty method* and the *nonsmooth exact penalty method*. The *augmented Lagrangian method* or the *multipliers method* explicitly uses Lagrange multiplier estimates in order to avoid the ill-conditioning of the quadratic penalty method. The best known methods for solving nonlinear optimization problems combine the penalty concept with the augmented Lagrangian in a penalty-barrier with quadratic approximation of the inequality constraints (SPENBAR), (Andrei, 1996a, b, c, 1998a, 2015a, 2017c), or the minimization of a modified augmented Lagrangian subject to linearized constraints (MINOS), (Murtagh, & Saunders, 1978, 1980, 1982, 1987, 1995), or the minimization of the augmented Lagrangian subject to simple bounds on variables (LANCELOT), (Conn, Gould, & Toint, 1992b). In this chapter, only the theory behind the SPENBAR and MINOS algorithms together with some numerical results for solving some applications from the LACOP collection are presented. *Sequential Quadratic Programming* (SQP) methods are very effective for solving optimization problems with significant nonlinearities in constraints. These methods are described in Chap. 15. They are active-set methods and generate steps by solving quadratic programming sub-problems at every iteration. These methods are used both in line-search and trust-region paradigm. Three implementations of these methods are discussed: a SQP algorithm for large-scale constrained optimization (SNOPT) (Gill, Murray, & Saunders 2002, 2005), a SQP algorithm with successive error restoration (NLPQLP) (Schittkowski, 1986, 2002, 2005, 2009, 2010), and active-set sequential linear-quadratic programming (KNITRO/ACTIVE) (Byrd, Gould, Nocedal, & Waltz, 2004a). The performances of all these methods are illustrated for solving large-scale problems from the LACOP collection. A very interesting and profitable combination of the generalized reduced gradient with the sequential linear-programming and with the sequential quadratic-programming, namely, CONOPT (Drud, 2011), is presented in Chap. 16. Comparisons of CONOPT with SQP versus MINOS, KNITRO/ACTIVE, and versus SNOPT for solving the applications from the LACOP collection subject to the total CPU computing time show that CONOPT is top performer. One of the most powerful methods for solving nonlinear optimization problems known as *interior point methods* is presented in Chap. 17. In this chapter we consider the interior point method for general nonlinear optimization problems in line-search and trust-region paradigms, respectively. Some aspects concerning the algorithmic developments for solving the primal-dual system focus on direct and iterative methods. Both the line-search interior point algorithm and the trust-region interior point algorithm illustrate the interior point algorithms. In the last part of this chapter the KNITRO/INTERIOR algorithm is being described, together with its numerical performances for solving large-scale general continuously nonlinear optimization problems. KNITRO/INTERIOR provides two procedures for computing the steps within the interior point approach. In the version INTERIOR-CG, each step is computed by using a projected conjugate gradient iteration. It factors a projection matrix and uses the conjugate gradient method to

approximately minimize a quadratic model of the barrier problem (Byrd, Hribar, & Nocedal, 1999). In the version INTERIOR-DIRECT, the algorithm attempts to compute a new iterate by solving the primal-dual KKT system using direct linear algebra (Waltz, Morales, Nocedal, & Orban, 2003). Four variants of KNITRO were tested for solving the applications from the LACOP collection. It seems that KNITRO which automatically chooses the algorithm based on the problem characteristics is the best. The filter methods developed by Fletcher and Leyffer (2002) as a new technique for the globalization of the nonlinear optimization algorithms are described in Chap. 18. The filter methods can use *sequential linear programming* or *sequential quadratic programming* in the context of trust-region methods. Chapter 19 presents an implementation of an interior point filter line-search algorithm called IPOPT, for large-scale nonlinear programming, proposed by Wächter and Biegler (2005a, 2005b). The idea of this algorithm is to combine the primal-dual interior point algorithms with filter line-search. For solving 15 large-scale applications from the LACOP collection, IPOPT (Table 19.2) needs 42.17 seconds. However, KNITRO (option 0) (Table 17.9) needs 17.81 seconds. Chapter 20 includes two direct methods for solving constrained nonlinear optimization problems. The first one is COBYLA by Powell (1993), while the second one is DFL by Liuzzi, Lucidi, and Sciandrone (2010). At every iteration of COBYLA, the objective function and constraints are linearly approximated by interpolation in vertices of a simplex structure. The corresponding linear programming problem completed with a constraint of trust-region type is solved, thus obtaining a new approximation of the solution of the original problem. DFL is an extension of the successive quadratic penalty method which does not involve the derivative information. The idea of this method is to approximately minimize a sequence of merit functions in which the penalization of the violation of the constraints is progressively enlarged.

Our strategy for presenting the state-of-the-art of optimization is to describe the most important modern methods for the unconstrained optimization (the steepest descent, the Newton method, the conjugate gradient, the quasi-Newton methods, the inexact Newton and the trust-region methods) and also for the constrained optimization (the quadratic programming, the penalty and the augmented Lagrangian methods, the sequential quadratic programming, the interior point, the filter methods) and to illustrate the computational performances of the most representative algorithms corresponding to the considered methods. Besides, the linear algebra procedures and techniques implementing the optimization algorithms are highly emphasized. As a final remark, the book deals with the basic aspects of the optimization theory and with the numerical performances of the optimization algorithms for solving large-scale optimization problems and real-applications.

The conclusion of the theoretical developments and of the intensive numerical studies is that there is no algorithm able to solve any nonlinear unconstrained or constrained optimization problem or application. The most modern, powerful nonlinear optimization algorithms presented and tested in this book combine and integrate different optimization techniques (the modified augmented Lagrangian, the sequential linear or quadratic programming, the interior point methods with filter line-search or trust-region) and include advanced computational linear algebra procedures.

## Notes and References

Continuous nonlinear optimization is practically present in any area of activity and is essential in running engineering complex structures and complex economy aggregates. Optimization traces its roots to calculus of variations and the work of Euler (1707–1783) and Lagrange (1736–1813). Optimization is often called *mathematical programming*, a somewhat confusing term introduced by Tjalling Koopmans (1910–1985) and consolidated *inter alia* by Leonid Kantorovich (1912–1986), George Dantzig (1914–2005), and by some others (see Lenstra, Rinnooy Kan, and Schrijver (1991),

Grötschel (2012)). The mathematical theory of optimization is found in many books: Luenberger (1973, 1984), Bazaraa, Sherali, and Shetty (1993), Bertsekas (1999), Boyd and Vandenberghe (2004), Nocedal and Wright (2006), Sun and Yuan (2006), Bartholomew-Biggs (2008), Luenberger and Ye (2016), and Andrei (2013e, 2015a), to cite only some of them. Now, the professional optimization algorithms work under the assistance of certain modern systems of modeling and optimization like GAMS, AMPL, TOMLAB, ALLO, etc.



# Fundamentals on Unconstrained Optimization. Stepsize Computation

2

Unconstrained optimization consists of minimizing a function which depends on a number of real variables without any restrictions on their values. When the number of variables is large, this problem becomes quite challenging. The methods for the unconstrained optimization are iterative. They start with an initial guess of the variables and generate a sequence of improved estimates of the minimum point until they terminate with a set of values for variables. At every iteration  $x_k$  an optimization algorithm computes the search direction  $d_k$  in the current point, which must be a descent one, and a stepsize  $\alpha_k$  taken along the search direction to obtain a new estimation of the minimum point,  $x_{k+1} = x_k + \alpha_k d_k$ . *The purpose of this chapter is to present the most important modern methods as well as their convergence properties for computing the stepsize  $\alpha_k$ , which is a crucial component of any unconstrained optimization algorithm.* For checking if this set of values of variables is indeed the solution of the problem, the optimality conditions should be used. If the optimality conditions are not satisfied, they may be used to improve the current estimate of the solution. The algorithms described in this book utilize the values of the minimizing function, of the first and possibly of the second derivatives of this function. Other methods based only on function's values (Golden-section search, Fibonacci, compass search, dichotomous search, quadratic interpolation – see (Antoniou, & Lu, 2007) are not considered in this book. We emphasize and recommend that the reader study the mathematical concepts and results included in Appendix A.

## 2.1 The Problem

In this book, the following unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad (2.1)$$

is considered, where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a real valued function  $f$  of  $n$  variables, smooth enough on  $\mathbb{R}^n$ . The interest is to find a *local minimizer* of this function, that is a point  $x^*$ , so that

$$f(x^*) \leq f(x) \text{ for all } x \text{ near } x^*. \quad (2.2)$$

If  $f(x^*) < f(x)$  for all  $x$  near  $x^*$ , then  $x^*$  is called *strict local minimizer*, or simple *minimizer* of function  $f$ . Often,  $f$  is referred to as the *objective function*, while  $f(x^*)$  as the *minimum* or the *minimum value*.

We emphasize that the local minimization problem is different from the *global minimization problem*, where a global minimizer, i.e., a point  $x^*$  so that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathbb{R}^n \quad (2.3)$$

is sought. This book deals with only the local minimization problems.

The function  $f$  in (2.1) may have any algebraic expression, and we suppose that it is twice continuously differentiable on  $\mathbb{R}^n$ . Let us introduce  $\nabla f(x)$  as the gradient of  $f$  and  $\nabla^2 f(x)$  its Hessian. Often, the gradient is denoted as  $g(x) = \nabla f(x)$ .

$$g(x) \equiv \nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}, \quad \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}.$$

For solving (2.1), plenty of methods are known (see Luenberger, 1973, 1984; Gill, Murray, & Wright, 1981; Bazaraa, Sherali, & Shetty, 1993; Bertsekas, 1999; Nocedal, & Wright, 2006; Sun, & Yuan, 2006; Bartholomew-Biggs, 2008; Griva, Nash, & Sofer, 2009; Andrei, 1999a, 2009e, 2015a, 2020a, 2021a). In general, for solving (2.1), the unconstrained optimization methods implement one of the following two strategies: the *line-search* and the *trust-region*. Besides these two strategies, for problems with a small number of variables (let us say 20), the *direct methods* are used. The direct methods use only the values of the minimizing function without appealing to the differential operators: the gradient or the Hessian. Often, the direct methods are called *derivative-free optimization* or the *zero-th order* method. There are a lot of direct methods (Rios, & Shainidis, 2013; Larson, Menickelly, Wild, 2019; Andrei, 2021a), etc.

In the *line-search strategy*, the corresponding algorithm chooses a direction  $d_k$  and searches along this direction from the current iterate  $x_k$  for a new iterate with a lower function value. Specifically, starting with an initial point  $x_0$ , the iterations are generated as

$$x_{k+1} = x_k + \alpha_k d_k, \quad k = 0, 1, \dots, \quad (2.4)$$

where  $d_k \in \mathbb{R}^n$  is the *search direction* along which the values of function  $f$  are reduced and  $\alpha_k \in \mathbb{R}$  is the *stepsize* determined by a line-search procedure. The main requirement is that the search direction  $d_k$ , at iteration  $k$  should be a *descent direction*. It is proved that the algebraic characterization of descent directions is that

$$d_k^T g_k < 0, \quad (2.5)$$

which is a very important criterion concerning the effectiveness of an algorithm. In (2.5),  $g_k = \nabla f(x_k)$  is the gradient of  $f$  in point  $x_k$ . In order to guarantee the global convergence, sometimes it is required that the search direction  $d_k$  satisfy the *sufficient descent* condition

$$g_k^T d_k \leq -c \|g_k\|^2, \quad (2.6)$$

where  $c$  is a positive constant.

In the *trust-region strategy*, the idea is to use the gathered information about the minimizing function  $f$  to construct a model function  $m_k$  whose behavior near the current point  $x_k$  is similar to that of the actual objective function  $f$ . In other words, the step  $p$  is determined by approximately solving the following subproblem:

$$\min_p m_k(x_k + p), \quad (2.7)$$

where the point  $x_k + p$  lies inside *the trust region*. If the step  $p$  does not produce a sufficient reduction of the function values, then it follows that the trust-region is too large. In this case the trust-region is shrunked and the model  $m_k$  in (2.7) is re-solved. Usually, the trust-region is a ball defined by  $\|p\|_2 \leq \Delta$ , where the scalar  $\Delta$  is known as the *trust-region radius*. Of course, elliptical and box-shaped trust regions may be used.

Usually, the model  $m_k$  in (2.7) is defined as a quadratic approximation of the minimizing function  $f$ :

$$m_k(x_k + p) = f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T B_k p, \quad (2.8)$$

where  $B_k$  is either the Hessian  $\nabla^2 f(x_k)$  or an approximation of it. Observe that whenever the size of the trust-region, i.e., the trust-region radius, is reduced after a failure of the current iterate, the step from  $x_k$  to the new point will be shorter and usually run to a different direction from the previous point.

By comparison, the line-search and the trust-region differ in the order in which they choose the *search direction* and the *stepsize* to move to the next iterate. The line-search starts with a descent direction  $d_k$  and then determines an appropriate distance along this direction, namely, the stepsize  $\alpha_k$ . In the trust-region, the maximum distance is first chosen, that is the trust-region radius  $\Delta_k$ . Then a direction and a step  $p_k$  that determine the best improvement of the function values subject to this distance constraint are determined. If this step is not satisfactory, then the distance measure  $\Delta_k$  is reduced, and the process is repeated.

For solving the unconstrained optimization problems, the *general principle* of the methods based on the differential operators (the gradient and the Hessian) consists in the construction in the current point of an *affine model* or a *quadratic model* of the minimizing function, in solving this model and in considering its solution as the next approximation of the minimum. This process is then repeated from this new approximation of the minimum point. In the next section, in order to establish the validity of this approach, it is essential to see the bounds of the errors corresponding to the models of the minimizing function and to notice some fundamentals on the convergence of the line-search methods.

## 2.2 Fundamentals on the Convergence of the Line-Search Methods

From the very beginning, two forms of the *fundamental theorem of calculus* are presented: one for the function-gradient and the other for the gradient-Hessian.

**Theorem 2.1** *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be twice continuously differentiable in a neighborhood of a line segment connecting the points  $x^* \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$ ; then*

$$f(x) = f(x^*) + \int_0^1 \nabla f(x^* + te) e dt \quad (2.9)$$

and

$$\nabla f(x) = \nabla f(x^*) + \int_0^1 \nabla^2 f(x^* + te) e dt, \quad (2.10)$$

where  $e = x - x^*$ . ◆

A direct consequence of Theorem 2.1 is the following form of *Taylor's theorem*, which is one of the most used results in optimization.

**Theorem 2.2** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be twice continuously differentiable in a neighborhood of a point  $x^* \in \mathbb{R}^n$ . Then for  $e \in \mathbb{R}^n$  and  $\|e\|$  sufficiently small it follows that

$$f(x) = f(x^*) + \nabla f(x^*)^T e + \frac{1}{2} e^T \nabla^2 f(x^*) e + o(\|e\|^2). \quad (2.11)$$

### Necessary Conditions for Optimality

The conditions which are satisfied at a local minimizer  $x^*$  are necessary conditions. Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be twice continuously differentiable. In the following, based on Taylor's theorem, we show that the gradient of the minimizing function  $f$  vanishes at a local minimum point and the Hessian of  $f$  is positive semidefinite. These are the *necessary conditions* for optimality. It is important to notice that the necessary conditions of (2.1) are in close connection with a nonlinear algebraic system which may be used to get fast and reliable algorithms for computing the minimum points of function  $f$ . The following theorem gives the necessary conditions for the optimality of  $x^*$ .

**Theorem 2.3** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be twice continuously differentiable and let  $x^*$  be a local minimum of  $f$ . Then

$$\nabla f(x^*) = 0.$$

Moreover,  $\nabla^2 f(x^*)$  is positive semidefinite.

**Proof** Let  $u \in \mathbb{R}^n$  be an arbitrary vector. Then Taylor's theorem says that for all real  $t$  sufficiently small

$$f(x^* + tu) = f(x^*) + t \nabla f(x^*)^T u + \frac{t^2}{2} u^T \nabla^2 f(x^*) u + o(t^2).$$

Since  $x^*$  is a local minimizer, then for  $t$  sufficiently small it follows that  $f(x^* + tu) - f(x^*) \geq 0$ . Hence, for  $t$  sufficiently small and all  $u \in \mathbb{R}^n$

$$\nabla f(x^*)^T u + \frac{t}{2} u^T \nabla^2 f(x^*) u + o(t) \geq 0. \quad (2.12)$$

Now, if we set  $t = 0$  and  $u = -\nabla f(x^*)$  we obtain

$$\|\nabla f(x^*)\|^2 = 0.$$

Setting in (2.12)  $\nabla f(x^*) = 0$ , dividing by  $t$  and setting  $t = 0$ , it follows that

$$\frac{1}{2} u^T \nabla^2 f(x^*) u \geq 0$$

for all  $u \in \mathbb{R}^n$ . ◆

The condition  $\nabla f(x^*) = 0$  is known as the *first-order necessary optimality condition* for unconstrained optimization, while a point satisfying this condition is called a *stationary point* or a *critical point* of the minimizing function  $f$ . Observe that the condition  $\nabla f(x^*) = 0$  is a nonlinear algebraic system. Therefore, to compute the minimizers of a function  $f$  we can use fast algorithms for solving nonlinear algebraic systems.

### Sufficient Conditions for Optimality

The conditions which guarantee that  $x^*$  is a local minimizer are the sufficient conditions. Obviously, the gradient of the minimizing function  $f$  also vanishes at a maximum point of  $f$ . In order to make a clear distinction between minima and maxima, the sufficient conditions for optimality are introduced. As it is known, to get a minimizer, the second derivative must be nonnegative. But, this alone is not sufficient. To make sure that we have a minimum point, we must require that the second derivative is strictly positive.

**Theorem 2.4** *Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be twice continuously differentiable in a neighborhood of  $x^*$ . Assume that  $\nabla f(x^*) = 0$  and that  $\nabla^2 f(x^*)$  is positive definite. Then,  $x^*$  is a local minimizer of function  $f$ .*

**Proof** Let  $u \in \mathbb{R}^n$  be an arbitrary vector with  $u \neq 0$ . Then, for sufficiently small  $t$  we can write

$$\begin{aligned} f(x^* + tu) &= f(x^*) + t \nabla f(x^*)^T u + \frac{t^2}{2} u^T \nabla^2 f(x^*) u + o(t^2) \\ &= f(x^*) + \frac{t^2}{2} u^T \nabla^2 f(x^*) u + o(t^2). \end{aligned}$$

Now, if  $\lambda > 0$  is the smallest eigenvalues of  $\nabla^2 f(x^*)$ , it follows that for  $t$  sufficiently small

$$f(x^* + tu) - f(x^*) \geq \frac{\lambda}{2} \|tu\|^2 + o(t^2) > 0.$$

Therefore  $x^*$  is a local minimizer of function  $f$ . ◆

Other details on the optimality conditions for the unconstrained optimization are given in Chap. 11.

### Bounds of the Distance Between the Approximations of Function $f$ and the Function Value in a Point

Let us consider function  $f$  twice continuously differentiable. In the following, let us present some bounds on the distance between the local affine model (local affine approximation):

$$m_1(x) = f(x) + \nabla f(x)^T d, \quad (2.13)$$

or the local quadratic model (local quadratic approximation):

$$m_2(x) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d \quad (2.14)$$

of the minimizing function and the function value in the point  $x + d$ , where  $d$  is the searching direction. From the very beginning, consider the differential function  $f: \mathbb{R} \rightarrow \mathbb{R}$ . Let  $D$  be an open, convex set in  $\mathbb{R}$ .

**Definition 2.1** A function  $f$  is Lipschitz continuous on  $D$  with the constant  $L$  if for any  $x, y \in D$ ,  $|f(x) - f(y)| \leq L\|x - y\|$ .

The following theorem shows a fundamental result from numerical computation by specifying that if  $f'(x)$  is Lipschitz continuous, then there is a bound of the distance between the local affine approximation  $f(x) + f'(x)(y - x)$  and the function value  $f(y)$ .

**Theorem 2.5** Let  $f: D \subset \mathbb{R} \rightarrow \mathbb{R}$  be a function for which  $f'$  is Lipschitz continuous on  $D$  with constant  $L$ . Then, for any  $x, y \in D$  the following estimation is true:

$$|f(y) - f(x) - f'(x)(y - x)| \leq \frac{L}{2}(y - x)^2. \quad (2.15)$$

**Proof** Obviously

$$f(y) - f(x) = \int_x^y f'(z) dz,$$

or

$$f(y) - f(x) - f'(x)(y - x) = \int_x^y [f'(z) - f'(x)] dz.$$

Now, consider  $z = x + t(y - x)$ , where  $dz = (y - x)dt$ . Therefore,

$$f(y) - f(x) - f'(x)(y - x) = \int_0^1 [f'(x + t(y - x)) - f'(x)](y - x) dt.$$

From the triangle inequality we get

$$|f(y) - f(x) - f'(x)(y - x)| \leq |y - x| \int_0^1 L|t(y - x)| dt = \frac{L}{2}|y - x|^2. \quad \blacklozenge$$

Observe that (2.15) resembles the bound of the error from Taylor's series with rest, in which the Lipschitz constant  $L$  is a bound of the  $|f''(\xi)|$ , for  $\xi \in D$ . The main advantage of the Lipschitz continuity is that, for the analysis of algorithms, the derivatives of high order need not be discussed.

In the following, let us present a bound of the distance between the local quadratic model and the function value in a point.

**Theorem 2.6** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the convex and open set  $D \subset \mathbb{R}^n$ . Assume that  $\nabla^2 f(x)$  is Lipschitz continuous in  $x \in D$  with constant  $L \geq 0$ . Then, for any  $x + d \in D$  it follows that

$$\left| f(x + d) - \left[ f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d \right] \right| \leq \frac{L}{6} \|d\|^3. \quad (2.16)$$

The proof of this theorem is similar to the proof of Theorem 2.5, and it is left as an exercise for the reader.  $\blacklozenge$

### Nonlinear Algebraic Systems

In the following, let us consider the simple case of nonlinear algebraic systems defined by the vectorial function  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , where  $F = [f_1(x), \dots, f_m(x)]^T$  and  $m = n$ . A continuous vectorial function is continuously differentiable at  $x \in \mathbb{R}^n$  if each component function  $f_i(x)$ ,  $i = 1, \dots, m$ , is continuous differentiable at  $x$ . The derivative of  $F$  at  $x$  is called the *Jacobian matrix* of  $F$  at  $x$  and is denoted as  $F'(x) = J(x) = \nabla F(x)^T$ .

Obviously

$$[F'(x)]_{ij} \triangleq [J(x)]_{ij} = \frac{\partial f_i(x)}{\partial x_j}, \quad i = 1, \dots, m, \quad j = 1, \dots, n.$$

For continuous vectorial functions, there is no mean value theorem. That is, in general, there may not exist a point  $z \in \mathbb{R}^n$  such that  $F(x + d) = F(x) + J(z)d$ . Even if each component  $f_i(x)$  satisfies  $f_i(x + d) = f_i(x) + \nabla f_i(z_i)^T d$ , the points  $z_i$  may be different.

By using the definition of Jacobian, it follows that if  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is continuously differentiable on an open set  $D \subset \mathbb{R}^n$ , then for any  $x, x + d \in D$ , we have

$$F(x + d) - F(x) = \int_0^1 J(x + td) dt = \int_x^{x+d} F'(\xi) d\xi. \quad (2.17)$$

**Definition 2.2** The function  $F : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  is Hölder continuous on  $D$  if there exists the constants  $L \geq 0$  and  $p \in (0, 1]$  such that for any  $x, y \in D$ ,

$$\|F(x) - F(y)\| \leq L \|x - y\|^p.$$

If  $p = 1$ , then  $F$  is Lipschitz continuous on  $D$  with the constant  $L$ .

For solving the algebraic nonlinear system  $F(x) = 0$  the same general principle is used to approximate the function  $F$  by an affine or a quadratic model. The following results show the bounds of the distance between the approximations of function  $F$ , affine or quadratic, and the function value in a point.

**Theorem 2.7** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be continuously differentiable on the convex open set  $D \subset \mathbb{R}^n$ . Assume that  $J$  is Lipschitz continuous at  $x \in D$  with the constant  $L \geq 0$ . Then, for any  $x + d \in D$  it follows that:

$$\|F(x + d) - F(x) - J(x)d\| \leq \frac{L}{2} \|d\|^2. \quad (2.18)$$

**Proof** We have

$$\begin{aligned} F(x + d) - F(x) - J(x)d &= \int_0^1 J(x + td)dtd - J(x)d \\ &= \int_0^1 [J(x + td) - J(x)]dtd. \end{aligned}$$

Therefore,

$$\begin{aligned} \|F(x + d) - F(x) - J(x)d\| &\leq \int_0^1 \|J(x + td) - J(x)\|\|d\|dt \\ &\leq \int_0^1 L\|td\|\|d\|dt = L\|d\|^2 \int_0^1 tdt = \frac{L}{2}\|d\|^2. \end{aligned} \quad \diamond$$

**Theorem 2.8** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be a continuously differentiable function on the open convex set  $D \subset \mathbb{R}^n$ . Then for any  $u, v, x \in D$  it follows that

$$\|F(u) - F(v) - J(x)(u - v)\| \leq \left[ \sup_{0 \leq t \leq 1} \|J(v + t(u - v)) - J(x)\| \right] \|u - v\|. \quad (2.19)$$

Moreover, assuming that  $J$  is Lipschitz continuous on  $D$ , then

$$\|F(u) - F(v) - J(x)(u - v)\| \leq L\sigma(u, v)\|u - v\| \quad (2.20)$$

and

$$\|F(u) - F(v) - J(x)(u - v)\| \leq L \frac{\|u - x\| + \|v - x\|}{2} \|u - v\|, \quad (2.21)$$

where  $\sigma(u, v) = \max \{\|u - x\|, \|v - x\|\}$ .

**Proof** From (2.17) and the mean value theorem it follows that

$$\begin{aligned} \|F(u) - F(v) - J(x)(u - v)\| &= \left\| \int_0^1 [J(v + t(u - v)) - J(x)](u - v)dt \right\| \\ &\leq \int_0^1 \|J(v + t(u - v)) - J(x)\|\|u - v\|dt \leq \left[ \sup_{0 \leq t \leq 1} \|J(v + t(u - v)) - J(x)\| \right] \|u - v\|, \end{aligned}$$

which is exactly (2.19). Now, since  $J$  is Lipschitz on  $D$ , then, as above, we get

$$\begin{aligned}\|F(u) - F(v) - J(x)(u - v)\| &\leq L \int_0^1 \|v + t(u - v) - x\| \|u - v\| dt \\ &= L \sup_{0 \leq t \leq 1} \|v + t(u - v) - x\| \|u - v\| = L\sigma(u, v)\|u - v\|\end{aligned}$$

which is exactly (2.20). The result (2.21) is proved in the same manner.  $\diamond$

**Theorem 2.9** Let  $F$  and  $J$  which satisfy the conditions of Theorem 2.7. Suppose that there exists  $J(x)^{-1}$ . Then, there exist  $\epsilon > 0$  and  $\beta > \alpha > 0$ , such that for all  $u, v \in D$ , when  $\max\{\|u - x\|, \|v - x\|\} \leq \epsilon$ , it follows that

$$\alpha\|u - v\| \leq \|F(u) - F(v)\| \leq \beta\|u - v\|. \quad (2.22)$$

**Proof** From the triangle inequality and from (2.20)

$$\begin{aligned}\|F(u) - F(v)\| &\leq \|J(x)(u - v)\| + \|F(u) - F(v) - J(x)(u - v)\| \\ &\leq [\|J(x)\| + L\sigma(u, v)]\|u - v\| \leq [\|J(x)\| + L\epsilon]\|u - v\|.\end{aligned}$$

Let  $\beta = \|J(x)\| + L\epsilon$ . Then, we obtain the inequality on the right side of (2.22). Similarly,

$$\begin{aligned}\|F(u) - F(v)\| &\geq \|J(x)(u - v)\| - \|F(u) - F(v) - J(x)(u - v)\| \\ &\geq \left[1/\|J(x)^{-1}\| - L\sigma(u, v)\right]\|u - v\| \geq \left[1/\|J(x)^{-1}\| - L\epsilon\right]\|u - v\|.\end{aligned}$$

Therefore, if  $1/(\|J(x)^{-1}\|L) > \epsilon$ , then we obtain the inequality on the left side of (2.22), where  $\alpha = 1/\|J(x)^{-1}\| - L\epsilon > 0$ .  $\diamond$

In conclusion, if  $x + d \in \mathbb{R}^n$  is close to  $x$ , then the affine approximation (2.13) or the quadratic approximation (2.14) of function  $f$  in  $x + d$  are very good approximations and may be used in an iterative scheme for solving (2.1). The iterative scheme involves solving these approximations (affine or quadratic), solution of which is considered the next approximations to the minimum point  $x^*$ . To ensure the convergence of the iterative scheme, some conditions have to be introduced, which will be presented and discussed in the following sections.

## 2.3 The General Algorithm for Unconstrained Optimization

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuous differentiable function bounded from below. As we have already seen, at iteration  $k$ , given the current point  $x_k$  and a descent search direction  $d_k$ , any optimization unconstrained algorithm computes the next approximation of the minimum point  $x^*$  by the iterative scheme

$$x_{k+1} = x_k + \alpha_k d_k, \quad (2.23)$$

where  $\alpha_k$  is the stepsize along the direction  $d_k$ . We mention that a descent direction  $d_k \neq 0$  in the point  $x_k$  satisfies the condition  $\nabla f(x_k)^T d_k < 0$ , called *descent condition*. Geometrically, the descent condition means that the angle between  $d_k$  and the negative gradient must be smaller than  $90^\circ$ . If  $\alpha_k \neq 0$ , we see that (2.23) satisfies the general principle of numerical computation, which recommends that *a new*

estimation of the solution of a problem should be computed as a small highly accurate modification of the previous estimation.

To start the computations, this iterative scheme must be initialized with an initial point  $x_0$ . This point has to be in  $\text{dom}f$  and additionally it is required that the level set  $S = \{x \in \text{dom}f : f(x) \leq f(x_0)\}$  be closed. This closeness of the level set ensures the elimination of the possibility that the algorithm converges to a point from the frontier of  $\text{dom}f$ .

The purpose of this section is to develop methods for an efficient computation of the scalar  $\alpha_k$ . Let

$$\varphi(\alpha) = f(x_k + \alpha d_k). \quad (2.24)$$

Now, the problem is starting from  $x_k$ , determine the stepsize  $\alpha_k$  in direction  $d_k$ , such that  $\varphi(\alpha_k) < \varphi(0)$ . This is known as the *line-search*. For this line-search, two possibilities are more important.

1. If  $\alpha_k$  is determined as the minimum of function  $f$  in the direction  $d_k$ , that is

$$f(x_k + \alpha_k d_k) = \min_{\alpha > 0} f(x_k + \alpha d_k) \quad (2.25)$$

or

$$\varphi(\alpha_k) = \min_{\alpha > 0} \varphi(\alpha), \quad (2.26)$$

then we say that the line-search is *exact* (or *perfect*).

2. On the other hand, if  $\alpha_k$  is selected such that an acceptable reducing of the objective is obtained along the direction  $d_k$  in point  $x_k$ , i.e.,  $f(x_k) - f(x_k + \alpha_k d_k) > 0$  is acceptable, then we say that the line-search is *inexact* (or *acceptable*).

Due to highly numerical costs, the exact line-search is not used for solving optimization problems, apart from some particular cases, such as the quadratic objective. In the vast majority of situations, the inexact line-search is used. The structure of an unconstrained optimization algorithm based on descent directions is as follows.

### **Algorithm 2.1** General unconstrained optimization algorithm

1.	Initialization. Consider an initial point $x_0 \in \mathbb{R}^n$ , as well as the convergence tolerance $0 < \varepsilon \ll 1$ sufficiently small. Set $k = 0$
2.	Compute the descent direction $d_k$
3.	Using the exact or the inexact line-search, compute the stepsize $\alpha_k$
4.	Compute $x_{k+1} = x_k + \alpha_k d_k$
5.	Test a criterion for stopping the iterations. For example, if $\ \nabla f(x_k)\  \leq \varepsilon$ , then stop; otherwise set $k = k + 1$ and go to step 2

◆

As we can see, the algorithm is very general and plenty of aspects remain to be clarified. For example, in step 2 of the algorithm, it is not specified how the descent direction  $d_k$  is computed. However, this will be presented in the next chapters, where different procedures for the  $d_k$  computation are discussed, for example, steepest descent, Newton, conjugate gradient, quasi-Newton, limited memory L-BFGS, inexact (truncated) Newton, etc. In step 3, the stepsize  $\alpha_k$  can be computed by the exact line-search or by the inexact line-search. In step 5 it is possible to implement other criteria for stopping the iterations, etc.

## 2.4 Convergence of the Algorithm with Exact Line-Search

In this section we are interested in the convergence of Algorithm 2.1, in which the stepsize is computed by the exact line-search (2.25) or (2.26). Let  $\varphi(\alpha) = f(x_k + \alpha d_k)$ . Observe that  $\varphi(0) = f(x_k)$  and  $\varphi(\alpha) \leq \varphi(0)$ . Condition (2.25) is very stringent and involves the computation of the global minimum of function  $\varphi(\alpha)$ , which is a very difficult task. Therefore, instead, we limit our search to the first stationary point of  $\varphi$ , i.e., determine  $\alpha_k$  as

$$\alpha_k = \min \left\{ \alpha \geq 0 : \nabla f(x_k + \alpha d_k)^T d_k = 0 \right\}. \quad (2.27)$$

Since by (2.25) the exact minimum of function  $f$  is obtained and by (2.27) a stationary point of function  $\varphi$  is determined, then the conditions (2.25) and (2.27) determine the exact line-search.

Now, consider  $\langle d_k, -\nabla f(x_k) \rangle$  as the angle between the vectors  $d_k$  and  $-\nabla f(x_k)$ . Then

$$\cos \langle d_k, -\nabla f(x_k) \rangle = -\frac{d_k^T \nabla f(x_k)}{\|d_k\| \|\nabla f(x_k)\|}. \quad (2.28)$$

The following theorem gives a bound of the reduction of the values of function  $f$  at every iteration of Algorithm 2.1 with exact line-search.

**Theorem 2.10** *Let  $\alpha_k > 0$  be solution of the problem (2.25) and assume that  $\|\nabla^2 f(x_k + \alpha d_k)\| \leq M$  for any  $\alpha > 0$ , where  $M$  is a positive constant. Then*

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \frac{1}{2M} \|\nabla f(x_k)\|^2 \cos^2 \langle d_k, -\nabla f(x_k) \rangle. \quad (2.29)$$

**Proof** From the hypothesis of the theorem, for any  $\alpha > 0$ , it follows that

$$f(x_k + \alpha d_k) \leq f(x_k) + \alpha d_k^T \nabla f(x_k) + \frac{\alpha^2}{2} M \|d_k\|^2. \quad (2.30)$$

Now, define

$$\bar{\alpha} = -\frac{d_k^T \nabla f(x_k)}{M \|d_k\|^2}.$$

Then, using (2.30) and (2.28) we get

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &\geq f(x_k) - f(x_k + \bar{\alpha} d_k) \geq -\bar{\alpha} d_k^T \nabla f(x_k) - \frac{\bar{\alpha}^2}{2} M \|d_k\|^2 \\ &= \frac{1}{2} \frac{(d_k^T \nabla f(x_k))^2}{M \|d_k\|^2} = \frac{1}{2M} \|\nabla f(x_k)\|^2 \frac{(d_k^T \nabla f(x_k))^2}{\|d_k\|^2 \|\nabla f(x_k)\|^2} \\ &= \frac{1}{2M} \|\nabla f(x_k)\|^2 \cos^2 \langle d_k, -\nabla f(x_k) \rangle \end{aligned}$$

◆

Observe that the theorem says that, at every iteration, the reduction of the values of function  $f$  is more emphasized when the angle  $\langle d_k, -\nabla f(x_k) \rangle$  is more acute. The following two theorems are two expressions of the same result that shows the convergence of Algorithm 2.1.

**Theorem 2.11** Let  $f(x)$  be a continuously differentiable function on the open set  $D \subset \mathbb{R}^n$  and suppose that the direction  $d_k$  is descent, i.e.,  $d_k^\top \nabla f(x_k) \leq 0$ , and the sequence  $\{x_k\}$  generated by Algorithm 2.1 with exact line-search satisfies a condition of monotony  $f(x_{k+1}) \leq f(x_k)$ . Let  $\bar{x} \in D$  be an accumulation point of  $\{x_k\}$  and a set  $K_1$  of indices with  $K_1 = \{k : \lim x_k = \bar{x}\}$ . Suppose that there is a positive constant  $M > 0$  such that for all  $k \in K_1$ ,  $\|d_k\| < M$ . If  $\bar{d}$  is an arbitrary accumulation point of the sequence  $\{d_k\}$ , then

$$\bar{d}^\top \nabla f(\bar{x}) = 0. \quad (2.31)$$

Moreover, if  $f(x)$  is twice continuous differentiable, then

$$\bar{d}^\top \nabla^2 f(\bar{x}) \bar{d} \geq 0. \quad (2.32)$$

**Proof** We prove only (2.31). Obviously, (2.32) may be proved similarly. Consider  $K_2 \subset K_1$  a set of indices for which  $\bar{d} = \lim_{k \in K_2} d_k$ . If  $\bar{d} = 0$ , then (2.31) is trivial satisfied, otherwise consider the following two cases.

- (i) There is a set of indices  $K_3 \subset K_2$  such that  $\lim_{k \in K_3} \alpha_k = 0$ . Since  $\alpha_k$  is the exact stepsize, it follows that  $d_k^\top \nabla f(x_k + \alpha_k d_k) = 0$ . Moreover, since  $\|d_k\|$  is uniformly bounded from above and  $\alpha_k \rightarrow 0$ , at limit we have that  $\bar{d}^\top \nabla f(\bar{x}) = 0$ .
- (ii) Now we refer to the case  $\liminf_{k \in K_2} \alpha_k = \bar{\alpha} > 0$ . Let  $K_4 \subset K_2$  be a set of indices for which  $\alpha_k \geq \bar{\alpha}/2$ . Suppose that (2.31) is not true, then  $\bar{d}^\top \nabla f(\bar{x}) < -\delta < 0$ . Therefore, there is a neighborhood  $N(\bar{x})$  of  $\bar{x}$  and a set of indices  $K_5 \subset K_4$  such that as soon as  $x \in N(\bar{x})$  and  $k \in K_5$ ,  $d_k^\top \nabla f(x) \leq -\delta/2 < 0$ . Let  $\hat{\alpha}$  be a positive number sufficiently small such that for all  $0 \leq \alpha \leq \hat{\alpha}$  and all  $k \in K_5$ ,  $x_k + \alpha d_k \in N(\bar{x})$ . Consider  $\alpha^* = \min\{\bar{\alpha}/2, \hat{\alpha}\}$ . Then, by using the monotony property of the algorithm, the exact line-search and the development in Taylor's series we have

$$\begin{aligned} f(\bar{x}) - f(x_0) &= \sum_{k=0}^{\infty} [f(x_{k+1}) - f(x_k)] \leq \sum_{k \in K_5} [f(x_{k+1}) - f(x_k)] \\ &\leq \sum_{k \in K_5} [f(x_k + \alpha^* d_k) - f(x_k)] \\ &= \sum_{k \in K_5} \alpha^* \nabla f(x_k + \tau_k d_k)^\top d_k \\ &\leq \sum_{k \in K_5} -\left(\frac{\delta}{2}\right) \alpha^* = -\infty, \end{aligned} \quad (2.33)$$

where  $0 \leq \tau_k \leq \alpha^*$ . The above contradiction shows that (2.31) is also true in case (ii).

The proof of (2.32) is similar. Instead of (2.33) it is sufficient to use the development in Taylor's series up to the second order to get

$$\begin{aligned}
f(\bar{x}) - f(x_0) &\leq \sum_{k \in K_5} [f(x_k + \alpha^* d_k) - f(x_k)] \\
&= \sum_{k \in K_5} \left[ \alpha^* \nabla f(x_k)^T d_k + \frac{(\alpha^*)^2}{2} d_k^T \nabla^2 f(x_k + \tau_k d_k) d_k \right] \\
&\leq \sum_{k \in K_5} \frac{(\alpha^*)^2}{2} d_k^T \nabla^2 f(x_k + \tau_k d_k) d_k \\
&\leq \sum_{k \in K_5} \left[ -\frac{1}{2} \left( \frac{\delta}{2} \right) (\alpha^*)^2 \right] = -\infty.
\end{aligned}$$

Therefore, we get a contradiction which proves (2.32).  $\blacklozenge$

**Theorem 2.12** Let  $\nabla f(x)$  be uniformly continuous on the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ . Also, let  $\theta_k = \langle d_k, -\nabla f(x_k) \rangle$ , where  $d_k$  is the search direction generated by Algorithm 2.1 with exact line-search. If

$$\theta_k \leq \frac{\pi}{2} - \mu, \quad \text{for certain } \mu > 0, \quad (2.34)$$

then either  $\nabla f(x_k) = 0$  for certain  $k$ , or  $f(x_k) \rightarrow -\infty$ , or  $\nabla f(x_k) \rightarrow 0$ .

**Proof** Suppose that for all  $k$ ,  $\nabla f(x_k) \neq 0$  and the function  $f(x_k)$  is bounded from below. Since  $\{f(x_k)\}$  is a sequence decreasing monotonically, then its limit exists. Hence,

$$f(x_k) - f(x_{k+1}) \rightarrow 0. \quad (2.35)$$

By contradiction, suppose that  $\nabla f(x_k) \rightarrow 0$  is not true. Then, there exists an  $\varepsilon > 0$  such that  $\|\nabla f(x_k)\| \geq \varepsilon$ . Therefore,

$$-\frac{\nabla f(x_k)^T d_k}{\|d_k\|} = \|\nabla f(x_k)\| \cos \theta_k \geq \varepsilon \sin \mu \triangleq \varepsilon_1 \quad (2.36)$$

Observe that

$$\begin{aligned}
f(x_k + \alpha d_k) &= f(x_k) + \alpha \nabla f(\xi_k)^T d_k \\
&= f(x_k) + \alpha \nabla f(x_k)^T d_k + \alpha [\nabla f(\xi_k) - \nabla f(x_k)]^T d_k \\
&\leq f(x_k) + \alpha \|d_k\| \left( \frac{\nabla f(x_k)^T d_k}{\|d_k\|} + \|\nabla f(\xi_k) - \nabla f(x_k)\| \right),
\end{aligned} \quad (2.37)$$

where  $\xi_k$  is on the line segment determined by  $x_k$  and  $x_k + \alpha d_k$ . Since  $\nabla f(x)$  is uniformly continuous on the level set  $S$ , it follows that there exists an  $\bar{\alpha}$  such that when  $0 \leq \alpha \|d_k\| \leq \bar{\alpha}$ , then

$$\|\nabla f(\xi_k) - \nabla f(x_k)\| \leq \frac{1}{2} \varepsilon_1. \quad (2.38)$$

From (2.35), (2.36), (2.37), and (2.38) we get

$$f\left(x_k + \bar{\alpha} \frac{d_k}{\|d_k\|}\right) \leq f(x_k) + \bar{\alpha} \left( \frac{\nabla f(x_k)^T d_k}{\|d_k\|} + \frac{1}{2} \varepsilon_1 \right) \leq f(x_k) - \frac{1}{2} \bar{\alpha} \varepsilon_1,$$

i.e.,

$$f(x_{k+1}) \leq f\left(x_k + \bar{\alpha} \frac{d_k}{\|d_k\|}\right) \leq f(x_k) - \frac{1}{2} \bar{\alpha} \varepsilon_1,$$

which contradicts (2.35). Therefore,  $\nabla f(x_k) \rightarrow 0$ , thus proving the theorem.  $\diamond$

In the following, let us present the *rate of convergence* of Algorithm 2.1 with exact line-search. For this, we need some technical results proved in the following three propositions.

**Proposition 2.1** *Let  $\varphi(\alpha)$  be at least twice continuously differentiable on the closed interval  $[0, b]$  and  $\varphi'(0) < 0$ . If the minimum of  $\varphi(\alpha)$  on  $[0, b]$  is  $\alpha^* \in (0, b)$ , then*

$$\alpha^* \geq \tilde{\alpha} = -\varphi'(0)/M, \quad (2.39)$$

where  $M$  is a positive number such that  $\varphi''(\alpha) \leq M$ , for all  $\alpha \in [0, b]$ .

**Proof** Consider the auxiliary function  $\psi(\alpha) = \varphi'(0) + M\alpha$  which has a unique zero  $\tilde{\alpha} = -\varphi'(0)/M$ . But, on  $[0, b]$ ,  $\varphi''(\alpha) \leq M$ , therefore

$$\varphi'(\alpha) = \varphi'(0) + \int_0^\alpha \varphi''(\sigma) d\sigma \leq \varphi'(0) + \int_0^\alpha M d\sigma = \psi(\alpha).$$

Now, taking  $\alpha = \alpha^*$  and noticing that  $\varphi'(\alpha^*) = 0$ , from the above inequality it follows that  $0 \leq \psi(\alpha^*) = \varphi'(0) + M\alpha^*$ , thus proving the proposition.  $\diamond$

**Proposition 2.2** *Let  $f(x)$  be twice continuously differentiable on  $\mathbb{R}^n$ . Then, for all  $x, d \in \mathbb{R}^n$  and for any scalar  $\alpha$  it follows that*

$$f(x + \alpha d) = f(x) + \alpha \nabla f(x)^T d + \alpha^2 \int_0^1 (1-t) [d^T \nabla^2 f(x + t\alpha d) d] dt. \quad (2.40)$$

**Proof** We have

$$\begin{aligned} f(x + \alpha d) - f(x) &= \int_0^1 df(x + t\alpha d) = - \int_0^1 [\alpha \nabla f(x + t\alpha d)^T d] d(1-t) \\ &= - \left[ (1-t) \alpha \nabla f(x + t\alpha d)^T d \right]_0^1 + \int_0^1 (1-t) d [\alpha \nabla f(x + t\alpha d)^T d] \\ &= \alpha \nabla f(x)^T d + \alpha^2 \int_0^1 [(1-t) d^T \nabla^2 f(x + t\alpha d) d] dt \end{aligned}$$

$\diamond$

**Proposition 2.3** *Let  $f(x)$  be a twice continuously differentiable function in a neighborhood of the minimum  $x^*$ . Suppose that there exist the constants  $\varepsilon > 0$  and  $M > m > 0$  such that for any  $y \in \mathbb{R}^n$ , as soon as  $\|x - x^*\| < \varepsilon$ ,*

$$m\|y\|^2 \leq y^T \nabla^2 f(x)y \leq M\|y\|^2. \quad (2.41)$$

Then

$$\frac{1}{2}m\|x - x^*\|^2 \leq f(x) - f(x^*) \leq \frac{1}{2}M\|x - x^*\|^2 \quad (2.42)$$

and

$$\|\nabla f(x)\| \geq m\|x - x^*\|. \quad (2.43)$$

**Proof** From Proposition 2.2 it follows that

$$\begin{aligned} f(x) - f(x^*) &= \nabla f(x^*)^T(x - x^*) + \int_0^1 (1-t)(x - x^*)^T \nabla^2 f(tx + (1-t)x^*)(x - x^*) dt \\ &= \int_0^1 (1-t)(x - x^*)^T \nabla^2 f(tx + (1-t)x^*)(x - x^*) dt. \end{aligned} \quad (2.44)$$

Now, from (2.41) and the mean value theorem we get

$$\begin{aligned} m\|x - x^*\|^2 \int_0^1 (1-t) dt &\leq \int_0^1 (1-t)(x - x^*)^T \nabla^2 f(tx + (1-t)x^*)(x - x^*) dt \\ &\leq M\|x - x^*\|^2 \int_0^1 (1-t) dt. \end{aligned} \quad (2.45)$$

From (2.44) and (2.45) it follows (2.42). For proving (2.43) observe that

$$\nabla f(x) = \nabla f(x) - \nabla f(x^*) = \int_0^1 \nabla^2 f(tx + (1-t)x^*)(x - x^*) dt.$$

Therefore,

$$\begin{aligned} \|\nabla f(x)\| \|x - x^*\| &\geq (x - x^*)^T \nabla f(x) = \int_0^1 (x - x^*)^T \nabla^2 f(tx + (1-t)x^*)(x - x^*) dt \\ &\geq m\|x - x^*\|^2, \end{aligned}$$

which proves (2.43).  $\blacklozenge$

The following theorem is central in establishing the *rate of convergence* of Algorithm 2.1 with exact line-search. It is proved that, if the minimizing function  $f$  is twice continuously differentiable with bounded Hessian, then the algorithm is at least *linear convergent* to a local minimum.

**Theorem 2.13** Let  $\{x_k\}$  be the sequence generated by Algorithm 2.1 with exact line-search, convergent to the minimum point  $x^*$  of function  $f(x)$ . Let  $f(x)$  be twice continuously differentiable in

a neighborhood of  $x^*$  and assume that there exist the positive constants  $\varepsilon > 0$  and  $M > m > 0$ , such that for all  $y \in \mathbb{R}^n$ , as soon as  $\|x - x^*\| < \varepsilon$ ,

$$m\|y\|^2 \leq y^T \nabla^2 f(x)y \leq M\|y\|^2.$$

Then, the sequence  $\{x_k\}$  is at least linear convergent to  $x^*$ .

**Proof** Consider  $\lim_{k \rightarrow \infty} x_k = x^*$ . Therefore, for  $k$  sufficiently large, we can assume that  $\|x_k - x^*\| \leq \varepsilon$ . Since  $\|x_{k+1} - x^*\| < \varepsilon$ , then there exists a positive  $\delta > 0$  such that

$$\|x_k + (\alpha_k + \delta)d_k - x^*\| = \|x_{k+1} - x^* + \delta d_k\| < \varepsilon. \quad (2.46)$$

Observe that  $\varphi(\alpha) = f(x_k + \alpha d_k)$  and  $\varphi'(\alpha) = \nabla f(x_k + \alpha d_k)^T d_k$ . But,  $d_k$  is a descent direction, then  $\varphi'(0) = \nabla f(x_k)^T d_k < 0$  and  $|\varphi'(0)| \leq \|\nabla f(x_k)\| \|d_k\|$ . Therefore, for a  $\rho \in (0, 1)$  we can write

$$\rho \|\nabla f(x_k)\| \|d_k\| \leq -\varphi'(0) \leq \|\nabla f(x_k)\| \|d_k\|$$

and

$$\varphi''(\alpha) = d_k^T \nabla^2 f(x_k + \alpha d_k) d_k \leq M \|d_k\|^2.$$

From Proposition 2.1 we know that the minimum  $\alpha_k$  of  $\varphi(\alpha)$  on  $[0, \alpha_k + \delta]$  satisfies

$$\alpha_k \geq \tilde{\alpha}_k = \frac{-\varphi'(0)}{M \|d_k\|^2} \geq \frac{\rho \|\nabla f(x_k)\|}{M \|d_k\|} \triangleq \bar{\alpha}_k. \quad (2.47)$$

Consider  $\bar{x}_k = x_k + \bar{\alpha}_k d_k$ . From (2.46) it follows that  $\|\bar{x}_k - x^*\| < \varepsilon$ . Therefore,

$$\begin{aligned} f(x_k + \alpha_k d_k) - f(x_k) &\leq f(x_k + \bar{\alpha}_k d_k) - f(x_k) \\ &= \bar{\alpha}_k \nabla f(x_k)^T d_k + \bar{\alpha}_k^2 \int_0^1 (1-t) [d_k^T \nabla^2 f(x_k + t\bar{\alpha}_k d_k) d_k] dt \quad (\text{from Proposition 2.2}) \\ &\leq \bar{\alpha}_k (-\rho) \|\nabla f(x_k)\| \|d_k\| + \frac{1}{2} M \bar{\alpha}_k^2 \|d_k\|^2 \quad (\text{from (2.47)}) \\ &\leq -\frac{\rho^2}{2M} \|\nabla f(x_k)\|^2 \leq -\frac{\rho^2}{2M} m^2 \|x_k - x^*\|^2 \quad (\text{from (2.47) and (2.43)}) \\ &\leq -\left(\frac{\rho m}{M}\right)^2 (f(x_k) - f(x^*)) \quad (\text{from (2.42)}) \end{aligned}$$

Hence,

$$\begin{aligned} f(x_{k+1}) - f(x^*) &= [f(x_{k+1}) - f(x_k)] + [f(x_k) - f(x^*)] \\ &\leq \left[1 - \left(\frac{\rho m}{M}\right)^2\right] [f(x_k) - f(x^*)]. \end{aligned} \quad (2.48)$$

Now, consider

$$\omega = \left[1 - \left(\frac{\rho m}{M}\right)^2\right]^{\frac{1}{2}}.$$

Obviously  $\omega \in (0, 1)$ . Therefore, (2.48) can be rewritten as

$$f(x_k) - f(x^*) \leq \omega^2 [f(x_{k-1}) - f(x^*)] \leq \dots \leq \omega^{2k} [f(x_0) - f(x^*)].$$

From (2.42) it follows that

$$\|x_k - x^*\|^2 \leq \frac{2}{m} [f(x_k) - f(x^*)] \leq \frac{2}{m} \omega^2 [f(x_{k-1}) - f(x^*)] \leq \frac{2}{m} \omega^2 \frac{M}{2} \|x_{k-1} - x^*\|^2,$$

i.e.

$$\|x_k - x^*\| \leq \sqrt{\frac{M}{m}} \omega \|x_{k-1} - x^*\|,$$

showing that the sequence  $\{x_k\}$  is at least linear convergent to  $x^*$ .  $\diamond$

The next theorem establishes a bound of the reduction of the values of the minimizing function  $f$  given by Algorithm 2.1 with exact line-search.

**Theorem 2.14** *Assume that the gradient of the minimizing function  $f(x)$  satisfies the following condition:*

$$(x - z)^T [\nabla f(x) - \nabla f(z)] \geq \eta \|x - z\|^2,$$

for any  $x, z \in \mathbb{R}^n$  and let  $\alpha_k$  be the stepsize obtained by the exact line-search. Then

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \frac{1}{2} \eta \|\alpha_k d_k\|^2.$$

**Proof** Since  $\alpha_k$  is obtained by the exact line-search, it follows that

$$\nabla f(x_k + \alpha_k d_k)^T d_k = 0.$$

Therefore, by the mean value theorem we get

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &= \int_0^{\alpha_k} -d_k^T \nabla f(x_k + t d_k) dt \\ &= \int_0^{\alpha_k} d_k^T [\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k + t d_k)] dt \geq \|d_k\|^2 \int_0^{\alpha_k} \eta (\alpha_k - t) dt = \frac{1}{2} \eta \|\alpha_k d_k\|^2, \end{aligned}$$

proving the theorem.  $\diamond$

## 2.5 Inexact Line-Search Methods

In the previous section we discussed the convergence properties of the general unconstrained optimization Algorithm 2.1 with exact line-search, i.e., the stepsize  $\alpha_k$  is computed as solution of the following minimizing problem

$$f(x_k + \alpha_k d_k) = \min_{\alpha > 0} f(x_k + \alpha d_k), \quad (2.49)$$

or

$$\alpha_k = \min_{\alpha>0} \left\{ \alpha \geq 0 : \nabla f(x_k + \alpha d_k)^T d_k = 0 \right\}. \quad (2.50)$$

The problems (2.49) or (2.50) are not easy to solve, especially when the current point is far away from the minimum point. On the other hand, for many optimization methods like the Newton or quasi-Newton, their rate of convergence does not depend on the exact line-search. Usually, these methods accept a unitary stepsize, and therefore the line-search in these methods is not critical. Therefore, in Algorithm 2.1, as soon as a stepsize which determines an acceptable reducing of the values of the minimizing function is obtained, then the exact line-search can be avoided, thus significantly reducing the computational effort.

The purpose of this section is to present a number of inexact line-search procedures which proved to be efficient and robust for solving unconstrained optimization problems.

### 1. Armijo Line-Search

Armijo (1966) introduced the following inexact line-search, known as *Armijo rule*. Consider the following scalars  $\beta \in (0, 1)$ ,  $\tau_k = -(\nabla f(x_k)^T d_k)/\|d_k\|^2$  and  $\rho \in (0, 1/2)$ . Set  $\alpha_k = \beta^{m_k} \tau_k$ , where  $m_k$  is the first nonnegative integer  $m$  for which

$$f(x_k) - f(x_k + \beta^m \tau_k d_k) \geq -\rho \beta^m \tau_k \nabla f(x_k)^T d_k, \quad (2.51)$$

i.e., try  $m = 0, 1, \dots$  until the above inequality is satisfied for certain  $m = m_k$ . The inequality (2.51) is called the *condition of Armijo*. In other words, the stepsize is computed successively as  $\alpha_k = \tau_k, \beta \tau_k, \beta^2 \tau_k, \dots$  until (2.51) is satisfied. The following theorem shows that there exists an interval for the stepsizes  $\alpha$  computed as above which satisfy the condition of Armijo.

**Theorem 2.15** *Consider  $x_k, d_k \in \mathbb{R}^n$  such that  $d_k \neq 0, \nabla f(x_k)^T d_k < 0$  and  $\rho \in (0, 1)$ . Then, there exists  $\varepsilon = \varepsilon(\rho)$  for which*

$$f(x_k) - f(x_k + \alpha d_k) \geq -\rho \alpha \nabla f(x_k)^T d_k, \quad (2.52)$$

for any  $\alpha \in (0, \varepsilon]$ .

**Proof** Obviously,

$$0 \neq \nabla f(x_k)^T d_k = \lim_{\alpha \rightarrow 0} \frac{f(x_k + \alpha d_k) - f(x_k)}{\alpha}.$$

Hence,

$$\lim_{\alpha \rightarrow 0} \frac{f(x_k + \alpha d_k) - f(x_k)}{\alpha \nabla f(x_k)^T d_k} = 1.$$

Therefore, there exists an  $\varepsilon > 0$  such that for any  $\alpha \in (0, \varepsilon]$ ,

$$0 < \frac{f(x_k + \alpha d_k) - f(x_k)}{\alpha \nabla f(x_k)^T d_k} \leq \rho,$$

i.e., the conclusion of the theorem. ◆

Now, suppose that Armijo line-search is initialized with the value  $\bar{\alpha}_k = -\nabla f(x_k)^T d_k / \|d_k\|^2$ . Then, the following theorem ensures that in mild conditions, the stepsize given by Armijo rule is bounded from below.

**Theorem 2.16** Suppose that  $d_k$  is a descent direction and  $\nabla f(x)$  satisfies the Lipschitz continuity  $\|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\|$  for any  $x, y \in S = \{x : f(x) \leq f(x_0)\}$ , where  $L$  is a positive constant. If the line-search satisfies the condition (2.51) of Armijo, then

$$\alpha_k \geq \min \left\{ 1, \frac{\beta(1-\rho)}{L} \frac{-g_k^T d_k}{\|d_k\|^2} \right\}.$$

**Proof** Let  $K_1 = \{k : \alpha_k = \tau_k\}$  and  $K_2 = \{k : \alpha_k < \tau_k\}$ . Then, for any  $k \in K_1$  we have

$$f_k - f_{k+1} \geq -\rho\tau_k \nabla f(x_k)^T d_k$$

and for any  $k \in K_2$ ,

$$f_k - f_{k+1} \geq -\rho\alpha_k \nabla f(x_k)^T d_k.$$

Using Armijo's rule, since for any  $k \in K_2$ ,  $\alpha_k/\beta \leq \tau_k$ , it follows that for any  $k \in K_2$ ,

$$f_k - f\left(x_k + \frac{\alpha_k}{\beta} d_k\right) < -\rho \frac{\alpha_k}{\beta} \nabla f(x_k)^T d_k.$$

Now, using the mean value theorem for the left side of the above inequality, it follows that there exists a  $\xi_k \in [0, 1]$  such that

$$\nabla f\left(x_k + \frac{\alpha_k}{\beta} \xi_k d_k\right)^T d_k > \frac{\rho}{\beta} \nabla f(x_k)^T d_k$$

for any  $k \in K_2$ .

Using the Cauchy-Schwarz inequality, from the Lipschitz continuity it follows that the above inequality gives

$$\begin{aligned} \frac{\alpha_k}{\beta} L \|d_k\|^2 &\geq \left\| \nabla f\left(x_k + \frac{\alpha_k}{\beta} \xi_k d_k\right) - \nabla f(x_k) \right\| \|d_k\| \\ &\geq \left( \nabla f\left(x_k + \frac{\alpha_k}{\beta} \xi_k d_k\right) - \nabla f(x_k) \right)^T d_k \\ &\geq \rho \nabla f(x_k)^T d_k - \nabla f(x_k)^T d_k = -(1-\rho) \nabla f(x_k)^T d_k \end{aligned}$$

for all  $k \in K_2$ . Combining this with the inequality corresponding to  $k \in K_1$ , the conclusion of the theorem is obtained.  $\blacklozenge$

**Theorem 2.17** (Termination of Armijo line-search) Let  $f$  be continuously differentiable with gradient  $\nabla f(x)$  Lipschitz continuous with constant  $L > 0$ , i.e.,  $\|\nabla(x) - \nabla(y)\| \leq L\|x - y\|$ , for any  $x, y$  from the level set  $S = \{x : f(x) \leq f(x_0)\}$ . Let  $d_k$  be a descent direction at  $x_k$ , i.e.,  $\nabla f(x_k)^T d_k < 0$ . Then, for fixed  $\gamma \in (0, 1)$ :

1. The Armijo condition  $f(x_k + \alpha d_k) \leq f(x_k) + \gamma \alpha g_k^T d_k$  (easily modified) is satisfied for all  $\alpha \in [0, \alpha_k^{\max}]$ , where

$$\alpha_k^{\max} = \frac{2(\gamma - 1)g_k^T d_k}{L \|d_k\|_2^2}.$$

2. For fixed  $\tau \in (0, 1)$  the stepsize generated by the backtracking-Armijo line-search terminates with

$$\alpha_k \geq \min \left\{ \alpha_k^0, \frac{2\tau(\gamma - 1)g_k^T d_k}{L \|d_k\|_2^2} \right\},$$

where  $\alpha_k^0$  is the initial stepsize at iteration  $k$ .

**Proof** Either  $\alpha_k^0$  already satisfies the Armijo condition, or there is a second to the last step in the Armijo backtracking algorithm which does not yet satisfy the Armijo condition. Therefore, the next step will multiply this second to the last one by  $\tau$ , which then satisfies the Armijo condition and the algorithms stops with  $\alpha_k$  satisfying the above inequality.  $\diamond$

## 2. Goldstein Line-Search

Goldstein (1965) presented the following rule, known as *Goldstein rule*. Consider the interval

$$J = \{\alpha > 0 : f(x_k + \alpha d_k) < f(x_k)\}. \quad (2.53)$$

In order to have a sufficient reduction of the values of the minimizing function  $f$ , we should select a value for  $\alpha$  which is far away from the end points of the interval. The following two conditions seem to be reasonable to reach this situation:

$$f(x_k + \alpha d_k) \leq f(x_k) + \rho \alpha \nabla f(x_k)^T d_k \quad (2.54)$$

and

$$f(x_k + \alpha d_k) \geq f(x_k) + (1 - \rho) \alpha \nabla f(x_k)^T d_k, \quad (2.55)$$

where  $0 < \rho < 1/2$ .

Obviously, these conditions may be written as

$$\delta_1 \alpha_k \nabla f(x_k)^T d_k \leq f(x_k + \alpha_k d_k) - f(x_k) \leq \delta_2 \alpha_k \nabla f(x_k)^T d_k, \quad (2.56)$$

where  $0 < \delta_2 < 1/2 < \delta_1 < 1$ . (2.56) is known as *Goldstein inexact line-search*, or *Goldstein rule*, or *Goldstein conditions*.

Let  $\varphi(\alpha) = f(x_k + \alpha d_k)$ . Then, (2.54) and (2.55) can be expressed as

$$\varphi(\alpha_k) \leq \varphi(0) + \rho \alpha_k \varphi'(0), \quad (2.57)$$

$$\varphi(\alpha_k) \geq \varphi(0) + (1 - \rho) \alpha_k \varphi'(0). \quad (2.58)$$

Observe that  $\rho < 1/2$  is a necessary condition. Otherwise, if, for example,  $\varphi(\alpha)$  is a quadratic function which satisfies  $\varphi'(0) < 0$  and  $\varphi''(0) > 0$ , then the global minimum  $\alpha^*$  of  $\varphi$  satisfies the relation  $\varphi(\alpha^*) = \varphi(0) + \frac{1}{2}\alpha^*\varphi'(0)$ . Therefore,  $\alpha^*$  satisfies (2.57) if and only if  $\rho < 1/2$ . This condition  $\rho < 1/2$  has a great effect on the Newton and quasi-Newton methods, allowing  $\alpha = 1$  in these methods and therefore ensuring the superlinear convergence of these methods.

### 3. Wolfe Line-Search

It is quite possible that Goldstein conditions (2.54) and (2.55) be too strong and therefore exclude the minimum points of function  $\varphi(\alpha)$ . That is why instead of (2.55), Wolfe (1969, 1971) introduced the following condition:

$$\nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma \nabla f(x_k)^T d_k, \quad (2.59)$$

where  $\sigma \in (\rho, 1)$ . In terms of function  $\varphi$ , (2.59) can be written as

$$\varphi'(\alpha_k) = \nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma \nabla f(x_k)^T d_k = \sigma \varphi'(0) > \varphi'(0).$$

The geometrical interpretation of (2.59) is that  $\varphi'(\alpha_k)$  computed in an acceptable point must be greater or equal to a multiple  $\sigma \in (0, 1)$  of  $\varphi'(0)$ . The relations (2.54) and (2.59) define the *weak Wolfe inexact line-search*, or the *weak Wolfe rule*, or the *weak Wolfe conditions*. In fact, these conditions are a very subtle extension of Goldstein conditions. Observe that (2.59) can be obtained from the mean value theorem and from (2.55). Indeed, let  $\alpha_k$  be a value of the stepsize which verifies (2.55), then

$$\alpha_k \nabla f(x_k + \theta_k \alpha_k d_k)^T d_k = f(x_k + \alpha_k d_k) - f(x_k) \geq (1 - \rho) \alpha_k \nabla f(x_k)^T d_k,$$

where  $\theta_k \in (0, 1)$ , which is exactly (2.59). Now, let us prove that there is an  $\alpha_k$  which satisfies the inexact Wolfe line-search (2.54) and (2.59).

**Theorem 2.18** Suppose that  $f$  is continuously differentiable and  $d_k$  is a descent direction for  $f$  in  $x_k$ . Suppose that  $f$  is bounded from below along the ray  $\{x_k + \alpha d_k : \alpha \geq 0\}$ . If  $0 < \rho < \sigma < 1$ , then there exists an interval  $[a, b]$ ,  $0 < a < b$ , such that the inexact weak Wolfe line-search

$$\begin{aligned} f(x_k + \alpha d_k) &\leq f(x_k) + \rho \alpha \nabla f(x_k)^T d_k, \\ \nabla f(x_k + \alpha d_k)^T d_k &\geq \sigma \nabla f(x_k)^T d_k, \end{aligned}$$

are satisfied for all  $\alpha \in [a, b]$ .

**Proof** As above, let  $\varphi(\alpha) = f(x_k + \alpha d_k)$ ,  $\alpha \geq 0$ . Since  $\varphi'(0) < 0$  and  $0 < \rho < \sigma < 1$ , it follows that for  $\alpha$  sufficiently small,  $\varphi(\alpha) < \varphi(0) + \rho \alpha \varphi'(0)$ . But, since  $\varphi$  is bounded from below, this relation is not satisfied for  $\alpha$  sufficiently large. Therefore, if we define  $\bar{\alpha} = \sup\{\hat{\alpha} > 0 : \varphi(\alpha) < \varphi(0) + \rho \alpha \varphi'(0), \forall \alpha \in (0, \hat{\alpha})\}$ , then  $\bar{\alpha}$  exists and is finite. Moreover,  $\bar{\alpha}$  satisfies (2.52) with equality, that is  $\varphi(\bar{\alpha}) = \varphi(0) + \rho \bar{\alpha} \varphi'(0)$ . Using the mean value theorem, it follows that there exists  $\beta \in (0, \bar{\alpha})$  such that

$$\varphi'(\beta) = \frac{\varphi(\bar{\alpha}) - \varphi(0)}{\bar{\alpha}} = \frac{\varphi(0) + \rho \bar{\alpha} \varphi'(0) - \varphi(0)}{\bar{\alpha}} = \rho \varphi'(0) > \sigma \varphi'(0),$$

since  $\sigma > \rho$  and  $\varphi'(0) < 0$ . But, from the continuity of  $\varphi'$ , there exists an interval  $(a, b) \subset (0, \bar{\alpha})$  centered in  $\beta$  such that  $\varphi'(\alpha) \geq \sigma\varphi'(0)$  for all  $\alpha \in (a, b)$ . Therefore, by construction, for any  $\alpha \in (a, b)$  the following relations  $\varphi(\alpha) < \varphi(0) + \rho\alpha\varphi'(0)$  and  $\varphi'(\alpha) \geq \sigma\varphi'(0)$  are verified, i.e., the weak Wolfe conditions.  $\blacklozenge$

The first weak Wolfe condition  $f(x_k + \alpha_k d_k) \leq f(x_k) + \rho\alpha_k \nabla f(x_k)^T d_k$  requires that the stepsize  $\alpha_k$  achieve a sufficient reduction of  $f$ . That is why (2.54) is also called the *condition of sufficient reduction*. In other words, this condition forces that the rate of reducing the function  $f$  in the direction  $d_k$  should be at least  $\rho \nabla f(x_k)^T d_k$ . In practice,  $\rho = 10^{-4}$ . Observe that any sufficiently small step may satisfy the condition of sufficient reduction. Consequently, in order to avoid very small steps, totally inefficient in the line-search, the second condition (2.59)  $\nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma \nabla f(x_k)^T d_k$  called the *curvature condition* is introduced. This condition ensures a bound of  $\nabla f(x_k + \alpha_k d_k)^T d_k$ . The typical value of  $\sigma$  is 0.9. In general, the smaller the value of  $\sigma$ , the more accurate the line-search. However, the smaller  $\sigma$  is, the bigger the computational effort is.

**Proposition 2.4** Suppose that  $d_k$  is a descent direction and  $\nabla f$  satisfies the Lipschitz condition

$$\|\nabla f(x) - \nabla f(x_k)\| \leq L\|x - x_k\|$$

for all  $x$  on the line segment connecting  $x_k$  and  $x_{k+1}$ , where  $L$  is a constant. If the line-search satisfies the Goldstein conditions (2.56), then

$$\alpha_k \geq \frac{1 - \delta_1}{L} \frac{|g_k^T d_k|}{\|d_k\|^2}. \quad (2.60)$$

If the line-search satisfies the weak Wolfe conditions, then

$$\alpha_k \geq \frac{1 - \sigma}{L} \frac{|g_k^T d_k|}{\|d_k\|^2}. \quad (2.61)$$

**Proof** If the Goldstein conditions hold, then by (2.56) and by the mean value theorem, we have

$$\begin{aligned} \delta_1 \alpha_k g_k^T d_k &\leq f(x_k + \alpha_k d_k) - f(x_k) \\ &= \alpha_k \nabla f(x_k + \xi d_k)^T d_k \leq \alpha_k g_k^T d_k + L \alpha_k^2 \|d_k\|^2, \end{aligned}$$

where  $\xi \in [0, \alpha_k]$ . From the above inequality we get (2.60).

Subtracting  $\nabla f(x_k)^T d_k$  from both sides of (2.59) and by using the Lipschitz condition, it follows that

$$(\sigma - 1) \nabla f(x_k)^T d_k \leq (\nabla f(x_{k+1}) - \nabla f(x_k))^T d_k \leq \alpha_k L \|d_k\|^2.$$

Observe that  $d_k$  is a descent direction and  $\sigma < 1$ , therefore (2.61) follows from the above inequality.  $\blacklozenge$

As a comparison, a disadvantage of the Goldstein conditions in contrast to the Wolfe conditions is that the first inequality from (2.56) may exclude all the minimizers of function  $\varphi$ . However, both these methods have much in common, and their convergence theory is very similar.

It is worth mentioning that a stepsize computed by the weak Wolfe line-search conditions (2.54) and (2.59) may not be sufficiently close to a minimizer of  $\varphi_k(\alpha)$ . In these situations, the *strong Wolfe*

*line-search conditions* may be used, which consist of (2.54) and, instead of (2.59), the following strengthened version

$$|\nabla f(x_k + \alpha_k d_k)^T d_k| \leq -\sigma \nabla f(x_k)^T d_k \quad (2.62)$$

is used. From (2.50) we see that, if  $\sigma \rightarrow 0$ , then the stepsize which satisfies (2.54) and (2.62) tends to be the optimal stepsize. Observe that, if a stepsize  $\alpha_k$  satisfies the strong Wolfe line-search, then it satisfies the weak Wolfe conditions.

The strong Wolfe conditions allow  $\sigma$  to be chosen to vary the accuracy of the step. If  $\rho$  is fixed at a value close to zero ( $\rho = 10^{-4}$ ), then a value of  $\sigma$  close to  $\rho$  gives a more accurate step with respect to closeness to a critical point of  $\nabla f(x_k + \alpha d_k)^T d_k$ . A value of  $\sigma$  close to 1 results in a more approximate step.

**Proposition 2.5** Suppose that the function  $f$  is continuously differentiable. Let  $d_k$  be a descent direction at point  $x_k$  and assume that  $f$  is bounded from below along the ray  $\{x_k + \alpha d_k : \alpha > 0\}$ . Then, if  $0 < \rho < \sigma < 1$ , there exists an interval of stepsizes  $\alpha$  satisfying the weak Wolfe conditions and the strong Wolfe conditions.

**Proof** Since  $\varphi_k(\alpha) = f(x_k + \alpha d_k)$  is bounded from below for all  $\alpha > 0$ , the line  $l(\alpha) = f(x_k) + \alpha \rho \nabla f(x_k)^T d_k$  must intersect the graph of  $\varphi$  at least once. Let  $\alpha' > 0$  be the smallest intersection value of  $\alpha$ , i.e.,

$$f(x_k + \alpha' d_k) = f(x_k) + \alpha' \rho \nabla f(x_k)^T d_k < f(x_k) + \rho \nabla f(x_k)^T d_k. \quad (2.63)$$

Hence, a sufficient decrease holds for all  $0 < \alpha < \alpha'$ .

Now, by the mean value theorem, there exists  $\alpha'' \in (0, \alpha')$  so that

$$f(x_k + \alpha' d_k) - f(x_k) = \alpha' \nabla f(x_k + \alpha'' d_k)^T d_k. \quad (2.64)$$

Since  $\rho < \sigma$  and  $\nabla f(x_k)^T d_k < 0$ , from (2.63) and (2.64) we get

$$\nabla f(x_k + \alpha'' d_k)^T d_k = \rho \nabla f(x_k)^T d_k > \sigma \nabla f(x_k)^T d_k. \quad (2.65)$$

Therefore,  $\alpha''$  satisfies the weak Wolfe line-search conditions (2.54) and (2.59) and the inequalities are strict. By smoothness assumption on  $f$ , there is an interval around  $\alpha''$  for which the Wolfe conditions hold. Since  $\nabla f(x_k + \alpha'' d_k)^T d_k < 0$ , it follows that the strong Wolfe line-search conditions (2.54) and (2.62) hold in the same interval.  $\blacklozenge$

We mention that the Wolfe line-search conditions are scale-invariant, i.e., multiplying the minimizing function by a constant or making an affine change of variables does not alter them. This line-search can be used in the most line-search methods for unconstrained optimization, particularly in the implementation of Newton, quasi-Newton, conjugate gradient, etc.

#### 4. Generalized Wolfe Line-Search

In the *generalized Wolfe line-search*, the absolute value in (2.62) is replaced by a pair of inequalities:

$$\sigma_1 \nabla f(x_k)^T d_k \leq \nabla f(x_k + \alpha_k d_k)^T d_k \leq -\sigma_2 \nabla f(x_k)^T d_k, \quad (2.66)$$

where  $0 < \rho < \sigma_1 < 1$  and  $\sigma_2 \geq 0$ . The particular case in which  $\sigma_1 = \sigma_2 = \sigma$  corresponds to the strong Wolfe line-search.

### 5. Hager-Zhang Line-Search

Hager and Zhang (2005) introduced the *approximate Wolfe line-search*

$$\sigma \nabla f(x_k)^T d_k \leq \nabla f(x_k + \alpha_k d_k)^T d_k \leq (2\rho - 1) \nabla f(x_k)^T d_k, \quad (2.67)$$

where  $0 < \rho < 1/2$  and  $\rho < \sigma < 1$ . Observe that the approximate Wolfe line-search (2.67) has the same form as the generalized Wolfe line-search (2.66), but with a special choice for  $\sigma_2$ . The first inequality in (2.67) is the same as (2.59). When  $f$  is quadratic, the second inequality in (2.67) is equivalent to (2.54).

In general, when  $\varphi_k(\alpha) = f(x_k + \alpha d_k)$  is replaced by a quadratic interpolating  $q(\cdot)$  that matches  $\varphi_k(\alpha)$  at  $\alpha = 0$  and  $\varphi'_k(\alpha)$  at  $\alpha = 0$  and  $\alpha = \alpha_k$ , (2.54) reduces to the second inequality in (2.67). Observe that the decay condition (2.54) is a component of the generalized Wolfe line-search, while in the approximate Wolfe line-search the decay condition is approximately enforced through the second inequality in (2.67). As shown by Hager and Zhang (2005), the first Wolfe condition (2.54) limits the accuracy of a conjugate gradient method to the order of the square root of the machine precision, while with the approximate Wolfe line-search we can achieve accuracy to the order of the machine precision.

The approximate Wolfe line-search is based on the derivative of  $\varphi_k(\alpha)$ . This can be achieved by using a quadratic approximation of  $\varphi_k$ . The quadratic interpolating polynomial  $q$  that matches  $\varphi_k(\alpha)$  at  $\alpha = 0$  and  $\varphi'_k(\alpha)$  at  $\alpha = 0$  and  $\alpha = \alpha_k$  (which is unknown) is given by

$$q(\alpha) = \varphi_k(0) + \varphi'_k(0)\alpha + \frac{\varphi'_k(\alpha_k) - \varphi'_k(0)}{2\alpha_k} \alpha^2.$$

Observe that the first weak Wolfe condition (2.54) can be written as  $\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho \alpha_k \varphi'_k(0)$ . Now, if  $\varphi_k$  is replaced by  $q$  in the first weak Wolfe condition, we get  $q(\alpha_k) \leq q(0) + \rho q'(\alpha_k)$ , which is rewritten as

$$\frac{\varphi'_k(\alpha_k) - \varphi'_k(0)}{2} \alpha_k + \varphi'_k(0) \alpha_k \leq \rho \alpha_k \varphi'_k(0),$$

and can be restated as

$$\varphi'_k(\alpha_k) \leq (2\rho - 1) \varphi'_k(0), \quad (2.68)$$

where  $\rho < \min\{0.5, \sigma\}$ , which is exactly the second inequality in (2.67).

In terms of function  $\varphi_k(\cdot)$  the approximate line-search aims at finding the stepsize  $\alpha_k$  which satisfies the weak Wolfe conditions

$$\varphi_k(\alpha) \leq \varphi_k(0) + \rho \varphi'_k(0)\alpha, \quad \text{and} \quad \varphi'_k(\alpha) \geq \sigma \varphi'_k(0), \quad (2.69)$$

which are called *LS1 conditions*, or the conditions (2.68) together with

$$\varphi_k(\alpha) \leq \varphi_k(0) + \varepsilon_k, \quad \text{and} \quad \varepsilon_k = \varepsilon |f(x_k)|, \quad (2.70)$$

where  $\varepsilon$  is a small positive parameter ( $\varepsilon = 10^{-6}$ ), which are called *LS2 conditions*.  $\varepsilon_k$  is an estimate for the error in the value of  $f$  at iteration  $k$ . With these, the approximate Wolfe line-search algorithm is as follows:

**Algorithm 2.2** Hager and Zhang line-search algorithm

- |    |  |
|----|--|
| 1. | Choose an initial interval $[a_0, b_0]$ and set $k = 0$  |
| 2. | If either LS1 or LS2 conditions are satisfied at $a_k$ , stop  |
| 3. | Define a new interval $[a, b]$ by using the $\text{secant}^2$ procedure: $[a, b] = \text{secant}^2(a_k, b_k)$  |
| 4. | If $b - a > \gamma(b_k - a_k)$ , then $c = (a + b)/2$ and use the $\text{update}$ procedure: $[a, b] = \text{update}(a, b, c)$ , where $\gamma \in (0, 1)$ . ( $\gamma = 0.66$ ) |
| 5. | Set $[a_k, b_k] = [a, b]$ and $k = k + 1$ and go to step 2   |

◆

The update procedure changes the current bracketing interval  $[a, b]$  into a new one  $[\bar{a}, \bar{b}]$  by using an additional point which is either obtained by a bisection step or by a secant step. The input data in the procedure update are the points  $a, b, c$ . The parameter in the procedure update is  $\theta \in (0, 1)$  ( $\theta = 0.5$ ). The output data are  $\bar{a}, \bar{b}$ .

**Algorithm 2.3** The update procedure

- |    |   |
|----|---|
| 1. | If $c \notin (a, b)$ , then set $\bar{a} = a$ , $\bar{b} = b$ and return  |
| 2. | If $\varphi'_k(c) \geq 0$ , then set $\bar{a} = a$ , $\bar{b} = c$ and return   |
| 3. | If $\varphi'_k(c) < 0$ and $\varphi_k(c) \leq \varphi_k(0) + \varepsilon_k$ , then set $\bar{a} = c$ , $\bar{b} = b$ and return   |
| 4. | If $\varphi'_k(c) < 0$ and $\varphi_k(c) > \varphi_k(0) + \varepsilon_k$ , then set $\hat{a} = a$ , $\hat{b} = c$ and perform the following steps:<br>(a) Set $d = (1 - \theta)\hat{a} + \theta\hat{b}$ . If $\varphi'_k(d) \geq 0$ , set $\bar{b} = d$ , $\bar{a} = \hat{a}$ and return<br>(b) If $\varphi'_k(d) < 0$ and $\varphi_k(d) \leq \varphi_k(0) + \varepsilon_k$ , then set $\hat{a} = d$ and go to step (a)<br>(c) If $\varphi'_k(d) < 0$ and $\varphi_k(d) > \varphi_k(0) + \varepsilon_k$ , then set $\hat{b} = d$ and go to step (a) |

◆

The update procedure finds the interval  $[\bar{a}, \bar{b}]$  so that

$$\varphi_k(\bar{a}) < \varphi_k(0) + \varepsilon_k, \quad \varphi'_k(\bar{a}) < 0 \quad \text{and} \quad \varphi'_k(\bar{b}) \geq 0. \quad (2.71)$$

Eventually, a nested sequence of intervals  $[a_k, b_k]$  is determined, which converges to the point that satisfies either LS1 (2.69) or LS2 (2.68) and (2.70) conditions.

The secant procedure updates the interval by secant steps. If  $c$  is obtained from a secant step based on the function values at  $a$  and  $b$ , then we write

$$c = \text{secant } (a, b) = \frac{a\varphi'_k(b) - b\varphi'_k(a)}{\varphi'_k(b) - \varphi'_k(a)}.$$

Since we do not know whether  $\varphi'$  is a convex or a concave function, then a pair of secant steps is generated by a procedure denoted  $\text{secant}^2$ , defined as follows. The input data are the points  $a$  and  $b$ . The outputs are  $\bar{a}$  and  $\bar{b}$  which define the interval  $[\bar{a}, \bar{b}]$ .

**Algorithm 2.4** Procedure  $\text{secant}^2$ 

- |    |   |
|----|---|
| 1. | Set $c = \text{secant } (a, b)$ and $[A, B] = \text{update } (a, b, c)$   |
| 2. | If $c = B$ , then $\bar{c} = \text{secant } (b, B)$   |
| 3. | If $c = A$ , then $\bar{c} = \text{secant } (a, A)$   |
| 4. | If $c = A$ or $c = B$ , then $[\bar{a}, \bar{b}] = \text{update } (A, B, \bar{c})$ . Otherwise, $[\bar{a}, \bar{b}] = [A, B]$ |

◆

The Hager and Zhang line-search procedure finds the stepsize  $\alpha_k$  that satisfies either LS1 or LS2 in a finite number of operations, as it is stated in the following theorem proved by Hager and Zhang (2005).

**Theorem 2.19** Suppose that  $\varphi_k(\alpha)$  is continuously differentiable on an interval  $[a_0, b_0]$ , where (2.71) holds. If  $\rho \in (0, 1/2)$ , then the Hager and Zhang line-search procedure terminates at a point satisfying either LS1 or LS2 conditions.  $\blacklozenge$

Under some additional assumptions, the convergence analysis of the secant<sup>2</sup> procedure was given by Hager and Zhang (2005), proving that the interval width generated by it is tending to zero, with the root convergence order  $1 + \sqrt{2}$ . This line-search procedure is implemented in CG-DESCENT, one of the most advanced conjugate gradient algorithms (see Chap. 5).

## 6. Dai and Kou Line-Search

In practical computations, the first Wolfe condition (2.54) may never be satisfied because of the numerical errors, even for tiny values of  $\rho$ . In order to avoid the numerical drawback of the weak Wolfe line-search, Hager and Zhang (2005) introduced a combination of the original weak Wolfe conditions and the approximate Wolfe conditions (2.67). Their line-search works well in numerical computations, but in theory it cannot guarantee the global convergence of the algorithm. Therefore, in order to overcome this deficiency of the approximate Wolfe line-search, Dai and Kou (2013) introduced the so-called *improved Wolfe line-search*: “given a constant parameter  $\epsilon > 0$ , a positive sequence  $\{\eta_k\}$  satisfying  $\sum_{k \geq 1} \eta_k < \infty$  as well as the parameters  $\rho$  and  $\sigma$  satisfying  $0 < \rho < \sigma < 1$ , Dai and Kou (2013) proposed the following modified Wolfe condition:

$$f(x_k + \alpha d_k) \leq f(x_k) + \min \{ \epsilon |g_k^T d_k|, \rho \alpha g_k^T d_k + \eta_k \}. \quad (2.72)$$

The line-search satisfying (2.72) and (2.59) is called the *improved Wolfe line-search*. If  $f$  is continuously differentiable and bounded from below, the gradient  $g$  is Lipschitz continuous and  $d_k$  is a descent direction (i.e.,  $\nabla f(x_k)^T d_k < 0$ ), then there must exist a suitable stepsize satisfying (2.59) and (2.72), since they are weaker than the weak Wolfe conditions.

## 7. Backtracking

Often, in practice, when a “good” search direction is known, then only the condition (2.54) is used for the stepsize computation. The main idea of this algorithm is to start the computations with  $\alpha_k = 1$ . If the new point  $x_k + \alpha_k d_k$  is not acceptable, then reduce the value of  $\alpha_k$  until the first condition of (2.54)  $f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k \nabla f(x_k)^T d_k$  is satisfied. This method is called *inexact line-search with backtracking*. A simple backtracking algorithm is as follows.

### Algorithm 2.5 Backtracking

- |    |  |
|----|--|
| 1. | Choose the scalars: $\rho \in (0, 1/2)$ , $0 < l < u < 1$ and set $\alpha_k = 1$   |
| 2. | Test if $f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k \nabla f(x_k)^T d_k$  |
| 3. | If $f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k \nabla f(x_k)^T d_k$ is not satisfied, then set $\alpha_k = \beta \alpha_k$ , where $\beta \in [l, u]$ and go to step 2; otherwise, stop with $\alpha_k$ obtained at previous step $\blacklozenge$ |

Observe that backtracking is very similar to Armijo's rule. Often, backtracking is used in the Newton or quasi-Newton methods, where it is known that, for the vast majority of the steps of these methods, their length is unitary, i.e., a single test of (2.54) is sufficient to get a convenient stepsize  $\alpha_k$ .

A more sophisticated variant of the inexact line-search with backtracking consists of determining  $\alpha_k$  not by its simple reduction through a factor taken from the interval  $(0, 1)$ , as in the above algorithm, but using the quadratic interpolation. Indeed, let  $\varphi(\alpha) = f(x_k + \alpha d_k)$ . From the very beginning we know  $\varphi(0) = f(x_k)$  and  $\varphi'(0) = \nabla f(x_k)^T d_k$ . Taking a unitary step, we find  $\varphi(1) = f(x_k + d_k)$ . If  $f(x_k + d_k)$  does not satisfy (2.54), then the following quadratic model may be used in order to approximate  $\varphi(\alpha)$ :

$$p(\alpha) = (\varphi(1) - \varphi(0) - \varphi'(0))\alpha^2 + \varphi'(0)\alpha + \varphi(0),$$

which takes into account the above interpolating values. From the minimum condition  $p'(\alpha) = 0$  we get a new value for  $\alpha$  as

$$\alpha = -\frac{\varphi'(0)}{2(\varphi(1) - \varphi(0) - \varphi'(0))}.$$

This value may be used in an iterative process as above to get an acceptable stepsize. In order to avoid too small stepsizes  $\alpha_k$ , some protections may be introduced. For example, if  $\|\alpha_k d_k\| < \eta$ , where  $\eta$  is a given parameter, then the computation of the stepsize is stopped. A simple variant of the inexact line-search with backtracking is as follows.

**Algorithm 2.6** *Backtracking – variant*

1.	Choose the scalars: $\rho \in (0, 1/2)$ , $\beta \in (0, 1)$ and set $\alpha_k = 1$
2.	If $f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k \nabla f(x_k)^T d_k$ , then, stop; otherwise go to step 3
3.	Set $\alpha_k = \beta \alpha_k$ and go to step 2

◆

Since  $d_k$  is a descent direction, then  $\nabla f(x_k)^T d_k < 0$ . Therefore, it follows that for sufficiently small values  $\alpha_k$  we have:

$$f(x_k + \alpha_k d_k) \cong f(x_k) + \alpha_k \nabla f(x_k)^T d_k < f(x_k) + \rho \alpha_k \nabla f(x_k)^T d_k,$$

which shows that the algorithm terminates in a finite number of iterations. The constant  $\rho$  is interpreted as the fraction of reducing the values of  $f$ , predicted by linear extrapolation. Usually,  $\rho$  is selected between 0.001 and 0.5 that means that we accept a reduction of function  $f$  between 1% and 50% from the prediction based on linear extrapolation. The parameter  $\beta$  is selected between 0.1 and 0.9. The following theorem shows the properties of the inexact line-search with backtracking.

**Theorem 2.20** *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a continuously differentiable function and the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  a compact. Suppose that for any  $k$ ,  $\nabla f(x_k)^T d_k < 0$ . Then, Algorithm 2.6 determines in a finite number of iterations a value  $\alpha_k > 0$  which satisfies the condition from step 2 of the algorithm. The sequence  $x_{k+1} = x_k + \alpha_k d_k$  satisfies  $f(x_{k+1}) < f(x_k)$  for any  $k$  and*

$$\lim_{k \rightarrow \infty} \frac{\nabla f(x_k)^T d_k}{\|d_k\|} = 0.$$

**Proof** We shall prove that the algorithm with backtracking cannot cycle indefinitely between steps 2 and 3. Indeed, if the algorithm cycles between these steps, then step 2 is never satisfied, and therefore

$$\frac{f(x_k + \beta^j d_k) - f(x_k)}{\beta^j} > \rho \nabla f(x_k)^T d_k.$$

But  $\beta \in (0, 1)$  and  $\beta^j \rightarrow 0$  when  $j \rightarrow \infty$  and therefore, for  $j \rightarrow \infty$  the above inequality becomes  $\nabla f(x_k)^T d_k > \rho \nabla f(x_k)^T d_k$ , which is not possible since  $\rho \in (0, 1/2)$  and  $\nabla f(x_k)^T d_k \neq 0$ .

Therefore, there exists a  $j_0$  such that

$$f(x_k + \beta^{j_0} d_k) - f(x_k) \leq \rho \beta^{j_0} \nabla f(x_k)^T d_k \neq 0.$$

Since  $\nabla f(x_k)^T d_k \neq 0$ , it follows that for any  $k$ ,  $f(x_{k+1}) < f(x_k)$ . Now, from the above inequality we get

$$\frac{f(x_k + \beta^{j_0} d_k) - f(x_k)}{\beta^{j_0} \|d_k\|} \leq \rho \frac{\nabla f(x_k)^T d_k}{\|d_k\|}.$$

Using the mean value theorem, it follows that

$$\frac{\nabla f(x_k + \beta^{j_0} d_k)^T d_k}{\|d_k\|} \leq \rho \frac{\nabla f(x_k)^T d_k}{\|d_k\|},$$

i.e., the conclusion of the theorem. ◆

It is worth mentioning that by using the same technique as above it is possible to obtain the inexact line-search with *approximative backtracking*. This is exactly the last inequality from (2.67).

The condition from the inexact line-search with backtracking is sufficient to achieve an improvement of the values of the minimizing function, but this is not strong enough to guarantee that any accumulation point of the algorithm will be a stationary point of function  $f$ . In fact, if  $n \geq 2$ , the descent direction  $d_k$  is very likely to be selected in such a way that the angle between  $d_k$  and  $-\nabla f(x_k)$  should be very close to  $90^\circ$ . In this case, the cosine of this angle tends to zero, case in which the general algorithm may converge to a nonstationary point. In order to avoid this situation, we require that the cosine of the above angle should be uniformly bounded away from zero. In other words, we require that the searching directions make an acute cone centered in  $-\nabla f(x_k)$ , i.e., they should satisfy the so-called *angle condition*

$$\nabla f(x_k)^T d_k \leq -\tau \|\nabla f(x_k)\| \|d_k\|,$$

where  $\tau \in (0, 1)$ .

### 8. Nonmonotone Line-Search Grippo, Lampariello, and Lucidi

The nonmonotone line-search for Newton's methods was introduced by Grippo, Lampariello, and Lucidi (1986). In this method, the stepsize  $\alpha_k$  satisfies the following condition:

$$f(x_k + \alpha_k d_k) \leq \max_{0 \leq j \leq m(k)} f(x_{k-j}) + \rho \alpha_k g_k^T d_k, \quad (2.73)$$

where  $\rho \in (0, 1)$ ,  $m(0) = 0$ ,  $0 \leq m(k) \leq \min \{m(k-1) + 1, M\}$  and  $M$  is a prespecified nonnegative integer. Theoretical analysis and numerical experiments showed the efficiency and robustness of this line-search for solving unconstrained optimization problems in the context of the Newton method.

The  $r$ -linear convergence for the nonmonotone line-search (2.73), when the objective function  $f$  is strongly convex, was proved by Dai (2002b).

Although these nonmonotone techniques based on (2.73) work well in many cases, there are some drawbacks. Firstly, a good function value generated in any iteration is essentially discarded due to the max in (2.73). Secondly, in some cases, the numerical performance is quite dependent on the choice of  $M$  (see Raydan, (1997)). Furthermore, it was pointed out by Dai (2002b) that, although an iterative method generates  $r$ -linearly convergent iterations for a strongly convex function, the iterates may not satisfy the condition (2.73) for  $k$  sufficiently large and for any fixed bound  $M$  on the memory.

### 9. Nonmonotone Line-Search Zhang and Hager

Zhang and Hager (2004) proposed another nonmonotone line-search technique by replacing the maximum function values in (2.73) with an average of function values. Suppose that  $d_k$  is a descent direction. Their line-search determines a stepsize  $\alpha_k$  as follows.

#### Algorithm 2.7 Zhang and Hager nonmonotone line-search

- |    |  |
|----|--|
| 1. | Choose a starting guess $x_0$ and the parameters: $0 \leq \eta_{\min} \leq \eta_{\max} \leq 1$ , $0 < \rho < \sigma < 1 < \beta$ and $\mu > 0$ . Set $C_0 = f(x_0)$ , $Q_0 = 1$ and $k = 0$                  |
| 2. | If $\ \nabla f(x_k)\ $ is sufficiently small, then stop  |
| 3. | Line-search update: Set $x_{k+1} = x_k + \alpha_k d_k$ , where $\alpha_k$ satisfies either the nonmonotone Wolfe conditions:   |
|    | $f(x_k + \alpha_k d_k) \leq C_k + \rho \alpha_k g_k^T d_k, \quad (2.74)$   |
|    | $\nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma d_k^T g_k, \quad (2.75)$   |
|    | or the nonmonotone Armijo conditions: $\alpha_k = \bar{\alpha}_k \beta^{h_k}$ , where $\bar{\alpha}_k > 0$ is the trial step and $h_k$ is the largest integer such that (2.74) holds and $\alpha_k \leq \mu$ |
| 4. | Choose $\eta_k \in [\eta_{\min}, \eta_{\max}]$ and set:  |
|    | $Q_{k+1} = \eta_k Q_k + 1, \quad (2.76)$   |
|    | $C_{k+1} = \frac{\eta_k Q_k C_k + f(x_{k+1})}{Q_{k+1}} \quad (2.77)$   |
| 5. | Set $k = k + 1$ and go to step 2   |

Observe that  $C_{k+1}$  is a convex combination of  $C_k$  and  $f(x_{k+1})$ . Since  $C_0 = f(x_0)$ , it follows that  $C_k$  is a convex combination of the function values  $f(x_0), f(x_1), \dots, f(x_k)$ . Parameter  $\eta_k$  controls the degree of nonmonotonicity. If  $\eta_k = 0$  for all  $k$ , then this nonmonotone line-search reduces to the monotone Wolfe or Armijo line-search. If  $\eta_k = 1$  for all  $k$ , then  $C_k = A_k$ , where

$$A_k = \frac{1}{k+1} \sum_{i=0}^n f(x_i).$$

**Theorem 2.21** *If  $\nabla f(x_k)^T d_k \leq 0$  for each  $k$ , then, for the iterates generated by the nonmonotone line-search of Zhang and Hager, we have  $f(x_k) \leq C_k \leq A_k$  for each  $k$ . Moreover, if  $\nabla f(x_k)^T d_k < 0$  and  $f(x)$  is bounded from below, then there exists  $\alpha_k$  satisfying either the Wolfe or the Armijo conditions of the line-search update.* ◆

Zhang and Hager (2004) proved the convergence of their algorithm.

**Theorem 2.22** Suppose that  $f$  is bounded from below and there exist the positive constants  $c_1$  and  $c_2$  such that  $\nabla f(x_k)^T d_k \leq -c_1 \|g_k\|^2$  and  $\|d_k\| \leq c_2 \|g_k\|$  for all sufficiently large  $k$ . Then, under the weak Wolfe line-search, if  $\nabla f$  is Lipschitz continuous, then the iterates  $x_k$  generated by the nonmonotone line-search of Zhang and Hager have the property that  $\liminf_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ . Moreover, if  $\eta_{\max} < 1$ , then  $\lim_{k \rightarrow \infty} \nabla f(x_k) = 0$ .  $\blacklozenge$

The numerical results reported by Zhang and Hager (2004) showed that this nonmonotone line-search is superior to the nonmonotone technique (2.73).

### 10. Nonmonotone Line-Search Gu and Mo

A modified version of the nonmonotone line-search (2.73) was proposed by Gu and Mo (2008). In this method, instead of an average of the successive objective function values introduced by Zhang and Hager (2004), the current nonmonotone term is a convex combination of the previous nonmonotone term and of the current value of the objective function, i.e., the stepsize  $\alpha_k$  is computed to satisfy the following condition:

$$f(x_k + \alpha_k d_k) \leq D_k + \rho \alpha_k g_k^T d_k, \quad (2.78)$$

where

$$\begin{cases} D_0 = f(x_0), & k = 0, \\ D_k = \theta_k D_{k-1} + (1 - \theta_k)f(x_k), & k \geq 1, \end{cases} \quad (2.79)$$

with  $0 \leq \theta_k \leq \theta_{\max} < 1$  and  $\rho \in (0, 1)$ . The theoretical and numerical results reported by Gu and Mo (2008) in the frame of the trust-region method showed the efficiency of this nonmonotone line-search scheme.

### 11. Nonmonotone Line-Search Huang, Wan, and Chen

Huang, Wan, and Chen (2014) proposed a new nonmonotone line-search as an improved version of the nonmonotone line-search technique proposed by Zhang and Hager. Their algorithm implementing the nonmonotone Armijo condition has the same properties as the nonmonotone line-search algorithm of Zhang and Hager as well as some other properties that certify its convergence in very mild conditions. Suppose that at  $x_k$  the search direction is  $d_k$ . The nonmonotone line-search proposed by Huang, Wan, and Chen is as follows:

#### Algorithm 2.8 Nonmonotone line-search Huang, Wan, and Chen

1.	Choose $0 \leq \eta_{\min} \leq \eta_{\max} < 1 < \beta$ , $\delta_{\max} < 1$ , $0 < \delta_{\min} < (1 - \eta_{\max})\delta_{\max}$ , the convergence tolerance $\varepsilon > 0$ small enough and $\mu > 0$
2.	If $\ g_k\  \leq \varepsilon$ , then the algorithm stops
3.	Choose $\eta_k \in [\eta_{\min}, \eta_{\max}]$ . Compute $Q_{k+1}$ and $C_{k+1}$ by (2.76) and (2.77), respectively. Choose $\delta_{\min} \leq \delta_k \leq \delta_{\max}/Q_{k+1}$ . Let $\alpha_k = \bar{\alpha}_k \beta^{h_k} \leq \mu$ be a stepsize satisfying
	$C_{k+1} = \frac{\eta_k Q_k C_k + f(x_k + \alpha_k d_k)}{Q_{k+1}} \leq C_k + \delta_k \alpha_k g_k^T d_k, \quad (2.80)$
	where $h_k$ is the largest integer such that (2.80) holds and $Q_k$ , $C_k$ , $Q_{k+1}$ and $C_{k+1}$ are computed as in the nonmonotone line-search of Zhang and Hager
4.	Set $x_{k+1} = x_k + \alpha_k d_k$ . Set $k = k + 1$ and go to step 2 $\blacklozenge$

If the minimizing function  $f$  is continuously differentiable and if  $g_k^T d_k \leq 0$  for each  $k$ , then there exists a trial step  $\bar{\alpha}_k$  such that (2.80) holds. The convergence of this nonmonotone line-search is obtained under the same conditions as in Theorem 2.22. The  $r$ -linear convergence is proved for strongly convex functions.

## 12. Nonmonotone Line-Search Ou and Liu

Based on (2.78), a new modified nonmonotone memory gradient algorithm for unconstrained optimization was elaborated by Ou and Liu (2017). Given  $\rho_1 \in (0, 1)$ ,  $\rho_2 > 0$  and  $\beta \in (0, 1)$  set  $s_k = -(g_k^T d_k)/\|d_k\|^2$  and compute the stepsize  $\alpha_k = \max \{s_k, s_k\beta, s_k\beta^2, \dots\}$  satisfying the line-search condition:

$$f(x_k + \alpha_k d_k) \leq D_k + \rho_1 \alpha_k g_k^T d_k - \rho_2 \alpha_k^2 \|d_k\|^2, \quad (2.81)$$

where  $D_k$  is defined by (2.79) and  $d_k$  is a descent direction, i.e.,  $g_k^T d_k < 0$ . Observe that if  $\rho_2 = 0$  and  $s_k \equiv s$  for all  $k$ , then the nonmonotone line-search (2.81) reduces to the nonmonotone line-search (2.78). The algorithm corresponding to this nonmonotone line-search presented by Ou and Liu is as follows:

### Algorithm 2.9 Ou and Liu nonmonotone line-search

1.	Consider a starting guess $x_0$ and select the parameters: convergence tolerance $\varepsilon \geq 0$ , $0 < \tau < 1$ , $\rho_1 \in (0, 1)$ , $\rho_2 > 0$ , $\beta \in (0, 1)$ and an integer $m > 0$ . Set $k = 0$
2.	If $\ g_k\  \leq \varepsilon$ , then stop
3.	Compute the direction $d_k$ by the following recursive formula:
	$d_k = \begin{cases} -g_k, & \text{if } k \leq m, \\ -\lambda_k g_k - \sum_{i=1}^m \lambda_{k-i} d_{k-i} & \text{if } k \geq m+1, \end{cases} \quad (2.82)$
	where
	$\lambda_{k-i} = \frac{\tau}{m} \frac{\ g_k\ ^2}{\ g_k\ ^2 +  g_k^T d_{k-i} }, \quad i = 1, \dots, m,$
	$\lambda_k = 1 - \sum_{i=1}^m \lambda_{k-i}$
4.	Using the above procedure, determine the stepsize $\alpha_k$ satisfying (2.82) and set $x_{k+1} = x_k + \alpha_k d_k$
5.	Set $k = k + 1$ and go to step 2

The algorithm has the following interesting properties. For any  $k \geq 0$ , it follows that  $g_k^T d_k \leq -(1 - \tau)\|g_k\|^2$ . For any  $k \geq m$ , it follows that  $\|d_k\| \leq \max_{1 \leq i \leq m} \{\|g_k\|, \|d_{k-i}\|\}$ . Moreover, for any  $k \geq 0$ ,  $\|d_k\| \leq \max_{0 \leq j \leq k} \{\|g_j\|\}$ .

**Theorem 2.23** *If the objective function is bounded from below on the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  and the gradient  $\nabla f(x)$  is Lipschitz continuous on an open convex set that contains  $S$ , then Algorithm 2.9 terminates in a finite number of iterates. Moreover, if the algorithm generates an infinite sequence  $\{x_k\}$ , then  $\lim_{k \rightarrow +\infty} \|g_k\| = 0$ .* ◆

The numerical results presented by Ou and Liu (2017) showed that this method is suitable for solving large-scale unconstrained optimization problems and is more stable than other similar methods.

### 13. Weak Wolfe Line-Search with Simple Bisection

The bisection method is a simple method to find a zero of a continuous function for which two values of opposite sign are known. To determine a stepsize  $\alpha_k$  which satisfies the weak Wolfe line-search conditions (2.54) and (2.59), the following simple algorithm based on the *bisection* concept may be used.

**Algorithm 2.10** *Weak Wolfe line-search with simple bisection*

1.	Choose $\alpha_k > 0$ and set $\alpha_k^{low} = \alpha_k^{high} = 0$
2.	If $\alpha_k$ satisfies (2.54), then go to Step 4
3.	Else (if $\alpha_k$ does not satisfy (2.54)), then set: $\alpha_k^{high} = \alpha_k$ and $\alpha_k = (\alpha_k^{low} + \alpha_k^{high})/2$ and go to Step 2
4.	If $\alpha_k$ satisfies (2.59), then stop
5.	Otherwise (if $\alpha_k$ does not satisfy (2.59)), then set: $\alpha_k^{low} = \alpha_k$ and $\alpha_k = \begin{cases} 2\alpha_k^{low}, & \text{if } \alpha_k^{high} = 0, \\ (\alpha_k^{low} + \alpha_k^{high})/2, & \text{if } \alpha_k^{high} > 0, \end{cases}$ and go to Step 2

◆

Let us prove that the above Algorithm 2.10 determines a value for  $\alpha_k$  which satisfies both Wolfe line-search conditions (2.54) and (2.59) in a finite number of steps.

**Theorem 2.24** Suppose that function  $f$  is continuously differentiable on  $\mathbb{R}^n$  and bounded from below on the half-line  $\{x_k + \alpha d_k : \alpha > 0\}$ . Then, Algorithm 2.10 terminates in finite time and generates a value for  $\alpha_k$  that satisfies the weak Wolfe conditions (2.54) and (2.59).

**Proof** Let us define  $\varphi(\alpha) = f(x_k + \alpha d_k)$  and introduce the following two sets:

$$\begin{aligned} S_1 &= \{\alpha > 0 : (2.54) \text{ holds}\}, \\ S_2 &= \{\alpha > 0 : (2.59) \text{ holds}\}. \end{aligned}$$

Observe that both  $S_1$  and  $S_2$  are closed in  $\mathbb{R}_+$ . Moreover, for  $\alpha$  sufficiently small since  $\varphi'$  is continuous and  $\rho < 1$ ,

$$\varphi(\alpha) = \varphi(0) + \int_0^\alpha \varphi'(\tau) d\tau < \varphi(0) + \int_0^\alpha \rho \varphi'(0) d\tau.$$

Therefore, there exists  $\delta_1 > 0$  such that  $[0, \delta_1] \subset S_1$ , i.e., there exists  $\alpha > 0$  satisfying the first Wolfe condition (2.54). Now, consider the second Wolfe condition (2.59). Let  $\alpha > 0$  and two sequences  $\{\alpha_k^{low,[i]}\}_{\mathbb{N}} \subset S_1$  and  $\{\alpha_k^{high,[i]}\}_{\mathbb{N}} \subset S_2$  be such that

$$\alpha_k^{low,[i]} < \alpha \text{ for any } i \in \mathbb{N}, \quad \alpha_k^{low,[i]} \xrightarrow[i \rightarrow \infty]{} \alpha, \tag{2.83}$$

$$\alpha_k^{high,[i]} > \alpha \text{ for any } i \in \mathbb{N}, \quad \alpha_k^{high,[i]} \xrightarrow[i \rightarrow \infty]{} \alpha. \tag{2.84}$$

With this, let us prove that  $\alpha \in S_2$ . Assume that  $\alpha \notin S_2$  and hence  $\varphi'(\alpha) \leq \sigma\varphi'(0)$ . Then, since  $\varphi'$  is continuous and  $\rho < \sigma$ , there exists a value  $\delta_2 > 0$  such that for any  $\theta \in [0, \delta_2]$  it follows that  $\varphi'(\alpha + \theta) \leq \sigma\varphi'(0)$ . Therefore, for all  $\theta \in [0, \delta_2]$  we have

$$\varphi(\alpha + \theta) = \varphi(\alpha) + \int_{\alpha}^{\alpha+\theta} \varphi'(\tau) d\tau < \varphi(0) + (\alpha + \theta)\sigma\varphi'(0).$$

But, since  $\alpha_k^{high,[i]}$  converges to  $\alpha$  from the right, it follows that there exists an index  $j$  large enough so that  $\alpha_k^{high,[j]} \in [\alpha, \alpha + \delta_2]$ , thus contradicting the assumption that  $\alpha_k^{high,[j]} \in S_1$ . Therefore,  $\alpha \in S_2$ .

Now, let us prove that the algorithm terminates in *finite time*. If the algorithm terminates in finite time, then  $\alpha_k$  generated by it satisfies both weak Wolfe line-search conditions. For this, let us define  $\alpha_k^{low,[i]}$ ,  $\alpha_k^{high,[i]}$ , and  $\alpha_k^{[i]}$  as the values of  $\alpha_k^{low}$ ,  $\alpha_k^{high}$ , and  $\alpha_k$  at the beginning of the iteration  $i$  of the algorithm. The following properties can be observed:

1. Observe that for all  $i$  it is impossible that  $\alpha_k^{low,[i]} = 0$ , because in this case  $\alpha_k^{[i]} = 2^{-i}\alpha_k^{[0]}$  and hence  $\alpha_k^{[i]} \in [0, \delta_1]$  and  $\alpha_k^{low}$  is updated to  $\alpha_k^{[i]} > 0$  (see Step 5 of the algorithm).
2. The sequence  $\left\{ \alpha_k^{low,[i]} \right\}_{\mathbb{N}}$  is an increasing one in  $S_1$  such that for all  $i$ ,  $\alpha_k^{low,[i]} < \alpha_k^{[i]}$ .  $\alpha_k^{low}$  can only be updated in Step 5 of Algorithm 2.10. It will be increased to the strictly larger value  $\alpha_k^{low,[i+1]} = \alpha_k^{[i]}$  and as we can see in Step 5 of the algorithm,  $\alpha_k^{[i+1]}$  takes on a strictly larger value than  $\alpha_k^{[i]}$ .
3. Initially, for a few iterations  $\alpha_k^{high,[i]} = 0$ . But, once it has taken on a value  $\alpha_k^{[i_0]} > 0$  at some iteration  $i_0$ , then this can only happen in Step 3 of Algorithm 2.10. Starting with the iteration  $i_0$ , the elements of the sequence  $\alpha_k^{high,[i]}$  with  $i \geq i_0$  decrease in  $S_1$  because  $\alpha_k^{high}$  is only updated in Step 3 of the algorithm to a value of  $\alpha_k$  that is strictly smaller than  $\alpha_k^{high}$  and  $\alpha_k$  is itself updated to a strictly smaller value.
4. All in all, only two possibilities can appear. Either  $\alpha_k^{high,[i]} = 0$  for all  $i$  and then  $\alpha_k^{low,[i]} = 2^{i-1}\alpha_k^{[0]}$  for all  $i$ , in which case Algorithm 2.10 finds that function  $f$  is unbounded. However, this case must be excluded since  $f$  is bounded in the assumptions of the theorem. Or, there exists an index  $i_0 \in \mathbb{N}$  such that  $\alpha_k^{high,[i_0]} > 0$  and therefore  $\alpha_k^{[i]} = (\alpha_k^{low,[i]} + \alpha_k^{high,[i]})/2$ , the sequence  $\left\{ \alpha_k^{low,[i]} \right\}_{\mathbb{N}}$  is increasing, the sequence  $\left\{ \alpha_k^{high,[i]} \right\}_{\mathbb{N}}$  is decreasing, and the interval  $[\alpha_k^{low,[i]}, \alpha_k^{high,[i]}]$  is halved in length at every iteration. Therefore, it follows that  $\alpha_k^{low,[i]}$  converges to a point  $\alpha_k$  from  $S_1$  and  $\alpha_k^{high,[i]}$  converges to the same point in  $S_1$ . In conclusion  $\alpha_k \in S_1 \cap S_2$ . Therefore,  $\alpha_k^{low,[i]} \in S_1 \cap S_2$  for  $i$  sufficiently large, proving that the algorithm terminates with this value.  $\blacklozenge$

Observe that this is a very simple procedure for computing a value of  $\alpha_k$  which satisfies the weak Wolfe line-search conditions (2.54) and (2.59) without any safeguarding to the too large or too small values of  $\alpha_k$ .

#### 14. Barzilai and Borwein Line-Search

A special nonmonotone line-search is the Barzilai and Borwein (1988) method. In this method, the next approximation to the minimum is computed as  $x_{k+1} = x_k - D_k g_k$ ,  $k = 0, 1, \dots$ , where  $D_k = \alpha_k I$ ,  $I$  being the identity matrix. The stepsize  $\alpha_k$  is computed as solution of the problem

$\min_{\alpha_k} \|s_k - D_k y_k\|$  or as solution of  $\min_{\alpha_k} \|D_k^{-1} s_k - y_k\|$ . In the first case,  $\alpha_k = (s_k^T y_k) / \|y_k\|^2$  and in the second one,  $\alpha_k = \|s_k\|^2 / (s_k^T y_k)$ , where  $s_k = x_{k+1} - x_k$  and  $y_k = g_{k+1} - g_k$ . Barzilai and Borwein proved that their algorithm is super-linearly convergent. Many researchers studied the Barzilai and Borwein algorithm, including Raydan (1997), Grippo and Sciandrone (2002), Dai, Hager, Schittkowski, and Zhang (2006), Dai and Liao (2002), Narushima, Wakamatsu, and Yabe, (2008), Liu and Liu (2019), etc.

The nonmonotone line-search methods were investigated by many authors; for example, see Dai (2002b) and the references therein. Observe that all these nonmonotone line-searches concentrate on modifying the first Wolfe condition (2.54). Also the approximate Wolfe line-search (2.67) of Hager and Zhang and the improved Wolfe line-search (2.72) and (2.59) of Dai and Kou modify the first Wolfe condition, responsible for a sufficient reduction of the objective function value. No numerical comparisons among the abovementioned nonmonotone line-searches were given.

As regards the stopping of the iterative scheme (2.4), one of the most popular criteria is  $\|g_k\| \leq \varepsilon$ , where  $\varepsilon$  is a small positive constant and  $\|\cdot\|$  is the Euclidian or  $l_\infty$  norm.

## 2.6 Convergence of the Algorithm with Inexact Line-Search

In this section we present the convergence of the general algorithm for unconstrained optimization with inexact line-search, i.e., Algorithm 2.1, in which the stepsize is computed by using the inexact line-searches. For this, the quality of the search direction needs to be studied. The quality of the search direction is defined by the angle between the search direction and the negative gradient. In order to ensure the convergence of Algorithm 2.1, the situations in which the search direction  $s_k = \alpha_k d_k$  is orthogonal to the negative gradient must be avoided. In other words, the angle  $\theta_k$  between  $s_k$  and  $-\nabla f(x_k)$  is uniformly bounded away from  $90^\circ$ , i.e., for any  $k$ :

$$\theta_k \leq \frac{\pi}{2} - \mu, \quad (2.85)$$

where  $\mu > 0$ . The inequality (2.85) is called *angle condition*. Consider  $g_k = \nabla f(x_k)$ , then the angle  $\theta_k \in [0, \pi/2]$  is defined as

$$\cos \theta_k = - \frac{g_k^T s_k}{\|g_k\| \|s_k\|}. \quad (2.86)$$

Observe that all the procedures for the stepsize computation (except for the Barzilai-Borwein) are based on the Armijo condition (2.51) or on the Wolfe condition (2.54) and (2.59). Therefore, the convergence analysis of Algorithm 2.1 with inexact line-search will be separately presented for the Armijo line-search and for the Wolfe line-search. The following theorem shows the convergence of Algorithm 2.1 in which the stepsize is computed by using the Armijo line-search (2.51) (Griva, Nash, and Sofer, 2009).

**Theorem 2.25** *Let  $f$  be a real-valued function continuously differentiable on  $\mathbb{R}^n$ . Let  $x_0$  be the initial point and consider the sequence  $\{x_k\}$ , defined by  $x_{k+1} = x_k + \alpha_k d_k$ , where  $\alpha_k \geq 0$  is the stepsize, a scalar, and  $d_k$  is the search direction, a vector of dimension  $n$ . Assume that:*

- (i) *The level set  $S = \{x : f(x) \leq f(x_0)\}$  is bounded.*
- (ii) *The gradient  $\nabla f(x)$  is Lipschitz continuous for all  $x$ , i.e.,  $\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$  for some constant  $0 < L < \infty$ .*

(iii) *The vectors  $d_k$  satisfy the sufficient descent condition*

$$-\frac{d_k^T \nabla f(x_k)}{\|d_k\| \|\nabla f(x_k)\|} \geq \varepsilon > 0.$$

(iv) *The search directions  $d_k$  are gradient related as*

$$\|d_k\| \geq m \|\nabla f(x_k)\|$$

*for all  $k$ , where  $m > 0$ , and bounded in norm*

$$\|d_k\| \leq M$$

*for all  $k$ .*

(v) *The stepsize  $\alpha_k$  satisfies the Armijo line-search*

$$f(x_k + \alpha d_k) \leq f(x_k) + \rho \alpha_k d_k^T \nabla f(x_k),$$

*where  $0 < \rho < 1$ .*

*Then,*

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

**Proof** Before presenting the proof, some comments are needed. Observe that the above theorem makes several assumptions to guarantee the convergence of Algorithm 2.1 with Armijo line-search. Two of them refer to the minimizing function: to have a bounded level set and its gradient to be Lipschitz continuous. The other three assumptions refer to the algorithm: the search direction has to satisfy the sufficient descent condition and is gradient related, and the stepsize has to satisfy the Armijo line-search.

The boundedness of the level set ensures that the function takes on its minimum value in a finite point. The Lipschitz continuity of the gradient of  $f$  is a technical assumption.

The search direction must first of all be a descent direction, that is,  $d_k^T \nabla f(x_k) < 0$ . However, the danger is that  $d_k$  satisfying  $d_k^T \nabla f(x_k) < 0$  might become arbitrarily close to being orthogonal to  $\nabla f(x_k)$  while still remaining a descent direction, thus dramatically slowing down the convergence of the algorithm. To avoid this situation, we assume that

$$-\frac{d_k^T \nabla f(x_k)}{\|d_k\| \|\nabla f(x_k)\|} \geq \varepsilon > 0,$$

for all  $k$ , where  $\varepsilon > 0$  is a specified tolerance. From (2.86), this condition can also be written as

$$\cos \theta_k \geq \varepsilon > 0,$$

where  $\theta_k$  is the angle between the search direction  $d_k$  and the negative gradient  $-\nabla f(x_k)$ . In other words, the search direction satisfies the angle condition.

The search directions  $d_k$  are gradient related if  $\|d_k\| \geq m \|\nabla f(x_k)\|$  for all  $k$ , where  $m > 0$  is a constant. In other words, this condition says that the norm of the search direction cannot become too much smaller than that of the gradient.

These conditions on the search direction can normally be guaranteed by the algorithm for minimizing function  $f$ .

Now, the assumptions that the stepsize produces a sufficient decrease in function  $f$  and it is not too small are ensured by the Armijo line-search. This ensures a *nontrivial* reduction in the function value at each iteration. Nontrivial is measured in terms of the Taylor series. More specifically, we require that the stepsize  $\alpha_k$  should produce a decrease in the function value that is at least a certain fraction of the decrease predicted by the linear approximation given by the Taylor series, i.e.,

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k d_k^T \nabla f(x_k),$$

where  $0 < \rho < 1$ . When  $\rho$  is near zero, this condition is easier to satisfy since only a small decrease in the function value is required. (Usually,  $\rho = 0.0001$ .)

With these comments, let us prove the theorem. The proof has five steps. Firstly, it is shown that  $f$  is bounded from below on  $S$ . Secondly, it is proved that  $\lim_{k \rightarrow \infty} f(x_k)$  exists. Thirdly, we prove that the following result  $\lim_{k \rightarrow \infty} \alpha_k \|\nabla f(x_k)\|^2 = 0$  holds. Then we show that if  $\alpha_k < 1$ , then  $\alpha_k \geq \gamma \|\nabla f(x_k)\|^2$  for an appropriate constant  $\gamma > 0$ . Finally, it is proved that  $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ .

1.  $f$  is bounded from below on  $S$ . Since  $f$  is continuous, it follows that the level set  $S$  is closed. But, by the assumption (i), the set  $S$  is bounded. Therefore,  $S$  is a compact. A continuous function on a compact set takes on its minimum value at some point in that set, that is,  $f(x) \geq C$ , for some constant  $C$  (see Theorem A3.1 – Weierstrass Extreme Value Theorem).
2.  $\lim_{k \rightarrow \infty} f(x_k)$  exists. The sufficient descent condition ensures that  $f(x_{k+1}) \leq f(x_k) \leq f(x_0)$  so that  $x_k \in S$  for all  $k$ . The sequence  $\{f(x_k)\}$  is monotone decreasing and bounded from below, so it has a limit, let us say  $\bar{f}$ .
3.  $\lim_{k \rightarrow \infty} \alpha_k \|\nabla f(x_k)\|^2 = 0$ . We have

$$\begin{aligned} f(x_0) - \bar{f} &= [f(x_0) - f(x_1)] + [f(x_1) - f(x_2)] + \cdots \\ &= \sum_{k=0}^{\infty} [f(x_k) - f(x_{k+1})] \geq - \sum_{k=0}^{\infty} \rho \alpha_k d_k^T \nabla f(x_k) \\ &\geq \sum_{k=0}^{\infty} \rho \alpha_k \varepsilon \|d_k\| \|\nabla f(x_k)\| \geq \sum_{k=0}^{\infty} \rho \alpha_k \varepsilon m \|\nabla f(x_k)\|^2. \end{aligned}$$

Since  $f(x_0) - \bar{f} \leq f(x_0) - C < \infty$  it follows that the above final summation is convergent, that is, the terms in this summation go to zero:

$$\lim_{k \rightarrow \infty} \rho \alpha_k \varepsilon m \|\nabla f(x_k)\|^2 = 0.$$

Since  $\rho$ ,  $\varepsilon$ , and  $m$  are fixed nonzero constants, it follows that  $\lim_{k \rightarrow \infty} \alpha_k \|\nabla f(x_k)\|^2 = 0$ .

4. If  $\alpha_k < 1$ , then  $\alpha_k \geq \gamma \|\nabla f(x_k)\|^2$  for an appropriate constant  $\gamma$ . To prove this, we use the backtracking line-search. If  $\alpha_k < 1$ , then the sufficient descent condition is violated when the stepsize  $2\alpha_k$  is tried, that is,

$$f(x_k + 2\alpha_k d_k) - f(x_k) > 2\rho \alpha_k d_k^T \nabla f(x_k).$$

Since  $\nabla f(x)$  is Lipschitz continuous, then by Theorem 2.7 it follows that

$$f(x_k + 2\alpha_k d_k) - f(x_k) - 2\alpha_k d_k^T \nabla f(x_k) \leq \frac{1}{2} L \|2\alpha_k d_k\|^2.$$

That is,

$$f(x_k) - f(x_k + 2\alpha_k d_k) \geq -2\alpha_k d_k^T \nabla f(x_k) - 2L\|\alpha_k d_k\|^2.$$

Combining this inequality with the violated condition when the stepsize was  $2\alpha_k$ , it follows that

$$\alpha_k L \|d_k\|^2 \geq -(1 - \rho) d_k^T \nabla f(x_k).$$

The sufficient descent condition (iii) and the gradient relatedness condition (iv) give

$$\alpha_k L \|d_k\|^2 \geq (1 - \rho) \varepsilon \|d_k\| \|\nabla f(x_k)\| \geq (1 - \rho) \varepsilon m \|\nabla f(x_k)\|^2.$$

Since  $\|d_k\| \leq M$ , it follows that  $\alpha_k \geq \gamma \|\nabla f(x_k)\|^2$  with

$$\gamma = \frac{(1 - \rho) \varepsilon m}{M^2 L} > 0.$$

5.  $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ . Observe that either  $\alpha_k = 1$  or  $\alpha_k \geq \gamma \|\nabla f(x_k)\|^2$ . Therefore,

$$\alpha_k \geq \min \left\{ 1, \gamma \|\nabla f(x_k)\|^2 \right\}.$$

Hence,

$$\alpha_k \|\nabla f(x_k)\|^2 \geq \left[ \min \left\{ 1, \gamma \|\nabla f(x_k)\|^2 \right\} \right] \|\nabla f(x_k)\|^2 \geq 0.$$

But, from step 3 above we know that  $\lim_{k \rightarrow \infty} \alpha_k \|\nabla f(x_k)\|^2 = 0$ . Since  $\gamma > 0$ , it follows that  $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ .  $\blacklozenge$

Now, we come back to see the convergence of Algorithm 2.1 in which the stepsize is determined by the Wolfe line-search (2.54) and (2.59). Let us show a very useful result, due to Zoutendijk (1970). From the proof of this result, we can get a clear idea how the properties of the minimizing function are linked to the Wolfe conditions.

**Theorem 2.26** Suppose that function  $f$  is bounded from below on  $\mathbb{R}^n$  and it is continuous differentiable in a neighborhood  $N$  of the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ . Suppose that the gradient  $g(x)$  off is Lipschitz continuous, i.e., there exists a positive constant  $L > 0$  such that

$$\|g(x) - g(y)\| \leq L \|x - y\| \quad (2.87)$$

for any  $x, y \in N$ . Consider an iteration  $x_{k+1} = x_k + \alpha_k d_k$ , where  $d_k$  is a descent direction and  $\alpha_k$  satisfies the weak Wolfe line-search (2.54) and (2.59). Then

$$\sum_{k \geq 1} \cos^2 \theta_k \|g_k\|^2 < \infty. \quad (2.88)$$

**Proof** From (2.59) we have

$$(g_{k+1} - g_k)^T d_k \geq (\sigma - 1) g_k^T d_k.$$

On the other hand, the Lipschitz condition (2.87) shows that

$$(g_{k+1} - g_k)^T d_k \geq \alpha_k L \|d_k\|^2.$$

Combining these two relations, we have

$$\alpha_k \geq \left( \frac{\sigma - 1}{L} \right) \frac{g_k^T d_k}{\|d_k\|^2}. \quad (2.89)$$

Now, from the first Wolfe condition (2.54) and (2.89), it follows that

$$f(x_{k+1}) \leq f(x_k) + \rho \left( \frac{\sigma - 1}{L} \right) \frac{(g_k^T d_k)^2}{\|d_k\|^2}.$$

From (2.86), the above relation can be rewritten as

$$f(x_{k+1}) \leq f(x_k) + c \cos^2 \theta_k \|g_k\|^2,$$

where  $c = \rho(\sigma - 1)/L$ . Summing these relations for  $k \geq 1$  and since function  $f$  is bounded from below, we get the conclusion of the theorem.  $\blacklozenge$

The relation (2.88) is called *Zoutendijk condition* (Zoutendijk, 1970). It is worth mentioning that this condition is directly involved in proving the global convergence of the general Algorithm 2.1. As we can see, if for any  $k$  the iterations  $x_{k+1} = x_k + \alpha_k d_k$  are in such a way that  $\cos \theta_k \geq \delta > 0$ , then from (2.88) we get  $\lim_{k \rightarrow \infty} \|g_k\| = 0$ . In other words, if the search direction does not tend to be orthogonal to the gradient, then the sequence of the gradients is convergent to zero. Observe that the Zoutendijk condition is a condition on the search direction. If, for example, the search direction is exactly  $-g_k$ , i.e., the steepest descent algorithm introduced for the first time by Cauchy (1847), then  $\cos \theta_k = 1$ . This shows that, in order to obtain the global convergence of the Cauchy algorithm, we need to execute an adequate line-search. This proves the importance of the line-search in the frame of the general Algorithm 2.1 with descent directions.

In this context, the following corollary of the Zoutendijk condition can be proved.

**Corollary 2.1** *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be bounded from below on  $\mathbb{R}^n$  and continuously differentiable in a neighborhood  $N$  of the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ . If  $\{x_k\}$  is the sequence generated by the general Algorithm 2.1 with weak Wolfe line-search, then*

$$\cos \theta_k \|\nabla f(x_k)\| \rightarrow 0. \quad \blacklozenge$$

If  $\cos \theta_k \geq \delta > 0$  for all  $k$ , then, from the Zoutendijk condition, it follows that  $\|\nabla f(x_k)\| \rightarrow 0$ , i.e., the general Algorithm 2.1 with weak Wolfe line-search converges to a stationary point of function  $f$ .

The limit  $\lim_{k \rightarrow \infty} \|g_k\| = 0$  is the best global convergence result we may obtain. Obviously, there is no guarantee that the algorithm converges to the minimum point. However, there is a guarantee that the iterates of the algorithm enter in a domain of attraction of the stationary points.

The following theorem shows the global convergence of Algorithm 2.1 with inexact line-search.

**Theorem 2.27** Suppose that in Algorithm 2.1 the search direction  $d_k$  satisfies the angle condition (2.85) and  $\alpha_k$  is computed by the Goldstein conditions (2.54) and (2.55) or by the weak Wolfe conditions (2.54) and (2.59). If  $\nabla f(x)$  exists and is uniformly continuous on the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ , then  $\nabla f(x_k) = 0$  for a  $k$  or  $f(x_k) \rightarrow -\infty$  or  $\nabla f(x_k) \rightarrow 0$ .

**Proof** Firstly, consider  $\alpha_k$  computed by the Goldstein conditions (2.54) and (2.55). Suppose that for all  $k$ ,  $g_k = \nabla f(x_k) \neq 0$  and  $f(x_k)$  is bounded from below. It follows that  $f(x_k) - f(x_{k+1}) \rightarrow 0$ , that is, from (2.54) we have  $-g_k^T s_k \rightarrow 0$ . Now, suppose that  $g_k \rightarrow 0$  is not true. Then, there exists a subsequence such that  $\|g_k\| \geq \varepsilon$  and  $\|s_k\| \rightarrow 0$ . Since  $\theta_k \leq \pi/2 - \mu$ , we get

$$\cos \theta_k \geq \cos \left( \frac{\pi}{2} - \mu \right) = \sin \mu,$$

i.e.,

$$-g_k^T s_k \geq \|g_k\| \|s_k\| \sin \mu \geq \varepsilon \|s_k\| \sin \mu.$$

Using Taylor's series, we can write

$$f(x_{k+1}) = f(x_k) + \nabla f(\xi_k)^T s_k,$$

where  $\xi_k$  is on the line segment connecting  $x_k$  with  $x_{k+1}$ . From the uniform continuity it follows that  $\nabla f(\xi_k) \rightarrow g_k$  when  $s_k \rightarrow 0$ . Therefore,

$$f(x_{k+1}) = f(x_k) + g_k^T s_k + o(\|s_k\|).$$

Hence,

$$\frac{f(x_k) - f(x_{k+1})}{-g_k^T s_k} \rightarrow 1,$$

which contradicts the second Goldstein condition (2.55). Therefore,  $g_k \rightarrow 0$ , thus proving the global convergence when  $\alpha_k$  is determined by the Goldstein conditions.

The global convergence for the case in which the stepsize is computed by the weak Wolfe conditions is proved similarly. Observe that from the uniform continuity of  $\nabla f(x)$ , it follows that

$$g_{k+1}^T d_k = g_k^T d_k + o(\|d_k\|),$$

such that

$$\frac{g_{k+1}^T d_k}{g_k^T d_k} \rightarrow 1.$$

But, this contradicts the relation  $g_{k+1}^T d_k / g_k^T d_k \leq \sigma < 1$  from (2.59). Therefore,  $g_k \rightarrow 0$ , thus proving the global convergence when  $\alpha_k$  is determined by the Wolfe conditions.  $\blacklozenge$

The convergence of the general unconstrained optimization Algorithm 2.1 with weak Wolfe inexact line-search is presented in the next theorem. Mainly, it is proved that the sequence generated by the algorithm satisfies the Zoutendijk condition.

**Theorem 2.28** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a continuously differentiable function bounded from below. Suppose that  $\nabla f(x)$  is uniformly continuous on the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  and  $\alpha_k$  is

determined by the weak Wolfe conditions (2.54) and (2.59). Then the sequence generated by Algorithm 2.1 satisfies

$$\lim_{k \rightarrow +\infty} \frac{\nabla f(x_k)^T s_k}{\|s_k\|} = 0, \quad (2.90)$$

i.e.,

$$\|\nabla f(x_k)\| \cos \theta_k \rightarrow 0. \quad (2.91)$$

**Proof** Since  $\nabla f(x_k)^T s_k < 0$  and  $f$  is bounded from below, it follows that the sequence  $\{x_k\}$  is well defined and  $\{x_k\} \subset S$ . Moreover, since  $\{f(x_k)\}$  is a decreasing sequence, then it is convergent.

We prove (2.90) by contradiction. Suppose that (2.90) is not true. Then, there exists an  $\varepsilon > 0$  and a subsequence indexed after the set of indices  $K$  such that for any  $k \in K$

$$-\frac{\nabla f(x_k)^T s_k}{\|s_k\|} \geq \varepsilon.$$

But, from (2.54) we have

$$f(x_k) - f(x_{k+1}) \geq \rho \|s_k\| \left( -\frac{\nabla f(x_k)^T s_k}{\|s_k\|} \right) \geq \rho \|s_k\| \varepsilon.$$

Since  $\{f(x_k)\}$  is convergent, it follows that the sequence  $\{s_k : k \in K\}$  is also convergent to zero. On the other hand, from (2.59), for any  $k \geq 0$  we have

$$(1 - \sigma) \left( -\nabla f(x_k)^T s_k \right) \leq (\nabla f(x_k + s_k) - \nabla f(x_k))^T s_k.$$

Therefore, for any  $k \in K$

$$\varepsilon \leq -\frac{\nabla f(x_k)^T s_k}{\|s_k\|} \leq \frac{1}{1 - \sigma} \|\nabla f(x_k + s_k) - \nabla f(x_k)\|. \quad (2.92)$$

Since we proved that  $\{s_k : k \in K\} \rightarrow 0$ , it follows that the right hand side of (2.92) tends to zero by the uniform continuity of  $\nabla f$  on set  $S$ . But this is a contradiction which actually proves the theorem.  $\blacklozenge$

Observe that (2.90) involves  $\|\nabla f(x_k)\| \cos \theta_k \rightarrow 0$ , which is exactly the Zoutendijk condition expressed in (2.88) in another form. The importance of the Zoutendijk condition is if  $\cos \theta_k \geq \delta > 0$ , then  $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ . If the hypothesis of uniform continuity is replaced with the Lipschitz condition of the gradient, then another result on the convergence of Algorithm 2.1 with inexact weak Wolfe line-search is obtained. But, before proving such a result, let us develop a technical result which gives a bound of the reduction of the values of the minimizing function  $f$ .

**Proposition 2.6** Let  $f : D \rightarrow \mathbb{R}$  be a continuously differentiable function on the open set  $D \subset \mathbb{R}^n$  with gradient  $\nabla f(x)$  Lipschitz continuous, i.e.,  $\|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\|$ , where  $L > 0$  is constant. If  $f(x_k + \alpha s_k)$  is bounded from below and  $\alpha > 0$ , then for any  $\alpha_k > 0$  which satisfies the weak Wolfe conditions (2.54) and (2.59), it follows that

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \gamma \|\nabla f(x_k)\|^2 \cos^2 \langle d_k, \nabla f(x_k) \rangle, \quad (2.93)$$

where  $\gamma > 0$  is a constant.

**Proof** From the Lipschitz continuity of  $\nabla f$  and (2.59) we get

$$\alpha_k L \|d_k\|^2 \geq d_k^T (\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)) \geq -(1 - \sigma) d_k^T \nabla f(x_k),$$

i.e.,

$$\begin{aligned} \alpha_k \|d_k\| &\geq \frac{1 - \sigma}{L \|d_k\|} \|d_k\| \|\nabla f(x_k)\| \cos \langle d_k, -\nabla f(x_k) \rangle \\ &= \frac{1 - \sigma}{L} \|\nabla f(x_k)\| \cos \langle d_k, -\nabla f(x_k) \rangle. \end{aligned}$$

Now, using (2.54) it follows that

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &\geq -\rho \alpha_k d_k^T \nabla f(x_k) \\ &= \rho \alpha_k \|d_k\| \|\nabla f(x_k)\| \cos \langle d_k, -\nabla f(x_k) \rangle \\ &\geq \rho \|\nabla f(x_k)\| \cos \langle d_k, -\nabla f(x_k) \rangle \frac{1 - \sigma}{L} \|\nabla f(x_k)\| \cos \langle d_k, -\nabla f(x_k) \rangle \\ &= \frac{\rho(1 - \sigma)}{L} \|\nabla f(x_k)\|^2 \cos^2 \langle d_k, -\nabla f(x_k) \rangle, \end{aligned}$$

which is exactly (2.93) with  $\gamma = \rho(1 - \sigma)/L$ .  $\blacklozenge$

**Theorem 2.29** Let  $f(x)$  be a continuously differentiable function on  $\mathbb{R}^n$  for which its gradient  $\nabla f(x)$  satisfies the Lipschitz condition  $\|\nabla f(y) - \nabla f(x)\| \leq L \|y - x\|$ , where  $L > 0$  is a constant. Let  $\alpha_k$  be the stepsize computed by the weak Wolfe conditions (2.54) and (2.59). If the angle condition (2.85) is satisfied, then, for the sequence  $\{x_k\}$  generated by Algorithm 2.1,  $\nabla f(x_k) = 0$  for a  $k$  or  $f(x_k) \rightarrow -\infty$  or  $\nabla f(x_k) \rightarrow 0$ .

**Proof** Suppose that for any  $k$ ,  $\nabla f(x_k) \neq 0$ . From Proposition 2.6 it follows that

$$f(x_k) - f(x_{k+1}) \geq \gamma \|\nabla f(x_k)\|^2 \cos^2 \theta_k,$$

where  $\gamma = \rho(1 - \sigma)/L$  is a positive constant independent of  $k$ . Then, for any  $k > 0$  we can write

$$f(x_0) - f(x_k) = \sum_{i=0}^{k-1} (f(x_i) - f(x_{i+1})) \geq \gamma \min_{0 \leq i \leq k} \|\nabla f(x_i)\|^2 \sum_{i=0}^{k-1} \cos^2 \theta_i. \quad (2.94)$$

Now, since  $\theta_k$  satisfies the angle condition (2.85), it follows that

$$\sum_{k=0}^{\infty} \cos^2 \theta_k = +\infty.$$

Therefore, from (2.94),  $\nabla f(x_k) \rightarrow 0$  or  $f(x_k) \rightarrow -\infty$ , thus proving the theorem.  $\blacklozenge$

The above theorem is a direct consequence of the Zoutendijk and of the angle conditions. Moreover, all these results have been obtained in mild conditions: function  $f$  is continuously differentiable and its gradient  $\nabla f(x)$  is Lipschitz continuous.

In the following, we point out a result on the reduction of the values of the minimizing function in conditions in which it is uniformly convex. At every iteration Proposition 2.6 gives a bound of the reduction of a general function with the gradient Lipschitz continuous. The estimation (2.93) is expressed in terms of the Zoutendijk condition. Uniform convexity leads us to a new estimation which depends on the bounds of the Hessian of the minimizing function. As it is known, a function  $f(x)$  is *uniformly convex* (or *strong convex*) if there exists a constant  $\eta > 0$  such that

$$(y - z)^T (\nabla f(y) - \nabla f(z)) \geq \eta \|y - z\|^2, \quad (2.95)$$

or there are the positive constants  $m$  and  $M$  ( $m < M$ ) such that

$$m \|y\|^2 \leq y^T \nabla^2 f(x)y \leq M \|y\|^2. \quad (2.96)$$

**Theorem 2.30** *Let  $f(x)$  be a uniformly convex function and Algorithm 2.1 in which  $\alpha_k$  satisfies (2.54). Then*

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \frac{\rho \eta}{1 + \sqrt{M/m}} \|\alpha_k d_k\|^2. \quad (2.97)$$

**Proof** The following two cases must be considered.

(i) Suppose that  $d_k^T \nabla f(x_k + \alpha_k d_k) \leq 0$ . In this case, we can write

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &= \int_0^{\alpha_k} -d_k^T \nabla f(x_k + td_k) dt \\ &\geq \int_0^{\alpha_k} d_k^T (\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k + td_k)) dt \\ &\geq \int_0^{\alpha_k} \|d_k\|^2 \eta (\alpha_k - t) dt = \frac{1}{2} \|\alpha_k d_k\|^2 \eta \\ &\geq \frac{\rho \eta}{1 + \sqrt{M/m}} \|\alpha_k d_k\|^2. \end{aligned}$$

(ii) Suppose that  $d_k^T \nabla f(x_k + \alpha_k d_k) > 0$ . Then there exists  $0 < \alpha^* < \alpha_k$  such that

$$d_k^T \nabla f(x_k + \alpha^* d_k) = 0.$$

Now, from the uniform convexity (2.96) it follows that

$$f(x_k) - f(x_k + \alpha^* d_k) \leq \frac{1}{2} M \|\alpha^* d_k\|^2 \quad (2.98)$$

and

$$f(x_k + \alpha_k d_k) - f(x_k + \alpha^* d_k) \geq \frac{1}{2}m\|(\alpha_k - \alpha^*)d_k\|^2. \quad (2.99)$$

Since  $f(x_k + \alpha_k d_k) < f(x_k)$ , from (2.98) and (2.99) it follows that

$$\alpha_k \leq \left(1 + \sqrt{\frac{M}{m}}\right)\alpha^*.$$

Therefore,

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &\geq -\rho\alpha_k d_k^\top \nabla f(x_k) \\ &\geq \rho\alpha_k d_k^\top (\nabla f(x_k + \alpha^* d_k) - \nabla f(x_k)) \\ &\geq \eta\rho\alpha_k \alpha^* \|d_k\|^2 \geq \frac{\rho\eta}{1 + \sqrt{M/m}} \|\alpha_k d_k\|^2. \end{aligned}$$

Hence, (2.97) is true in both cases.  $\diamond$

The convergence of the general unconstrained optimization algorithm with inexact line-search presented in this section is based on the descent condition  $\nabla f(x_k)^\top d_k < 0$  and on the hypothesis that the search direction  $d_k$  is bounded in norm away from zero. Moreover, the convergence is proved by assuming that the minimizing function  $f$  is at least twice continuously differentiable and its gradient is Lipschitz continuous. Therefore, for smooth enough functions, as we can find in real optimization applications, the above results ensure the convergence of Algorithm 2.1 with inexact line-search.

In the following, we consider another approach of the convergence of Algorithm 2.1 with inexact line-search for uniformly (strong) convex functions when the search direction satisfies a *sufficient descent condition* (Shi, 2004). Indeed, let us assume that the search direction  $d_k$  satisfies the sufficient descent condition

$$g_k^\top d_k \leq -c\|g_k\|^2, \quad (2.100)$$

where  $c > 0$  is a constant.

**Theorem 2.31** *If:*

- (i) *function  $f$  is bounded from below on the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ , where  $x_0 \in \mathbb{R}^n$  is the initial point,*
- (ii) *the gradient  $g(x) = \nabla f(x)$  is Lipschitz continuous on a convex set  $N$  which includes  $S$ , i.e. there exists  $L > 0$  such that  $\|g(x) - g(y)\| \leq L\|x - y\|$ , for any  $x, y \in N$ ,*
- (iii)  *$d_k$  is a descent direction which satisfies  $g_k^\top d_k < 0$ ,*

*then the general unconstrained optimization Algorithm 2.1 with inexact line-searches given by Armijo, Goldstein, the weak Wolfe, or the strong Wolfe generates an infinite sequence  $\{x_k\}$  that satisfies the Zoutendijk condition*

$$\lim_{k \rightarrow \infty} \left( \frac{-g_k^\top d_k}{\|d_k\|} \right)^2 = 0. \quad (2.101)$$

**Proof** For the algorithm of Armijo with inexact line-search (2.51) let  $K_1 = \{k : \alpha_k = \tau_k\}$  and  $K_2 = \{k : \alpha_k < \tau_k\}$ . Then, from (2.51) it follows that

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &\geq -\rho \tau_k g_k^T d_k, \quad k \in K_1, \\ f(x_k) - f(x_k + \alpha_k d_k) &\geq -\rho \alpha_k g_k^T d_k, \quad k \in K_2. \end{aligned}$$

Since for any  $k \in K_2$ ,  $\alpha_k/\beta \leq \tau_k$ , we have

$$f(x_k) - f(x_k + (\alpha_k/\beta)d_k) < -\rho \alpha_k g_k^T d_k / \beta, \quad k \in K_2.$$

By the mean value theorem applied to the left side member of the above inequality, there exists a  $\theta_k \in [0, 1]$  such that

$$-\alpha_k \nabla f(x_k + \alpha_k \theta_k d_k / \beta)^T d_k / \beta < -\rho \alpha_k g_k^T d_k / \beta, \quad k \in K_2,$$

i.e., for any  $k \in K_2$

$$\nabla f(x_k + \alpha_k \theta_k d_k / \beta)^T d_k > \rho g_k^T d_k.$$

From (ii) and from the Cauchy-Schwarz inequality, it follows that for any  $k \in K_2$

$$\begin{aligned} \alpha_k L \|d_k\|^2 / \beta &\geq \|\nabla f(x_k + \alpha_k \theta_k d_k / \beta) - \nabla f(x_k)\| \|d_k\| \\ &\geq (\nabla f(x_k + \alpha_k \theta_k d_k / \beta) - \nabla f(x_k))^T d_k \geq -(1 - \rho) g_k^T d_k. \end{aligned}$$

Therefore, for any  $k \in K_2$

$$\alpha_k \geq -\frac{\beta(1 - \rho) g_k^T d_k}{L \|d_k\|^2}.$$

Hence, for any  $k \in K_2$

$$f(x_k) - f(x_k + \alpha_k d_k) > \frac{\beta \rho (1 - \rho)}{L} \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2.$$

But,  $\tau_k = -(\nabla f(x_k)^T d_k) / \|d_k\|^2$ . Hence, for any  $k \in K_1$

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \rho \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2.$$

If we set  $\eta = \min \{\rho, \beta \rho (1 - \rho)/L\}$ , then, from the above relations we get

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \eta \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2.$$

From (i) we get (2.101), since the search direction is a descent one.

For the algorithm of Goldstein (2.55), by using the mean value theorem as well as the Cauchy-Schwarz inequality, it follows that there exists  $\theta_k \in [0, 1]$ , such that

$$\begin{aligned} (1 - \rho) \alpha_k \nabla f(x_k)^T d_k &\leq f(x_k + \alpha_k d_k) - f(x_k) = \alpha_k \nabla f(x_k + \alpha_k \theta_k d_k)^T d_k \\ &= \alpha_k [\nabla f(x_k + \alpha_k \theta_k d_k) - \nabla f(x_k)]^T d_k + \alpha_k \nabla f(x_k)^T d_k \\ &\leq \alpha_k \nabla f(x_k)^T d_k + \alpha_k \|\nabla f(x_k + \alpha_k \theta_k d_k) - \nabla f(x_k)\| \|d_k\| \\ &\leq \alpha_k \nabla f(x_k)^T d_k + \alpha_k^2 L \|d_k\|^2. \end{aligned}$$

Therefore,

$$\alpha_k \geq \frac{\rho}{L} \left( \frac{-g_k^T d_k}{\|d_k\|^2} \right).$$

Now, taking into consideration (2.54), we get

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \frac{\rho^2}{L} \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2.$$

Therefore, from (i) it follows that the relation (2.101) is true for Algorithm 2.1 with Goldstein inexact line-search.

For the general algorithm with weak Wolfe inexact line-search, from (ii), from (2.59) and from the Cauchy-Schwarz inequality we get

$$\begin{aligned} \alpha_k L \|d_k\|^2 &\geq \|\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)\| \|d_k\| \\ &\geq [\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)]^T d_k \geq -(1 - \sigma) g_k^T d_k. \end{aligned}$$

Therefore,

$$\alpha_k \geq \frac{1 - \sigma}{L} \left( \frac{-g_k^T d_k}{\|d_k\|^2} \right).$$

Now, from (2.54) it follows that

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \frac{\rho(1 - \sigma)}{L} \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2.$$

Therefore, from (ii) it follows that (2.101) is also true for Algorithm 2.1 with inexact weak Wolfe line-search.  $\blacklozenge$

In conclusion, for the inexact line-search procedures of Armijo, Goldstein, and Wolfe, we proved that

1. the stepsize  $\alpha_k$  is lower bounded, and
2. for any  $k$

$$f(x_k + \alpha_k d_k) - f(x_k) \leq -\eta_0 \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2, \quad (2.102)$$

where  $\eta_0 > 0$ . These two results are very important and are to be used in proving the convergence of the general Algorithm 2.1.

In order to establish the convergence of Algorithm 2.1, in the following we shall consider the class of *strongly convex functions*. We mention that the strongly convex functions represent a natural class of functions with very good properties that may be used to prove convergence theorems. If  $f$  is twice continuously differentiable and strongly convex on  $\mathbb{R}^n$ , then the hypotheses (i) and (ii) of Theorem 2.31 are satisfied,  $f(x)$  has an unique minimum point  $x^*$ , and there exist the constants  $0 < m \leq M$  such that for any  $x, y \in \mathbb{R}^n$

$$m\|y\|^2 \leq y^T \nabla^2 f(x)y \leq M\|y\|^2, \quad (2.103)$$

$$\frac{1}{2}m\|x - x^*\|^2 \leq f(x) - f(x^*) \leq \frac{1}{2}M\|x - x^*\|^2, \quad (2.104)$$

$$m\|x - y\|^2 \leq (\nabla f(x) - \nabla f(y))^T(x - y) \leq M\|x - y\|^2, \quad (2.105)$$

i.e.,

$$m\|x - x^*\|^2 \leq \nabla f(x)^T(x - x^*) \leq M\|x - x^*\|^2. \quad (2.106)$$

From the Cauchy-Schwarz inequality and the mean value theorem applied to (2.106), we get

$$m\|x - x^*\| \leq \|\nabla f(x)\| \leq M\|x - x^*\|. \quad (2.107)$$

**Theorem 2.32** Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a strongly convex function, twice continuously differentiable and

$$g_k^T d_k \leq -\tau \|g_k\| \|d_k\|, \quad (2.108)$$

where  $0 < \tau \leq 1$ , then the general Algorithm 2.1 with inexact line-search procedures of Armijo, Goldstein, and weak Wolfe generates an infinite sequence  $\{x_k\}$  such that

$$\lim_{k \rightarrow \infty} \|g_k\| = 0, \quad (2.109)$$

and  $\{x_k\}$  converges at least linearly to  $x^*$ .

**Proof** Since  $f$  is strongly convex, it follows that there exist the constants  $0 < m \leq M$  such that for any  $x \in \mathbb{R}^n$

$$M\|x - x^*\| \geq \|\nabla f(x)\| \geq m\|x - x^*\|.$$

From (2.105) observe that  $m \leq L$ . From (2.102), by using the new descendent condition (2.108), we can write

$$\begin{aligned} f(x_k) - f(x_k + \alpha_k d_k) &\geq \eta_0 \left( \frac{-g_k^T d_k}{\|d_k\|} \right)^2 = \eta_0 \left( \frac{-g_k^T d_k}{\|d_k\| \|g_k\|} \right)^2 \|g_k\|^2 \\ &\geq \eta_0 \tau^2 \|g_k\|^2 \geq \eta_0 \tau^2 m^2 \|x_k - x^*\|^2 \geq \eta_0 \tau^2 m^2 \frac{2}{M} (f(x_k) - f(x^*)). \end{aligned}$$

Denote:

$$\gamma = \sqrt{2 \frac{\eta_0 \tau^2 m^2}{M}} \text{ and } \zeta = \sqrt{1 - \gamma^2}.$$

Hence

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \gamma^2 (f(x_k) - f(x^*)). \quad (2.110)$$

Now, let us prove that for the Armijo, Goldstein, and weak Wolfe inexact line-search,  $\gamma \leq 1$ . For the Armijo inexact line-search, since

$$\eta_0 = \min \{\rho, \beta\rho(1 - \rho)/L\} \leq \beta\rho(1 - \rho)/L,$$

it follows that

$$\gamma^2 = 2 \frac{\eta_0 \tau^2 m^2}{M} \leq \frac{2\beta\rho(1 - \rho)\tau^2 m^2}{LM} \leq \frac{\beta(1 - \rho)\tau^2 m}{M} \leq \beta\tau^2(1 - \rho) < 1.$$

For the Goldstein inexact line-search,  $\eta_0 = \rho^2/L$ , then

$$\gamma^2 = 2 \frac{\eta_0 \tau^2 m^2}{M} = \frac{2\rho^2 \tau^2 m^2}{LM} \leq \frac{\rho \tau^2 m}{M} \leq \tau^2 \rho < 1.$$

For the weak Wolfe inexact line-search,  $\eta_0 = \rho(1 - \sigma)/L$ , and therefore

$$\gamma^2 = 2 \frac{\eta_0 \tau^2 m^2}{M} = \frac{2\rho(1 - \sigma)\tau^2 m^2}{LM} \leq \frac{(1 - \sigma)\tau^2 m}{M} \leq (1 - \sigma)\tau^2 < 1.$$

Therefore, for all these inexact line-search procedures,  $\gamma \leq 1$ . Now, from (2.110) it follows that

$$f(x_{k+1}) - f(x_k) \leq -\gamma^2(f(x_k) - f(x^*)),$$

or

$$f(x_k) - f(x_{k-1}) \leq -\gamma^2(f(x_{k-1}) - f(x^*)).$$

Subtracting  $f(x^*)$  from both members of the above inequality, we get

$$\begin{aligned} f(x_k) - f(x^*) &\leq (1 - \gamma^2)(f(x_{k-1}) - f(x^*)) \\ &= \zeta^2(f(x_{k-1}) - f(x^*)) \leq \dots \leq \zeta^{2k}(f(x_0) - f(x^*)). \end{aligned}$$

But,

$$\begin{aligned} \|x_k - x^*\|^2 &\leq \frac{2}{m}(f(x_k) - f(x^*)) \leq \frac{2}{m}\zeta^{2k}(f(x_0) - f(x^*)) \\ &= \frac{2(f(x_0) - f(x^*))}{m}\zeta^{2k} = \omega^2\zeta^{2k}, \end{aligned}$$

where

$$\omega^2 = \frac{2(f(x_0) - f(x^*))}{m}.$$

Therefore,  $\|x_k - x^*\| \leq \omega\zeta^k$ , thus establishing the linear convergence of the sequence  $\{x_k\}$  to  $x^*$ .  $\blacklozenge$

The above theorem shows that for strongly convex continuously differentiable functions, if the search direction  $d_k$  satisfies the descent condition (2.108), then the general unconstrained optimization algorithm with inexact line-search procedures of Armijo, Goldstein, and Wolfe for minimizing the function  $f$  is convergent in the sense of (2.109). It is possible to prove that the theorem is also true in case in which the condition (2.108) is replaced by the sufficient descent condition  $g_k^T d_k \leq -c \|g_k\|^2$ , where  $c$  is a positive constant.

## 2.7 Three Fortran Implementations of the Inexact Line-Search

For the stepsize computation, the most used algorithms are based on the *inexact line-search*. In these algorithms  $\alpha_k$  is computed to approximately minimize  $f$  along the ray  $\{x_k + \alpha d_k : \alpha \geq 0\}$  or at least to sufficiently reduce  $f$ . Plenty of inexact line-search procedures were proposed: Goldstein (1965), Armijo (1966), Wolfe (1969, 1971), Powell (1976), Dennis and Schnabel (1983), Fletcher (1987), Potra and Shi (1995), Lemaréchal (1981), Shanno (1983), Moré and Thuente (1990), Hager and Zhang (2005), and many others.

Any line-search procedure requires an initial estimate  $\alpha_0$  of the stepsize and generates a sequence  $\{\alpha_k\}$  that converges to a stepsize satisfying the conditions imposed by the user (e.g., Armijo, Goldstein, Wolfe, etc.), or it informs that such a stepsize does not exist. Mainly, line-search procedures have two phases: a *bracketing phase* that finds an interval  $[a, b]$  containing an acceptable value for the stepsize and a *selection phase* that locates the final stepsize. The selection phase reduces the bracketing interval during the search and usually interpolates some values of the function and its gradient obtained from the earlier steps of searching in order to guess the location of the minimum.

In the following, we shall present three implementations of the inexact line-search: *backtracking* (Armijo), the *Wolfe line-search with simple bisection*, and the *Wolfe line-search with safeguarding and cubic interpolation*. They enable the reader to see the subtlety and refinement of the implementation in the computing programs of line-search algorithms.

1. *Backtracking (Armijo's line-search)*. Figure 2.1 presents a Fortran implementation of the Armijo line-search.

The Armijo line-search with backtracking is not efficient and robust for solving large-scale unconstrained optimization problems. This was introduced here just to show that it is an important component of the Wolfe line-search conditions. The Armijo line-search does not guarantee reasonable progress of the line-search. Theorem 2.15 only shows that (2.52) is always satisfied for sufficiently small  $\alpha_k$ , provided that  $d_k$  is a descent direction, but we are not sure that  $\alpha_k$  becomes unacceptably small. In order to avoid such cases, the curvature condition has to be introduced.

2. *Wolfe line-search with simple bisection*. Figure 2.2 presents a Fortran implementation of the Wolfe line-search with simple bisection. This is a simple procedure for computing a value of  $\alpha_k$  which satisfies the Wolfe line-search conditions (2.54) and (2.59) without any safeguarding against the too large or too small values of  $\alpha_k$ .
3. *Wolfe line-search with safeguarding and cubic interpolation*. This is a *variant* of a line-search procedure which is simple enough to generate safeguarded stepsizes satisfying the Wolfe conditions (2.54) and (2.59) (proposed by Shanno, (1983), with minor modifications by Andrei, 1995). Suppose that we are at the iteration  $k$ . In order to have a simple interpretation of the procedure and a clear description, a Fortran version of it is presented in Fig. 2.3.

The inputs of this procedure are  $n$  the number of variables,  $x = x_k$  a vector with the current values of variables,  $f = f(x_k)$  the value of the minimizing function in  $x$ ,  $d = d_k$  the current search direction,  $gtd = \nabla f(x_k)^T d_k$  the scalar product of the current gradient and the search direction, and  $dnorm = \|d_k\|$  the  $l_2$ -norm of the search direction. The outputs of the procedure are  $alpha = \alpha_k$  the stepsize satisfying the Wolfe line-search conditions,  $xnew = x_{k+1} = x_k + \alpha_k d_k$  the new point,  $fnew = f(x_{k+1})$  the function value in the new point,  $gnew = \nabla f(x_{k+1})$  the gradient of the minimizing function in the new point,

$f_{\text{gcnt}}$  the number of function and its gradient calls,  $l_{\text{scnt}}$  indicates that the line-search procedure performed a number of iterations, and  $l_{\text{flag}}$  indicates that the number of iterations in the line-search procedure is greater than a prespecified threshold.

In Fig. 2.3, **maxls** is the maximum number of iterations in the line-search procedure, **epsm** is the epsilon machine, and **evalfg(n, xnew, fnew, gnew)** is the subroutine that implements the algebraic expressions of the minimizing function and its gradient. In the input, this subroutine has **n** as the number of variables and **xnew** as the new point. In the output, it computes **fnew** as the value of function  $f$  in the new point and **gnew** as the gradient of  $f$  in the new point.

We can see that a line-search procedure is complicated. To be reliable, it must incorporate a lot of features. Firstly, observe that the Wolfe conditions are implemented in a complicated form, which takes into consideration both the ratio between the rate of the decrease of  $f$  in the direction  $d_k$  at the new point and the rate of the decrease in the direction  $d_k$  at the current point  $x_k$ . It also takes some precautions to avoid too small or too large values of the stepsize. Observe that in the selection phase of the procedure, the cubic interpolation is used. The cubic interpolation provides a good model for the minimizing function in the searching interval. Suppose we have an interval  $[\bar{a}, \bar{b}]$  containing the desirable stepsize and two previous stepsizes estimates  $\alpha_{i-1}$  and  $\alpha_i$  in this interval. A cubic function is used to interpolate the values,  $\varphi_k(\alpha_{i-1})$ ,  $\varphi_k(\alpha_i)$ ,  $\varphi'_k(\alpha_{i-1})$ , and  $\varphi'_k(\alpha_i)$ , where  $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ . Bulirsch and Stoer (1980) proved that such a cubic function always exists and is unique. The minimizer of this cubic function in  $[\bar{a}, \bar{b}]$ , that is, a new estimation of the stepsize, is either at one of the endpoints or in the interior, case in which it is given by

```

*-----*
* Subroutine back(n,x,d,fx,gtd,rho,sigma, alpha,xnew,fxnew,gradnew,
*                  *          fgcnt,lscnt, nexp)
*-----*
* Subroutine back implements the backtracking algorithm. (Armijo)
*
* Input parameters:
* =====
* n      (integer)   the number of variables in the function to be
*                   minimized.
* x      (double)    vector containing the current estimation to the
*                   minimizer.
* d      (double)    vector containing the current searching direction.
* fx     (double)    function value in the current point.
* gtd    (double)    scalar product: (gradient)T * direction.
* rho    (double)    rho parameter in backtracking.
* sigma(double)    parameter for reducing in backtracking.
* nexp   (integer)   the number of the numerical experiment.
*
* Output parameters:
* =====
* alpha  (double)   steplength satisfying the backtracking condition.
* xnew   (double)   vector containing the new estimation to the minimum
*                   of function: xnew(i) = x(i) + alpha*d(i).
* fxnew  (double)   function value in xnew.
* gradnew (double)  the gradient in xnew.
* fgcnt  (integer)  the number of function and its gradient evaluations.
* lscnt  (integer)  the number of linear searching.
*
* Calling subroutines:
* =====
* call evalfg(n,x,f,g)
* This is a user supplied subroutine which calculates the function and
* its gradient at x and places them in f and g respectively.
*
* Remark:

```

Fig. 2.1 Subroutine *back* which implements backtracking (Armijo)

```

* =====
* Usually, the subroutine back is called with the following values of
* rho and sigma parameters: rho = 0.0001 and sigma = 0.8.
* These values do not have a significative influence on the performances
* of the searching subroutine.
*-----.

      real*8 x(n),d(n), xnew(n), gradnew(n)
      real*8 fx, fxnew, fy, gtd
      real*8 rho, sigma, alpha
      real*8 y(100000), gy(100000)
      integer fgcnt, lsiter
*
      alpha=1.d0
      lsiter=1
*
1     continue
      do i=1,n
        y(i)=x(i) + alpha*d(i)
      end do
c1 ----- ! function evaluation
      call evalfg(n,y,fy,gy, nexp)
      fgcnt = fgcnt + 1

      if(fy .le. fx + rho*alpha*gtd) go to 10

      if(fy .gt. fx + rho*alpha*gtd) then
        alpha = alpha * sigma
        lsiter=lsiter+1
        go to 1
      end if
*
10    continue

      do i=1,n
        xnew(i) = x(i) + alpha*d(i)
      end do
c2 ----- ! function evaluation
      call evalfg(n,xnew,fxnew,gradnew, nexp)
      fgcnt = fgcnt + 1

      if ( lsiter .ne. 0 ) then
        lscnt = lscnt + 1
      end if

      return
end

```

**Fig. 2.1** (continued)

$$\alpha_{i+1} = \alpha_i - (\alpha_i - \alpha_{i-1}) \left[ \frac{\varphi'_k(\alpha_i) + b - a}{\varphi'_k(\alpha_i) - \varphi'_k(\alpha_{i-1}) + 2b} \right], \quad (2.111)$$

where

$$a = \varphi'_k(\alpha_{i-1}) + \varphi'_k(\alpha_i) - 3 \frac{\varphi_k(\alpha_{i-1}) - \varphi_k(\alpha_i)}{\alpha_{i-1} - \alpha_i}, \quad (2.112)$$

$$b = (a^2 - \varphi'_k(\alpha_{i-1})\varphi'_k(\alpha_i))^{1/2}. \quad (2.113)$$

In Fig. 2.3 the new estimate  $\alpha_{i+1}$  is computed as **alphatemp**. The interpolation process can be repeated by discarding the data at one of the stepsizes  $\alpha_{i-1}$  or  $\alpha_i$  and replacing it by  $\varphi_k(\alpha_{i+1})$  and  $\varphi'_k(\alpha_{i+1})$ . Observe that the interpolation step that determines a new estimation to the stepsize is

```

*
*      subroutine LSbis(n,x,f,d,gtd,alpha,xnew,fnew,gnew,fgcnt, nexp)
*-----
* Subroutine LSbis implements the Wolfe line-search conditions with
* bisection. (Andrei)
*
* Input parameters:
* =====
* n      (integer)    the number of variables in the function to be
*                   minimized.
* x      (double)     vector containing the current estimation to the
*                   minimizer.
* f      (double)     the function value in current point.
* d      (double)     vector containing the current searching direction.
* gtd   (double)     scalar product: (gradient)T * direction.
* nexp  (integer)    the number of the numerical experiment.
*
* Output parameters:
* =====
* alpha  (double)    steplength satisfying the backtracking condition.
* xnew   (double)    vector containing the new estimation to the minimum
*                   of function: xnew(i) = x(i) + alpha*d(i).
* fxnew  (double)    function value in xnew.
* gnew   (double)    the gradient in xnew.
* fgcnt (integer)   the number of function and gradient evaluations.
*
* Calling subroutines:
* =====
* call evalfg(n,x,f,g)
* This is a user supplied subroutine which calculates the function and
* its gradient at x and places them in f and g respectively.
*
* Remark:
* =====
* The Wolfe line-search conditions use two parameters. In this subroutine
* these parameters are cw1=0.0001d0 used in the first Wolfe condition
* and cw2=0.8d0 used in the second Wolfe condition.
*-----.

      parameter(ia=50000)
      double precision x(n), d(n), xnew(n), gnew(n)
      double precision xtemp(ia), ftemp, gtemp(ia)
      double precision f, fnew, gtd, gtd1
      double precision alpha, alphalow, alphahigh
      integer fgcnt, nexp, ils

c      alpha      = 1.d0
c      alphalow   = 0.d0
c      alphahigh  = 0.d0

      cw1 = 0.0001d0
      cw2 = 0.8d0

      ils = 0

10    continue
      ils = ils + 1
      if(ils .gt. 20) return

      do i=1,n
        xtemp(i) = x(i) + alpha*d(i)
      end do
      call evalfg(n,xtemp,ftemp,gtemp, nexp)
      fgcnt = fgcnt + 1

c Test W1 (The First Wolfe condition)

      if(ftemp .le. f + cw1*alpha*gtd) then
        go to 50
      else

```

**Fig. 2.2** Subroutine *LSbis* which generate stepsizes satisfying the weak Wolfe line-search with simple bisection

```

        alphahigh = alpha
        alpha = (alphalow + alphahigh)/2.d0
        go to 10
    end if

50    continue

    gtd1 = 0.d0
    do i=1,n
        gtd1 = gtd1 + gtemp(i)*d(i)
    end do

c Test W2 (The second Wolfe condition)

    if(gtd1 .ge. cw2*gtd) then
        go to 100
    else
        alphalow = alpha
        if(alphahigh .eq. 0.d0) alpha = 2.d0*alphalow
        if(alphahigh .gt. 0.d0) alpha = (alphalow+alphahigh)/2.d0
    end if

    go to 10

100   continue

    do i=1,n
        xnew(i) = x(i) + alpha*d(i)
    end do

    call evalfg(n,xnew,fnew,gnew, nexp)
    fgcnt = fgcnt + 1

    return
end

```

**Fig. 2.2** (continued)

```

*
*      subroutine wolfeLS (n,x,f,d,gtd,dnorm, alpha,xnew,fnew,gnew,
*      +                           fgcnt,lscnt,lsflag, nexp)
*-----
* Subroutine wolfeLS implements the weak Wolfe line-search conditions.
*
C      SCALAR ARGUMENTS
      integer n,fgcnt,lscnt,lsflag
      double precision f,gtd,dnorm,alpha,fnew

C      ARRAY ARGUMENTS
      double precision x(n),d(n),xnew(n),gnew(n)

C      LOCAL SCALARS
      integer i,lsiter
      double precision alphap,alphatemp,fp,dp,gtdnew,a,b

      lsflag = 0
*
* Maximum number of Line-Search is maxls (now=20)
* maxls=20

```

**Fig. 2.3** Subroutine *wolfeLS* which generate safeguarded stepsizes satisfying the inexact weak Wolfe line-search with cubic interpolation

```

alphan = 0.0d0
fp      = f
dp      = gtd

do i = 1,n
    xnew(i) = x(i) + alpha * d(i)
end do
c1 ----- ! function evaluation
call evalfg(n,xnew,fnew,gnew, nexp)
fgcnt = fgcnt + 1

gtdnew = 0.0d0
do i = 1,n
    gtdnew = gtdnew + gnew(i) * d(i)
end do

lsiter = 0

* Test whether the Wolfe line-search conditions have been met.

10   if ( alpha * dnorm .gt. 1.0d-30 .and. lsiter .lt. maxls .and.
+     .not. ( gtdnew .eq. 0.0d0 .and. fnew .lt. f ) .and.
+     ( ( fnew .gt. f + 1.0d-04 * alpha * gtd .or.
+       abs( gtdnew / gtd ) .gt. 0.9d0 ) .or. ( lsiter .eq. 0 .and.
+       abs( gtdnew / gtd ) .gt. 0.5d0 ) ) ) then

* Test whether the new point has a negative slope and a higher function
* value than that corresponding to alpha=0. In this case, the search has
* passed through a local max and is heading for a distant local minimum.
* Reduce alpha, compute the new corresponding point, its function value
* and gradient, as well as gtdnew.
* Repeat this test until a good point has been found.

20   if ( alpha * dnorm .gt. 1.0d-30 .and. fnew .gt. f .and.
+     gtdnew .lt. 0.0d0 ) then

        alpha = alpha / 3.0d0

        do i = 1,n
            xnew(i) = x(i) + alpha * d(i)
        end do
c2 ----- ! function evaluation
call evalfg(n,xnew,fnew,gnew, nexp)
fgcnt = fgcnt + 1

        gtdnew = 0.0d0
        do i = 1,n
            gtdnew = gtdnew + gnew(i) * d(i)
        end do

        alphan = 0.0d0
        fp      = f
        dp      = gtd

        goto 20

    end if

* Cubic interpolation to find a new trial point corresponding to alphatemp.

        a = dp + gtdnew - 3.0d0 * ( fp - fnew ) / ( alphan - alpha )
        b = a ** 2 - dp * gtdnew

        if ( b .gt. 0.0d0 ) then
            b = sqrt( b )
        else
            b = 0.0d0
        end if

        alphatemp = alpha - ( alpha - alphan ) * ( gtdnew + b - a ) /
+                   ( gtdnew - dp + 2.0d0 * b )

```

Fig. 2.3 (continued)

```

* Test whether the line minimum has been bracketed.
if ( gtdnew / dp .le. 0.0d0 ) then

* Here the minimum has been bracketed.
* Test whether the trial point lies sufficiently within the bracketed
* interval. If it does not, choose alphatemp as the midpoint of the
* interval.

      if ( 0.99d0 * max( alpha, alphap ) .lt. alphatemp .or.
+          alphatemp .lt. 1.01d0 * min( alpha, alphap ) ) then
          alphatemp = ( alpha + alphap ) / 2.0d0
      end if

      else

* Here the minimum has not been bracketed.
* The trial point is too small, double the largest prior point.

      if ( gtdnew .lt. 0.0d0 .and.
+          alphatemp .lt. 1.01d0 * max( alpha, alphap ) ) then
          alphatemp = 2.0d0 * max( alpha, alphap )
      end if

* The trial point is too large, halve the smallest prior point.

      if ( ( gtdnew .gt. 0.0d0 .and.
+          alphatemp .gt. 0.99d0 * min( alpha, alphap ) ) .or.
+          alphatemp .lt. 0.0d0 ) then
          alphatemp = min( alpha, alphap ) / 2.0d0
      end if

      end if

* Save and continue the search.

      alphap = alpha
      fp    = fnew
      dp    = gtdnew

      alpha = alphatemp

      do i = 1,n
          xnew(i) = x(i) + alpha * d(i)
      end do
c3 ----- ! function evaluation
call evalfg(n,xnew,fnew,gnew, nexp)
fgcnt = fgcnt + 1

      gtdnew = 0.0d0
      do i = 1,n
          gtdnew = gtdnew + gnew(i) * d(i)
      end do

      lsiter = lsiter + 1

      goto 10

c----- End if for stopping criteria
      end if

      if ( lsiter .ge. maxls ) then
          lsflag = 1
      end if

      if ( lsiter .ne. 0 ) then
          lscnt = lscnt + 1
      end if

      return
end

```

Fig. 2.3 (continued)

safeguarded in order to ensure that the new stepsize is not too close to the endpoints of the interval. Some more details may be found, for example, in (Dennis, & Schnabel, 1983), (Shanno, 1983), and (Nocedal, & Wright, 2006).

## 2.8 Numerical Studies: Stepsize Computation

As we have already seen, for the stepsize  $\alpha_k$  computation along the descent direction  $d_k$ , there are a lot of algorithms for which plenty of theoretical developments subject to their definition and convergence were presented. The problem we face, problem which is one of the greatest provocations of any theory, is that of its implementation in computer programs able to solve a large diversity of optimization problems with different structures and complexities. Therefore, in the following, we shall consider three line-search algorithms for the stepsize computation: *backtracking* (see subroutine *back* in Fig. 2.1), the *weak Wolfe line-search* (Shanno-Phua) (Shanno, & Phua, 1976, 1980), (Shanno, 1983), (see subroutine *WolfeLS* in Fig. 2.3), and the *strong Wolfe line-search* (Moré-Thuente) (Moré, & Thuente, 1992; Andrei, 2021b) (see subroutines *MTlines* in Andrei, 2021b, pp. 15–22). We want to see their performances in the context of two unconstrained optimization algorithms: the *steepest descent* and the *Hestenes-Stiefel conjugate gradient*. We mention that these unconstrained optimization algorithms will be analyzed in Chaps. 3 and 5, respectively.

Firstly, let us consider the steepest descent algorithm equipped with the above three line-searches. The search direction of the steepest descent is  $d_k = -g_k$ .

**Example 2.1** Consider the function

$$f_1(x) = \sum_{i=1}^{n-1} (x_i - 1)^2 + \left( \sum_{j=1}^n x_j^2 - 0.25 \right)^2$$

with the initial point  $x_0 = [1, 2, \dots, n]$ , and  $n = 100$ . The performances of the steepest descent algorithm with the above three line-searches are presented in Table 2.1.

**Example 2.2** For the function

$$f_2(x) = (3x_1 - 2x_1^2)^2 + \sum_{i=2}^{n-1} (3x_i - 2x_i^2 - x_{i-1} - 2x_{i+1} + 1)^2 + (3x_n - 2x_n^2 - x_{n-1} + 1)^2$$

with the initial point  $x_0 = [-1, -1, \dots, -1]$  and  $n = 100$ , Table 2.2 shows the performances of the steepest descent algorithm equipped with three line-searches.

**Example 2.3** Now, consider the function

$$f_3(x) = \sum_{i=1}^{n/2} (1.5 - x_{2i-2} + x_{2i-1}x_{2i})^2 + (2.25 - x_{2i-1} + x_{2i-1}x_{2i}^2)^2 + (2.625 - x_{2i-1} + x_{2i-1}x_{2i}^3)^2$$

with the initial point  $x_0 = [1, 0.8, \dots, 1, 0.8]$  and  $n = 100$ . Table 2.3 presents the performances of the steepest descent algorithm equipped with three line-searches.

**Table 2.1** Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Steepest descent algorithm. Function  $f_1(x)$ 

Line-search	#iter	#fg	cpu(s)	$f^*$	$\ g\ $
Backtracking	137	2836	62.47	75	0.953999e-06
Weak Wolfe	14	48	1.18	75	0.680075e-06
Strong Wolfe	20	103	2.39	75	0.248833e-06

**Table 2.2** Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Steepest descent algorithm. Function  $f_2(x)$ 

Line-search	#iter	#fg	cpu(s)	$f^*$	$\ g\ $
Backtracking	607	13665	330.81	0.397067	0.827037e-06
Weak Wolfe	78	195	4.58	0.222e-13	0.826909e-06
Strong Wolfe	93	289	6.85	0.1575e-13	0.797402e-06

**Table 2.3** Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Steepest descent algorithm. Function  $f_3(x)$ 

Line-search	#iter	#fg	cpu(s)	$f^*$	$\ g\ $
Backtracking	1142	19617	477.28	0.1254e-12	0.984514e-06
Weak Wolfe	1087	2218	52.27	0.5976e-12	0.681328e-06
Strong Wolfe	300	898	21.73	0.1337e-11	0.928115e-06

**Table 2.4** Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Hestenes-Stiefel conjugate gradient algorithm. Function  $f_1(x)$ 

Line-search	#iter	#fg	cpu(s)	$f^*$	$\ g\ $
Backtracking	229	7977	132.15	9453.238852	0.606498e-04
Weak Wolfe	115	2912	52.94	9453.238852	0.127230e-05
Strong Wolfe	23	107	2.20	9453.238852	0.576972e-05

In the next numerical experiment, let us see the performances of the three line-search algorithms for the stepsize  $\alpha_k$  computation (backtracking, weak Wolfe, strong Wolfe) in the context of the Hestenes-Stiefel conjugate gradient algorithm. The search direction in this algorithm is computed as  $d_{k+1} = -g_k + \beta_k d_k$ , in which the conjugate gradient parameter  $\beta_k$  is computed as  $\beta_k = g_{k+1}^T y_k / y_k^T d_k$ , where  $y_k = g_{k+1} - g_k$ . The Hestenes-Stiefel conjugate gradient will be presented and analyzed in Chap. 5.

**Example 2.4** Table 2.4 shows the performances of the line-search algorithms, backtracking, weak Wolfe, and strong Wolfe, for the minimization of function  $f_1(x)$  presented in Example 2.1 with the same initial point and  $n = 10000$  and using the Hestenes-Stiefel conjugate gradient algorithm.

**Table 2.5** Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Hestenes-Stiefel conjugate gradient algorithm. Function  $f_2(x)$ 

Line-search	#iter	#fg	cpu(s)	$f^*$	$\ g\ $
Backtracking	264	5070	9.39	1.961211	0.627392e-05
Weak Wolfe	44	82	0.82	0.302616e-13	0.189612e-05
Strong Wolfe	68	164	1.48	0.712527	0.322331e-05

**Table 2.6** Performances of the inexact line-searches: backtracking, weak Wolfe, and strong Wolfe. Hestenes-Stiefel conjugate gradient algorithm. Function  $f_3(x)$ 

Line-search	#iter	#fg	cpu(s)	$f^*$	$\ g\ $
Backtracking	84	1075	10.87	0.581314e-10	0.691753e-04
Weak Wolfe	12	22	0.39	0.103860e-14	0.319524e-07
Strong Wolfe	16	34	0.60	0.147450e-12	0.104290e-05

In the above tables: #iter represents the number of iterations to get a solution, #fg is the number of function and its gradient evaluations, cpu(s) is the CPU computing time in seconds,  $f^*$  is the value of the minimizing function in the minimum point,  $\|g\|$  is the norm of the gradient in the minimum point

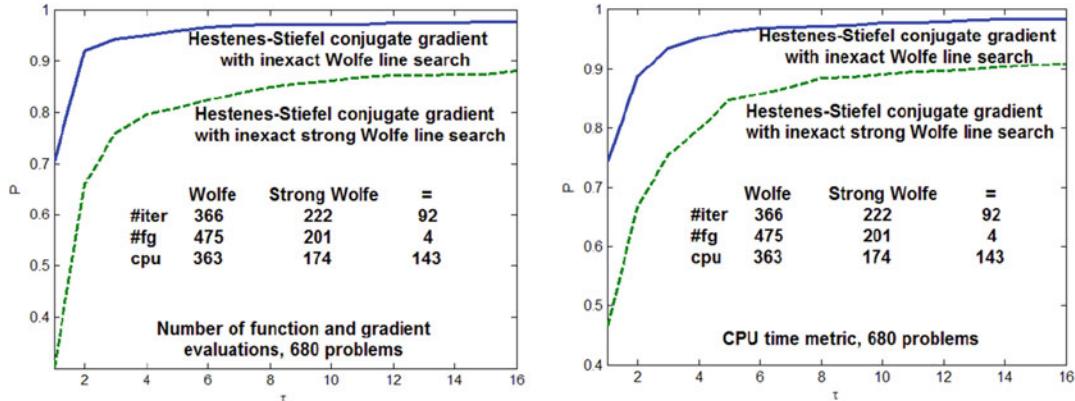
**Example 2.5** Table 2.5 shows the performances of the line-search algorithms, backtracking, weak Wolfe, and strong Wolfe, for the minimization of function  $f_2(x)$  given in Example 2.2 with the same initial point and  $n = 10000$  and using the Hestenes-Stiefel conjugate gradient algorithm.

**Example 2.6** Table 2.6 presents the performances of the line-search algorithms, backtracking, weak Wolfe, and strong Wolfe, for the minimization of function  $f_3(x)$  with  $n = 10000$  and using the Hestenes-Stiefel conjugate gradient algorithm.

From the above tables we can see that the inexact weak and the inexact strong Wolfe line-searches are the best line-search procedures for the stepsize computation. Observe that the weak Wolfe line-search is more efficient than the strong Wolfe line-search. However, from this limited set of numerical experiments, it cannot be concluded which version of the Wolfe line-search (weak or strong) is the better. That is why we next illustrate the running of some conjugate gradient algorithms for solving the problems from the UOP collection (Andrei, 2020a).

## Numerical Studies

1. In the following, we consider the Hestenes-Stiefel conjugate gradient algorithm in which the stepsize is computed by *the weak Wolfe line-search with cubic interpolation* (Shanno) (see Fig. 2.3) and the *strong Wolfe line-search* Moré-Thuente (see Andrei, 2021b) for solving the problems from the UOP collection (Andrei, 2020a, pp. 455–466). The UOP collection includes 80 functions in generalized or extended form. For each test function ten numerical experiments with the number of variables  $n = 1000, 2000, \dots, 10000$  have been considered, thus obtaining a number of 800 problems. All the algorithms have been coded in double precision Fortran and compiled with f77 (default compiler settings) and run on an Intel Pentium 4, 1.8 GHz workstation. For the comparison of algorithms, we follow the Remark 1.1



**Fig. 2.4** Performances of the Hestenes-Stiefel conjugate gradient algorithm with the inexact weak Wolfe line-search versus the inexact strong Wolfe line-search

The iterations are stopped if the inequality  $\|g_k\|_\infty \leq 10^{-6}$  is satisfied, where  $\|\cdot\|_\infty$  is the maximum absolute component of a vector. In line-search algorithms, we set  $\rho = 0.0001$  and  $\sigma = 0.8$ . The maximum number of iterations was limited to 2000.

Figure 2.4 presents the Dolan and Moré (2002) performances profiles of the Hestenes-Stiefel conjugate gradient algorithm in which the stepsize is computed by *the weak Wolfe line-search with cubic interpolation* (Shanno) and the *strong Wolfe line-search* (Moré-Thuente).

In a performance profile plot, the top curve corresponds to the method that solved the most problems in a time that was within a given factor of the best time. The percentage of the test problems for which a method is the fastest is given on the left axis of the plot. The right side of the plot gives the percentage of the test problems that were successfully solved by these algorithms, respectively. The left side of the plot is a measure of the *efficiency* of an algorithm, while the right side is a measure of the *robustness* of an algorithm. It is obvious that the Hestenes-Stiefel with inexact weak Wolfe line-search is more efficient and more robust versus the same algorithm with strong Wolfe line-search.

From Fig. 2.4, when comparing weak Wolfe versus strong Wolfe line-searches subject to the number of iterations or subject to the CPU time metric, we can see that the weak Wolfe is top performer. The table inside the plot of Fig. 2.4 shows that, subject to the number of iterations, the weak Wolfe is better in 366 problems (i.e., it achieved the minimum number of iterations in 366 problems). The strong Wolfe is better in 222 problems and they achieve the same number of iterations in 92 problems, etc. Out of 800 problems considered in this numerical study, only for 680 problems does the criterion (1.3) hold. Therefore, subject to the number of iterations metric or subject to the CPU time metric, on average, the weak Wolfe appears to slightly generate the best stepsizes.

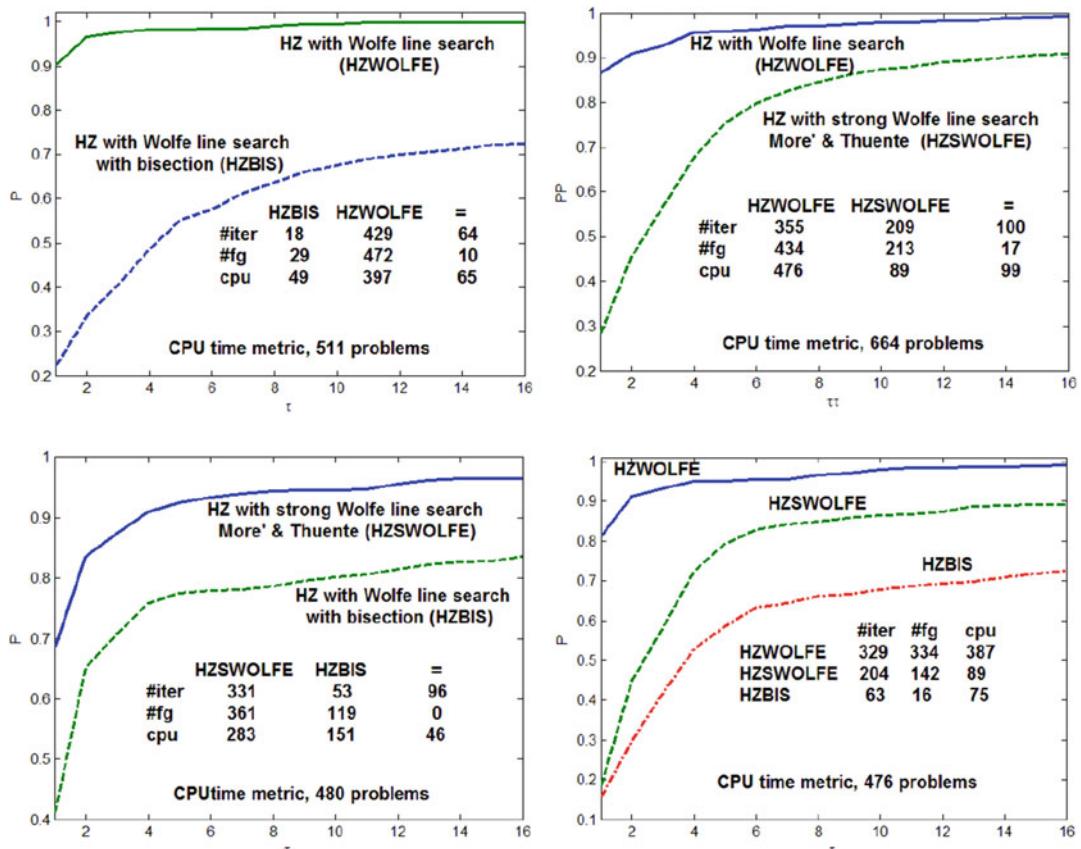
- Now, let us report some numerical results obtained with a Fortran implementation of the Hager and Zhang conjugate gradient algorithm (Hager, & Zhang, 2005), in which the stepsize is computed by using the *weak Wolfe line-search with a simple bisection technique* (see Fig. 2.2), *the weak Wolfe line-search with cubic interpolation* (Shanno) (see Fig. 2.3), and the *strong Wolfe line-search* (Moré-Thuente) (see Andrei, 2021b) for solving 800 problems from the UOP collection with the number of variables in the range [1000, 10000].

The conjugate gradient parameter in the Hager and Zhang conjugate gradient algorithm (HZ) is computed as

$$\begin{aligned}\beta_k &= \max \{\beta_k^N, \eta_k\}, \\ \beta_k^N &= \frac{1}{y_k^T d_k} \left( y_k - 2 \frac{\|y_k\|^2}{y_k^T d_k} \right)^T g_{k+1}, \\ \eta_k &= \frac{-1}{\|d_k\| \min \{\eta, \|g_k\|\}}, \quad \eta = 0.01.\end{aligned}\tag{2.114}$$

(This algorithm will be presented in Chap. 5.) Figure 2.5 presents the Dolan and Moré performance profiles of these algorithms: HZ with *weak Wolfe line-search* (HZWOLFE), HZ with *weak Wolfe line-search with simple bisection* (HZBIS), and HZ with *strong Wolfe line-search* (HZSWOLFE).

From Fig. 2.5 we can see that the strong Wolfe line-search is more expensive than the weak Wolfe line-search. For example, subject to the CPU computing time, HZWOLFE is faster in solving 476 problems (out 664), and HZSWOLFE is faster in solving only 89 problems. However, the strong Wolfe line-search has the advantage that, by decreasing  $\sigma$ , we have a direct control of the quality of the search by forcing the accepted value of the stepsize  $\alpha$  to be near enough to a local minimum. This is important in nonlinear conjugate gradient methods for solving large-scale unconstrained optimization.



**Fig. 2.5** Performance profiles of the Hager-Zhang algorithm with different line-search procedures

**Table 2.7** Global performances of HZ for solving 800 unconstrained optimization problems from the UOP collection with different line-searches

Algorithm and line-search procedure	#iter	#fg	cpu(s)
HZ with weak Wolfe line-search with simple bisection	941932	5608849	8459.79
HZ with weak Wolfe line-search with cubic interpolation	422540	2243545	1654.15
HZ with strong Wolfe line-search	534378	4618333	8230.91

It is interesting to notice the global performances of the Hager-Zhang conjugate gradient algorithm (HZ) with three line-search procedures: the *weak Wolfe line-search with bisection*, the *weak Wolfe line-search with cubic interpolation*, and the *strong Wolfe line-search*. Table 2.7 presents the global performances of HZ for solving this set of 800 unconstrained optimization problems with different line-searches.

The conclusion of these intensive numerical studies is that the weak Wolfe line-search is more efficient and more robust versus the backtracking and versus the Wolfe line-search with bisection and versus the strong Wolfe line-search. A comparison between the subroutine *WolfeLS* which implements the weak Wolfe line-search with safeguarded stepsize and cubic interpolation (Fig. 2.3) on one side and the subroutines *LSbis* (Fig. 2.2) and *back* (Fig. 2.1) shows the importance of the line-search in the frame of unconstrained optimization algorithms. The strong Wolfe line-search in the implementation of Moré and Thuente (see subroutine *MTlines* in Andrei, (2021b), pp. 15–22) is more sophisticated. It implements the safeguarding of stepsizes by introducing some large limits on them, uses the quadratic or the cubic interpolation, possibly taking the average of these quadratic and cubic steps. However, in our numerical experiments, we found that the weak Wolfe line-search with safeguarding and cubic interpolation implemented in the subroutine *WolfeLS* is more efficient and more robust than the strong Wolfe line-search implemented in the subroutine *MTlines*.

## Notes and References

This chapter was dedicated to the stepsize computation in the frame of unconstrained optimization algorithms. The stepsize computation is an important component of any optimization algorithm. Starting from  $x_k$ , the stepsize computation is to determine  $\alpha_k$ , in the direction  $d_k$  such that  $\varphi(\alpha_k) < \varphi(0)$ , where  $\varphi(\alpha) = f(x_k + \alpha d_k)$ . This is a one-dimensional minimization and can be achieved by using the traditional methods: dichotomous search, Fibonacci and golden-section search, Powell quadratic (1964) and Davidon cubic interpolation (1959), two-point interpolation, etc. These methods have not been discussed in this book, but the interested reader can find them, for example, in Antoniou and Lu (2007) or Walsh (1975).

The *modern* line-searches presented in Sect. 2.5 are based on the Armijo and Wolfe line-search conditions. No numerical studies on their performances are known. An excellent presentation of the theoretical developments concerning the convergence of the general algorithm with descent direction for unconstrained optimization with both exact and inexact line-searches was given by Sun and Yuan (2006). We followed their presentation. Details on the Wolfe line-search are also given by Dennis and Schnabel (1983) and Nocedal and Wright (2006). See also (Pulkkinen, 2008), (Bartholomew-Biggs, 2008), and (Andrei, 2009e). Nocedal and Wright (2006) presented important details on computing the stepsize by the strong Wolfe line-search concerning the initial stepsize selection, cubic interpolation, selection of the bracketing interval  $[\alpha_{lo}, \alpha_{hi}]$ , selection of the stepsize  $\alpha_j$  in this interval and replacement of one of the endpoints by  $\alpha_j$ , stopping test, etc.

More details on the stepsize computation for unconstrained optimization algorithms are given by (Gill, & Murray, 1974a; Murray, & Overton, 1979; Gill, Murray, & Wright, 1981; Lemaréchal, 1981; Fletcher, 1987; Dennis, & Schnabel, 1983; Kelley, 1999; Shi, & Shen 2015; Gould, 2006; Nocedal, & Wright, 2006; Yuan, 2006; Sun, & Yuan, 2006; Griva, Nash, & Sofer, 2009; Andrei, 2020a; etc.).



# Steepest Descent Methods

3

The steepest descent method was designed by Cauchy (1847) and is the simplest of the gradient methods for the optimization of general continuously differential functions in  $n$  variables. Its importance is due to the fact that it gives the fundamental ideas and concepts of all unconstrained optimization methods. It introduces a pattern common to many optimization methods. In this pattern, an iteration consists of two parts: the choice of a descent search direction  $d_k$  followed at once by a line-search to find a suitable stepsize  $\alpha_k$ . The search direction in the steepest descent method is exactly the negative gradient.

Actually, this is not an efficient method for solving large-scale and complex unconstrained optimization problems and applications. This simple strategy of moving along the negative gradient works well for particular functions with near-circular contours, but practical optimization problems or real applications may involve functions with narrow curving valleys which need a more sophisticated approach.

It is common practice in optimization algorithms to introduce the steepest descent when the search direction is not well defined. In fact, one point that emphasizes the importance of the steepest descent in optimization algorithms is exactly this one. All the respectable optimization algorithms include this stratagem: when the search direction is not well defined, try the steepest descent. This stratagem is often used as protection to make sure the algorithm is running appropriately.

---

## 3.1 The Steepest Descent

The unconstrained problem considered is

$$\min \{f(x) : x \in \mathbb{R}^n\}, \quad (3.1)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a continuously differentiable function lower bounded. If  $x_k$  is the current point which represents an approximation to the minimum point  $x^*$ , then  $f(x)$  can be approximated by its affine model

$$f(x_k + d) \approx f(x_k) + \nabla f(x_k)^T d,$$

where  $d \in \mathbb{R}^n$  is small, i.e.,  $\|d\|$  is small.

Observe that, if the above approximation of function  $f$  is good enough, then we want to choose the direction  $d$  so that  $\nabla f(x_k)^T d$  is as small as possible. Let us normalize the direction  $d$  so that  $\|d\| = 1$ . Then, among all the directions  $d$  with  $\|d\| = 1$ , the direction

$$\tilde{d} = \frac{-\nabla f(x_k)}{\|\nabla f(x_k)\|}$$

makes the smallest inner product with the gradient  $\nabla f(x_k)$ . Indeed, this follows from the following inequalities:

$$\nabla f(x_k)^T d \geq -\|\nabla f(x_k)\| \|d\| = \nabla f(x_k)^T \frac{-\nabla f(x_k)}{\|\nabla f(x_k)\|} = -\nabla f(x_k)^T \tilde{d}.$$

Therefore, the un-normalized direction  $d_k = -\nabla f(x_k)$  is called the *direction of steepest descent* at point  $x_k$ .

Observe that  $d_k = -\nabla f(x_k)$  is a descent direction as long as  $\nabla f(x_k) \neq 0$ . To see this, we can write  $\nabla f(x_k)^T d_k = -\nabla f(x_k)^T \nabla f(x_k) < 0$ , so long as  $\nabla f(x_k) \neq 0$ . With these, the following algorithm, called the steepest descent algorithm, may be presented.

### Algorithm 3.1 Steepest descent

- |    |  |
|----|--|
| 1. | Choose an initial point $x_0$ and a convergence tolerance $\varepsilon > 0$ sufficiently small for stopping the iterations.<br>Set $k = 0$ |
| 2. | Compute $\nabla f(x_k)$ . Set $d_k = -\nabla f(x_k)$   |
| 3. | If $\ d_k\  \leq \varepsilon$ , then stop; otherwise, continue with step 4   |
| 4. | Compute the stepsize $\alpha_k$ by the exact or by the inexact line-search procedures  |
| 5. | Set $x_{k+1} = x_k + \alpha_k d_k$ , $k = k + 1$ and go to step 2  |

◆

Since  $d_k = -\nabla f(x_k)$  is a descent direction, it follows that  $f(x_{k+1}) < f(x_k)$ . In step 4, the exact or the inexact line-search may be used. Let us consider a numerical example to see the running of the steepest descent algorithm.

**Example 3.1** Consider the function

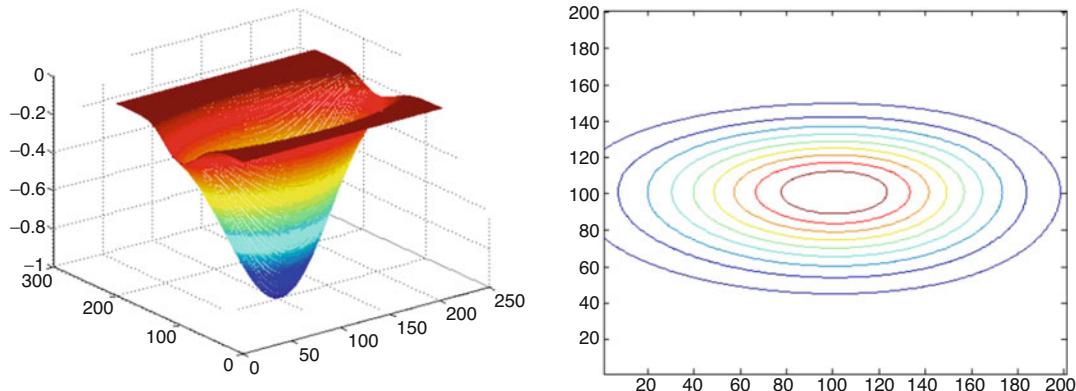
$$f_1(x) = -(0.5 + 0.5x_1)^4 x_2^4 \exp\left(2 - (0.5 + 0.5x_1)^4 - x_2^4\right),$$

illustrated in Fig. 3.1.

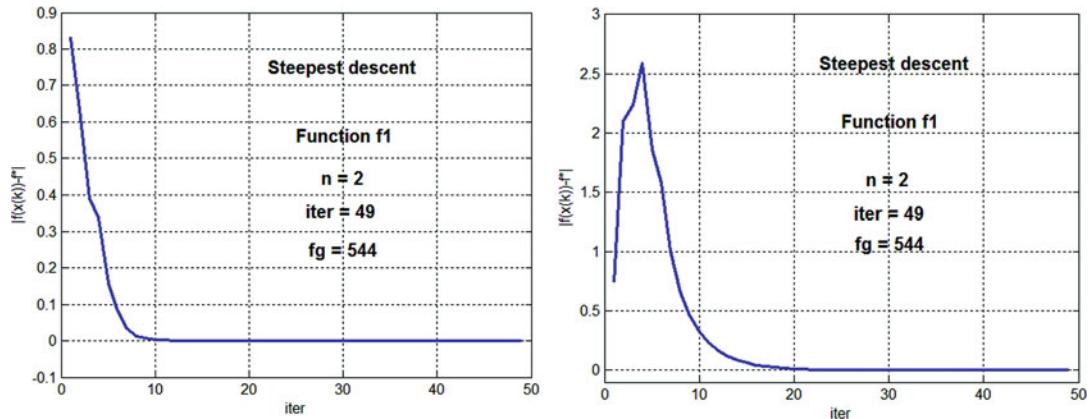
The local minimum point of this function is  $x^* = [1, 1]$ , in which  $f_1^* = -1$ . For the initial point  $x_0 = [0.1, 0.8]$  the steepest descent algorithm, in which in step 4 the stepsize is computed by backtracking, gives the solution in 49 iterations and 544 evaluations of the minimizing function and its gradient. The evolutions of the error  $|f_1(x_k) - f_1^*|$  as well as of  $\|g_k\|_2$  are presented in Fig. 3.2a, b, respectively.

Observe that the error  $|f_1(x_k) - f_1^*|$  converges to zero as a geometrical series, i.e., the convergence is *linear*. From Fig. 3.2a we can see that the error  $|f_1(x_k) - f_1^*|$  is drastically reduced in the first few iterations, then the algorithm slowly converges to the solution. This is a characteristic of the steepest descent algorithm. If instead of backtracking the weak Wolfe line-search is used, then the solution is obtained in 13 iterations.

The evolution of the iterations in the steepest descent algorithm is in “zigzag.” Indeed, if the line search is exact, then  $g_{k+1}^T g_k = d_{k+1}^T d_k = 0$ . Therefore, the successive gradients are orthogonal, i.e.,



**Fig. 3.1** Representation of function  $f_1(x)$



**Fig. 3.2** (a) Evolution of the error  $|f_1(x_k) - f_1^*|$ . (b) Evolution of  $\|g_k\|_2$

the successive search directions are orthogonal. On the other hand, if the line-search is inexact,  $\|g_k\|$  becomes smaller and smaller along the iterations. Therefore, from Taylor's development, we have  $f(x_k - \alpha g_k) = f(x_k) - \alpha g_k^T g_k + o(\|\alpha g_k\|)$ , where the first order term  $\alpha g_k^T g_k = \alpha \|g_k\|^2$  is very small. Hence, the reduction of the values of the minimizing function will be very small. This is the main drawback of the steepest descent algorithm.

### Convergence of the Steepest Descent Method for Quadratic Functions

Consider the quadratic function

$$f(x) = \frac{1}{2} x^T Q x - b^T x, \quad (3.2)$$

where  $Q \in \mathbb{R}^{n \times n}$  is a symmetric and positive definite matrix. Let  $\lambda_{\min}$  and  $\lambda_{\max}$  be the smallest and the largest eigenvalues of  $Q$ . As we know, a method of descent directions generates a sequence

$$x_{k+1} = x_k + \alpha_k d_k,$$

where  $d_k$  is a descent direction for which

$$\nabla f(x_k)^T d_k < 0,$$

such that  $f(x_k + \alpha d_k) < f(x_k)$  for small values of  $\alpha$ . For the above quadratic function, the value of the stepsize  $\alpha$  which minimizes  $f$  along the line through  $x_k$  in the direction  $d_k$  can be analytically computed as

$$\alpha = \frac{d_k^T r_k}{d_k^T Q d_k}, \quad (3.3)$$

where  $e_k = x^* - x_k$  is the *error* at iteration  $x_k$  and

$$r_k = Qe_k = b - Qx_k \quad (3.4)$$

is the *residual* from iteration  $k$ .

In the steepest descent method,

$$d_k = -\nabla f(x_k) = r_k. \quad (3.5)$$

Now, defining  $\|x\|_Q^2 = x^T Q x$ , then the  $Q$ - norm of the error is

$$\begin{aligned} \|e_{k+1}\|_Q^2 &= \|x^* - (x_k + \alpha_k d_k)\|_Q^2 = \|e_k - \alpha_k d_k\|_Q^2 = \\ &= \|e_k\|_Q^2 - 2\alpha_k e_k^T Q d_k + \alpha_k^2 \|d_k\|_Q^2 \\ &= \|e_k\|_Q^2 - 2 \frac{r_k^T r_k}{r_k^T Q r_k} e_k^T Q r_k + \frac{(r_k^T r_k)^2}{(r_k^T Q r_k)^2} r_k^T Q r_k \\ &= \|e_k\|_Q^2 \left(1 - \frac{(r_k^T r_k)^2}{r_k^T Q r_k r_k^T Q^{-1} r_k}\right) \end{aligned} \quad (3.6)$$

In order to prove the convergence of the steepest descent method, we need a theoretical result known as the inequality of Kantorovich.

## Inequality of Kantorovich

Let  $\lambda_{\max}$  and  $\lambda_{\min}$  be the largest and the smallest eigenvalues of a symmetric and positive definite matrix  $Q$ , respectively. Then

$$\beta = \frac{(d^T Q d)(d^T Q^{-1} d)}{(d^T d)^2} \leq \frac{(\lambda_{\max} + \lambda_{\min})^2}{4\lambda_{\max}\lambda_{\min}}.$$

**Proof** Let  $Q = RDR^T$ . Then  $Q^{-1} = RD^{-1}R^T$ , where  $R = R^T$  is an orthonormal matrix and the eigenvalues of  $Q$  are  $0 < \lambda_{\min} = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n = \lambda_{\max}$ , and  $D = \text{diag}(\lambda_1, \dots, \lambda_n)$ . Then

$$\beta = \frac{(d^T R D R^T d)(d^T R D^{-1} R^T d)}{(d^T R R^T d)(d^T R R^T d)} = \frac{v^T D v v^T D^{-1} v}{v^T v v^T v},$$

where  $v = R^T d$ . Now, let  $\gamma_i = v_i^2 / (v^T v)$ . Then,  $\gamma_i \geq 0$  and  $\sum_{i=1}^n \gamma_i = 1$ . Hence,

$$\beta = \sum_{i=1}^n \gamma_i \lambda_i \sum_{i=1}^n \gamma_i \left( \frac{1}{\lambda_i} \right) = \frac{\sum_{i=1}^n \gamma_i (1/\lambda_i)}{1 / \sum_{i=1}^n \gamma_i \lambda_i}.$$

The largest value of  $\beta$  is obtained when  $\gamma_1 + \gamma_n = 1$ . Therefore,

$$\begin{aligned} \beta &\leq \frac{\gamma_1(1/\lambda_{\min}) + \gamma_n(1/\lambda_{\max})}{1/(\gamma_1\lambda_{\min} + \gamma_n\lambda_{\max})} = \frac{(\gamma_1\lambda_{\min} + \gamma_n\lambda_{\max})(\gamma_1\lambda_{\max} + \gamma_n\lambda_{\min})}{\lambda_{\min}\lambda_{\max}} \\ &\leq \frac{(\lambda_{\min}/2 + \lambda_{\max}/2)(\lambda_{\max}/2 + \lambda_{\min}/2)}{\lambda_{\min}\lambda_{\max}} = \frac{(\lambda_{\max} + \lambda_{\min})^2}{4\lambda_{\max}\lambda_{\min}}. \end{aligned}$$
◆

At this moment, applying the Kantorovich inequality to the right side term of (3.6) and noticing that

$$1 - \frac{4\lambda_{\min}\lambda_{\max}}{(\lambda_{\min} + \lambda_{\max})^2} = \frac{(\lambda_{\max} - \lambda_{\min})^2}{(\lambda_{\max} + \lambda_{\min})^2},$$

it follows that

$$\|e_{k+1}\|_Q \leq \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \|e_k\|_Q \equiv \frac{\kappa - 1}{\kappa + 1} \|e_k\|_Q, \quad (3.7)$$

where  $\kappa = \lambda_{\max}/\lambda_{\min}$  is the condition number of  $Q$ . In other words, *the convergence of the steepest descent method for quadratic functions is linear*.

**Example 3.2** Let us see the running of the steepest descent algorithm for minimizing a quadratic function. Consider the function

$$f_2(x) = \frac{1}{2}x^T Q x,$$

where  $Q = \text{diag}(1, 2, \dots, n)$ . Observe that  $Q$  is a positive definite matrix with the eigenvalues  $1, 2, \dots, n$ . The condition number of  $Q$  is  $n$ . For this function,  $\nabla f(x) = Qx$  and the stepsize is computed as

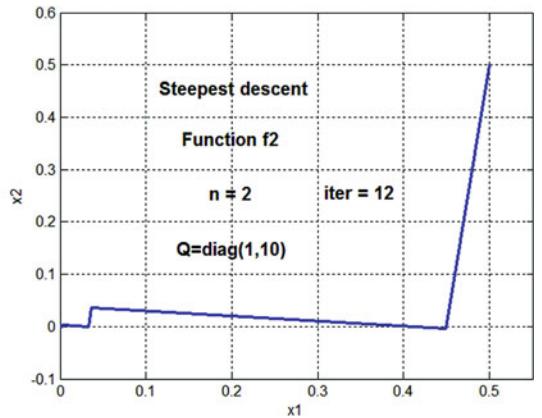
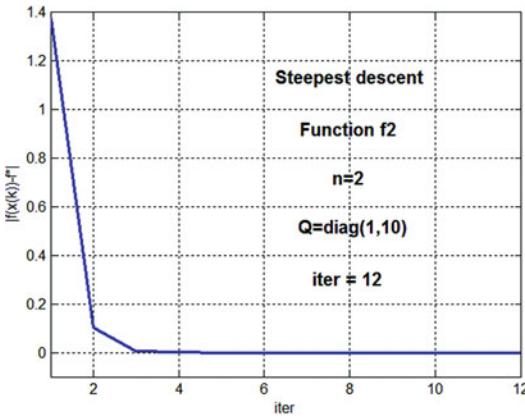
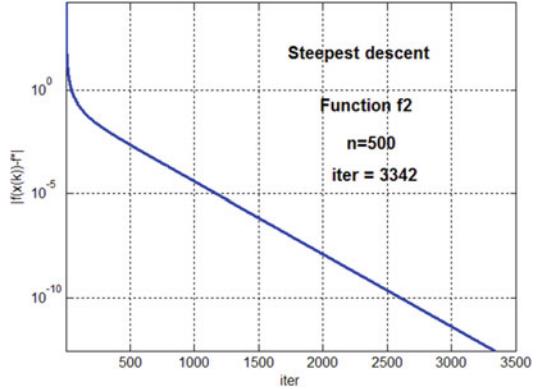
$$\alpha_k = \frac{g_k^T g_k}{g_k^T Q g_k}, \quad \text{where } g_k = Qx_k.$$

As we know, for this *optimal* selection of  $\alpha_k$ , the rate of convergence of the steepest descent algorithm is linear

$$\|x_k - x^*\|_Q \leq \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \|x_{k-1} - x^*\|_Q,$$

where  $\|z\|_Q = z^T Q z$ , and  $\lambda_{\min}$  and  $\lambda_{\max}$  are the smallest and the largest eigenvalues of  $Q$ , respectively. The solution of this problem is vector zero, while the value of the function in the minimum point is also zero. Considering  $n = 500$ ,  $\epsilon = 10^{-6}$  and the initial point  $x_0 = [0.5, 0.5, \dots, 0.5]^T$ , then, along the iterations, the steepest descent algorithm gives the following evolution of the error  $|f_2(x_k) - f_2(x^*)|$  from Fig. 3.3.

**Fig. 3.3** Evolution of the error  $|f_2(x_k) - f_2(x^*)|$ .  
 $n = 500$



**Fig. 3.4** (a) Evolution of the error  $|f_2(x_k) - f_2(x^*)|$ .  $n = 2$ . (b) Zigzag evolution

This solution is obtained in 3342 iterations. Observe the linear convergence of this algorithm, especially in the final part of the iterations. For large values of  $n$ , it follows that  $(\lambda_{\max} - \lambda_{\min}) / (\lambda_{\max} + \lambda_{\min})$  tends to 1, thus proving the linear convergence.

Now, considering  $Q = \text{diag}(1, 10)$ , then the error  $|f_2(x_k) - f_2(x^*)|$  has the evolution from Fig. 3.4a. The trajectory of variables is illustrated in Fig. 3.4b. In Fig. 3.4a we can see the linear convergence of the algorithm, while in Fig. 3.4b the trajectory of variables is in *zigzag in wright angles*.

The rate of convergence of the steepest descent algorithm depends on the ratio between the largest axis and the smallest axis of the ellipsoid associated to the matrix  $Q$ . The bigger this ratio, the slower the convergence. Details on the convergence of the steepest descent algorithm for quadratic functions are presented in the following theorem.

**Theorem 3.1** Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x) \equiv \frac{1}{2} x^T Q x, \quad (3.8)$$

where  $Q$  is a symmetric and positive definite matrix with  $\lambda_{\min}$  and  $\lambda_{\max}$  the smallest and the largest eigenvalues, respectively. Let  $x^*$  be the solution of the problem (3.8). Then, the sequence  $\{x_k\}$  generated by the steepest descent algorithm converges to  $x^*$ , the rate of convergence is at least linear, and the following relations hold:

$$\frac{f(x_{k+1}) - f(x^*)}{f(x_k) - f(x^*)} \leq \frac{(\kappa - 1)^2}{(\kappa + 1)^2} = \frac{(\lambda_{\max} - \lambda_{\min})^2}{(\lambda_{\max} + \lambda_{\min})^2}, \quad (3.9)$$

$$\frac{\|x_{k+1} - x^*\|_Q}{\|x_k - x^*\|_Q} \leq \frac{\kappa - 1}{\kappa + 1} = \left( \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right), \quad (3.10)$$

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq \sqrt{\kappa} \frac{\kappa - 1}{\kappa + 1} = \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} \left( \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right), \quad (3.11)$$

where  $\kappa = \lambda_{\max}/\lambda_{\min}$ .

**Proof** Consider the steepest descent algorithm  $x_{k+1} = x_k - \alpha_k g_k$ , where  $\alpha_k = \frac{g_k^T g_k}{g_k^T Q g_k}$  and  $g_k = Qx_k$ . Then

$$\begin{aligned} \frac{f(x_k) - f(x_{k+1})}{f(x_k)} &= \frac{\frac{1}{2} x_k^T Q x_k - \frac{1}{2} (x_k - \alpha_k g_k)^T Q (x_k - \alpha_k g_k)}{\frac{1}{2} x_k^T Q x_k} \\ &= \frac{\alpha_k g_k^T Q x_k - \frac{1}{2} \alpha_k^2 g_k^T Q g_k}{\frac{1}{2} x_k^T Q x_k} = \frac{(g_k^T g_k)^2}{(g_k^T Q g_k)(g_k^T Q^{-1} g_k)}. \end{aligned}$$

Now, from the Kantorovich inequality, we get

$$\frac{f(x_{k+1})}{f(x_k)} = 1 - \frac{(g_k^T g_k)^2}{(g_k^T Q g_k)(g_k^T Q^{-1} g_k)} = 1 - \frac{4\lambda_{\max}\lambda_{\min}}{(\lambda_{\max} + \lambda_{\min})^2} = \left( \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right)^2,$$

which is exactly (3.9).

Now, let  $e_k = x_k - x^*$ . Since  $Q$  is symmetric and positive definite, it follows that

$$\lambda_{\min} e_k^T e_k \leq e_k^T Q e_k \leq \lambda_{\max} e_k^T e_k.$$

Moreover, since  $x^* = 0$ , we get

$$\|x_k - x^*\|_Q^2 = e_k^T Q e_k = x_k^T Q x_k = 2f(x_k).$$

Now, using the above inequalities, it follows that for any  $k \geq 0$ ,

$$\lambda_{\min} \|x_k - x^*\|^2 \leq 2f(x_k) \leq \lambda_{\max} \|x_k - x^*\|^2.$$

Therefore,

$$\frac{\lambda_{\min} \|x_{k+1} - x^*\|^2}{\lambda_{\max} \|x_k - x^*\|^2} \leq \frac{\|x_{k+1} - x^*\|_Q^2}{\|x_k - x^*\|_Q^2} \leq \left( \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right)^2,$$

which gives (3.10) and (3.11). ◆

This is one of the best estimations we can obtain for the steepest descent in certain conditions. From (3.9) we can write

$$f(x_{k+1}) - f(x^*) \leq \left[ \frac{\kappa(Q) - 1}{\kappa(Q) + 1} \right]^2 (f(x_k) - f(x^*)),$$

where  $\kappa(Q)$  is the condition number of  $Q$ . (See Appendix A.)

For general nonlinear functions, it is possible to show that the steepest descent method with an exact line-search also converges linearly, with a constant rate that is bounded by the *fundamental ratio*  $[(\kappa(\nabla^2 f(x^*)) - 1)/(\kappa(\nabla^2 f(x^*)) + 1)]^2$ .

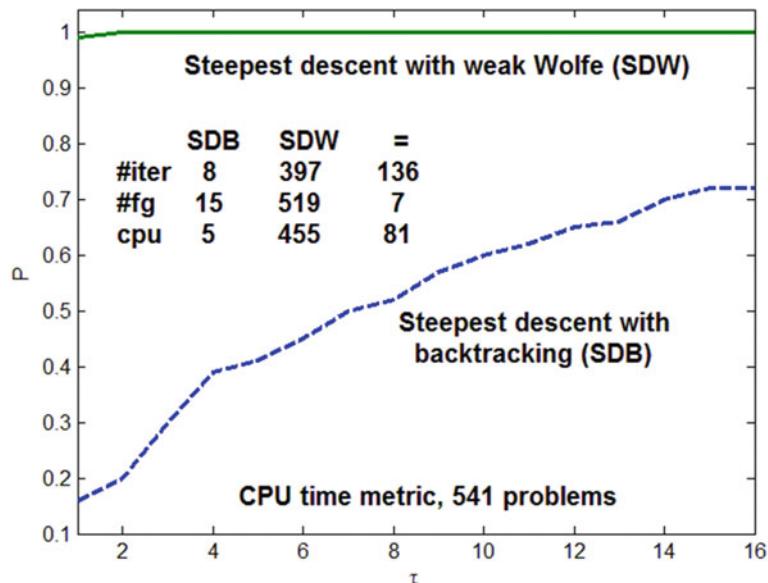
For strongly convex functions for which the gradient is Lipschitz continuous, Nemirovsky and Yudin (1983) define the global estimate of the rate of convergence of an iterative method as  $f(x_{k+1}) - f(x^*) \leq ch(x_1 - x^*, m, L, k)$ , where  $c$  is a constant,  $h(\cdot)$  is a function,  $m$  is a lower bound on the smallest eigenvalue of the Hessian  $\nabla^2 f(x)$ ,  $L$  is the Lipschitz constant, and  $k$  is the iteration number. The faster the rate at which  $h$  converges to zero as  $k \rightarrow \infty$ , the more efficient the algorithm.

Now, we have an algorithm for minimizing continuously differentiable functions, the steepest descent. In step 4 of this algorithm, for stepsize computation, both the exact or the inexact line-search procedures may be used. In the following, before discussing the convergence properties of the steepest descent algorithm, let us present a numerical study referring to a comparison between the steepest descent with backtracking versus the steepest descent with weak Wolfe line-searches. This study is relevant for the importance of the line-search in gradient descent algorithms.

## Numerical Study

Consider the problems from the UOP collection (Andrei, 2020a), which includes 80 large-scale unconstrained optimization problems in generalized or extended form. For each test function from this collection, we have considered 10 numerical experiments with the number of variables increasing as  $n = 100, 200, \dots, 1000$ . Therefore, a set of 800 unconstrained optimization problems are solved in our numerical experiments, both with steepest descent with backtracking (SDB) and steepest descent with weak Wolfe (SDW). Both SDB and SDW use the same stopping criterion given by  $\|\nabla f(x_k)\|_2 \leq \epsilon_g$  or  $\alpha_k |g_k^T d_k| \leq \epsilon_f |f(x_{k+1})|$ , where  $\epsilon_g = 10^{-6}$  and  $\epsilon_f = 10^{-16}$ . In backtracking,  $\rho = 0.0001$ ,  $\beta = 0.8$ . Figure 3.5 presents the performances of these algorithms subject to the CPU computing time. From Fig. 3.5 we can see that SDW is more efficient and more robust than SDB. This is not a surprise, because the backtracking (see Fig. 2.1) is a very crude line-search procedure. In contrast, the weak

**Fig. 3.5** Steepest descent with weak Wolfe (SDW) versus steepest descent with backtracking (SDB)



Wolfe (see Fig. 2.3) is more advanced, being able to generate an acceptable (inexact) stepsize which emphasizes the minimization of function  $f$ . For example, the table inside the plot shows that, subject to the CPU time, SDW is faster in 455 problems, while SDB is faster only in solving 5 problems. Out of 800 problems considered in this numerical study, only for 541 problems does the criterion (1.3) hold.

## Convergence of the Steepest Descent Method for General Functions

In the following, let us discuss the global convergence and the local rate of convergence of the steepest descent method for general continuously differentiable functions.

**Theorem 3.2** Assume that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is convex and differentiable and additionally  $\nabla f$  is Lipschitz continuous, i.e.,  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ , for any  $x, y$ , with constant  $L > 0$ . Then the steepest descent with fixed stepsize  $\delta \leq 1/L$  satisfies

$$f(x_k) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\delta k}.$$

**Proof** The theorem says that the steepest descent algorithm has the convergence rate  $O(1/k)$ , that is, in order to get  $f(x_k) - f(x^*) \leq \varepsilon$ , it needs  $O(1/\varepsilon)$  iterations. The convergence is more emphasized if the initial point  $x_0$  is close to the minimum point  $x^*$ .

From the Lipschitz continuity with constant  $L$ , it follows that

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2,$$

for all  $x, y$ . Now, in the above inequality, when introducing  $y = x - \delta \nabla f(x)$ , we get

$$f(y) \leq f(x) - \left(1 - \frac{L\delta}{2}\right)\delta\|\nabla f(x)\|^2.$$

Considering  $x^+ = x - \delta \nabla f(x)$  and taking  $0 < \delta \leq 1/L$ , it follows that

$$\begin{aligned} f(x^+) &\leq f(x^*) + \nabla f(x)^T(x - x^*) - \frac{\delta}{2}\|\nabla f(x)\|^2 \\ &= f(x^*) + \frac{1}{2\delta}\left(\|x - x^*\|^2 - \|x^+ - x^*\|^2\right). \end{aligned}$$

Summing over iterations we get

$$\sum_{i=1}^k (f(x_i) - f(x^*)) \leq \frac{1}{2\delta} \left( \|x_0 - x^*\|^2 - \|x_k - x^*\|^2 \right) \leq \frac{1}{2\delta} \|x_0 - x^*\|^2.$$

Since  $\{f(x_k)\}$  is a nonincreasing sequence, it follows that

$$f(x_k) - f(x^*) \leq \frac{1}{k} \sum_{i=1}^k (f(x_i) - f(x^*)) \leq \frac{\|x_0 - x^*\|^2}{2\delta k}. \quad \blacklozenge$$

**Theorem 3.3** Assume that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is convex and differentiable and  $\nabla f$  is Lipschitz continuous with constant  $L > 0$ . Then the steepest descent with backtracking line-search satisfies

$$f(x_k) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\delta_{\min} k},$$

where  $\delta_{\min} = \min \{1, \beta/L\}$  and  $\beta$  is as in Algorithm 2.5 or Algorithm 2.6.

**Proof** The proof is the same as the proof of Theorem 3.2.  $\diamond$

Observe that if  $\beta$  is not too small, then we do not lose much in the convergence compared to the fixed stepsize ( $\beta/L$  versus  $1/L$ ).

Now, let us discuss the convergence of the steepest descent algorithm for minimizing strongly convex functions. Strong convexity of  $f$  means that for some  $m > 0$ ,  $\nabla^2 f(x) \geq mI$  for any  $x$ . For strongly convex functions, we have a better lower bound than the one from the usual convexity

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{m}{2}\|y - x\|^2,$$

for all  $x, y$ .

**Theorem 3.4** Assume that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is strongly convex, and its gradient is Lipschitz continuous, i.e.,  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$  for any  $x, y$ , with constant  $L > 0$ . Then the steepest descent with fixed stepsize  $\delta \leq 2/(m + L)$  or with backtracking line-search satisfies

$$f(x_k) - f(x^*) \leq c^k \frac{L}{2}\|x_0 - x^*\|^2,$$

where  $0 < c < 1$ .  $\diamond$

The theorem says that under strong convexity, the rate of convergence is  $O(c^k)$ , i.e., exponentially fast. In other words, to get  $f(x_k) - f(x^*) \leq \varepsilon$ , the algorithm needs  $O(\log(1/\varepsilon))$  iterations. The constant  $c$  depends on the condition number  $L/m$ . A higher condition number involves a slower rate of convergence.

Observe that these results are obtained under the Lipschitz continuity of the gradient of the minimizing function and under the strong convexity of the minimizing function. A question arises: how realistic are these conditions? The Lipschitz continuity of  $\nabla f$  means that  $\nabla^2 f(x) \leq LI$ , where  $L$  is the Lipschitz constant and  $I$  is the identity matrix. For example, consider  $f(x) = \frac{1}{2}\|y - Ax\|^2$ . Then,  $\nabla^2 f(x) = A^T A$ , so  $\nabla f$  is Lipschitz with  $L = \sigma_{\max}^2(A) = \|A\|^2$ . On the other hand, strong convexity means that  $\nabla^2 f(x) \geq mI$ , where  $I$  is the identity matrix. For example, consider again the function  $f(x) = \frac{1}{2}\|y - Ax\|^2$ . Then,  $\nabla^2 f(x) = A^T A$  and  $m = \sigma_{\min}^2(A)$ .

**Theorem 3.5** Let function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be continuously differentiable. Then, any accumulation point of the sequence  $\{x_k\}$  generated by the steepest descent Algorithm 3.1 with exact line-search is a stationary point of function  $f$ .

**Proof** Consider  $\bar{x}$  an accumulation point of the sequence  $\{x_k\}$  and  $K$  an infinite set of indices such that  $\lim_{k \in K} x_k = \bar{x}$ . Let  $d_k = -\nabla f(x_k)$ . Since  $f$  is continuously differentiable, the sequence  $\{d_k : k \in K\}$  is uniformly bounded and  $\|d_k\| = \|\nabla f(x_k)\|$ . Since the hypotheses of Theorem 2.11 are satisfied, it follows that  $\|\nabla f(\bar{x})\|^2 = 0$ , that is,  $\nabla f(\bar{x}) = 0$ .  $\diamond$

**Theorem 3.6** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on  $\mathbb{R}^n$  and  $\|\nabla^2 f(x)\| \leq M$ , where  $M$  is a positive constant. Let  $x_0$  be the initial point and  $\varepsilon > 0$  a positive constant sufficiently small. Then, the sequence generated by the steepest descent Algorithm 3.1 satisfies the condition of termination after a finite number of iterations or  $\lim_{k \rightarrow \infty} f(x_k) = -\infty$  or  $\lim_{k \rightarrow \infty} \nabla f(x_k) = 0$ .

**Proof** Consider the infinite case. From Algorithm 3.1 and Theorem 2.10, it follows that

$$f(x_k) - f(x_{k+1}) \geq \frac{1}{2M} \|\nabla f(x_k)\|^2.$$

Hence,

$$f(x_0) - f(x_k) = \sum_{i=0}^{k-1} [f(x_i) - f(x_{i+1})] \geq \frac{1}{2M} \sum_{i=0}^{k-1} \|\nabla f(x_i)\|^2.$$

Therefore,  $\lim_{k \rightarrow \infty} f(x_k) = -\infty$ , or  $\lim_{k \rightarrow \infty} \nabla f(x_k) = 0$ .  $\blacklozenge$

The next theorem establishes the rate of convergence of the steepest descent algorithm with exact line-search.

**Theorem 3.7** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  which satisfies the hypothesis of Theorem 2.12. If the sequence  $\{x_k\}$  generated by the steepest descent algorithm converges to  $x^*$ , then the rate of convergence is linear.

**Proof** The theorem is a direct consequence of Theorem 2.12.

The rate of convergence of the steepest descent algorithm for general twice continuously differentiable functions is established by the following theorem.

**Theorem 3.8** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function in a neighborhood of  $x^*$  with  $\nabla f(x^*) = 0$  and the Hessian  $\nabla^2 f(x^*)$  a positive definite matrix. Let  $\lambda_{\max}$  and  $\lambda_{\min}$  be the largest and the smallest eigenvalues of  $\nabla^2 f(x^*)$ , respectively, which satisfy  $0 < m \leq \lambda_{\min} \leq \lambda_{\max} \leq M$ . Let  $\{x_k\}$  be the sequence generated by the steepest descent Algorithm 3.1 convergent to  $x^*$ . Let

$$\frac{f(x_{k+1}) - f(x^*)}{f(x_k) - f(x^*)} = \beta_k. \quad (3.12)$$

Then, for any  $k$ ,  $\beta_k < 1$  and

$$\limsup_{k \rightarrow +\infty} \beta_k \leq \frac{M - m}{M} < 1. \quad (3.13)$$

**Proof** From Theorem 2.10 it follows that

$$\begin{aligned} [f(x_k) - f(x^*)] - [f(x_{k+1}) - f(x^*)] &= f(x_k) - f(x_{k+1}) \\ &\geq \frac{1}{2M} \|\nabla f(x_k)\|^2. \end{aligned}$$

Using the definition of  $\beta_k$ , we have

$$(1 - \beta_k)[f(x_k) - f(x^*)] \geq \frac{1}{2M} \|\nabla f(x_k)\|^2.$$

Therefore, from the hypothesis of the theorem, we get

$$\beta_k \leq 1 - \frac{\|\nabla f(x_k)\|^2}{2M[f(x_k) - f(x^*)]} < 1. \quad (3.14)$$

Suppose that

$$\frac{x_k - x^*}{\|x_k - x^*\|} \rightarrow \bar{d}.$$

Obviously,

$$\|\nabla f(x_k)\|^2 = \|x_k - x^*\|^2 \left( \|\nabla^2 f(x^*) \bar{d}\|^2 + o(1) \right)$$

and

$$f(x_k) - f(x^*) = \frac{1}{2} \|x_k - x^*\|^2 \left( \bar{d}^T \nabla^2 f(x^*) \bar{d} + o(1) \right).$$

From the above inequalities, it follows that

$$\lim_{k \rightarrow \infty} \frac{\|\nabla f(x_k)\|^2}{f(x_k) - f(x^*)} = \frac{2 \|\nabla^2 f(x^*) \bar{d}\|^2}{\bar{d}^T \nabla^2 f(x^*) \bar{d}} \geq 2m. \quad (3.15)$$

Hence, from (3.14) and (3.15) we obtain

$$\limsup_{k \rightarrow \infty} \beta_k \leq 1 - \liminf_{k \rightarrow \infty} \frac{\|\nabla f(x_k)\|^2}{2M[f(x_k) - f(x^*)]} \leq 1 - \frac{m}{M} < 1,$$

thus proving the theorem. ◆

We may conclude that:

- ◆ The steepest descent algorithm is linear convergent, that is, the error  $f(x_k) - f^*$  tends to zero like a geometric series.
- ◆ The rate of convergence depends on the condition number of the Hessian of the minimizing function (which is unknown). The convergence may be very slow even for functions relatively well conditioned. When the condition number of the Hessian is large, then the steepest descent algorithm converges so slowly that it does not have any practical value.
- ◆ The steepest descent with weak Wolfe line-search is way more efficient and more robust than the steepest descent with backtracking.
- ◆ The parameters used in backtracking do not have a significant effect on the convergence of the algorithm. The exact line-search improves the convergence, but without any significant effect.

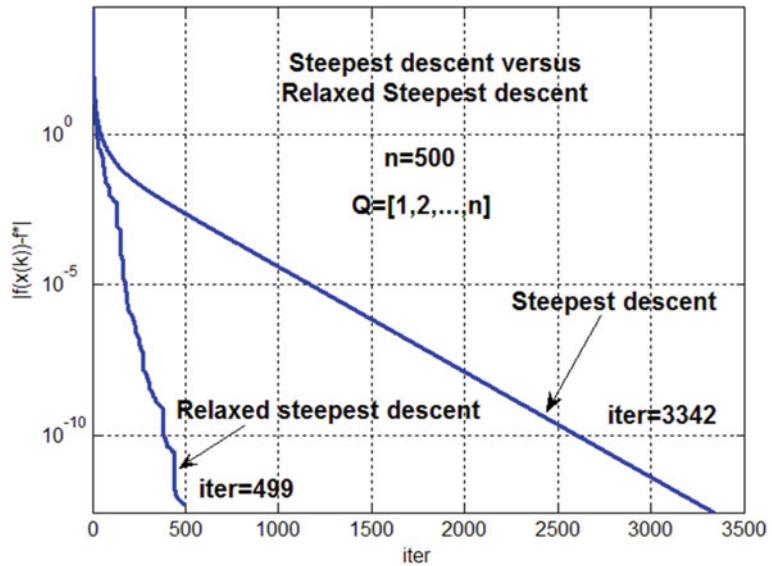
## 3.2 The Relaxed Steepest Descent

It is interesting to see what happens when the stepsize  $\alpha_k$  computed by a line-search procedure is randomly modified in  $[0, 1]$ , that is, the new estimation to the minimum  $x^*$  is computed as

$$x_{k+1} = x_k + \theta_k \alpha_k d_k, \quad (3.16)$$

where  $d_k = -\nabla f(x_k)$  and  $\theta_k$  is a random variable uniformly distributed in the interval  $[0, 1]$ . Thus, a new algorithm called *relaxed steepest gradient* algorithm is obtained (Andrei, 2004b, 2005a).

**Fig. 3.6** Evolution of the error  $|f(x_k) - f(x^*)|$  for steepest descent versus relaxed steepest descent



**Example 3.3** Consider the function

$$f(x) = \frac{1}{2}x^T Q x,$$

where  $Q = \text{diag}(1, 2, \dots, n)$  and  $n = 500$ , from Example 3.2. Taking the initial point  $x_0 = [0.5, 0.5, \dots, 0.5]^T$  and  $\epsilon = 10^{-6}$  in the criterion for stopping the iterations, then, along the iterations, the steepest descent and the relaxed steepest descent algorithms give the following evolution of the error  $|f(x_k) - f(x^*)|$  from Fig. 3.6. ♦

In this case of quadratic positive definite functions, the relaxed steepest descent algorithm is more efficient than the classical steepest descent algorithm. Indeed, for solving the problem, the steepest descent algorithm needs 3342 iterations, while the relaxed steepest descent only 499 iterations. A slightly better evolution subject to the number of iterations is obtained when  $\theta_k$  is selected as a random variable uniformly distributed in the interval  $[0, 2]$ . For this quadratic function, taking different values for the number of variables  $n$ , Table 3.1 shows the number of iterations for the steepest descent algorithm and its relaxed version, where  $\theta_k \in [0, 1]$  and  $\theta_k \in [0, 2]$ .

This behavior of the relaxed steepest descent algorithm illustrates a *very serious limitation* of the optimal stepsize selection as well as a cognitive misleading, since a small modification of the stepsize determines major changes in the performances of the steepest descent algorithm. In fact, for quadratic positive definite functions, in very mild conditions on  $\theta_k \in [0, 2]$ , the following convergence result of the relaxed steepest descent algorithm may be established (Raydan & Svaiter, 2002).

**Theorem 3.9** *For the quadratic problem*

$$\min f(x) = \frac{1}{2}x^T Q x - b^T x,$$

where  $Q \in \mathbb{R}^{n \times n}$  is a symmetric and positive definite matrix, if the sequence  $\theta_k \in [0, 2]$  has an accumulation point  $\theta^* \in [0, 2]$ , then the sequence  $\{x_k\}$ :

**Table 3.1** Number of iterations of the steepest descent and of the relaxed steepest descent for the quadratic problem from Example 3.3

<i>n</i>	Steepest descent	Relaxed steepest descent $\theta_k \in [0,1]$	Relaxed steepest descent $\theta_k \in [0,2]$
1000	6682	812	860
2000	13358	950	1676
3000	20034	1627	1684
4000	26710	1407	1780
5000	33386	1615	1962
6000	40062	2198	1992
7000	46738	2684	1850
8000	534414	2440	2348
9000	60092	1765	3028
10000	66768	3005	3231

$$x_{k+1} = x_k - \theta_k \alpha_k g_k; \quad \alpha_k = \frac{g_k^T g_k}{g_k^T Q g_k}, \quad g_k = Qx_k,$$

generated by the relaxed steepest descent algorithm is linearly convergent to  $x^*$ .

**Proof** Observe that, for any  $k$ , the function

$$\Phi_k(\theta) = f(x_k - \theta \alpha_k g_k)$$

is a convex polynom of second order which has a minimum in point  $\theta = 1$ . Moreover,  $\Phi_k(0) = \Phi_k(2)$  and for any  $\theta \in [0, 2]$ ,  $\Phi_k(\theta) \leq \Phi_k(0)$ . Therefore, for all  $k$ ,  $f(x_{k+1}) \leq f(x_k)$ . Since  $f$  is lower bounded, it follows that

$$\lim_{k \rightarrow \infty} (f(x_k) - f(x_{k+1})) = 0.$$

We can write

$$f(x_k - \theta \alpha_k g_k) = f(x_k) - \left(\theta - \frac{1}{2}\theta^2\right) \frac{(g_k^T g_k)^2}{g_k^T Q g_k}.$$

Observe that for  $\theta \in [0, 2]$ ,  $\theta - \theta^2/2 \geq 0$ . Therefore,  $\{f(x_k)\}$  is a decreasing sequence convergent to  $f(x^*)$ .

There is a  $\xi \in (0, 1)$  such that  $\xi < \theta^* < 2 - \xi$ . Since  $\Phi_k(\theta)$  is convex for all  $\theta \in (\xi, 2 - \xi)$  we have  $\Phi_k(\theta) < \Phi_k(\xi)$ . But,

$$\Phi_k(\xi) < \Phi_k(0) - \xi(\Phi_k(0) - \Phi_k(1)).$$

Since

$$\Phi_k(0) - \Phi_k(1) = \frac{1}{2} \frac{(g_k^T g_k)^2}{g_k^T Q g_k},$$

it follows that

$$\Phi_k(\xi) < \Phi_k(0) - \frac{\xi}{2} \frac{(g_k^T g_k)^2}{g_k^T Q g_k}.$$

Therefore,

$$\Phi_k(0) - \Phi_k(\xi) > \frac{\xi}{2} \frac{(g_k^T g_k)^2}{g_k^T Q g_k} \geq \frac{\xi}{2} \frac{g_k^T g_k}{\lambda_{\max}},$$

that is

$$f(x_k) - f(x_k - \xi \alpha_k g_k) > \frac{\xi}{2\lambda_{\max}} \|g_k\|_2^2.$$

But,  $f(x_k) - f(x_k - \xi \alpha_k g_k) \rightarrow 0$ , showing that  $g_k \rightarrow 0$  and hence  $x \rightarrow x^*$ , solution of the problem. Since  $\{f(x_k)\}$  is a decreasing sequence, this tends to  $f(x^*)$ , that is  $x \rightarrow x^*$ .  $\blacklozenge$

The relaxed steepest descent algorithm can be used for solving general continuously differentiable functions (Andrei, 2004b). The following example illustrates the performances of the steepest descent algorithm (SDB) versus the relaxed steepest descent (RSDB), where the stepsize in both algorithms is computed by backtracking.

**Example 3.4** Consider the function

$$f(x) = (x_1 - 3)^2 + \sum_{i=2}^n \left( x_1 - 3 - 2(x_1 + x_2 + \cdots + x_i) \right)^2.$$

Let  $x_0 = [0.001, \dots, 0.001]$  be the initial point and the criterion for stopping the iterations  $\|\nabla f(x_k)\| \leq \varepsilon_g$ , where  $\varepsilon_g = 10^{-6}$ . Then the number of iterations and the average stepsize required by SDB and by RSDB with backtracking ( $\rho = 0.0001$  and  $\sigma = 0.8$ ), for different values of  $n$ , is as in Table 3.2.

These numerical examples show the *limits of the steepest descent* algorithm. Observe that a very simple, small multiplicative modification of the stepsize through a random variable uniformly distributed in the interval  $[0, 1]$  determines major changes in the behavior of the steepest descent algorithm. In other words, the steepest descent algorithm has very reduced robustness at the variations of the stepsize. In fact, for strongly convex functions, the following result can be proved (Raydan & Svaiter, 2002).

**Table 3.2** Number of iterations and the average stepsize for SDB and RSDB with backtracking for Example 3.4

<i>n</i>	SDB		RSDB	
	#iter	Average stepsize	#iter	Average stepsize
10	186997	0.103413	40462	0.327121
20	253806	0.047808	101595	0.217972
30	410108	0.031707	105885	0.172842
40	780362	0.024489	122293	0.146184
50	829749	0.020759	144316	0.126295

**Theorem 3.10** If the sequence  $\{\theta_k\}$  has an accumulation point  $\theta^* \in (0, 1)$ , then the sequence  $\{x_k\}$  generated by the relaxed steepest descent algorithm converges to  $x^*$ .

**Proof** Consider the function  $\varphi(\theta) = f(x_k - \theta \alpha_k g_k)$ , where  $g_k = \nabla f(x_k)$ . We can write

$$f(x_k - \theta \alpha_k g_k) = f(x_k) - \theta \alpha_k g_k^T g_k + \frac{1}{2} \theta^2 \alpha_k^2 g_k^T \nabla^2 f(x_k) g_k.$$

Since  $f$  is strongly convex, it follows that  $\varphi(\theta)$  is a convex function and  $\varphi(0) = f(x_k)$ . From the strong convexity of  $f$  we have

$$f(x_k - \theta \alpha_k g_k) \leq f(x_k) - \left( \theta - \frac{M\alpha_k}{2} \theta^2 \right) \alpha_k \|g_k\|_2^2.$$

But,  $\theta - \frac{M\alpha_k}{2} \theta^2$  is a concave function and negative on  $(0, 2/M\alpha_k)$  with maximum value  $1/2M\alpha_k$  in  $1/M\alpha_k$ . Therefore, for all  $k$ ,  $f(x_{k+1}) \leq f(x_k)$ . Since  $f$  is lower bounded, it follows that

$$\lim_{k \rightarrow \infty} (f(x_k) - f(x_{k+1})) = 0. \quad \blacklozenge$$

In the following, let us assume that function  $f$  is strongly convex and the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  is closed. Then, the following theorem can be proved, which shows the linear convergence of the relaxed steepest descent algorithm with backtracking.

**Theorem 3.11** For strongly convex functions, the steepest descent algorithm relaxed through the sequence  $\{\theta_k\}$ , where  $\theta_k$  is a random variable uniformly distributed in the interval  $[0, 1]$  and where the stepsize is computed by backtracking with parameters  $\rho$  and  $\beta$ , is linear convergent and

$$f(x_k) - f^* \leq \left( \prod_{i=0}^{k-1} c_i \right) (f(x_0) - f^*), \quad (3.17)$$

where

$$c_i = 1 - \min \{2m\rho\theta_i, 2m\rho\theta_i\beta/M\} < 1. \quad (3.18)$$

**Proof** Observe that the backtracking uses the parameters  $\rho$  and  $\beta$  as in Algorithm 2.5 or as in Algorithm 2.6. Consider  $0 < \theta < 1$ , then

$$f(x_k - \theta \alpha_k g_k) \leq f(x_k) - \left( \theta - \frac{M\alpha_k}{2} \theta^2 \right) \alpha_k \|g_k\|_2^2.$$

Note that  $\theta - \frac{M\alpha_k}{2} \theta^2$  is a concave function and for all  $0 \leq \theta \leq \frac{1}{M\alpha_k}$ ,

$$\theta - \frac{M\alpha_k}{2} \theta^2 \geq \frac{\theta}{2}.$$

Therefore,

$$f(x_k - \theta \alpha_k g_k) \leq f(x_k) - \frac{\theta}{2} \alpha_k \|g_k\|_2^2 \leq f(x_k) - \rho \theta \alpha_k \|g_k\|_2^2,$$

since  $\rho \leq 1/2$ . Hence, the backtracking terminates with  $\alpha_k = 1$  or with a value  $\alpha_k \geq \beta/M$ . With this, at iteration  $k$  a lower bound of the reduction of function  $f$  is obtained. In the first case we have

$$f(x_{k+1}) \leq f(x_k) - \rho\theta_k \|g_k\|_2^2,$$

while in the second one,

$$f(x_{k+1}) \leq f(x_k) - \rho\theta_k \frac{\beta}{M} \|g_k\|_2^2.$$

Therefore,

$$f(x_{k+1}) \leq f(x_k) - \min\{\rho\theta_k, \rho\theta_k\beta/M\} \|g_k\|_2^2.$$

Obviously,

$$f(x_{k+1}) - f^* \leq f(x_k) - f^* - \min\{\rho\theta_k, \rho\theta_k\beta/M\} \|g_k\|_2^2.$$

But  $\|g_k\|_2^2 \geq 2m(f(x_k) - f^*)$ , (see Appendix A). Combining these relations we find that

$$f(x_{k+1}) - f^* \leq (1 - \min\{2m\rho\theta_k, 2m\rho\theta_k\beta/M\})(f(x_k) - f^*).$$

Denote  $c_k = 1 - \min\{2m\rho\theta_k, 2m\rho\theta_k\beta/M\}$ . Therefore, for any  $k = 0, 1, \dots$

$$f(x_{k+1}) - f^* \leq c_k(f(x_k) - f^*),$$

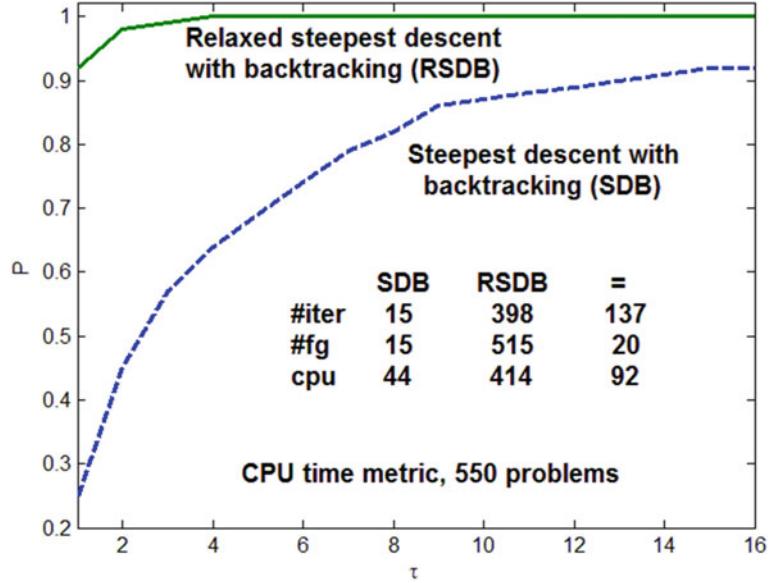
which proves the suboptimality from iteration  $k$ . Since  $c_k < 1$  the sequence  $\{f(x_k)\}$  converges to  $f^*$  like a geometrical series with a factor which depends on the bound on the condition number  $M/m$ , the backtracking parameters, the sequence  $\{\theta_k\}$  of the random numbers uniformly distributed in the interval  $[0,1]$  as well as on the initial suboptimality. Therefore, the relaxed steepest descent algorithm is linear convergent.  $\blacklozenge$

## Numerical Study: SDB Versus RSDB

This numerical study is devoted to comparing the performances of the steepest descent algorithm (SDB) with the relaxed steepest descent algorithm (RSDB), both of them being equipped with backtracking. To achieve this, let us consider the collection UOP (Andrei, 2020a), which includes 80 large-scale unconstrained optimization problems in generalized or extended form. For each test function from this collection, we have considered 10 numerical experiments with the number of variables increasing as  $n = 100, 200, \dots, 1000$ . Therefore, a set of 800 unconstrained optimization problems are solved in our numerical experiments with SDB and RSDB unconstrained optimization algorithms. Figure 3.7 presents the performances of these algorithms subject to the CPU computing time. Observe that the relaxed steepest descent with backtracking (RSDB) is way more efficient and more robust than the steepest descent with backtracking (SDB).

For example, subject to the CPU computing time, we can see that RSDB is faster in solving 414 problems, while SDB is faster in solving only 44 problems. Out of 800 problems solved in this numerical study, only for 550 does the criterion (1.3) hold.

**Fig. 3.7** Relaxed steepest descent with backtracking (RSDB) versus steepest descent with backtracking (SDB)



### 3.3 The Accelerated Steepest Descent

Let us present in this section the *accelerated steepest descent* algorithms for solving the unconstrained optimization problem (3.1) (Andrei, 2005b, 2006a). Suppose that function  $f$  is twice continuously differentiable. At the current iteration  $k$ ,  $x_k$ ,  $f_k$ ,  $g_k$ , and the search direction  $d_k = -g_k$  are known. Now, by the weak Wolfe line-search (2.54) and (2.59), the stepsize  $\alpha_k$  can be computed, and thus the following point  $z = x_k - \alpha_k g_k$  is determined. The first Wolfe condition (2.54) shows that the stepsize  $\alpha_k > 0$  satisfies

$$f(z) = f(x_k - \alpha_k g_k) \leq f(x_k) - \rho \alpha_k g_k^T g_k.$$

With these, let us introduce the *accelerated steepest descent algorithm* by means of the following iterative scheme:

$$x_{k+1} = x_k - \theta_k \alpha_k g_k, \quad (3.19)$$

where  $\theta_k > 0$  is a parameter which is to be determined in such a way as to improve the algorithm. Now, we have

$$f(x_k - \alpha_k g_k) = f(x_k) - \alpha_k g_k^T g_k + \frac{1}{2} \alpha_k^2 g_k^T \nabla^2 f(x_k) g_k + o\left(\|\alpha_k g_k\|^2\right).$$

On the other hand, for  $\theta > 0$  it follows that

$$f(x_k - \theta \alpha_k g_k) = f(x_k) - \theta \alpha_k g_k^T g_k + \frac{1}{2} \theta^2 \alpha_k^2 g_k^T \nabla^2 f(x_k) g_k + o\left(\|\theta \alpha_k g_k\|^2\right).$$

Therefore,

$$f(x_k - \theta \alpha_k g_k) = f(x_k - \alpha_k g_k) + \Psi_k(\theta), \quad (3.20)$$

where

$$\begin{aligned}\Psi_k(\theta) = & \frac{1}{2} (\theta^2 - 1) \alpha_k^2 g_k^T \nabla^2 f(x_k) g_k + (1 - \theta) \alpha_k g_k^T g_k \\ & + \theta^2 \alpha_k o\left(\alpha_k \|g_k\|^2\right) - \alpha_k o\left(\alpha_k \|g_k\|^2\right).\end{aligned}\quad (3.21)$$

Let us denote

$$\begin{aligned}a_k &\equiv \alpha_k g_k^T g_k \geq 0, \\ b_k &\equiv \alpha_k^2 g_k^T \nabla^2 f(x_k) g_k, \\ \varepsilon_k &\equiv o\left(\alpha_k \|g_k\|^2\right).\end{aligned}$$

Observe that  $a_k \geq 0$  and for convex functions,  $b_k \geq 0$ . Therefore,

$$\Psi_k(\theta) = \frac{1}{2} (\theta^2 - 1) b_k + (1 - \theta) a_k + \theta^2 \alpha_k \varepsilon_k - \alpha_k \varepsilon_k. \quad (3.22)$$

But,  $\Psi'_k(\theta) = (b_k + 2\alpha_k \varepsilon_k)\theta - a_k$  and  $\Psi'_k(\theta_m) = 0$  for

$$\theta_m = \frac{a_k}{b_k + 2\alpha_k \varepsilon_k}. \quad (3.23)$$

Observe that  $\Psi'_k(0) = -a_k \leq 0$ . Therefore, assuming that  $b_k + 2\alpha_k \varepsilon_k > 0$ , then  $\Psi_k(\theta)$  is a convex quadratic function with the minimum value in point  $\theta_m$  and

$$\Psi_k(\theta_m) = -\frac{(a_k - (b_k + 2\alpha_k \varepsilon_k))^2}{2(b_k + 2\alpha_k \varepsilon_k)} \leq 0.$$

Considering  $\theta = \theta_m$  in (3.20) and since  $b_k \geq 0$ , it follows that for every  $k$ ,

$$f(x_k - \theta_m \alpha_k g_k) = f(x_k - \alpha_k g_k) - \frac{(a_k - (b_k + 2\alpha_k \varepsilon_k))^2}{2(b_k + 2\alpha_k \varepsilon_k)} \leq f(x_k - \alpha_k g_k),$$

which is a possible improvement of the values of function  $f$  (when  $a_k - (b_k + 2\alpha_k \varepsilon_k) \neq 0$ ).

Therefore, using this simple multiplicative modification of the stepsize  $\alpha_k$  as  $\theta_k \alpha_k$ , where  $\theta_k = \theta_m = a_k / (b_k + 2\alpha_k \varepsilon_k)$ , it follows that

$$\begin{aligned}f(x_{k+1}) &= f(x_k - \theta_k \alpha_k g_k) \leq f(x_k) - \rho \alpha_k g_k^T g_k - \frac{(a_k - (b_k + 2\alpha_k \varepsilon_k))^2}{2(b_k + 2\alpha_k \varepsilon_k)} \\ &= f(x_k) - \left[ \rho a_k + \frac{(a_k - (b_k + 2\alpha_k \varepsilon_k))^2}{2(b_k + 2\alpha_k \varepsilon_k)} \right] \leq f(x_k),\end{aligned}\quad (3.24)$$

since

$$\rho a_k + \frac{(a_k - (b_k + 2\alpha_k \varepsilon_k))^2}{2(b_k + 2\alpha_k \varepsilon_k)} \geq 0,$$

where  $\rho \in (0, 1/2)$ .

Now, neglecting the contribution of  $\varepsilon_k$  in (3.24), an improvement of the function values is still obtained as

$$f(x_{k+1}) \leq f(x_k) - \left[ \rho a_k + \frac{(a_k - b_k)^2}{2b_k} \right] \leq f(x_k). \quad (3.25)$$

In order to get the algorithm, a procedure for the computation of  $b_k$  is needed. For this, at point  $z = x_k - \alpha_k g_k$ ,

$$f(z) = f(x_k - \alpha_k g_k) = f(x_k) - \alpha_k g_k^T g_k + \frac{1}{2} \alpha_k^2 g_k^T \nabla^2 f(\tilde{x}_k) g_k,$$

where  $\tilde{x}_k$  is a point on the line segment connecting  $x_k$  and  $z$ . On the other hand, at point  $x_k = z + \alpha_k d_k$ ,

$$f(x_k) = f(z + \alpha_k g_k) = f(z) + \alpha_k g_z^T g_k + \frac{1}{2} \alpha_k^2 g_z^T \nabla^2 f(\bar{x}_k) g_z,$$

where  $g_z = \nabla f(z)$  and  $\bar{x}_k$  is a point on the line segment connecting  $x_k$  and  $z$ . Having in view the local character of the searching and that the distance between  $x_k$  and  $z$  is small enough, we can consider  $\tilde{x}_k = \bar{x}_k = x_k$ . So, by adding the above equalities, the following value for  $b_k$  is obtained:

$$b_k = -\alpha_k y_k^T g_k, \quad (3.26)$$

where  $y_k = g_z - g_k$ . Observe that the computation of  $b_k$  needs an additional evaluation of the gradient in point  $z$ . Therefore, neglecting the contribution of  $\varepsilon_k$  and considering  $\theta_k = \theta_m = a_k/b_k$  in (3.19), the following algorithm can be presented.

**Algorithm 3.2** Accelerated steepest descent (with Wolfe line-search)

- |    |   |
|----|---|
| 1. | Choose a starting point $x_0 \in \text{dom } f$ and compute: $f_0 = f(x_0)$ and $g_0 = \nabla f(x_0)$ . Select $\varepsilon_A > 0$ sufficiently small and positive values $0 < \rho < \sigma < 1$ used in the Wolfe line-search conditions. Set $d_0 = -g_0$ and $k = 0$  |
| 2. | Test a criterion for stopping the iterations. If the test is satisfied, then stop; otherwise, continue with step 3  |
| 3. | Using the weak Wolfe line-search conditions (2.54) and (2.59), determine the stepsize $\alpha_k$  |
| 4. | Update the variables $x_{k+1} = x_k - \alpha_k g_k$ and compute $f_{k+1}$ and $g_{k+1}$ . Compute $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$   |
| 5. | Acceleration scheme:<br>(a) Compute: $z = x_k - \alpha_k g_k$ , $g_z = \nabla f(z)$ and $y_k = g_z - g_k$<br>(b) Compute: $a_k = \alpha_k g_k^T g_k$ , and $b_k = -\alpha_k y_k^T g_k$<br>(c) if $ b_k  < \varepsilon_A$ , then go to step 6<br>(d) if $ b_k  \geq \varepsilon_A$ , then compute $\theta_k = a_k/b_k$ . Compute $z = x_k - \theta_k \alpha_k g_k$ , $g_z = \nabla f(z)$ , $s_z = z - x_k$ and $y_z = g_z - g_k$ . If $y_z^T s_z > 0$ , then set $x_{k+1} = z$ and compute $f_{k+1} = f(x_{k+1})$ , $g_{k+1} = \nabla f(x_{k+1})$ , $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$ . Go to step 6. Otherwise, if $y_z^T s_z \leq 0$ , then go to step 6 |
| 6. | Set $k = k + 1$ and go to step 2  |

◆

Usually,  $\varepsilon_A$  is epsilon machine. The un-accelerated steepest descent algorithm can immediately be obtained from the above algorithm by skipping step 5. If  $f$  is bounded along the direction  $-g_k$ , then there exists a stepsize  $\alpha_k$  satisfying the Wolfe line-search conditions (2.54) and (2.59). Under reasonable assumptions, the Wolfe conditions are sufficient to prove the global convergence of the algorithm.

The algorithm is equipped with an acceleration scheme (see step 5) introduced by Andrei (2005b, 2006a). This scheme modifies the stepsize determined by the Wolfe line-search conditions in such a way as to improve the reduction of the minimizing function values along the iterations. It is proved that this acceleration scheme is linear convergent. However, when the acceleration scheme is used, in some unconstrained optimization algorithms, it is necessary to monitor whether  $y_k^T s_k > 0$ . The Wolfe line-search implemented in step 3 of the algorithm ensures that  $y_k^T s_k > 0$ , which is crucial in quasi-

Newton unconstrained optimization algorithms. If  $\nabla^2 f(x_k)$  is positive definite, it is easy to see that in the new point  $x_{k+1} = x_k - \theta_k \alpha_k g_k$  we have  $y_k^T s_k > 0$  for  $k = 0, 1, \dots$ . In general, for solving the unconstrained optimization problems by using, for example, the accelerated conjugate gradient or the accelerated quasi-Newton methods, if the new  $y_z^T s_z \leq 0$ , then the algorithm updates  $x_{k+1} = x_k + \alpha_k d_k$ . Otherwise, if  $y_z^T s_z > 0$ , then the algorithm updates  $x_{k+1} = x_k + \theta_k \alpha_k d_k$ , where the acceleration factor  $\theta_k$  is computed as in Algorithm 3.2. Having in view that the contribution of the acceleration to reducing the values of the minimizing function is small, it follows that by acceleration, the condition  $y_k^T s_k > 0$  is conserved along the iterations and hence the above scheme is very seldom used.

Observe that, if  $|a_k| > b_k$ , then  $\theta_k > 1$ . In this case,  $\theta_k \alpha_k > \alpha_k$  and it is also possible that  $\theta_k \alpha_k \leq 1$  or  $\theta_k \alpha_k > 1$ . Hence, the stepsize  $\theta_k \alpha_k$  can be greater than 1. On the other hand, if  $|a_k| \leq b_k$ , then  $\theta_k \leq 1$ . In this case,  $\theta_k \alpha_k \leq \alpha_k$ , so the stepsize  $\theta_k \alpha_k$  is reduced. Therefore, if  $|a_k| \neq b_k$ , then  $\theta_k \neq 1$  and the stepsize  $\alpha_k$  computed by the Wolfe conditions will be modified by its increasing or its decreasing through factor  $\theta_k$ .

Neglecting  $\varepsilon_k$  in (3.22), we can see that  $\Psi_k(1) = 0$  and if  $|a_k| \leq b_k/2$ , then  $\Psi_k(0) = -a_k - b_k/2 \leq 0$  and  $\theta_k < 1$ . Therefore, for any  $\theta \in [0, 1]$ ,  $\Psi_k(\theta) \leq 0$ . Consequently, for any  $\theta \in (0, 1)$  it follows that  $f(x_k + \theta \alpha_k d_k) < f(x_k)$ . In this case, for any  $\theta \in [0, 1]$ ,  $\theta \alpha_k \leq \alpha_k$ . However, in our algorithm we selected  $\theta_k = \theta_m$  as the point achieving the minimum value of  $\Psi_k(\theta)$ .

In the following, for strongly convex functions, let us prove the *linear convergence* of the acceleration scheme (Andrei, 2006a). For strongly convex functions, it is easy to prove that,  $\|\nabla f(x)\|^2 \geq 2m(f(x) - f(x^*))$ , for all  $x \in S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ , where  $x^*$  is a local solution of (3.1). (See Appendix A.)

**Proposition 3.1** Suppose that  $f$  is a strongly convex function on the level set  $S = \{x : f(x) \leq f(x_0)\}$ . Then, the sequence  $\{x_k\}$  generated by the accelerated steepest descent algorithm 3.2 converges linearly to  $x^*$ , solution to the problem (3.1).

**Proof** From (3.24),  $f(x_{k+1}) \leq f(x_k)$  for all  $k$ . Since  $f$  is bounded below, it follows that

$$\lim_{k \rightarrow \infty} (f(x_k) - f(x_{k+1})) = 0.$$

Now, since  $f$  is strongly convex, there exist positive constants  $m$  and  $M$  so that  $mI \leq \nabla^2 f(x) \leq MI$  on  $S$ . Suppose that  $x_k - \alpha g_k \in S$  and  $x_k - \theta_m \alpha g_k \in S$  for all  $\alpha > 0$ , where  $\theta_m$  is the acceleration factor. Therefore,

$$f(x_k - \theta_m \alpha g_k) \leq f(x_k - \alpha g_k) - \frac{(a_k - b_k)^2}{2b_k}. \quad (3.27)$$

But, from the strong convexity, the following quadratic upper bound of  $f(x_k - \alpha g_k)$  is true

$$f(x_k - \alpha g_k) \leq f(x_k) - \alpha \|g_k\|_2^2 + \frac{1}{2} M \alpha^2 \|g_k\|_2^2.$$

Observe that  $-\alpha + M\alpha^2/2$  is a convex function. Therefore, for any  $0 \leq \alpha \leq 1/M$ , it follows that

$$-\alpha + M\alpha^2/2 \leq -\alpha/2.$$

Hence,

$$\begin{aligned} f(x_k - \alpha g_k) &\leq f(x_k) - \alpha \|g_k\|_2^2 + \frac{1}{2} M \alpha^2 \|g_k\|_2^2 \\ &\leq f(x_k) - \frac{\alpha}{2} \|g_k\|_2^2 \leq f(x_k) - \rho \alpha \|g_k\|_2^2, \end{aligned}$$

since  $\rho \leq 1/2$ .

The line-search with backtracking terminates either with  $s = 1$ , or with a value  $\alpha \geq \beta/M$ . This gives a lower bound on the reduction of the minimizing function  $f$ . For  $\alpha = 1$  we have

$$f(x_k - \alpha g_k) \leq f(x_k) - \rho \|g_k\|_2^2$$

and for  $\alpha \geq \beta/M$

$$f(x_k - \alpha g_k) \leq f(x_k) - \frac{\rho\beta}{M} \|g_k\|_2^2.$$

Therefore, for  $0 \leq \alpha \leq 1/M$ , we have

$$f(x_k - \alpha g_k) \leq f(x_k) - \min \left\{ \rho, \frac{\rho\beta}{M} \right\} \|g_k\|_2^2. \quad (3.28)$$

On the other hand,

$$\frac{(a_k - b_k)^2}{2b_k} \geq \frac{(\alpha \|g_k\|_2^2 - \alpha^2 M \|g_k\|_2^2)^2}{2\alpha^2 M \|g_k\|_2^2} = \frac{(1 - \alpha M)^2}{2M} \|g_k\|_2^2.$$

Now, as above, for  $\alpha = 1$

$$\frac{(a_k - b_k)^2}{2b_k} \geq \frac{(1 - M)^2}{2M} \|g_k\|_2^2.$$

For  $\alpha \geq \beta/M$

$$\frac{(a_k - b_k)^2}{2b_k} \geq \frac{(1 - \beta)^2}{2M} \|g_k\|_2^2.$$

Hence,

$$\frac{(a_k - b_k)^2}{2b_k} \geq \min \left\{ \frac{(1 - M)^2}{2M}, \frac{(1 - \beta)^2}{2M} \right\} \|g_k\|_2^2. \quad (3.29)$$

From (3.27), (3.28) and (3.29) we get

$$\begin{aligned} f(x_k - \theta_m \alpha g_k) &\leq f(x_k) - \min \left\{ \rho, \frac{\rho\beta}{M} \right\} \|g_k\|_2^2 \\ &\quad - \min \left\{ \frac{(1 - M)^2}{2M}, \frac{(1 - \beta)^2}{2M} \right\} \|g_k\|_2^2. \end{aligned} \quad (3.30)$$

Therefore,

$$f(x_k) - f(x_{k+1}) \geq \left[ \min \left\{ \rho, \frac{\rho\beta}{M} \right\} + \min \left\{ \frac{(1 - M)^2}{2M}, \frac{(1 - \beta)^2}{2M} \right\} \right] \|g_k\|_2^2.$$

But,  $f(x_k) - f(x_{k+1}) \rightarrow 0$  and consequently  $g_k$  tends to zero, that is,  $x_k$  converges to  $x^*$ . Having in view that  $f(x_k)$  is a nonincreasing sequence, it follows that  $f(x_k)$  converges to  $f(x^*)$ . From (3.30) we see that

$$f(x_{k+1}) \leq f(x_k) - \left[ \min \left\{ \rho, \frac{\rho\beta}{M} \right\} + \min \left\{ \frac{(1-M)^2}{2M}, \frac{(1-\beta)^2}{2M} \right\} \right] \|g_k\|_2^2.$$

Combining this with

$$\|g_k\|_2^2 \geq 2m(f(x_k) - f^*)$$

and subtracting  $f^*$  from both sides of the above inequality, we have

$$f(x_{k+1}) - f^* \leq c(f(x_k) - f^*), \quad (3.31)$$

where

$$c = 1 - \min \left\{ 2m\rho, \frac{2m\rho\beta}{M} \right\} - \min \left\{ \frac{(1-M)^2 m}{M}, \frac{(1-\beta)^2 m}{M} \right\} < 1.$$

Therefore,  $f(x_k)$  converges to  $f(x^*)$  at least as fast as a geometric series with a factor that depends on the parameter  $\rho$  in the first Wolfe condition and on the bounds  $m$  and  $M$ , i.e., the convergence is at least linearly.  $\blacklozenge$

**Remark 3.1** Basically, the acceleration scheme modifies the stepsize  $\alpha_k$  in a multiplicative way to improve the reduction of the function values along the iterations. In the accelerated algorithm, instead of  $x_{k+1} = x_k + \alpha_k d_k$ , the new estimation of the minimum point is computed as

$$x_{k+1} = x_k + \eta_k \alpha_k d_k, \quad (3.32)$$

where the acceleration factor  $\eta_k$  is computed as

$$\eta_k = -\frac{a_k}{b_k}, \quad (3.33)$$

where  $a_k = \alpha_k g_k^T g_k$ ,  $b_k = -\alpha_k (g_z - g_k)^T g_k$ ,  $g_z = \nabla f(z)$  and  $z = x_k - \alpha_k g_k$ . Hence, if  $|b_k| \geq \varepsilon_A$ , where  $\varepsilon_A > 0$  is sufficiently small, then the new estimation of the solution is computed as  $x_{k+1} = x_k + \eta_k \alpha_k d_k$ ; otherwise  $x_{k+1} = x_k + \alpha_k d_k$ .  $\blacklozenge$

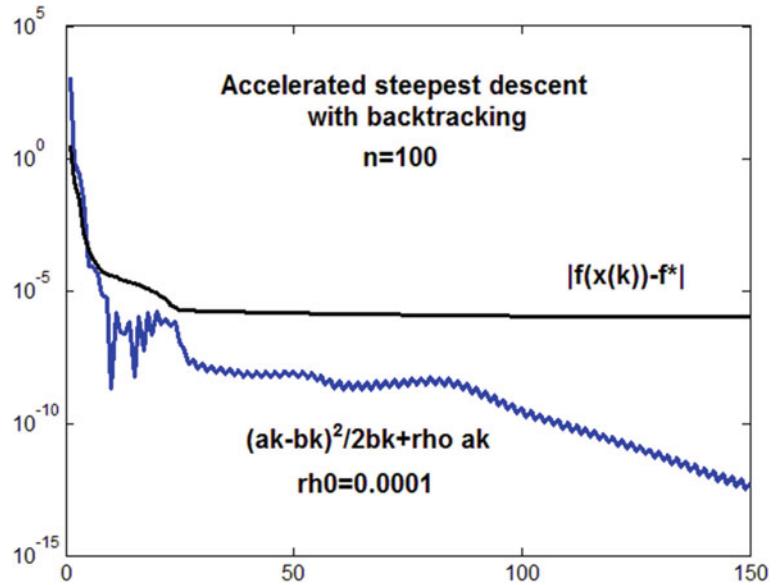
**Example 3.5** In the following, let us see the running of the accelerated steepest descent with backtracking for minimizing the function

$$f(x) = \sum_{i=1}^n \left( \left( n - \sum_{j=1}^n \cos x_j \right) + i(1 - \cos x_i) - \sin x_i \right)^2.$$

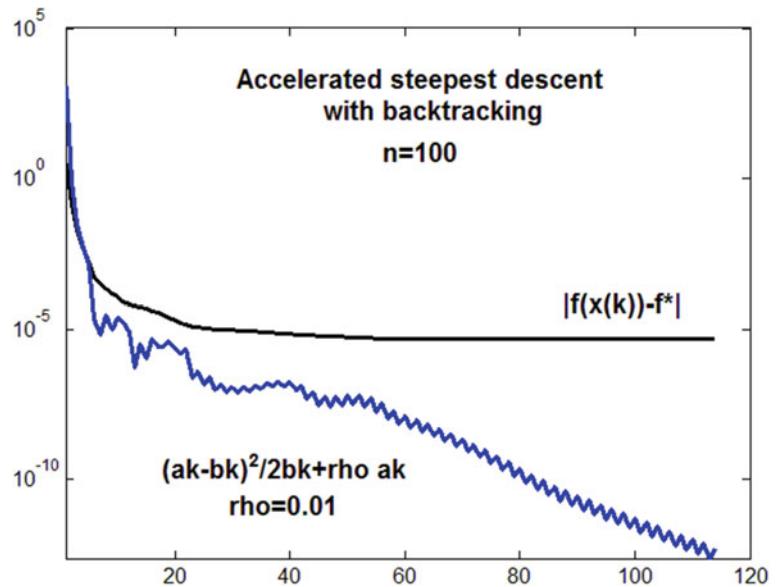
Consider  $n = 100$ , the initial point  $x_0 = [0.2, \dots, 0.2]$  and the parameters  $\rho = 0.0001$  and  $\beta = 0.8$  in the backtracking procedure. The criterion for stopping the iterations is  $\|\nabla f(x_k)\|_2 \leq \varepsilon_g$ . For minimizing this function, the algorithm needs 150 iterations to get the solution. In Fig. 3.8 observe the evolution of the error  $|f(x_k) - f^*|$  and the evolution of the contribution  $(a_k - b_k)^2/2b_k + \rho a_k$  at the reduction of the function  $f$ ; see (3.25). Figure 3.9 presents the evolution of the same elements, but this time for  $\rho = 0.01$ . In this case, for minimizing this function, the algorithm needs 114 iterations.

Observe that the quantity  $(a_k - b_k)^2/2b_k + \rho a_k$  ( $a_k < 0$ ) is indeed much smaller than the error  $|f(x_k) - f^*|$ , but it contributes to the reduction of the minimizing function even if  $\rho$  is small in the first weak Wolfe condition.

**Fig. 3.8** Accelerated steepest descent with backtracking.  $\rho = 0.0001$ . Error  $|f(x_k) - f^*|$  and contribution  $(a_k - b_k)^2 / 2b_k + \rho a_k$



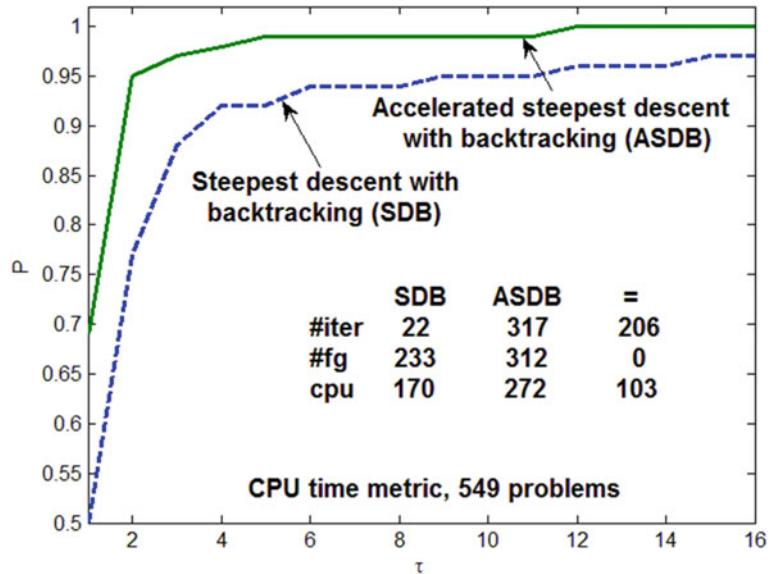
**Fig. 3.9** Accelerated steepest descent with backtracking.  $\rho = 0.01$ . Error  $|f(x_k) - f^*|$  and contribution  $(a_k - b_k)^2 / 2b_k + \rho a_k$



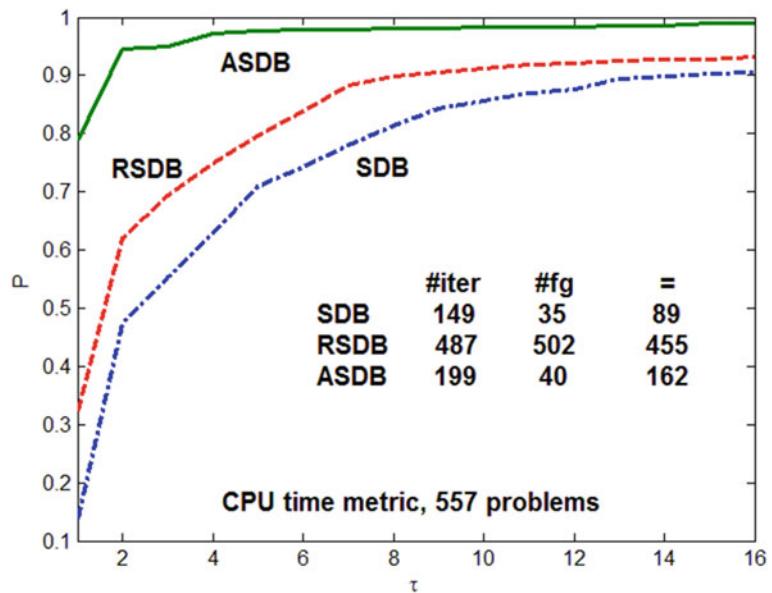
## Numerical Study

This numerical study presents a comparison of the accelerated steepest descent with backtracking (ASDB) versus the steepest descent with backtracking (SDB) and versus the relaxed steepest descent with backtracking (RSDB). For this, the collection UOP (Andrei, 2020a), which includes 80 large-scale unconstrained optimization problems in generalized or extended form, is used. For each test function from this collection, 10 numerical experiments have been considered, with the number of variables increasing as  $n = 100, 200, \dots, 1000$ . Therefore, a set of 800 unconstrained optimization problems are solved. Figure 3.10 presents the performance profiles of ASDB versus SDB. We can see

**Fig. 3.10** Accelerated steepest descent with backtracking (ASDB) versus steepest descent with backtracking (SDB)



**Fig. 3.11** Accelerated steepest descent with backtracking (ASDB) versus relaxed steepest descent with backtracking (RSDB) and versus steepest descent with backtracking (SDB)



that the accelerated steepest descent algorithm ASDB is more efficient and more robust than its un-accelerated variant SDB. Subject to the CPU computing time, ASDB is faster in solving 272 problems, while SDB is faster in solving only 170 problems. Out of 800 problems, only for 549 of them does the criterion (1.3) hold.

Figure 3.11 presents the performance profiles of ASDB versus SDB and versus RSDB. Observe that RSDB is more efficient and more robust than SDB, but ASDB is the best.

In conclusion, if we want to minimize the continuous differentiable function  $f$  and if the current point is  $x_k$ , then, the best direction of moving to find better points is the negative gradient  $-\nabla f(x_k)$ , that is, the steepest descent direction. However, as soon as we move away from  $x_k$  along the steepest

descent direction for obtaining  $x_{k+1}$ , even if the value of the minimizing function decreases, this direction ceases to be a good one. The purpose of the acceleration is to find points on the steepest descent direction for which the reduction of the value of the minimizing function is better.

### 3.4 Comments on the Acceleration Scheme

The acceleration scheme presented in Remark 3.1 was developed for the steepest descent algorithm, but it can be applied to any other unconstrained optimization algorithms based on the descent direction, for example, the conjugate gradient, the quasi-Newton, the limited-memory BFGS, and the truncated Newton. In general, the acceleration scheme is recommended for those unconstrained optimization methods for which the evolution of the stepsize is very hieratic, having up to one or two orders of magnitude along the iterations, like the conjugate gradient algorithms. The value of  $\eta_k$  computed as in (3.33) is small, and so the variables are not dramatically modified by acceleration. When the acceleration scheme is used in conjunction with the conjugate gradient or the quasi-Newton or the limited-memory quasi-Newton BFGS methods, it is necessary to monitor whether  $y_k^T s_k > 0$ . If  $\nabla^2 f(x)$  is positive definite, then in the new point  $x_{k+1} = x_k + \eta_k \alpha_k d_k$  it follows that  $y_k^T s_k > 0$  for all iterations  $k = 0, 1, \dots$ . For solving unconstrained optimization problems by using the accelerated scheme, Algorithm 3.2 may consider

$$x_{k+1} = \begin{cases} x_k + \eta_k \alpha_k d_k, & \text{if } y_k^T s_k > 0, \\ x_k + \alpha_k d_k, & \text{if } y_k^T s_k \leq 0. \end{cases} \quad (3.34)$$

However, our intensive numerical experiments with large-scale optimization problems have proved that (3.34) does not need to be implemented. Having in view that the contribution  $(a_k - b_k)^2 / 2b_k + \rho a_k$  is small, it follows that the condition  $y_k^T s_k > 0$  is conserved along the iterations by acceleration.

The accelerated scheme proved to be a major improvement of the unconstrained optimization algorithms (Dener, Denchfield, & Munson, 2019). We should emphasize that another acceleration scheme of the optimization algorithms is preconditioning. Notice that preconditioning is quite dependent on the matrix of preconditioning. If the preconditioning matrix contains useful information about the inverse Hessian of the objective function, it is better to use it into a quasi-Newton context than into a preconditioned conjugate gradient one. For example, for solving large-scale unconstrained optimization problems, preconditioning the nonlinear conjugate gradient methods based on the Perry-Shanno scheme remains an open question with very little consensus (Andrei, 2019d) (see Chap. 5). On the other hand, the acceleration of the conjugate gradient algorithms based on the modification of the stepsize is well understood and has proved to be extremely effective in practice (see Chap. 5).

#### Notes and References

The steepest descent algorithm designed by A.L. Cauchy (1789–1857) is *not an efficient minimization method*. This simple strategy of proceeding along the negative gradient works well for functions with circular or near-circular contours, but practical optimization problems or real applications may involve functions with narrow curving valleys which need more sophisticated approaches, as we are going to see in the next chapters of this book. We have insisted on presenting the steepest descent method as it emphasizes the *pattern* of line-search methods. As we said, in this pattern, a particular iteration  $k$  consists of two operations: the choice of a *descent search direction*  $d_k$  followed by a *line-search* along this direction to get a suitable (inexact) stepsize  $\alpha_k$ .

The relaxed steepest descent algorithm was defined by Andrei (2005a). The development of the accelerated conjugate gradient algorithms was presented by Andrei (2009c, 2009d, 2010a). Discussions and comments on the accelerated scheme (3.32) and (3.33) were given by Babaie-Kafaki and Rezaee (2018) and by Sun, Liu, and Liu (2021). A class of accelerated conjugate gradient methods based on a modified secant equation was presented by Ou and Lin (2020).



# The Newton Method

# 4

In the panoply of the optimization methods and in general, for solving problems that have an algebraic mathematical model, the Newton method has a central position. The idea of this method is to approximate the mathematical model through a local affine or a local quadratic model. This chapter is dedicated to presenting the Newton method for solving *algebraic nonlinear systems* on the one hand and to *minimizing smooth enough functions* on the other one. It is proved that, initialized near solution, the Newton method is quadratic convergent to a minimum point of the minimizing function. Some modifications of the Newton method and the *composite Newton method* are also presented.

---

## 4.1 The Newton Method for Solving Nonlinear Algebraic Systems

Let us consider an algebraic nonlinear system  $F(x) = 0$ , where under  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is a continuous differentiable vectorial function with the components  $f_i(x)$ ,  $i = 1, \dots, n$ , scalar functions. We are interested in finding a point  $x^*$  such that  $F(x^*) = 0$ . Let  $J(x)$  be the Jacobian matrix of  $F$  in point  $x$  defined as the  $n \times n$  matrix

$$J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

Then, given the estimation  $x_k$  of  $x^*$ , the Newton method computes the next estimation  $x_{k+1}$  by solving a local linear approximation of  $F$  (*local affine model*) in  $x_k$ , that is,

$$M_k(x) \equiv F(x_k) + J(x_k)(x - x_k) = 0. \quad (4.1)$$

Supposing that for any  $k$ , the Jacobian matrix  $J(x_k)$  is nonsingular, then  $x_{k+1}$  is defined as

$$x_{k+1} = x_k - J(x_k)^{-1}F(x_k), k = 0, 1, 2, \dots \quad (4.2)$$

This is the *Newton method for solving nonlinear algebraic system of equations*. If  $J$  is a singular matrix, then the Newton method is not defined, and some transformations have to be considered in order to overcome this situation. In the following, suppose that  $J(x^*)$  is nonsingular. By the continuity of  $J$  it follows that  $J(x_k)$  is nonsingular for any  $x_k$  near enough to  $x^*$ .

For solving the system  $F(x) = 0$ , an initial point  $x_0$  from  $\text{dom}(f_i)$ ,  $i = 1, \dots, n$ , must be specified and then apply (4.2) until a criterion for stopping the iterations has been satisfied.

**Algorithm 4.1** *Newton for  $F(x) = 0$*

- |    |   |
|----|---|
| 1. | <i>Initialization.</i> Consider the initial point $x_0 \in \mathbb{R}^n$ as well as the convergence tolerance $\varepsilon > 0$ sufficiently small. Set $k = 0$ . |
| 2. | Compute $F(x_k)$ and $J(x_k) = \nabla F(x_k)$   |
| 3. | Solve the linear algebraic system $J(x_k)d_k = -F(x_k)$   |
| 4. | If $\ d_k\ _\infty \leq \varepsilon$ , then stop; otherwise continue with step 5  |
| 5. | Set $x_{k+1} = x_k + d_k$ , $k = k + 1$ and continue with step 2  |

◆

Observe that the most delicate operations in this algorithm are in step 2, where the Jacobian matrix must be evaluated in the current point, and also in step 3, where a linear system has to be solved, known as the *Newton system*. In step 4, some other stopping criteria could be used. However, we will prove that the Newton method initialized with a “good” starting point is quadratic convergent to a local solution. Therefore, for stopping the iterations, the following criterion  $\|x_{k+1} - x_k\| \leq \varepsilon$ , is also convenient.

**Example 4.1** Consider the vectorial function  $F$  defined as

$$F(x) = \begin{bmatrix} x_1^2 + x_2^2 - 2 \\ \exp(x_1 - 1) - x_2 \end{bmatrix}.$$

Applying the Newton method with initial point  $x_0 = [1.4 \ 0.8]$ , the following results are obtained as in Tables 4.1 and 4.2. Observe that only six iterations are needed to get the solution.

From Table 4.2 we can see that

$$\frac{\|x^* - x_2\|}{\|x^* - x_1\|^2} = 0.477, \quad \frac{\|x^* - x_3\|}{\|x^* - x_2\|^2} = 0.499, \quad \frac{\|x^* - x_4\|}{\|x^* - x_3\|^2} = 0.5$$

thus showing that

$$\frac{\|x^* - x_{k+1}\|}{\|x^* - x_k\|^2}$$

is asymptotically constant when  $k \rightarrow \infty$ .

Since the error  $\|x^* - x_k\|$  is very fast reduced to zero, this cannot be numerically verified. ◆

**Table 4.1** Iterations generated by the Newton method

$k$	$x_k^1$	$x_k^2$
0	.14000000000000E+01	.80000000000000E+00
1	.1070918449595E+01	.1000892713209E+01
2	.1002402519583E+01	.9999422607128E+00
3	.1000002882268E+01	.9999999985242E+00
4	.1000000000004E+01	.1000000000000E+01
5	.1000000000000E+01	.1000000000000E+01

**Table 4.2** Evolution of  $\|x^* - x_k\|$  and  $\|F(x_k)\|$ 

$k$	$\ x^* - x_k\ $	$\ F(x_k)\ $
0	.447213595499958E+00	.9157627488965E+00
1	.709240680574689E-01	.1654342180140E+00
2	.240321330115156E-02	.5302195270017E-02
3	.288226844402807E-05	.6442977199172E-05
4	.415378842433256E-11	.9288153280959E-11
5	.000000000000000E+00	.0000000000000E+00

Observe that the local affine model  $M_k(x)$  which approximates  $F(x)$  in the current point  $x_k$  is very efficient for the determination of the next estimation of the solution. In the following theorem, the quadratic convergence is proved, which is the main result of the Newton method.

**Theorem 4.1** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a vectorial continuous differentiable function and  $F(x^*) = 0$ . If:

- (i) Jacobian  $J(x^*)$  of  $F$  in  $x^*$  is nonsingular.
- (ii) Jacobian  $J(x)$  is continuously Lipschitz in a neighborhood of  $x^*$ ,

then, for all  $x_0$  near enough to  $x^*$ , the Newton method defined by (4.2) generates a sequence  $\{x_k\}$  which is quadratic convergent to  $x^*$ .

Before proving this theorem, we need some technical results as follows. We can say that  $J : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$  is Lipschitz continuous on the open set  $D \in \mathbb{R}^n$  if there exists a positive constant  $L$ , called the *Lipschitz constant*, such that  $\|J(x) - J(y)\| \leq L\|x - y\|$  for all  $x, y \in D$ . In other words, the difference  $J(x) - J(y)$  is proportional with  $x - y$ . Observe that the Lipschitz condition on  $J$  is stronger than the continuity of  $J$ , but weaker than the twice differentiability of  $J$ . Mainly,  $L$  gives a measure of the nonlinearity of function  $F$  in  $x^*$ .

Another important point is the quadratic convergence. It is important from two points of view. On the one hand, the quadratic convergence guarantees that, if the Newton method can be initialized near solution, then it is rapid convergent to the solution. On the other hand, a very simple stopping criterion, namely,  $\|x_k - x_{k-1}\| < \epsilon$ , can be associated to the Newton method, where  $\epsilon > 0$  is a sufficiently small convergence tolerance.

**Proposition 4.1** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be continuously differentiable and  $a, b \in \mathbb{R}^n$ . Then

$$F(b) = F(a) + \int_0^1 J(a + t(b-a))(b-a)dt, \quad (4.3)$$

where  $J$  is the Jacobian matrix of  $F$ .

**Proposition 4.2** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be integrable on the interval  $[a, b]$ , then

$$\left\| \int_a^b F(t)dt \right\| \leq \int_a^b \|F(t)\| dt. \quad (4.4)$$

**Proposition 4.3** Let  $J : \mathbb{R}^m \rightarrow \mathbb{R}^{m \times m}$  be a continuously function with matrices values. If  $J(x^*)$  is nonsingular, then there exists  $\delta > 0$  such that for all  $x \in \mathbb{R}^m$  with  $\|x - x^*\| < \delta$ ,  $J(x)$  is nonsingular and

$$\|J(x)\| \leq 2\|J(x^*)\|, \quad (4.5)$$

$$\|J(x)^{-1}\| < 2\|J(x^*)^{-1}\|, \quad (4.6)$$

$$\frac{1}{2} \|J(x^*)^{-1}\|^{-1} \|x - x^*\| \leq \|F(x)\| \leq 2\|J(x^*)\| \|x - x^*\|. \quad (4.7)$$

**Proof** Let  $\delta > 0$  be sufficiently small. Then, for any  $x$  for which  $\|x - x^*\| < \delta$ , we have  $\|J(x)\| \leq \|J(x^*)\| + L\|x - x^*\|$ , where  $L$  is the Lipschitz constant associated to  $J(x)$ . Therefore, (4.5) holds if  $L\delta < \|J(x^*)\|$ .

Result (4.6) is a direct consequence of the Banach lemma (see Appendix A) if  $\|I - J(x^*)^{-1}J(x)\| < 1/2$ . This follows from

$$\delta < \frac{\|J(x^*)^{-1}\|^{-1}}{2L}$$

since

$$\begin{aligned} \|I - J(x^*)^{-1}J(x)\| &= \|J(x^*)^{-1}(J(x^*) - J(x))\| \\ &\leq L\|J(x^*)^{-1}\| \|x - x^*\| \leq L\delta\|J(x^*)^{-1}\| \leq 1/2. \end{aligned} \quad (4.8)$$

For proving (4.7) observe that if  $\|x - x^*\| < \delta$ , then  $t\|x - x^*\| < \delta$  holds for any  $0 \leq t \leq 1$ . From (4.5) it follows that

$$\|F(x)\| \leq \int_0^1 \|J(x^* + t(x-x^*))\| \|x - x^*\| dt \leq 2\|J(x^*)\| \|x - x^*\|,$$

which is the right inequality in (4.7). For proving the left inequality observe that

$$\begin{aligned} J(x^*)^{-1}F(x) &= J(x^*)^{-1} \int_0^1 J(x^* + t(x - x^*)) dt \\ &= (x - x^*) - \int_0^1 (I - J(x^*)^{-1}J(x^* + t(x - x^*))) (x - x^*) dt, \end{aligned}$$

and therefore, from (4.8) we get

$$\|J(x^*)^{-1}F(x)\| \geq \|x - x^*\| \left( 1 - \left\| \int_0^1 (I - J(x^*)^{-1}J(x^* + t(x - x^*))) dt \right\| \right) \geq \frac{1}{2} \|x - x^*\|$$

Therefore,

$$\frac{1}{2} \|x - x^*\| \leq \|J(x^*)^{-1}F(x)\| \leq \|J(x^*)^{-1}\| \|F(x)\|,$$

which completes the proof.  $\diamond$

Proposition 4.3 shows that the set of nonsingular matrices is an open set. The second part of the proposition follows from the fact that if the application  $x \rightarrow J(x)$  is continuous, then  $x \rightarrow J(x)^{-1}$  is also continuous as soon as  $J(x)^{-1}$  is definite. With these results, let us prove Theorem 4.1.

**Proof of Theorem 4.1** Suppose that the estimation  $x_k$  is sufficiently closed to  $x^*$  and  $J(x_k)$  is nonsingular. Consider the iteration

$$x_{k+1} = x_k - J(x_k)^{-1}F(x_k).$$

Now, subtracting  $x^*$  from both members we get

$$x_{k+1} - x^* = x_k - x^* - J(x_k)^{-1}F(x_k).$$

But  $F(x^*) = 0$ , therefore

$$x_{k+1} - x^* = x_k - x^* - J(x_k)^{-1}(F(x_k) - F(x^*)).$$

Now, from Proposition 4.1 we can estimate the difference  $F(x_k) - F(x^*)$  as

$$\begin{aligned} F(x_k) - F(x^*) &= \int_0^1 J(x^* + t(x_k - x^*))(x_k - x^*) dt \\ &= \int_0^1 J(x^*)(x_k - x^*) dt + \int_0^1 (J(x^* + t(x_k - x^*)) - J(x^*))(x_k - x^*) dt \\ &= J(x^*)(x_k - x^*) + \int_0^1 (J(x^* + t(x_k - x^*)) - J(x^*))(x_k - x^*) dt. \end{aligned}$$

Therefore,

$$\begin{aligned}
& \|F(x_k) - F(x^*) - J(x^*)(x_k - x^*)\| \\
&= \left\| \int_0^1 (J(x^* + t(x_k - x^*)) - J(x^*))(x_k - x^*) dt \right\| \\
&\leq \int_0^1 \| (J(x^* + t(x_k - x^*)) - J(x^*))(x_k - x^*) \| dt \\
&\leq \int_0^1 \|J(x^* + t(x_k - x^*)) - J(x^*)\| \|x_k - x^*\| dt \\
&\leq \int_0^1 L t \|x_k - x^*\|^2 dt = \frac{L}{2} \|x_k - x^*\|^2.
\end{aligned}$$

With these

$$\begin{aligned}
x_{k+1} - x^* &= x_k - x^* - J(x_k)^{-1}(F(x_k) - F(x^*)) \\
&= x_k - x^* - J(x_k)^{-1}(J(x^*)(x_k - x^*) + F(x_k) - F(x^*) - J(x^*)(x_k - x^*)) \\
&= (I - J(x_k)^{-1}J(x^*))(x_k - x^*) - J(x_k)^{-1}(F(x_k) - F(x^*) - J(x^*)(x_k - x^*)).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\|x_{k+1} - x^*\| &\leq \left\| (I - J(x_k)^{-1}J(x^*))(x_k - x^*) \right\| + \\
&\quad \|J(x_k)^{-1}(F(x_k) - F(x^*) - J(x^*)(x_k - x^*))\| \\
&\leq \|I - J(x_k)^{-1}J(x^*)\| \|x_k - x^*\| + \\
&\quad \|J(x_k)^{-1}\| \|F(x_k) - F(x^*) - J(x^*)(x_k - x^*)\| \\
&\leq \|I - J(x_k)^{-1}J(x^*)\| \|x_k - x^*\| + \frac{L}{2} \|J(x_k)^{-1}\| \|x_k - x^*\|^2.
\end{aligned}$$

From the Lipschitz continuity, we obtain the following estimation:

$$\begin{aligned}
\|I - J(x_k)^{-1}J(x^*)\| &= \|J(x_k)^{-1}(J(x_k) - J(x^*))\| \\
&\leq \|J(x_k)^{-1}\| \|J(x_k) - J(x^*)\| \leq L \|J(x_k)^{-1}\| \|x_k - x^*\|.
\end{aligned}$$

That is,

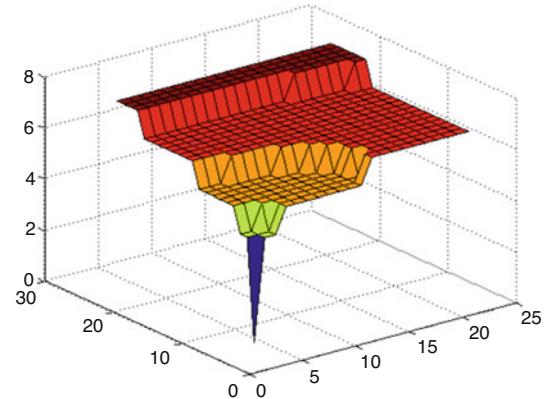
$$\|x_{k+1} - x^*\| \leq \frac{3L}{2} \|J(x_k)^{-1}\| \|x_k - x^*\|^2. \quad (4.9)$$

Now, using Proposition 4.3, it follows that for all  $x_k$  near enough to  $x^*$  we have  $\|J(x_k)^{-1}\| \leq 2M$ , where  $M = \|J(x^*)^{-1}\|$ . Therefore, for  $x_k$  sufficiently closed to  $x^*$ ,

$$\|x_{k+1} - x^*\| \leq 3LM \|x_k - x^*\|^2, \quad (4.10)$$

thus proving the quadratic convergence of the Newton method.  $\blacklozenge$

**Fig. 4.1** The number of iterations for solving the system from Example 4.1 when the Newton method is initialized in points from the domain  $[1, 3] \times [1, 3]$



We can see that if  $\|x_k - x^*\| \leq \frac{1}{6LM}$ , then  $\|x_{k+1} - x^*\| \leq \frac{1}{2} \|x_k - x^*\|$ , which is the progress of the method near solution.

**Example 4.2** Figure 4.1 shows the number of iterations needed by the Newton method initialized in a lattice of points in the domain  $[1, 3] \times [1, 3]$  for solving the nonlinear system from Example 4.1.

Note that near the solution, the method requires a relative small number of iterations. Moreover, a number of “plateaus” with initial points may be identified. For every point from these plateaus, the number of iterations is the same.

The Newton method has excellent local convergence properties. Far away from the solution, there is no guarantee that the quadratic convergence property is conserved. It is quite possible for the method not to be well defined in the sense that there exists an estimation  $x_k$  in which the Jacobian  $J(x_k)$  is a singular matrix. In such circumstances it is necessary to modify the method in order to obtain its global convergence. Theorem 4.1 makes us sure that, if the Jacobian matrix of  $F(x)$  in point  $x^*$  is nonsingular and the Lipschitz continuous and the initial point is in a neighborhood of the solution, then the Newton method is quadratic convergent in the sense of (4.10). The next theorem emphasizes this behavior of the Newton method.

**Theorem 4.2** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a continuously differentiable function on a convex set  $D \subset \mathbb{R}^n$ . Suppose that there exist  $x^* \in \mathbb{R}^n$  solution of the system  $F(x) = 0$  and the constants  $\hat{\eta}, \gamma > 0$  such that the ball  $B(x^*, \hat{\eta})$  centered in  $x^*$  of radius  $\hat{\eta}$  is included in  $D$ . Moreover, assume that there exists  $J(x^*)^{-1}$  which satisfies the condition  $\|J(x^*)^{-1}\| \leq \gamma$  and  $J(x)$  is Lipschitz continuous on the ball  $B(x^*, \hat{\eta})$  with constant  $L$ . Then, there exists  $\varepsilon > 0$  such that for any  $x_0 \in B(x^*, \varepsilon)$ , the sequence  $\{x_k\}$

$$x_{k+1} = x_k - J(x_k)^{-1}F(x_k), \quad k = 0, 1, \dots \quad (4.11)$$

generated by the Newton method is well definite, convergent to  $x^*$  and satisfies

$$\|x_{k+1} - x^*\| \leq \gamma L \|x_k - x^*\|^2, \quad k = 0, 1, \dots \quad (4.12)$$

**Proof** Select  $\varepsilon$  such that  $J(x)$  is nonsingular for any  $x \in B(x^*, \varepsilon)$ . Let us prove that the convergence is quadratic since the local error given by the local affine model used to generate each iteration of the Newton method is at most  $O(\|x_k - x^*\|^2)$ . Define

$$\varepsilon = \min \left\{ \widehat{\eta}, \frac{1}{2\gamma L} \right\}. \quad (4.13)$$

By induction after  $k$  we prove that (4.12) is satisfied at each iteration and

$$\|x_{k+1} - x^*\| \leq \frac{1}{2} \|x_k - x^*\|, \quad (4.14)$$

that is,

$$x_{k+1} \in B(x^*, \varepsilon). \quad (4.15)$$

Let us now prove that  $J(x_0)$  is nonsingular. From  $\|x_0 - x^*\| \leq \varepsilon$ , the Lipschitz continuity of  $J$  in  $x^*$  and from (4.12) it follows that

$$\begin{aligned} \|J(x^*)^{-1}[J(x_0) - J(x^*)]\| &\leq \|J(x^*)^{-1}\| \|J(x_0) - J(x^*)\| \\ &\leq \gamma L \|x_0 - x^*\| \leq \gamma L \varepsilon \leq 1/2. \end{aligned}$$

Now, from the lemma of Banach (see Appendix A), we have that  $J(x_0)$  is nonsingular and

$$\|J(x_0)^{-1}\| \leq \frac{\|J(x^*)^{-1}\|}{1 - \|J(x^*)^{-1}[J(x_0) - J(x^*)]\|} \leq 2\|J(x^*)^{-1}\| \leq 2\gamma.$$

Therefore,  $x_1$  is well definite and

$$\begin{aligned} x_1 - x^* &= x_0 - x^* - J(x_0)^{-1}F(x_0) \\ &= x_0 - x^* - J(x_0)^{-1}[F(x_0) - F(x^*)] \\ &= J(x_0)^{-1}[F(x^*) - F(x_0) - J(x_0)(x^* - x_0)]. \end{aligned}$$

Note that  $F(x^*) - F(x_0) - J(x_0)(x^* - x_0)$  is exactly the difference between  $F(x^*)$  and the affine model evaluated in  $x^*$ . Therefore, from Theorem 2.7 and the above relation, it follows that

$$\begin{aligned} \|x_1 - x^*\| &\leq \|J(x_0)^{-1}\| \|F(x^*) - F(x_0) - J(x_0)(x^* - x_0)\| \\ &\leq 2\gamma \frac{L}{2} \|x_0 - x^*\|^2 = \gamma L \|x_0 - x^*\|^2, \end{aligned}$$

thus proving (4.12). Since  $\|x_0 - x^*\| \leq 1/(2\gamma L)$ , it follows that  $\|x_1 - x^*\| \leq \frac{1}{2} \|x_0 - x^*\|$ , that is,  $x_1 \in B(x^*, \varepsilon)$ , thus completing the case  $k = 0$ . As above, the proof continues by induction.  $\blacklozenge$

In the above theorem the constants  $\gamma$  and  $L$  can be concentrated in a single constant  $L_{rel} = \gamma L$ . Since

$$\|J(x^*)^{-1}[J(x) - J(x^*)]\| \leq \|J(x^*)^{-1}\| \|J(x) - J(x^*)\| \leq \gamma L \|x - x^*\| = L_{rel} \|x - x^*\|$$

for  $x \in B(x^*, \widehat{\eta})$ , we can see that  $L_{rel}$  is a Lipschitz constant which measures the *relative nonlinearity* of function  $F$  in point  $x^*$ . In this context, Theorem 4.2 says that *the convergence radius of the Newton method is inversely proportional to the relative nonlinearity of  $F$  in point  $x^*$* . The relative nonlinearity is a fundamental concept which defines the behavior of the Newton method. All the convergence results can be easily reformulated by using this concept. However, we preferred to use the absolute nonlinearity, that is, the Lipschitz constant  $L$ , since this gives us a possibility for a deep analysis of the Newton method.

Theorem 4.2 is very powerful. The bound  $\varepsilon$  given by (4.13) is an estimation of the most disadvantageous situation, in which the domain of the quadratic convergence extends in the direction from  $x^*$  in which  $F$  is the most nonlinear. On the other hand, in the direction from  $x^*$  in which  $F$  is the least nonlinear, the domain of the quadratic convergence is much bigger.

If the condition  $J(x)$  is Lipschitz continuous on  $B(x^*, \hat{\eta})$  with constant  $L$  is replaced with the condition of Hölder's continuity, that is,  $\|J(x) - J(y)\| \leq L\|x - y\|^\delta$ , where  $\delta \in (0, 1]$ , then Theorem 4.2 remains true, where this time (4.12) is expressed as

$$\|x_{k+1} - x^*\| \leq \gamma L \|x_k - x^*\|^{1+\delta}, \quad k = 0, 1, \dots$$

### Applications: The Newton Method for Solving Nonlinear Algebraic Systems of Equations

In the following, let us present the running and the performances of the Newton method for solving some applications from the collection SMUNO (see Appendix B) as well as from the MINPACK-II collection.

**Application S5** (*Stationary solution of a chemical reactor*) The application S5 from the SMUNO collection is expressed as a nonlinear algebraic system and solved with the Newton method:

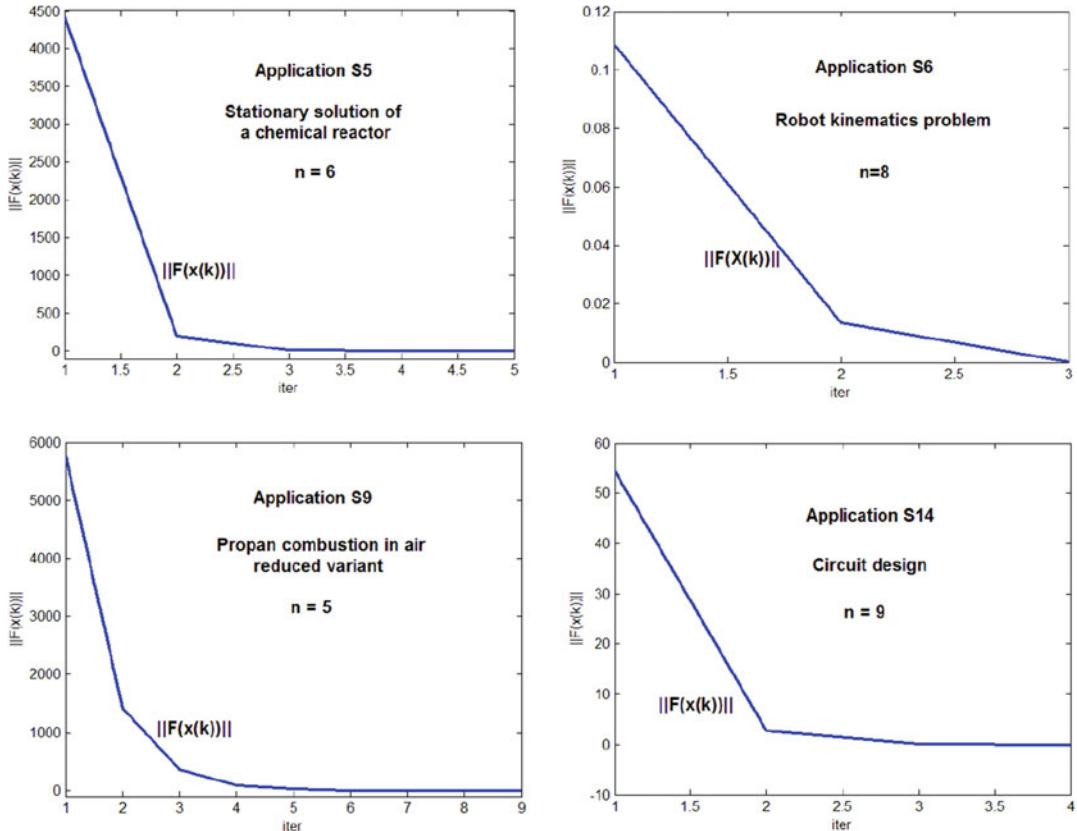
$$\begin{aligned} 1 - x_1 - k_1 x_1 x_6 + r_1 x_4 &= 0, \\ 1 - x_2 - k_2 x_2 x_6 + r_2 x_5 &= 0, \\ -x_3 + 2k_3 x_4 x_5 &= 0, \\ k_1 x_1 x_6 - r_1 x_4 - k_3 x_4 x_5 &= 0, \\ 1, 5(k_2 x_2 x_6 - r_2 x_5) - k_3 x_4 x_5 &= 0, \\ 1 - x_4 - x_5 - x_6 &= 0, \end{aligned}$$

where  $k_1 = 31.24$     $k_2 = 0.272$     $k_3 = 303.03$     $r_1 = 2.062$     $r_2 = 0.02$ .

Table 4.3 contains the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method in 5 iterations, and the functions values in  $x^*$ . Figure 4.2 presents the evolution of  $\|F(x_k)\|$ .

**Table 4.3** Stationary solution of a chemical reactor. Initial point, solution, functions values in these points. Newton method. 5 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	1.09	-35.164864	0.97424361	0.244e-14
2	1.05	-0.2403040	0.98282907	-0.241e-16
3	3.05	3626.94637	0.05151276	-0.104e-10
4	0.99	-1779.9233	0.93567106	0.524e-11
5	6.05	-1814.7127	0.00009083	0.524e-11
6	1.09	-7.13	0.06423809	-0.416e-16



**Fig. 4.2** Evolution of  $\|F(x_k)\|$  for applications S5, S6, S9, and S14

**Application S6 (Robot kinematics problem)** The application S6 from the SMUNO collection is considered as a nonlinear algebraic system and solved with the Newton method:

$$\begin{aligned}
 & 4.731(10^{-3})x_1x_3 - 0.3578x_2x_3 - \\
 & 0.1238x_1 + x_7 - 1.637(10^{-3})x_2 - 0.9338x_4 - 0.3571 = 0, \\
 & 0.2238x_1x_3 + 0.7623x_2x_3 + \\
 & 0.2638x_1 - x_7 - 0.07745x_2 - 0.6734x_4 - 0.6022 = 0, \\
 & x_6x_8 + 0.3578x_1 + 4.731(10^{-3})x_2 = 0, \\
 & -0.7623x_1 + 0.2238x_2 + 0.3461 = 0, \\
 & x_1^2 + x_2^2 - 1 = 0, \\
 & x_3^2 + x_4^2 - 1 = 0, \\
 & x_5^2 + x_6^2 - 1 = 0, \\
 & x_7^2 + x_8^2 - 1 = 0.
 \end{aligned}$$

Table 4.4 shows the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method in 3 iterations, and the functions values in  $x^*$ . Figure 4.2 presents the evolution of  $\|F(x_k)\|$ .

**Table 4.4** Robot kinematics problem. Initial point, solution, functions values in these points. Newton method. 3 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	0.164	0.0026823	0.16443166	0.111e-15
2	-0.98	-0.0098180	-0.98638847	-0.111e-15
3	-0.94	0.1050028	-0.94706369	-0.433e-17
4	-0.32	0.0017588	-0.32104573	0.555e-16
5	-0.99	-0.0127040	-0.99823316	-0.111e-15
6	-0.056	-0.014	0.05941842	-0.111e-15
7	0.41	-0.016764	0.41103315	0.522e-09
8	-0.91	-0.0038	-0.91162039	0

**Table 4.5** Propan combustion in air – reduced variant. Initial point, solution, functions values in these points. Newton method. 9 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	10	109.85000	0.003114068	-0.160e-09
2	10	209.54267	34.59792428	-0.515e-09
3	0.05	-0.3487418	0.065041778	-0.368e-09
4	50.5	5098.5172	0.859378096	0.158e-06
5	0.05	2659.2930	0.036951859	0.788e-07

**Application S9** (*Propan combustion in air – reduced variant*) This application from the SMUNO collection is written here as a nonlinear algebraic system:

$$\begin{aligned} x_1x_2 + x_1 - 3x_5 &= 0, \\ 2x_1x_2 + x_1 + 2R_{10}x_2^2 + x_2x_3^2 + R_7x_2x_3 + R_9x_2x_4 + R_8x_2 - Rx_5 &= 0, \\ 2x_2x_3^2 + R_7x_2x_3 + 2R_5x_3^2 + R_6x_3 - 8x_5 &= 0, \\ R_9x_2x_4 + 2x_4^2 - 4Rx_5 &= 0, \\ x_1x_2 + x_1 + R_{10}x_2^2 + x_2x_3^2 + R_7x_2x_3 + R_9x_2x_4, + R_8x_2 + R_5x_3^2 + R_6x_3 + x_4^2 - 1 &= 0 \end{aligned}$$

where

$$\begin{aligned} R_5 &= 0.193 & R_6 &= 0.4106217541E - 3 & R_7 &= 0.5451766686E - 3 \\ R_8 &= 0.44975E - 6 & R_9 &= 0.3407354178E - 4 & R_{10} &= 0.9615E - 6 \\ R &= 10. \end{aligned}$$

Table 4.5 contains the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method in 9 iterations, and the functions values in  $x^*$ . Figure 4.2 illustrates the evolution of  $\|F(x_k)\|$  corresponding to this application.

**Application S14 (Circuit design)** This application taken from the SMUNO collection is as follows:

$$\begin{aligned} & (1 - x_1 x_2) x_3 \left\{ \exp [x_5 (g_{1k} - g_{3k} x_7 10^{-3} - g_{5k} x_8 10^{-3})] - 1 \right\} \\ & + g_{4k} x_2 - g_{5k} = 0, \quad k = 1, \dots, 4, \\ & (1 - x_1 x_2) x_4 \left\{ \exp [x_6 (g_{1k} - g_{2k} - g_{3k} x_7 10^{-3} - g_{4k} x_9 10^{-3})] - 1 \right\} \\ & + g_{4k} - g_{5k} x_1 = 0, \quad k = 1, \dots, 4, \\ & x_1 x_3 - x_2 x_4 = 0, \end{aligned}$$

where

$$g = \begin{bmatrix} 0.4850 & 0.7520 & 0.8690 & 0.9820 \\ 0.3690 & 1.2540 & 0.7030 & 1.4550 \\ 5.2095 & 10.0677 & 22.9274 & 20.2153 \\ 23.3037 & 101.7790 & 111.4610 & 191.2670 \\ 28.5132 & 111.8467 & 134.3884 & 211.4823 \end{bmatrix}.$$

Table 4.6 contains the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method in 4 iterations, and the functions values in  $x^*$ . Figure 4.2 illustrates the evolution of  $\|F(x_k)\|$  corresponding to this application.

**Application N1 Solid fuel ignition. Application from the MINPACK-II collection** (Averick, Carter, & Moré, 1991), (Averick, Carter, Moré, & Xue, 1992) (*Bratu's problem*).

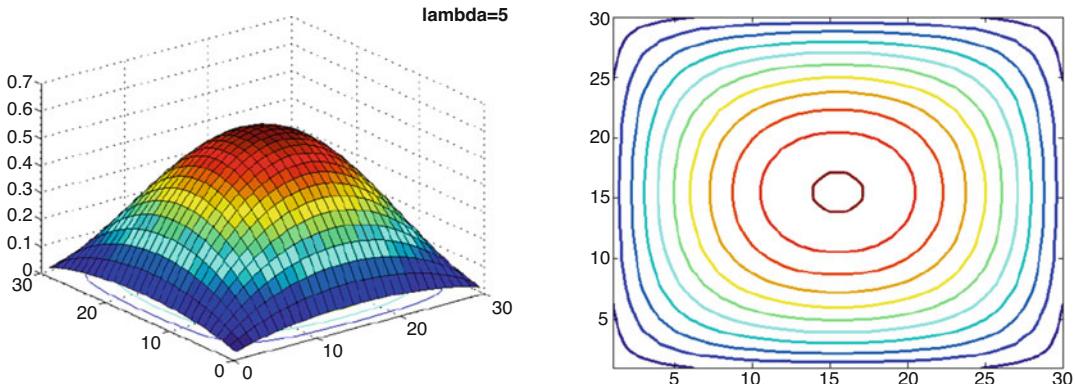
A steady-state model of solid fuel ignition can be described in terms of the solution  $u_\lambda$  of the boundary value problem

$$-\Delta u(x) = \lambda \exp[u(x)], \quad x \in D, \quad u(x) = 0 \text{ for } x \in \partial D$$

where  $\Delta$  is the Laplace operator,  $D$  is a domain in  $\mathbb{R}^2$  with boundary  $\partial D$ , and  $\lambda \in \mathbb{R}$  is a parameter. This problem is known as Bratu's problem. The model simulates a thermal reaction process in a rigid material (Aris, 1975). On the rectangular domain  $D = (l_1, u_1) \times (l_2, u_2)$  the problem can be

**Table 4.6** Circuit design. Initial point, solution, functions values in these points. Newton method. 4 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	0.7	0.95525047	0.89999995	0.124e-13
2	0.5	3.04155543	0.44998747	0.206e-12
3	0.9	-4.16621316	1.00000648	0.973e-12
4	1.9	-2.11846214	2.00006854	0.167e-11
5	8.1	5.636125856	7.99997144	-0.355e-14
6	8.1	22.31397917	7.99969268	0.0
7	5.9	24.95943321	5.00003127	0.255e-12
8	1	42.18784331	0.99998772	0.284e-13
9	1.9	-0.32000000	2.00005248	0.277e-14



**Fig. 4.3** Solution of Bratu's problem

represented as a finite dimensional version using a finite-difference formulation. The vertices  $z_{i,j} \in D$  are determined by choosing the grid spacings  $h_x$  and  $h_y$  and by defining

$$z_{i,j} = (l_1 + ih_x, l_2 + jh_y), \quad 0 \leq i \leq n_x + 1, \quad 0 \leq j \leq n_y + 1$$

such that  $z_{n_x+1,n_y+1} = (u_1, u_2)$ . The approximations  $u_{i,j}$  to  $u(z_{i,j})$  can be obtained by using central differences to approximate the Laplacian operator. This leads to a system of  $n = n_x n_y$  nonlinear equations:

$$\frac{h_y}{h_x} (2u_{i,j} - u_{i+1,j} - u_{i-1,j}) + \frac{h_x}{h_y} (2u_{i,j} - u_{i,j+1} - u_{i,j-1}) = \lambda h_x h_y \exp(u_{i,j})$$

where  $1 \leq i \leq n_x$  and  $1 \leq j \leq n_y$ . The solution of Bratu's problem requires the determination of a path  $u(\cdot)$  such that  $f(u(\lambda), \lambda) = 0$ , where  $f: \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n$  is the mapping that defines the finite dimensional version of Bratu's problem.

Considering  $n_x = 30$ ,  $n_y = 30$ , then a nonlinear algebraic system with  $n = 900$  variables is obtained. For  $\lambda = 5$ , the Newton method determines a solution of the above system with an accuracy smaller than  $10^{-6}$  in 2 iterations. Figure 4.3 presents the solution of this application.

**Application N2 Flow in a drive cavity. Application from the MINPACK-II collection** (Averick, Carter, & Moré, 1991), (Averick, Carter, Moré, & Xue, 1992).

The steady flow of a viscous incompressible fluid in a planar region  $D$  is described by the Navier-Stokes equations

$$-\nu \Delta u + (u \cdot \nabla) u + \nabla p = 0, \quad \nabla \cdot u = 0,$$

where  $u: D \rightarrow \mathbb{R}^2$  is the velocity field of the fluid,  $p: D \rightarrow \mathbb{R}$  is the pressure, and  $\nu$  is the viscosity parameter (the reciprocal of the Reynolds number  $R$ ). In a classical driven cavity problem, the region  $D$  is the unit square in  $\mathbb{R}^2$ , and the boundary conditions are

$$u(\xi_1, \xi_2) = \begin{cases} (0, 1), & \xi_2 = 1, \\ (0, 0), & 0 \leq \xi_2 < 1. \end{cases}$$

One of the main difficulties associated with this formulation is that there are no boundary conditions on the pressure  $p$ . In the stream function-vorticity formulation, the pressure is eliminated, and the problem is expressed in terms of the stream function  $\varphi$  and vorticity  $\omega$ . The components  $u_1$  and  $u_2$  of the velocity vector  $u$  are expressed in terms of the stream function  $\varphi$  by  $u_1 = \partial_y \varphi$ ,  $u_2 = -\partial_x \varphi$ . Thus, the incompressibility condition  $\nabla \cdot u = 0$  is automatically satisfied. The vorticity  $\omega \equiv \partial_x u_2 - \partial_y u_1$  is thus given by  $-\Delta \varphi = \omega$ . The stream function-vorticity formulation of the driven cavity problem is then

$$\begin{aligned} \nu \Delta^2 \varphi + [(\partial_y \varphi)(\partial_x \omega) - (\partial_x \varphi)(\partial_y \omega)] &= 0, \\ -\Delta \varphi &= \omega. \end{aligned}$$

The formulation of the driven cavity problem in terms of the stream function requires the solution of the following boundary value problem

$$\Delta^2 \varphi - \text{Re} [(\partial_y \varphi)(\partial_x \varphi) - (\partial_x \varphi)(\partial_y \varphi)] = 0.$$

with the boundary conditions

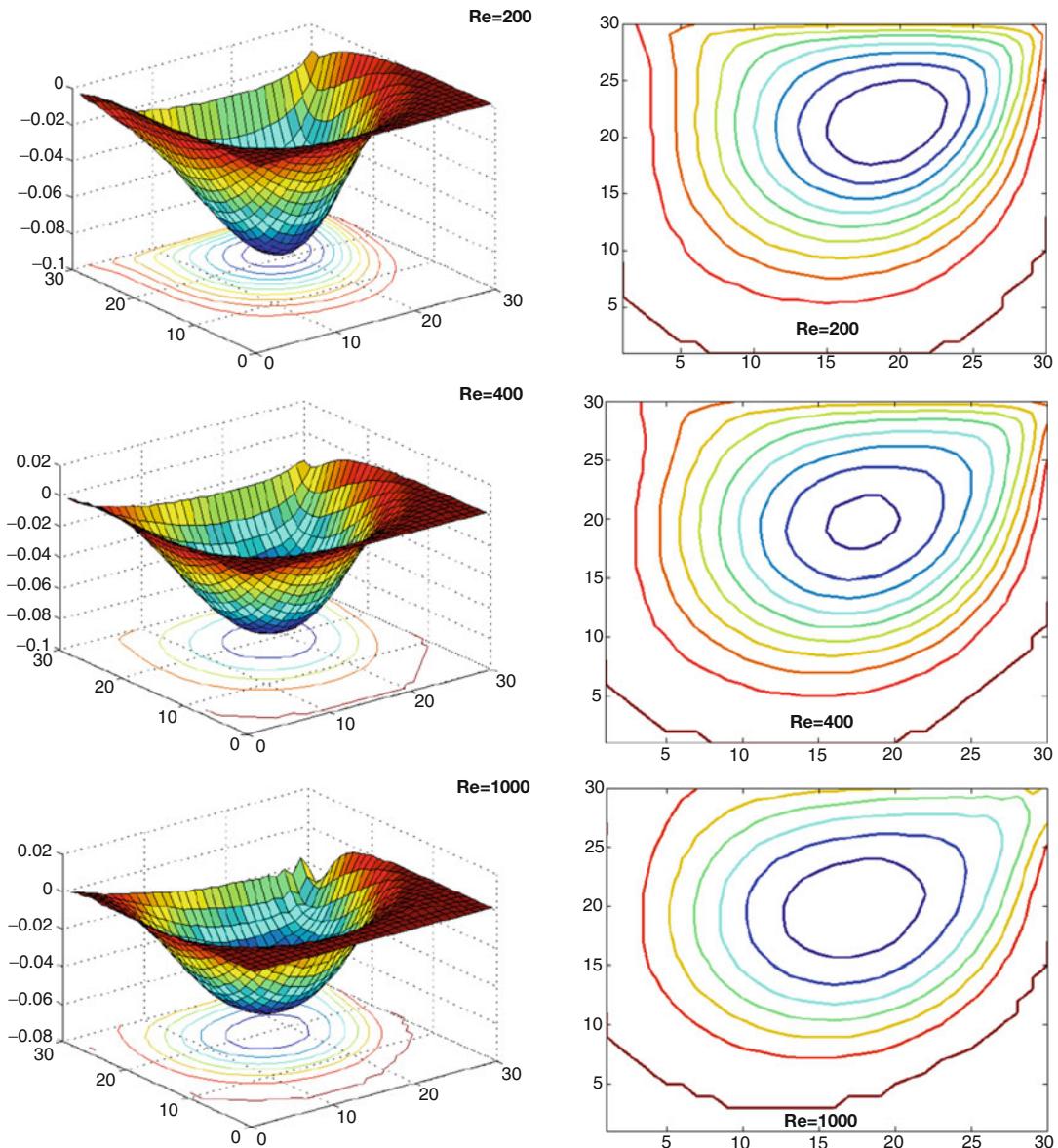
$$\varphi(\xi_1, \xi_2) = \partial_x \varphi(\xi_1, \xi_2) = 0, \quad \partial_y \varphi(\xi_1, \xi_2) = \begin{cases} 1, & \text{if } \xi_2 = 1, \\ 0, & \text{if } 0 \leq \xi_2 < 1. \end{cases}$$

( $\text{Re}$  is the Reynolds number.) The finite dimensional formulation of the driven cavity problem follows Schreiber and Keller (1983). The vertices  $z_{i,j} \in D$  are determined by choosing the grid spacings  $h_x$  and  $h_y$  and by defining

$$z_{i,j} = (ih_x, jh_y), \quad 0 < i \leq n_x + 1, \quad 0 < j \leq n_y + 1,$$

such that  $z_{n_x+1, n_y+1} = (1, 1)$ . The approximations  $u_{i,j}$  to  $\varphi(z_{i,j})$  are obtained by using central differences to approximate the Laplacian operator and the partial derivatives  $\partial_y \varphi$  and  $\partial_x \varphi$ . These approximations lead to a system of  $n = n_x n_y$  equations in the  $n$  unknown  $u_{i,j}$  of the form  $f(u) = Au + b - \text{Re}\Phi(u)$ , where  $A$  is the discrete biharmonic operator,  $b$  contains boundary information, and  $\Phi$  is the discrete representation of the nonlinear term.

An interesting feature of this formulation is that the discrete biharmonic approximation becomes poorly conditioned as the dimension  $n$  increases. Thus, the problem becomes difficult to solve even for moderate values of the Reynolds number  $\text{Re}$ . This difficulty can be overcome by preconditioning the problem. Thus,  $f$  is preconditioned with the fast biharmonic solver *bihar*. The preconditioned problem  $A^{-1}f(u) = 0$  with  $(n_x = 30, n_y = 30)$ , that is,  $n = 900$  variables, is solved for various values of the Reynolds number. Figure 4.4 illustrates the streamlines for three Reynolds numbers:  $\text{Re} = 200$ ,  $\text{Re} = 400$ , and  $\text{Re} = 1000$ , respectively. Considering  $n_x = 30$ ,  $n_y = 30$ , and  $\text{Re} = 200$ , the Newton method determines a solution of the above system in 4 iterations, with an accuracy smaller than  $10^{-6}$ . The graphical representation of the solution is given in Fig. 4.4a. For  $n_x = 30$ ,  $n_y = 30$ , and  $\text{Re} = 400$ , the Newton method gives a solution in 5 iterations, like in Fig. 4.4b. Figure 4.4c shows the solution of the problem for  $n_x = 30$ ,  $n_y = 30$ , and  $\text{Re} = 1000$  obtained by the Newton method in 7 iterations.



**Fig. 4.4** (a) Flow in a drive cavity ( $Re = 200$ ). (b) Flow in a drive cavity ( $Re = 400$ ). (c) Flow in a drive cavity ( $Re = 1000$ )

## 4.2 The Gauss-Newton Method

Many systems of nonlinear equations are expressed in the form of a vector of  $m$  independent functions, each of them depending on  $n$  variables:  $f_1(x), \dots, f_m(x)$ , where  $x \in \mathbb{R}^n$ . Let  $f = [f_1(x), \dots, f_m(x)]^T$ . The solution of the system  $f_i(x) = 0$ ,  $i = 1, \dots, m$ , can be obtained by forming a real-valued function

$$F(x) = \sum_{i=1}^m f_i(x)^2. \quad (4.16)$$

If  $F(x)$  is minimized by using an unconstrained optimization algorithm, then the individual functions  $f_i(x)$ ,  $i = 1, \dots, m$ , are minimized in the *least-squares sense*. A method for solving the above class of problems, known as the *Gauss-Newton method*, can be developed by applying the Newton method for minimizing function  $F$ .

Let  $J(x)$  be the Jacobian matrix of  $F$  in point  $x$  defined as the  $m \times n$  matrix

$$J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

In this case, the Jacobian matrix need not be a square matrix. By differentiating  $F$  with respect to  $x_j$ , we get

$$\frac{\partial F}{\partial x_j} = \sum_{i=1}^m 2f_i(x) \frac{\partial f_i}{\partial x_j}, \text{ for } j = 1, \dots, n. \quad (4.17)$$

Therefore, if  $\nabla F(x)$  is the gradient of  $F$  in point  $x$ , it follows that the gradient of  $F$  may be expressed as

$$\nabla F = 2J^T f. \quad (4.18)$$

Assuming that the functions  $f_i(x)$ ,  $i = 1, \dots, m$ , are twice continuously differentiable, then (4.17) yields

$$\frac{\partial^2 F}{\partial x_j \partial x_k} = 2 \sum_{i=1}^m \frac{\partial f_i}{\partial x_j} \frac{\partial f_i}{\partial x_k} + 2 \sum_{i=1}^m f_i(x) \frac{\partial^2 f_i}{\partial x_j \partial x_k},$$

for  $j, k = 1, \dots, n$ .

If the second derivatives of  $f_i(x)$  are neglected, we have

$$\frac{\partial^2 F}{\partial x_j \partial x_k} \cong 2 \sum_{i=1}^m \frac{\partial f_i}{\partial x_j} \frac{\partial f_i}{\partial x_k}.$$

Thus, the Hessian of  $F$  can be approximated as

$$\nabla^2 F(x) \cong 2J^T(x)J(x). \quad (4.19)$$

Since the gradient and the Hessian of  $F$  in the current point  $x_k$  are known and assuming that the Hessian (4.19) is nonsingular, then the Newton method can be applied for solving this problem as

$$\begin{aligned}x_{k+1} &= x_k - \left(2J(x_k)^T J(x_k)\right)^{-1} \left(2J(x_k)^T f(x_k)\right) \\&= x_k - \left(J(x_k)^T J(x_k)\right)^{-1} \left(J(x_k)^T f(x_k)\right), k = 0, 1, \dots\end{aligned}$$

When the current point  $x_k$  is in the neighborhood of the solution  $x^*$ , then the functions  $f_i(x_k)$ ,  $i = 1, \dots, m$ , can be accurately represented by the linear approximation of the Taylor series and therefore, the matrix (4.19) becomes an accurate representation of the Hessian of  $F(x_k)$ . In this case, the method converges very rapidly. If the functions  $f_i(x)$ ,  $i = 1, \dots, m$ , are linear, then  $F(x)$  is quadratic, the matrix (4.19) is the exact Hessian, and the problem is solved in one iteration. Of course, the Gauss-Newton method breaks down if the Hessian (4.19) becomes singular. In this case, the remedy is the modification of the Gauss-Newton method by using the methods described in Sect. 4.5.

### 4.3 The Newton Method for Function Minimization

Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function and consider the minimizing problem

$$\min_{x \in \mathbb{R}^n} f(x). \quad (4.20)$$

Obviously, in the current point  $x_k$  we can compute  $f(x_k)$ , the gradient  $\nabla f(x_k)$ , and the Hessian  $\nabla^2 f(x_k)$ , where by definition

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \dots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}.$$

If  $f$  is a smooth function, then  $\nabla^2 f(x)$  is a symmetric matrix. Suppose that with a minimal effort we can get the analytical expression of the Hessian. With these elements the *quadratic model of function in point  $x_k$*  can be defined as

$$m_k^2(x) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T \nabla^2 f(x_k) (x - x_k). \quad (4.21)$$

For solving the problem (4.20), the main idea of the Newton method is quite simple: in an iterative manner, minimize the quadratic model (4.21) and take its solution  $x_{k+1}$  as a new approximation to the minimum point. In this new point  $x_{k+1}$  compute again a new quadratic model and take it as a new approximation to  $x^*$ , etc.

Minimizing (4.21) leads to the iterative formula

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k), \quad (4.22)$$

known as the *Newton method*.

**Algorithm 4.2** *Newton for minf(x)*

1. *Initialization.* Consider an initial point  $x_0 \in \mathbb{R}^n$  as well as the convergence tolerance  $\varepsilon > 0$  sufficiently small for stopping the iterations. Set  $k = 0$
2. Compute  $\nabla f(x_k)$  and  $\nabla^2 f(x_k)$
3. If  $\|\nabla f(x_k)\| \leq \varepsilon$ , then stop; otherwise, continue with step 4
4. Solve the linear algebraic system  $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$
5. Set  $x_{k+1} = x_k + d_k$ ,  $k = k + 1$  and continue with step 2

◆

Note that in this algorithm, the most difficult operations are in steps 2 and 4, in which the Hessian matrix has to be evaluated in the current point and a linear algebraic system needs to be solved. However, the Newton method has some very important properties which define its efficiency. The most important is that its convergence is quadratic when initialized near the solution. All the modifications of this method refer to either the evaluation of the Hessian or to the solving of the linear algebraic system  $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$ , known as the *Newton system*.

**Example 4.3** Let us compute the minimum point of the function

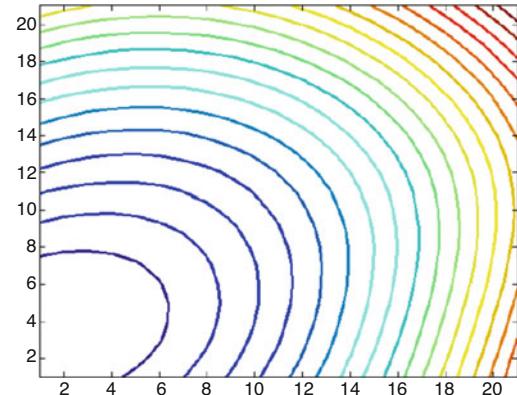
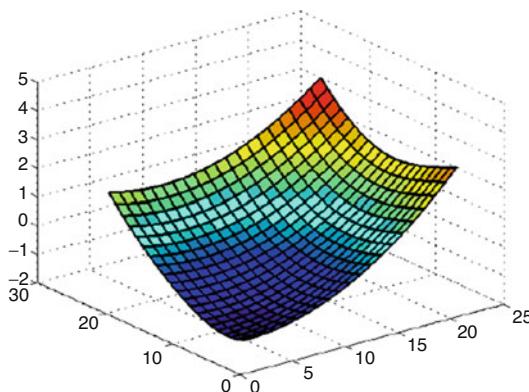
$$f(x) = x_1 \exp(-x_2^2) + x_1^2 + x_1 + x_2^2 - x_2$$

Figure 4.5 shows the representation of the function in the domain  $[-1, 1] \times [0, 2]$ .

The solution of this problem is  $x^* = [-0.966616, 0.262870]$  and  $f(x^*) = -1.1281165$ . Considering a lattice of the initial points in the domain  $[-1, 1] \times [0, 2]$ , then the number of the iterations required by the Newton method for minimizing function  $f$  is shown in Fig. 4.6. We can see that near the solution, the method needs a relatively small number of iterations. Moreover, there are some plateaus with initial points for which the number of iterations is the same.

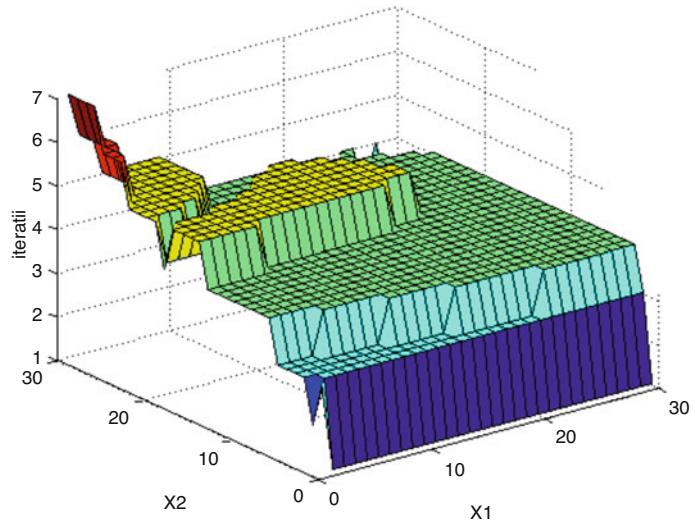
The following theorem shows that the Newton method (4.22) is well defined and its rate of convergence is quadratic. The theorem is based on the Lipschitz constant of the Hessian.

**Theorem 4.3** *Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be twice continuously differentiable and the current point  $x_k$  sufficiently close to the solution  $x^*$  of the problem (4.20) for which  $\nabla f(x^*) = 0$ . If the Hessian matrix  $\nabla^2 f(x^*)$  is positive definite and  $\nabla^2 f(x)$  is Lipschitz continuous, that is,*



**Fig. 4.5** Representation of function  $f$

**Fig. 4.6** Number of iterations of the Newton method for different initial points from the domain  $[-1, 1] \times [0, 2]$



$$\left| \nabla^2 f(x)_{ij} - \nabla^2 f(y)_{ij} \right| \leq L \|x - y\|, \text{ for all } 1 \leq i, j \leq n$$

where  $\nabla^2 f(x)_{ij}$  is the  $(i, j)$ -element of the Hessian and  $L$  is a positive constant, then for any  $k$  the Newton iteration (4.22) is well defined and the sequence  $\{x_k\}$  generated by the algorithm is quadratic convergent to  $x^*$ .

**Proof** Let  $h_k = x_k - x^*$ . From the Taylor development, it follows that

$$0 = \nabla f(x^*) = \nabla f(x_k) - \nabla^2 f(x_k) h_k + O(\|h_k\|^2).$$

Since  $f$  is twice continuously differentiable,  $x_k$  is sufficiently close to  $x^*$  and  $\nabla^2 f(x^*)$  is positive definite, it follows that it is quite reasonable to suppose that the points  $x_k$  are in a neighborhood of  $x^*$ ,  $\nabla^2 f(x_k)$  are positive definite, and  $\nabla^2 f(x_k)^{-1}$  are upper bounded. Therefore, at the iteration  $k$  the Newton method is defined. Multiplying the above relation by  $\nabla^2 f(x_k)^{-1}$  we get

$$\begin{aligned} 0 &= \nabla^2 f(x_k)^{-1} \nabla f(x_k) - h_k + O(\|h_k\|^2) \\ &= -d_k - h_k + O(\|h_k\|^2) = -h_{k+1} + O(\|h_k\|^2) \end{aligned}$$

By definition of  $O(\cdot)$ , there exists a constant  $C$  such that

$$\|h_{k+1}\| \leq C \|h_k\|^2. \quad (4.23)$$

If  $x_k \in \Omega = \{x : \|h\| \leq \gamma/C, h = x - x^*, \gamma \in (0, 1)\}$ , then

$$\|h_{k+1}\| \leq \gamma \|h_k\| \leq \gamma^2/C < \gamma/C. \quad (4.24)$$

Therefore,  $x_{k+1} \in \Omega$ . By induction, it follows that the Newton iteration is well defined for all  $k$  and  $\|h_k\| \rightarrow 0$  when  $k \rightarrow \infty$ , thus proving the convergence of the Newton method. From (4.23) we can see that the rate of convergence of the Newton method is quadratic.  $\diamond$

The following theorem shows the quadratic convergence of the Newton method, but this time the theorem is based on the assumption that the Hessian matrix is bounded.

**Theorem 4.4** *Let  $f$  be a twice continuously differentiable function defined over  $\mathbb{R}^n$ . Assume that:*

- *there exists  $\mu > 0$  such that*

$$\nabla^2 f(x) \geq \mu I \text{ for any } x \in \mathbb{R}^n, \quad (4.25)$$

- *there exists  $M > 0$  such that*

$$\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq M \|x - y\|_2 \text{ for any } x, y \in \mathbb{R}^n. \quad (4.26)$$

Let  $\{x_k\}$  be the sequence generated by the Newton method (4.22), and let  $x^*$  be the unique minimizer of  $f$  over  $\mathbb{R}^n$ . Then

$$\|x_{k+1} - x^*\|_2 \leq \frac{M}{2\mu} \|x_k - x^*\|_2^2. \quad (4.27)$$

If  $\|x_0 - x^*\|_2 \leq \frac{\mu}{M}$ , then

$$\|x_k - x^*\|_2 \leq \frac{2\mu}{M} \left(\frac{1}{2}\right)^{2^k}. \quad (4.28)$$

**Proof** Firstly, let us prove (4.27). By the fundamental theorem of calculus, it follows that

$$\begin{aligned} x_{k+1} - x^* &= x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k) - x^* \\ &= x_k - x^* + [\nabla^2 f(x_k)]^{-1} (\nabla f(x^*) - \nabla f(x_k)) \\ &= x_k - x^* + [\nabla^2 f(x_k)]^{-1} \int_0^1 [\nabla^2 f(x_k + t(x^* - x_k))] (x^* - x_k) dt \\ &= [\nabla^2 f(x_k)]^{-1} \int_0^1 [\nabla^2 f(x_k + t(x^* - x_k)) - \nabla^2 f(x_k)] (x^* - x_k) dt. \end{aligned}$$

Then,

$$\begin{aligned} \|x_{k+1} - x^*\|_2 &\leq \left\| [\nabla^2 f(x_k)]^{-1} \right\|_2 \left\| \int_0^1 [\nabla^2 f(x_k + t(x^* - x_k)) - \nabla^2 f(x_k)] (x^* - x_k) dt \right\|_2 \\ &\leq \left\| [\nabla^2 f(x_k)]^{-1} \right\|_2 \int_0^1 \|[\nabla^2 f(x_k + t(x^* - x_k)) - \nabla^2 f(x_k)] (x^* - x_k)\|_2 dt \\ &\leq \left\| [\nabla^2 f(x_k)]^{-1} \right\|_2 \int_0^1 \|\nabla^2 f(x_k + t(x^* - x_k)) - \nabla^2 f(x_k)\|_2 \|x^* - x_k\|_2 dt \\ &\leq \frac{1}{\mu} \int_0^1 Mt \|x_k - x^*\|_2^2 dt \\ &= \frac{M}{2\mu} \|x_k - x^*\|_2^2. \end{aligned}$$

For proving (4.28) observe that for  $k = 0$ ,

$$\|x_0 - x^*\|_2 \leq \frac{\mu}{M} = \frac{2\mu}{M} \left(\frac{1}{2}\right)^{2^0}.$$

Suppose that for an integer  $k$ , (4.28) is satisfied, i.e.,  $\|x_k - x^*\|_2 \leq \frac{2\mu}{M} \left(\frac{1}{2}\right)^{2^k}$ . Now, let us show that this holds for  $k + 1$ . We have

$$\|x_{k+1} - x^*\|_2 \leq \frac{M}{2\mu} \|x_k - x^*\|_2^2 \leq \frac{M}{2\mu} \left(\frac{2\mu}{M} \left(\frac{1}{2}\right)^{2^k}\right)^2 = \frac{2\mu}{M} \left(\frac{1}{2}\right)^{2^{k+1}}. \quad \blacklozenge$$

The interpretation of the condition (4.25) is as follows. As we know, the Taylor series of  $f$  near  $x_k$  is

$$f(x) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T \nabla^2 f(x_k) (x - x_k) + \dots \quad (4.29)$$

The condition  $\nabla^2 f(x) \geq \mu I$  determines that the quadratic approximation of  $f$  in  $x_k$  is consistent in the sense that the quadratic term in (4.29) is large enough such that the iterative formula for the Newton method  $x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} g_k$ ,  $k = 0, 1, \dots$ , is well defined.

On the other hand, the condition  $\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq M \|x - y\|_2$  can be rewritten as

$$\frac{\|\nabla^2 f(x) - \nabla^2 f(y)\|_2}{\|x - y\|_2} \leq M,$$

where  $0 < M < \infty$ . Therefore, this condition tells us that the tensor of the derivative of the third order of  $f$  in  $x_k$  is small enough such that the rest of the Taylor series can be very well neglected. In conclusion, for the iterative process  $x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} g_k$ ,  $k = 0, 1, \dots$ , to work, both these conditions (4.25) and (4.26) require that the quadratic approximation of  $f$  in  $x_k$  should be a good one.

## 4.4 The Newton Method with Line-Search

The Newton method is a local method. When the initial point  $x_0$  is far away from the solution, we are not sure that  $\nabla^2 f(x_k)$  is positive definite, and there is no guarantee that the Newton direction  $d_k$  is descendent. Since, as we know, the line-search is a strategy of globalization, then we can define the *Newton method with line-search*, (also called the *damped Newton method*) which guarantees the global convergence.

The Newton method with line-search is defined as

$$d_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k), \quad (4.30)$$

$$x_{k+1} = x_k + \alpha_k d_k, \quad (4.31)$$

where  $\alpha_k$  is the stepsize, which can be determined either by the exact line-search, or by the inexact line-search. The Newton algorithm with line-search is as follows:

**Algorithm 4.3** *Newton with line-search for  $\min f(x)$* 

1. *Initialization.* Consider an initial point  $x_0 \in \mathbb{R}^n$  as well as the convergence tolerance  $\varepsilon > 0$  sufficiently small for stopping the iterations. Set  $k = 0$
2. Compute  $\nabla f(x_k)$  and  $\nabla^2 f(x_k)$
3. If  $\|\nabla f(x_k)\| \leq \varepsilon$ , then stop; otherwise, continue with step 4
4. Solve the linear algebraic system  $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$
5. Compute  $\alpha_k$  by the exact line-search  

$$f(x_k + \alpha_k d_k) = \min_{\alpha \geq 0} f(x_k + \alpha d_k)$$
or by the inexact line-search  

$$\alpha_k = \operatorname{argmin}_{\alpha \geq 0} f(x_k + \alpha d_k)$$
6. Set  $x_{k+1} = x_k + \alpha_k d_k$ ,  $k = k + 1$  and continue with step 2

◆

The following theorem shows the global convergence of Algorithm 4.3.

**Theorem 4.5** *Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the open and convex set  $D \subset \mathbb{R}^n$ . Suppose that for any  $x_0 \in D$  there exists a constant  $m > 0$  such that  $f(x)$  satisfies*

$$u^T \nabla^2 f(x) u \geq m \|u\|^2, \text{ for any } u \in \mathbb{R}^n \text{ and } x \in S, \quad (4.32)$$

where  $S = \{x : f(x) \leq f(x_0)\}$  is the level set corresponding to the initial point. Then the sequence  $\{x_k\}$  generated by Algorithm 4.3 with exact line-search satisfies:

- (i) If the sequence  $\{x_k\}$  is finite, then  $\nabla f(x_k) = 0$  for a certain  $k$ .
- (ii) If the sequence  $\{x_k\}$  is infinite, then  $\{x_k\}$  is convergent to the unique minimum  $x^*$  of  $f$ .

**Proof** From (4.32) it follows that  $f(x)$  is a strict convex function on  $\mathbb{R}^n$ . Therefore, its stationary point is the unique global minimum. Also, the level set  $S$  is convex, bounded, and closed. Since the sequence  $\{f(x_k)\}$  is monotonously decreasing, then  $\{x_k\} \subset S$ , that is,  $\{x_k\}$  is bounded. Therefore, there is a limit point  $\bar{x} \in S$  such that  $x_k \rightarrow \bar{x}$  and  $f(x_k) \rightarrow f(\bar{x})$ . Now, since  $f$  is twice continuously differentiable, from Theorem 2.12 it follows that  $\nabla f(x_k) \rightarrow \nabla f(\bar{x}) = 0$ . Finally, observe that, since the stationary point is unique, then the sequence  $\{x_k\}$  is convergent to  $\bar{x}$ , the unique minim. ◆

Now, let us see the convergence of the Newton algorithm with inexact line-search. If the weak Wolfe line-search (2.54) and (2.59) are used, then from (2.93) in Proposition 2.6, it follows that

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \bar{\eta} \|\nabla f(x_k)\|^2 \cos^2 \langle d_k, -\nabla f(x_k) \rangle, \quad (4.33)$$

where  $\bar{\eta}$  is a constant independent of  $k$ . With these, the global convergence of the Newton method with inexact line-search can be proved.

**Theorem 4.6** *Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the open and convex set  $D \subset \mathbb{R}^n$ . Suppose that for any  $x_0 \in \mathbb{R}^n$  there exists the constant  $m > 0$  such that  $f(x)$  satisfies (4.32) on the level set  $S$ . If the line-search satisfies (4.33), then the sequence  $\{x_k\}$  generated by Algorithm 4.3 with inexact line-search satisfies*

$$\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0 \quad (4.34)$$

and  $\{x_k\}$  is convergent to the unique minimum  $x^*$  of function  $f$ .

**Proof** Since  $f(x)$  satisfies (4.32), then  $f(x)$  is uniformly convex on  $S$ . Also, from (4.33) it follows that  $f(x)$  is strictly monotonously decreasing and the sequence  $\{x_k\}$  is bounded. Therefore, there exists a constant  $M > 0$  such that for any  $k$ ,

$$\|\nabla^2 f(x_k)\| \leq M. \quad (4.35)$$

Then, from (4.33) we get

$$\begin{aligned} \cos \langle d_k, -\nabla f(x_k) \rangle &= \frac{-d_k^T \nabla f(x_k)}{\|d_k\| \|\nabla f(x_k)\|} = \frac{\nabla f(x_k)^T \nabla^2 f(x_k)^{-1} \nabla f(x_k)}{\|\nabla^2 f(x_k)^{-1} \nabla f(x_k)\| \|\nabla f(x_k)\|} \\ &= \frac{d_k^T \nabla^2 f(x_k) d_k}{\|d_k\| \|\nabla^2 f(x_k) d_k\|} \geq \frac{m}{M} \end{aligned}$$

Again, from (4.33) it follows that

$$\infty > \sum_{k=0}^{\infty} [f(x_k) - f(x_{k+1})] \geq \sum_{k=0}^{\infty} \bar{\eta} \frac{m^2}{M^2} \|\nabla f(x_k)\|^2, \quad (4.36)$$

thus proving (4.34). Note that  $f(x)$  is uniformly convex, which means that it has only a stationary point. Therefore, (4.34) shows that the sequence  $\{x_k\}$  is convergent to the unique minimum  $x^*$  of function  $f$ .  $\blacklozenge$

The following result emphasizes the local rate of the convergence of the Newton method. Recall that for all  $x$  from a vicinity of a minimum point  $x^*$  such that  $\nabla^2 f(x^*)$  is positive definite the Hessian  $\nabla^2 f(x)$  will also be positive definite. The Newton method will be well defined in this vicinity and will converge quadratically, provided that the stepsizes  $\alpha_k$  are always 1 eventually.

**Theorem 4.7** Suppose that  $f$  is twice continuously differentiable and that its Hessian  $\nabla^2 f(x)$  is Lipschitz continuous in a neighborhood of a solution  $x^*$  at which the sufficient optimality conditions (Theorem 11.6) are satisfied. Consider the iteration  $x_{k+1} = x_k + d_k^N$ , where  $d_k^N = -\nabla^2 f(x_k)^{-1} g_k$ . Then:

- (i) If the starting point  $x_0$  is sufficiently close to  $x^*$ , then the sequence  $\{x_k\}$  generated by the Newton method converges to  $x^*$ .
- (ii) The rate of convergence of  $\{x_k\}$  is quadratic.
- (iii) The sequence of the gradients norms  $\{\|g_k\|\}$  converges quadratically to zero.

**Proof** From the optimality condition  $\nabla f(x^*) = 0$  and by the definition of the Newton method, it follows that

$$\begin{aligned} x_k + d_k^N - x^* &= x_k - x^* - \nabla^2 f(x_k)^{-1} g_k \\ &= \nabla^2 f(x_k)^{-1} [\nabla^2 f(x_k)(x_k - x^*) - (\nabla f(x_k) - \nabla f(x^*))]. \end{aligned}$$

But

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

Therefore,

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

where  $L$  is the Lipschitz constant of  $\nabla^2 f(x)$  for  $x$  near  $x^*$ . Since  $\nabla^2 f(x^*)$  is nonsingular, there is a radius  $r > 0$  such that  $\| \nabla^2 f(x_k)^{-1} \| \leq 2 \| \nabla^2 f(x^*)^{-1} \|$  (see Theorem A2.5) for all  $x_k$  with  $\| x_k - x^* \| \leq r$ . Therefore,

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

where  $\tilde{L} = L \| \nabla^2 f(x^*)^{-1} \|$  and  $d_k^N = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ . Choosing  $x_0$  so that  $\| x_0 - x^* \| \leq \min \{ r, 1/(2\tilde{L}) \}$ , then the above inequality may be used inductively to deduce that the sequence  $\{x_k\}$  converges to  $x^*$  and the rate of convergence is quadratic.

To prove that the gradient norms converge to zero quadratically, we use the relations  $x_{k+1} - x_k = d_k^N$  and  $\nabla f(x_k) + \nabla^2 f(x_k) d_k^N = 0$ , to obtain that

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

showing (iii). ◆

The following theorem shows that in some mild conditions, when the iterates generated by the Newton method are approaching the solution, then the Wolfe (or the Goldstein) line-search conditions will accept the stepsize  $\alpha_k = 1$  for all large  $k$ .

**Theorem 4.8** Suppose that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable. Consider the iteration  $x_{k+1} = x_k + \alpha_k d_k$ , where  $d_k$  is a descent direction and  $\alpha_k$  satisfies the Wolfe line-search (2.54) and (2.59) with  $\rho \leq 1/2$ . If the sequence  $\{x_k\}$  converges to a point  $x^*$  such that  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite and if the search direction satisfies

$$\lim_{k \rightarrow \infty} \frac{\|\nabla f(x_k) + \nabla^2 f(x_k) d_k\|}{\|d_k\|} = 0,$$

then:

- (i) The stepsize  $\alpha_k = 1$  is admissible for all  $k$  greater than a certain index  $k_0$ .
- (ii) If  $\alpha_k = 1$  for all  $k > k_0$ , then  $\{x_k\}$  converges to  $x^*$  superlinearly.  $\blacklozenge$

Observe that if  $\rho > 1/2$ , then the line-search would exclude the minimizer of a quadratic and the unit stepsize may not be admissible.

## 4.5 Analysis of Complexity

This section is devoted to analyzing the complexity of the Newton method, namely, an estimation of the number of iterations for solving an unconstrained optimization problem. From the very beginning let us introduce some concepts. At the same time, in order to simplify the notation, instead of  $x_{k+1} = x_k + \alpha_k d_k$ , the following notation will be used  $x^+ = x + ad$ . The Newton method is defined by two concepts: the *Newton step* and the *Newton decrement*. The Newton decrement can be used as a criterion for stopping the iterations (Andrei, 2004a).

**The Newton Step** For  $x \in \text{dom } f$ , the vector

$$d_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x) \quad (4.37)$$

is called *the Newton step* for function  $f$  in point  $x$ . If  $\nabla^2 f(x)$  is positive definite, it follows that, if  $\nabla f(x) \neq 0$ , then

$$\nabla f(x)^T d_{nt} = -\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) < 0.$$

Therefore, if  $x$  is not the minimum point, then the Newton step is always a descent direction. Therefore, in order to minimize the second order approximation (4.21) of  $f$  in  $x$ , the idea is to add the Newton step at the current point. Observe that if  $f$  is quadratic, then  $x + d_{nt}$  is the exact minimum of  $f$ . If  $f$  is close to a quadratic, then our intuition tells us that  $x + d_{nt}$  is a very good estimation of the minimum  $x^*$  of  $f$ . Moreover, since  $f$  is strictly convex and twice continuously differentiable, the quadratic model of  $f$  will be very exact when  $x$  is near  $x^*$ . Therefore, as soon as  $x$  is near enough to  $x^*$ , it results that the point  $x + d_{nt}$  is a very good approximation of  $x^*$ , i.e., it is very close to  $x^*$ . If the optimality condition  $\nabla f(x^*) = 0$  is linearized around  $x$ , then

$$\nabla f(x + d) \cong \nabla f(x) + \nabla^2 f(x)d = 0$$

and, as we can see, this is a linear algebraic system in  $d$  with solution  $d = d_{nt}$ . Therefore, the Newton step  $d_{nt}$  is exactly what is necessary to add to  $x$  in order satisfy the optimality condition.

A property of the Newton step concerning its invariance at the linear coordinate or affine transformations needs to be mentioned. Suppose that  $T \in \mathbb{R}^{n \times n}$  is a nonsingular matrix and define  $\bar{f}(y) = f(Ty)$ . Then

$$\nabla \bar{f}(y) = T^T \nabla f(x), \quad \nabla^2 \bar{f}(y) = T^T \nabla^2 f(x) T,$$

where  $x = Ty$ . With these, the Newton step for  $\bar{f}$  in the variables  $y$  is

$$\begin{aligned} \bar{d}_{nt} &= -\left(T^T \nabla^2 f(x) T\right)^{-1} (T^T \nabla f(x)) \\ &= -T^{-1} \nabla^2 f(x)^{-1} \nabla f(x) = T^{-1} d_{nt}, \end{aligned}$$

where  $d_{nt}$  is the Newton step of  $f$  in the variables  $x$ . Therefore, the Newton steps for  $f$  and  $\bar{f}$  are connected by the same linear transformation and  $x + d_{nt} = T(y + \bar{d}_{nt})$ .

**The Newton Decrement** The Newton decrement in point  $x$  is defined by the quantity

$$r(x) = \left( \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) \right)^{1/2}. \quad (4.38)$$

The Newton decrement has a very important role in the analysis of the Newton method and can be used as a criterion for stopping the iterations in the algorithm of this method. Indeed, the quantity  $f(x) - \inf_y \hat{f}(y)$  where  $\hat{f}$  is the quadratic approximation of  $f$  in  $x$  can be expressed as

$$f(x) - \inf_y \hat{f}(y) = f(x) - \hat{f}(x + d_{nt}) = \frac{1}{2} r(x)^2.$$

Therefore,  $r^2/2$  is an estimation of the suboptimality  $f(x) - f^*$  based on the quadratic approximation of  $f$  in  $x$ . Observe that the Newton decrement can be expressed as

$$r(x) = (d_{nt}^T \nabla^2 f(x) d_{nt})^{1/2}, \quad (4.39)$$

where  $d_{nt}$  is the Newton step. Therefore, it follows that  $r$  is exactly the norm of the Newton step subject to the Hessian of function  $f$

$$r(x) = \|d_{nt}\|_{\nabla^2 f(x)} = (d_{nt}^T \nabla^2 f(x) d_{nt})^{1/2}. \quad (4.40)$$

Finally, we mention that the Newton decrement is affine invariant. In other words, the Newton decrement of  $\bar{f}(y) = f(Ty)$  in  $y$ , where  $T$  is nonsingular, is the same as the Newton decrement of  $f$  in  $x = Ty$ .

In order to have a general analysis of the convergence of the Newton method, we consider that a line-search is performed along the Newton direction. This leads to the *Newton method with line-search*, which is different from the *pure Newton method*, in which at every iteration the stepsize is one. In the Newton algorithm with line-search, at every iteration the Newton step and the Newton decrement are computed. These involve the inverse of the Hessian matrix of the minimizing function  $f$ . Numerically, this is the most delicate operation. However, from the viewpoint of the convergence, this operation does not have any importance since it is the bounds of the Hessian which are the most important for the strongly convex functions considered in our analysis.

Suppose that  $f$  is a twice continuous differentiable function, strongly convex with the constant  $m$ , that is, for any  $x \in S$ ,  $\nabla^2 f(x) \geq mI$ . As we know, this involves that there is another constant  $M > 0$  such

that, for any  $x \in S$ ,  $\nabla^2 f(x) \leq MI$ . Additionally, suppose that the Hessian matrix of  $f$  is Lipschitz continuous on  $S$  with the constant  $L$ , that is, for all  $x, y \in S$ ,  $\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq L\|x - y\|_2$ . The Lipschitz constant  $L$  can be interpreted as a bound of the third order derivative of function  $f$ , which is zero for quadratic functions. Moreover,  $L$  gives a measure of how well function  $f$  can be approximated by a quadratic model around the current point. The smaller the Lipschitz constant, the better the approximation of  $f$  in the current point by the quadratic model, i.e., the Newton method will work better.

**Theorem 4.9** *Let  $f$  be a twice continuously differentiable function, strongly convex with constants  $m$  and  $M$ , for which the Hessian matrix is Lipschitz continuous with the constant  $L$ . Then, there exist the real numbers  $\eta$  and  $\gamma$  with the properties  $0 < \eta \leq m^2/L$  and  $\gamma > 0$ , such that the following two conditions hold:*

- If  $\|\nabla f(x)\|_2 \geq \eta$ , then

$$f(x_{k+1}) - f(x_k) \leq -\gamma. \quad (4.41)$$

- If  $\|\nabla f(x)\|_2 < \eta$ , then the backtracking line-search selects  $\alpha_k = 1$  and

$$\frac{L}{2m^2} \|\nabla f(x_{k+1})\|_2 \leq \left( \frac{L}{2m^2} \|\nabla f(x_k)\|_2 \right)^2 \quad (4.42)$$

Before proving the theorem, some comments are needed (Andrei, 2004a). Let us analyze the above implications. Indeed, the first condition shows that even if the norm of the gradient is greater than a positive constant, then the algorithm achieves a reduction of the values of the minimizing function. The second condition is stronger and has the following interpretation. Suppose that at iteration  $k$  we have  $\|\nabla f(x_k)\|_2 < \eta$ . Since  $\eta \leq m^2/L$ , it follows that  $\|\nabla f(x_{k+1})\|_2 < \eta$ , that is, the second condition is also satisfied at iteration  $k + 1$ . Recursively, continuing this interpretation, we conclude that as soon as the second condition holds, it will be satisfied for all the next iterations, i.e., for all  $j \geq k$  we have  $\|\nabla f(x_j)\|_2 < \eta$ . Therefore, for all  $j \geq k$ , the algorithm will take a Newton step of unitary steplength  $\alpha_j = 1$  and

$$\frac{L}{2m^2} \|\nabla f(x_{j+1})\|_2 \leq \left( \frac{L}{2m^2} \|\nabla f(x_j)\|_2 \right)^2 \quad (4.43)$$

Applying this inequality recursively, for all  $j \geq k$  we get

$$\frac{L}{2m^2} \|\nabla f(x_j)\|_2 \leq \left( \frac{L}{2m^2} \|\nabla f(x_k)\|_2 \right)^{2^{j-k}} \leq \left( \frac{1}{2} \right)^{2^{j-k}}.$$

Therefore,

$$f(x_j) - f^* \leq \frac{1}{2m} \|\nabla f(x_j)\|_2^2 \leq \frac{2m^3}{L^2} \left( \frac{1}{2} \right)^{2^{j-k+1}} \quad (4.44)$$

This inequality shows that as soon as the second condition has been satisfied, the convergence of the algorithm is *extremely rapid*. Keeping in mind that in this situation the stepsizes are all equal to

one, then from (4.44) we can see that the error is quadratically convergent to zero, that is, at every iteration the number of the correct digits in the value of the minimizing function is doubled.

Every iteration of the Newton algorithm may be divided into two stages. The first one is called the *damped Newton stage*, since the algorithm may select the stepsize  $\alpha < 1$ . The second one appears as soon as  $\|\nabla f(x)\|_2 < \eta$  has been satisfied and is called the *pure Newton stage* or the *Newton quadratic convergent stage*. In this stage, the stepsize is one.

With these, let us determine the complexity of the Newton algorithm with line-search. At first determine an upper bound of the number of iterations in the damped Newton stage. Indeed, since at every iteration,  $f$  decreases at least with  $\gamma$ , it follows that the number of iterations in the damped Newton stage should not be larger than

$$\frac{f(x_0) - f^*}{\gamma}.$$

For finding a bound of the number of iterations corresponding to the pure Newton stage, consider (4.44). Indeed, we can see that  $f(x) - f^* \leq \varepsilon$  after not more than

$$\log_2 \log_2 \left( \frac{\varepsilon_0}{\varepsilon} \right)$$

iterations, where  $\varepsilon_0 = 2m^3/L^2$ .

Therefore, the total number of iterations after which  $f(x) - f^* \leq \varepsilon$  is upper bounded by

$$\frac{f(x_0) - f^*}{\gamma} + \log_2 \log_2 \left( \frac{\varepsilon_0}{\varepsilon} \right) \quad (4.45)$$

Note that the term  $\log_2 \log_2(\varepsilon_0/\varepsilon)$ , which upper bounds the number of iterations in the Newton quadratic convergent stage, *increases extremely slowly* with the accuracy  $\varepsilon$ . Practically, if we consider, for example,  $\varepsilon = 5 \cdot 10^{-20} \varepsilon_0$ , then the number of iterations corresponding to the pure Newton stage is not larger than 6.

Hence, we can say that the total number of iterations associated to the Newton method for minimizing function  $f$  with a very good approximation is upper bounded by

$$\frac{f(x_0) - f^*}{\gamma} + 6. \quad (4.46)$$

Now, let us prove the above theorem. This consists of analyzing the stages of the Newton method.

**Proof of Theorem 4.9 Damped Newton stage.** We need to establish (4.41). Suppose that  $\|\nabla f(x)\|_2 \geq \eta$ . At the very beginning let us establish a lower bound on the stepsize computed by backtracking. From the strong convexity, it follows that on  $S$  we have  $\nabla^2 f(x) \leq MI$ . Hence,

$$\begin{aligned} f(x + \alpha d_{nt}) &\leq f(x) + \alpha \nabla f(x)^T d_{nt} + \frac{M \|d_{nt}\|_2^2}{2} \alpha^2 \\ &\leq f(x) - \alpha r(x)^2 + \frac{M}{2m} \alpha^2 r(x)^2 \end{aligned}$$

Observe that  $\hat{\alpha} = m/M$  verifies the termination condition of the backtracking, since

$$f(x + \hat{\alpha} d_{nt}) \leq f(x) - \frac{m}{2M} r(x)^2 \leq f(x) - \rho \hat{\alpha} r(x)^2.$$

Therefore, the backtracking gives a stepsize of length  $\alpha \geq \beta m/M$ , thus involving a reduction of the function values

$$\begin{aligned} f(x^+) - f(x) &\leq -\rho \alpha r(x)^2 \leq -\rho \beta \frac{m}{M} r(x)^2 \\ &\leq -\rho \beta \frac{m}{M^2} \|\nabla f(x)\|_2^2 \leq -\rho \beta \eta^2 \frac{m}{M^2}, \end{aligned}$$

That is, (4.41) is satisfied for

$$\gamma = \rho \beta \eta^2 \frac{m}{M^2}. \quad (4.47)$$

*Newton quadratic convergent stage.* Now, let us establish (4.42). Suppose that  $\|\nabla f(x)\|_2 < \eta$ . Let us prove that, if

$$\eta \leq 3(1 - 2\rho) \frac{m^2}{L},$$

then the *backtracking selects stepsizes of unitary length*. But the Lipschitz continuity of the Hessian matrix

$$\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq L \|x - y\|_2$$

determines that for all  $\alpha \geq 0$

$$-\alpha L \|d_{nt}\|_2^3 \leq d_{nt}^T (\nabla^2 f(x + \alpha d_{nt}) - \nabla^2 f(x)) d_{nt} \leq \alpha L \|d_{nt}\|_2^3.$$

This gives the possibility to determine an upper bound for  $\varphi(\alpha) = f(x + \alpha d_{nt})$ . Indeed,

$$-\alpha L \|d_{nt}\|_2^3 \leq \varphi''(\alpha) - \varphi''(0) \leq \alpha L \|d_{nt}\|_2^3.$$

By integration twice we get

$$\varphi(\alpha) \leq \varphi(0) + \alpha \varphi'(0) + \frac{\alpha^2}{2} \varphi''(0) + \frac{\alpha^3}{6} L \|d_{nt}\|_2^3, \quad (4.48)$$

and

$$f(x + \alpha d_{nt}) \leq f(x) + \alpha \nabla f(x)^T d_{nt} + \frac{\alpha^2}{2} d_{nt}^T \nabla^2 f(x) d_{nt} + \frac{\alpha^3}{6} L \|d_{nt}\|_2^3. \quad (4.49)$$

Analyzing the first three terms in (4.49), observe that they are the second order approximation of  $f(x + \alpha d_{nt})$  around  $\alpha = 0$ . The fourth gives an upper bound of the deviation from the quadratic function. The bound (4.49) holds for any direction  $d$ . Therefore, applied to the Newton direction, we get

$$\begin{aligned} f(x + \alpha d_{nt}) &\leq f(x) - \alpha \left(1 - \frac{\alpha}{2}\right) \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) + \frac{\alpha^3}{6} L \|\nabla^2 f(x)^{-1} \nabla f(x)\|_2^3 \\ &\leq f(x) - \alpha \left(1 - \frac{\alpha}{2}\right) r(x)^2 + \alpha^3 \frac{L}{6m^{3/2}} r(x)^3. \end{aligned} \quad (4.50)$$

Suppose that  $\|\nabla f(x)\|_2 \leq \eta \leq 3(1 - 2\rho)m^2/L$ . From the strong convexity of function  $f$ , it follows that

$$r(x) \leq 3(1 - 2\rho)m^{3/2}/L$$

and

$$\begin{aligned} f(x + d_{nt}) &\leq f(x) - \frac{1}{2}r(x)^2 + \frac{L}{6m^{3/2}}r(x)^3 \\ &\leq f(x) - \rho r(x)^2 = f(x) + \rho \nabla f(x)^T d_{nt}, \end{aligned}$$

that is, the stepsize  $\alpha = 1$  is accepted by backtracking.

Now, let us examine the rate of convergence. From the Lipschitz continuity we have

$$\begin{aligned} \|\nabla f(x^+)\|_2 &= \|\nabla f(x + d_{nt}) - \nabla f(x) - \nabla^2 f(x)d_{nt}\|_2 \\ &= \left\| \int_0^1 (\nabla^2 f(x + \alpha d_{nt}) - \nabla^2 f(x))d_{nt} d\alpha \right\|_2 \\ &\leq \frac{L}{2} \|d_{nt}\|_2^2 = \frac{L}{2} \|\nabla^2 f(x)^{-1} \nabla f(x)\|_2^2 \leq \frac{L}{2m^2} \|\nabla f(x)\|_2^2, \end{aligned} \quad (4.51)$$

that is exactly (4.42).  $\blacklozenge$

In conclusion, the Newton algorithm selects unitary stepsizes and satisfies (4.42) if  $\|\nabla f(x)\|_2 < \eta$ , where

$$\eta = \min \{1, 3(1 - 2\rho)\} \frac{m^2}{L}. \quad (4.52)$$

Introducing (4.47) and (4.52) in (4.46), we obtain that the total number of iterations corresponding to the Newton method is bounded by

$$\frac{f(x_0) - f^*}{\rho \beta \left( \min \{1, 3(1 - 2\rho)\} \frac{m^2}{L} \right)^2 \frac{m}{M^2}} + 6. \quad (4.53)$$

Observe that:

- ◆ The estimation (4.53) depends on three constants  $m$ ,  $M$ , and  $L$  which are unknown for the vast majority of practical optimization applications. Moreover, this estimation is valid only for twice continuously differentiable, strongly convex functions.
- ◆ The number of iterations depends on the initial point  $x_0$ . (See Figs. 4.1 and 4.6.) The closer to  $x^*$  the point  $x_0$  is, that is the smaller the initial suboptimality  $f(x_0) - f^*$  is, the smaller the number of iterations in the damped Newton stage will be. This emphasizes once more the local character of the Newton method as well as the importance of choosing a good initial point.
- ◆ The number of iterations in the Newton method does not explicitly depend on the number of variables of the problem.
- ◆ The parameters  $\rho$  and  $\beta$  from backtracking are not important in the convergence of the damped Newton stage.

Therefore, the bound (4.53) does not have a practical value, but a conceptual one, showing that for twice continuously differentiable, strongly convex, functions the Newton algorithm converges quadratically to the solution if the initial point is in a neighborhood of the minimum point. If the initial point is far way from the solution, nothing can be said about convergence.

**Example 4.4** Consider the minimization of the function

$$f(x) = (x_1 - 3)^2 + \sum_{i=2}^n \left( x_1 - 3 - 2(x_1 + x_2 + \cdots + x_i)^2 \right)^2$$

The solution of this problem is  $x^* = [3, -3, 0, \dots, 0]$ , for which  $f(x^*) = 0$ . Observe that the function is complex, and its Hessian matrix is a full matrix.

Taking  $n = 50$  and different initial points, then the initial value of function  $f$  and the number of iterations (#iter) for the Newton method with backtracking are as in Table 4.7

For  $n = 100$ , the number of iterations of the Newton method for minimizing function  $f$  are presented in Table 4.8.

For  $n = 300$ , the Newton method gives the results from Table 4.9.

From the Tables 4.7–4.9, we can see that the bigger the initial suboptimality  $f(x_0) - f^*$ , the bigger the number of iterations, which confirms the structure of (4.53).

**Table 4.7** Initial points, value of function in initial points, number of iterations.  $n = 50$ ,  $\varepsilon = 10^{-8}$ ,  $\rho = 0.0001$  and  $\beta = 0.8$

Nr.	$x_0$	$f(x_0)$	#iter
1	[0.001, ..., 0.001]	450.215229	9
2	[0.01, ..., 0.01]	500.96877	16
3	[0.1, ..., 0.1]	31666.3496	28
4	[1, ..., 1]	263010248.01	44

**Table 4.8** Initial points, value of function in initial points, number of iterations.  $n = 100$ ,  $\varepsilon = 10^{-8}$ ,  $\rho = 0.0001$  and  $\beta = 0.8$

Nr.	$x_0$	$f(x_0)$	#iter
1	[0.001, ..., 0.001]	903.46713	13
2	[0.01, ..., 0.01]	1380.68874	17
3	[0.1, ..., 0.1]	860222.816	40
4	[1, ..., 1]	$0.82040405 - 10^{10}$	54

**Table 4.9** Initial points, value of function in initial points, number of iterations.  $n = 300$ ,  $\varepsilon = 10^{-8}$ ,  $\rho = 0.0001$  and  $\beta = 0.8$

Nr.	$x_0$	$f(x_0)$	#iter
1	[0.001, ..., 0.001]	2808.6649	16
2	[0.01, ..., 0.01]	33102.2686	23
3	[0.1, ..., 0.1]	$0.197075349 - 10^9$	66
4	[1, ..., 1]	$0.196030836 - 10^{13}$	85

### Advantages and Disadvantages of the Newton Method

The Newton method has some very important *advantages* which establish its central position in the frame of unconstrained optimization methods:

- In general the convergence of the Newton method is rapid, being quadratic near  $x^*$ . The algorithm is immediately in the attraction basin of  $x^*$ , that is, the iterations are beyond the damped stage. The algorithm gives a very accurate solution in 5 or 6 iterations at most.
- The Newton method is affine invariant, i.e., it is not sensitive to any change of the coordinate system or to the condition number of the level sets.
- The backtracking parameters  $\rho$  and  $\beta$  have little influence over the performances of the Newton method. The exact line-search does not significantly improve the performances of the method. In the most practical implementations of backtracking,  $\rho = 0.0001$  and  $\beta = 0.8$ .
- The Newton method is indifferent to the dimension of the problem. Its performances in solving problems with a large number of variables are similar to its performances in solving problems with a small number of variables.

The most important *disadvantages* of the Newton method are:

- The method depends on the initial point. The difficulties of convergence appear in the damped stage. In general, for nonconvex functions, if the initial point is far away from  $x^*$ , then the method can be divergent. The hypothesis of convexity is crucial in the Newton method.
- The practical implementation of this method requires the computation and storage of the Hessian of the minimizing function as well as the solving of the Newton system. These are very difficult tasks, especially for complicated, strongly nonlinear unconstrained optimization problems or for problems with a large number of variables. The quasi-Newton methods, which inherit some properties of the Newton method, can be used for solving these difficult problems, but at a superlinear rate of convergence. In other words, *what we gain in the computational effort we lose in the convergence*.

## 4.6 The Modified Newton Method

As known, given the initial point  $x_0$ , the Newton method is defined by the following computational scheme:

$$x_{k+1} = x_k + \alpha_k d_k, \quad (4.54)$$

$$d_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k), \quad (4.55)$$

for  $k = 0, 1, \dots$ . The Newton method is well definite if and only if the following conditions hold: (i) the Hessian  $\nabla^2 f(x_k)$  is nonsingular and (ii) the approximation (4.21) is valid. If condition (i) is violated, then (4.55) may have an infinite number of solutions or no solution at all. On the other hand, if condition (ii) is violated, then  $d_k$  may not yield the solution in one iteration and, if  $\nabla^2 f(x_k)$  is not positive definite,  $d_k$  may not even yield a reduction in the minimizing function.

Therefore, one of the most important difficulties with this method is that the Hessian matrix  $\nabla^2 f(x_k)$  might not be positive definite. In this case we have no certainty that the quadratic model approximating function  $f$  around  $x_k$  has a minimum point. If  $\nabla^2 f(x_k)$  is not defined, then the quadratic

model  $m_k^2(x)$  is unbounded. In order to overcome these situations, some modifications of the Newton method have been suggested, the most important ones discussed in this section being Levenberg (1944) and Marquardt (1963), Goldfeld, Quandt, and Trotter (1966), Zwart (1969), Matthews and Davies (1971), Goldstein and Price (1967), and Gill and Murray (1972). Other modified Newton methods like Fiacco and McCormick (1968), Fletcher and Freeman (1977), second order Armijo step rule by Goldfarb (1980), and by Moré and Sorensen (1979) were analyzed by Sun and Yuan (2006). Some other papers on using the negative curvature directions in optimization were given by (Forsgren, Gill, & Murray, 1995), (Facchinei, & Lucidi, 1998), (Gill, Kungurtsev, & Robinson, 2017a, 2017b), and (Goldfarb, Mu, Wright, & Zhou, 2017).

### (a) The Levenberg and Marquardt Method

A simple technique to ensure that the search direction is a descent one is given by the Levenberg-Marquardt modification of the Newton method:

$$x_{k+1} = x_k - (\nabla^2 f(x_k) + \mu_k I)^{-1} \nabla f(x_k), \quad k = 0, 1, \dots, \quad (4.56)$$

where  $\mu_k \geq 0$ .

Consider the symmetric matrix  $\nabla^2 f(x)$ , which may not be positive definite and let  $\lambda_1, \dots, \lambda_n$  be its eigenvalues with eigenvectors  $v_1, \dots, v_n$ . Obviously, the eigenvalues  $\lambda_1, \dots, \lambda_n$  are real, but may not all be positive. Now, let us consider the matrix  $G = \nabla^2 f(x) + \mu I$ , where  $\mu \geq 0$ . The eigenvalues of  $G$  are  $\lambda_1 + \mu, \dots, \lambda_n + \mu$ . Indeed,

$$\begin{aligned} Gv_i &= (\nabla^2 f(x) + \mu I)v_i = \nabla^2 f(x)v_i + \mu Iv_i \\ &= \lambda_i v_i + \mu v_i = (\lambda_i + \mu)v_i, \end{aligned}$$

which shows that for all  $i = 1, \dots, n$ ,  $v_i$  is also an eigenvector of  $\nabla^2 f(x)$  with eigenvalue  $\lambda_i + \mu$ . Therefore, if the parameter  $\mu$  is sufficiently large, then we are sure that all the eigenvalues of  $G$  are positive, that is,  $G$  is positive definite. In other words, if the parameter  $\mu_k$  in the Levenberg-Marquardt modification of the Newton method (4.56) is sufficiently large, then the search direction  $d_{k+1} = -(\nabla^2 f(x_k) + \mu_k I)^{-1} \nabla f(x_k)$  always is a descent direction. If  $\mu_k \rightarrow 0$ , then the Levenberg-Marquardt modification of the Newton method (4.56) approaches the behavior of the pure Newton method. On the other hand, if  $\mu_k \rightarrow \infty$ , then the algorithm approaches a pure gradient method with small stepsizes. In practice, the algorithm starts with a small value of  $\mu_k$ , and then slowly it is increased until a descent iteration is obtained, that is, until  $f(x_{k+1}) < f(x_k)$ .

### (b) The Goldfeld, Quandt, and Trotter Method

When the Hessian  $G_k \equiv \nabla^2 f(x_k)$  becomes nonpositive, this method replaces the Hessian by the  $n \times n$  identity matrix  $I_n$ . This approach can be implemented by considering

$$\widehat{G}_k = \frac{G_k + \gamma I_n}{1 + \gamma}, \quad (4.57)$$

where  $\gamma$  is set to a large value if  $G_k$  is nonpositive definite or to a small value if  $G_k$  is positive definite. Obviously, if  $\gamma$  is large, then  $\widehat{G}_k \approx I_n$  and therefore  $d_k \approx -\nabla f(x_k)$ . As known, a nonpositive definite  $G_k$  arises at points far from the solution, where the steepest descent method is most effective in reducing the value of  $f(x)$ . Therefore, the above modification (4.57) leads to an algorithm that combines the properties of the Newton and steepest descent methods.

### (c) The Zwart Method

The idea of this method is to form a modified matrix

$$\widehat{G}_k = U^T G_k U + \varepsilon, \quad (4.58)$$

where  $U$  is a real unitary matrix (i.e.,  $U^T U = I_n$ ) and  $\varepsilon$  is a diagonal  $n \times n$  matrix with diagonal elements  $\varepsilon_i$ ,  $i = 1, \dots, n$ . Obviously, for the real symmetric matrix  $G_k$ , there exists a real unitary (or orthogonal) matrix  $U$  such that  $U^T G_k U$  is a diagonal matrix with diagonal elements  $\lambda_i$ ,  $i = 1, \dots, n$ , where  $\lambda_i$  are the eigenvalues of  $G_k$ . Therefore, from (4.58), we can see that  $\widehat{G}_k$  is a diagonal matrix with the elements  $\lambda_i + \varepsilon_i$ ,  $i = 1, \dots, n$ . If  $\varepsilon_i$ ,  $i = 1, \dots, n$ , are selected as

$$\varepsilon_i = \begin{cases} 0, & \lambda_i > 0, \\ \delta - \lambda_i, & \lambda_i \leq 0, \end{cases}$$

where  $\delta$  is a positive constant, then  $\widehat{G}_k$  will be positive definite. The matrix  $U^T G_k U$  can be computed by solving the characteristic equation  $\det(G_k - \lambda I_n) = 0$ . This method involves a minimal disturbance of  $G_k$ , and therefore the convergence properties of the Newton method are preserved. However, the difficulty with this method is the determination of the  $n$  roots of the characteristic polynomial of  $G_k$ .

### (d) The Matthews and Davies Method

This method simultaneously determines the modification of  $G_k$  and the Newton search direction  $d_k$ . As known, for the matrix  $G_k$ , a diagonal matrix  $D$  can be computed as  $D = L G_k L^T$ , where  $L = L_{n-1} \cdots L_2 L_1$ , is a unit lower triangular matrix and  $L_1, L_2, \dots, L_{n-1}$  are the elementary matrices (see Appendix A). If  $G_k$  is positive definite, then  $D$  is positive definite and vice-versa. If  $D$  is not positive definite, then a positive definite diagonal matrix  $\widehat{D}$  can be formed by replacing each zero or negative element in  $D$  by a positive element. In this way, a positive definite matrix  $\widehat{G}_k$  can be formed as

$$\widehat{G}_k = L^{-1} \widehat{D} (L^T)^{-1}.$$

Therefore, the search direction can be computed as solution of the system  $\widehat{G}_k d_k = -\nabla f(x_k)$ , that is,

$$L^{-1} \widehat{D} (L^T)^{-1} d_k = -\nabla f(x_k). \quad (4.59)$$

Denoting  $\widehat{D}(L^T)^{-1} d_k = z_k$ , then (4.59) can be expressed as  $L^{-1} z_k = -\nabla f(x_k)$ . Therefore,

$$z_k = -L \nabla f(x_k) = -L_{n-1} \cdots L_2 L_1 \nabla f(x_k).$$

With this, the search direction  $d_k$  can be computed as  $d_k = L^T \widehat{D}^{-1} z_k$ .

### (e) The Goldstein and Price Method

When  $\nabla^2 f(x_k)$  is not positive definite, then, instead of the direction  $d_k$  from (4.55), the steepest descent method is used. From the angle rule

$$\theta \leq \frac{\pi}{2} - \mu$$

for a certain  $\mu > 0$  where  $\theta$  is the angle between  $d_k$  and  $-\nabla f(x_k)$ , then the search direction in this method is computed as

$$d_k = \begin{cases} -\nabla^2 f(x_k)^{-1} g_k, & \cos \theta \geq \eta, \\ -g_k, & \text{otherwise,} \end{cases}$$

where  $\eta > 0$  is a given constant. Obviously, the direction  $d_k$  satisfies the condition  $\cos \theta \geq \eta$ , thus obtaining the convergence of the method.

#### (f) The Gill and Murray Method

In this method, when  $\nabla^2 f(x_k)$  is not positive definite, then the Hessian matrix  $\nabla^2 f(x_k)$  is modified as  $\nabla^2 f(x_k) + \nu_k I$ , where  $\nu_k > 0$  is selected in such a way that the matrix  $\nabla^2 f(x_k) + \nu_k I$  is positive definite and well conditioned. At iteration  $k$  the following steps are performed: set  $\bar{G}_k = \nabla^2 f(x_k) + \nu_k I$ , where  $\nu_k = 0$  if  $\nabla^2 f(x_k)$  is positive definite or  $\nu_k > 0$  otherwise; solve the modified Newton system  $\bar{G}_k d_k = -g_k$  subject to  $d_k$  and update the variables as  $x_{k+1} = x_k + d_k$ .

For the selection of the parameter  $\nu_k$ , Gill and Murray (1974a) suggested the following procedure based on the Cholesky factorization of  $G_k \equiv \nabla^2 f(x_k)$  as  $G_k + E = LDL^T$ , where  $E$  is a diagonal matrix with nonnegative elements. If  $E = 0$ , then consider  $\nu_k = 0$ . If  $E \neq 0$ , then, by using the Gershgorin circles, compute an upper bound  $a_1$  of  $\nu_k$  as

$$a_1 = \left| \min_{1 \leq i \leq n} \left\{ (G_k)_{ii} - \sum_{j \neq i} |(G_k)_{ij}| \right\} \right| \geq \left| \min_i \lambda_i \right|.$$

Observe that

$$a_2 = \max_i \{e_{ii}\}$$

is also an upper bound of  $\nu_k$ , where  $e_{ii}$  is the  $i$ -th diagonal element of  $E$ . With these, an estimation of  $\nu_k$  is given by

$$\nu_k = \min \{a_1, a_2\},$$

which gives a positive definite matrix  $\bar{G}_k$ .

In the following, let us present a *Cholesky factorization which is numerically stable* and can be used in the Newton method as well as in the frame of some other optimization methods. The Cholesky factorization  $G_k = LDL^T$  of the positive definite matrix  $G_k$  is expressed as

$$d_{jj} = g_{jj} - \sum_{p=1}^{j-1} d_{pp} l_{jp}^2, \quad (4.60)$$

$$l_{ij} = \frac{1}{d_{jj}} \left( g_{ij} - \sum_{p=1}^{j-1} d_{pp} l_{jp} l_{ip} \right), \quad i \geq j+1, \quad (4.61)$$

where  $g_{ij}$  represents the elements of  $G_k$  and  $d_{jj}$  are the diagonal elements of  $D$ . At this moment, for defining a *stable Cholesky factorization* for the factors  $L$  and  $D$ , the following conditions are imposed:

- (a) All the diagonal elements of  $D$  are positive
- (b) The elements of the matricial factors  $L$  and  $D$  are uniformly bounded, that is,

$$d_{kk} > \delta > 0 \text{ for any } k \text{ and } |r_{ik}| \leq \beta \text{ for } i > k, \quad (4.62)$$

where  $r_{ik} = l_{ik}\sqrt{d_{kk}}$ ,  $\beta$  is a given positive number, and  $\delta$  is a positive number sufficiently small.

In the following, let us present the step  $j$  of this factorization. Suppose that the first  $j - 1$  columns of the factors  $L$  and  $D$  have been computed, that is, for  $k = 1, \dots, j - 1$ , the elements  $d_{kk}$  and  $l_{ik}$ ,  $i = 1, \dots, n$ , have been determined so that they satisfy the requirements from (4.62). Now, compute

$$\gamma_j = \left| \xi_j - \sum_{p=1}^{j-1} d_{pp} l_{jp}^2 \right|,$$

where  $\xi_j$  is  $g_{jj}$  and set

$$\bar{d} = \max \{\gamma_j, \delta\}.$$

In order to establish that  $\bar{d}$  can be accepted as the  $j$ -th element of  $D$ , test if  $r_{ij} = l_{ij}\sqrt{\bar{d}}$  satisfies (4.62). If it does, set  $d_{jj} = \bar{d}$  and compute the  $j$ -th column of  $L$  where  $l_{ij} = r_{ij}/\sqrt{d_{jj}}$ . Otherwise, if  $r_{ij}$  does not satisfy (4.62), then set

$$d_{jj} = \left| \xi_j - \sum_{p=1}^{j-1} d_{pp} l_{jp}^2 \right|,$$

where  $\xi_j = g_{jj} + e_{jj}$ , in which  $e_{jj}$  is selected in such a way so that  $\max|r_{ij}| = \beta$ , with which, as above, compute the  $j$ -th column of  $L$ .

At step  $n$  of this procedure, we obtain a Cholesky factorization of  $\bar{G}_k$  as

$$\bar{G}_k = LDL^T = G_k + E,$$

where  $E$  is a diagonal matrix with diagonal elements  $e_{ii}$  nonnegative. For a given matrix  $G_k$ , the elements of the matrix  $E$  depend on  $\beta$ . Gill and Murray (1974a) proved that if  $n > 1$ , then

$$\|E(\beta)\|_\infty \leq \left( \frac{\xi}{\beta} + (n-1)\beta \right)^2 + 2(\gamma + (n-1)\beta^2) + \delta,$$

where  $\xi$  and  $\gamma$  are the maximum of the absolute value of the nondiagonal elements of  $G_k$  and the maximum of the diagonal elements of this matrix, respectively. Since for  $\beta^2 = \xi/\sqrt{n^2 - 1}$  the above bound is minimized, then we can consider

$$\beta^2 = \max \left\{ \gamma, \xi/\sqrt{n^2 - 1}, \varepsilon_M \right\}, \quad (4.63)$$

where  $\varepsilon_M$  represents the accuracy of the machine. Introducing  $\varepsilon_M$  in (4.63) is motivated by the case in which  $\|G_k\|$  is too small. With these developments, the following modified Cholesky factorization algorithm may be presented, in which  $c_{ip} = l_{ip}d_{pp}$ ,  $p = 1, \dots, j$ ;  $i = j, \dots, n$  are the parameters stored in  $G_k$  for reducing the memory requirements.

**Algorithm 4.4** Modified Cholesky factorization – Gill and Murray

- |    |   |
|----|---|
| 1. | Compute $\beta$ as in (4.63). Set: $j = 1, c_{ii} = g_{ii}, i = 1, \dots, n$ . Consider a value for $\delta$  |
| 2. | Determine the smallest index $q$ such that $ c_{qq}  = \max_{j \leq i \leq n}  c_{ii} $ . Permute line $q$ with line $i$ and column $q$ with column $i$   |
| 3. | Compute line $i$ of $L$ and determine the maximum module of $l_{ij}d_{jj}$<br>Set $l_{jp} = c_{jp}/d_{pp}, p = 1, \dots, j - 1$<br>Compute $c_{ij} = g_{ij} - \sum_{p=1}^{j-1} l_{jp}c_{ip}, i = j + 1, \dots, n$<br>Determine $\theta_j = \max_{j+1 \leq i \leq n}  c_{ij} $ , (if $j = n$ , then $\theta_j = 0$ ) |
| 4. | Compute the $j$ -th diagonal element of $D$ : $d_{jj} = \max \left\{ \delta,  c_{jj} , \theta_j^2/\beta^2 \right\}$ and update the element $e_{jj}$ as $e_{jj} = d_{jj} - c_{jj}$   |
| 5. | Update $c_{ii} = c_{ii} - c_{ij}^2/d_{jj}, i = j + 1, \dots, n$ . Set $j = j + 1$ and continue with step 2  |

◆

The modified Cholesky factorization is important in the unconstrained optimization, particularly in the modified Newton method. Note that if  $\bar{G}_k = G_k + E_k$  is positive definite and the condition number of this matrix is uniformly bounded, that is,  $\|\bar{G}_k\| \|\bar{G}_k^{-1}\| \leq \kappa$ , where  $\kappa \geq 0$ , then we have

$$-\frac{\nabla f(x_k)^T s_k}{\|s_k\|} \geq \frac{1}{\kappa} \|\nabla f(x_k)\|.$$

With these, under the Wolfe inexact line-search, from Theorem 2.27 it follows that the sequence  $\{\nabla f(x_k)\}$  is convergent to zero. In other words, for the modified Newton method, the following result holds.

**Theorem 4.10** Let  $f: D \subset \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the open set  $D$  and the level set  $S = \{x : f(x) \leq f(x_0)\}$  a compact. If the sequence  $\{x_k\}$  is generated by the modified Newton method, then

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

◆

We can see that in order to get a well-defined Newton method, there are a lot of procedures for modifying the Hessian matrix of the minimizing function. We point out that there are no computational studies about the efficiency of all these modifications of the Newton method and about their classification for seeing which one is the best.

## 4.7 The Newton Method with Finite-Differences

As we have already seen, at every iteration the Newton method requires the evaluation of the Hessian matrix of the minimizing function  $f$  and the solving of the Newton system  $\nabla^2 f(x_k)d_k = -\nabla f(x_k)$  for determining the Newton search direction  $d_k$ . For the evaluation of the Hessian matrix  $\nabla^2 f(x_k)$  in the current point  $x_k$ , we have two possibilities. The first one consists of the analytical determination of the elements of the Hessian. The second one is automatic differentiation. The automatic differentiation is a powerful solution and is included in some professional systems for modeling and optimization like GAMS (Brooke, Kendrick, Meeraus, Raman, & Rosenthal, 1998), (Brooke, Kendrick, Meeraus, &

Raman, 2005), and AMPL (Fourer, Gay, & Kernighan, 2002). An alternative for these two possibilities is to use the *finite-difference approximation* of the derivatives.

In this section, the Newton method with finite-differences is presented, both for solving nonlinear algebraic systems of equations and for functions minimization. At the start, let us discuss the approximation of derivatives by finite-differences. For function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  the estimation of  $f'(x)$  by finite differencing using only values of  $f(x)$  is discussed in Appendix A.

Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . Then the element  $(i, j)$  of the Jacobian matrix  $J(x)$  of function  $F(x)$  can be approximated as

$$\frac{\partial f_i(x)}{\partial x_j} \cong a_{ij} = \frac{f_i(x + he_j) - f_i(x)}{h}, \quad (4.64)$$

where  $f_i(x)$  is the  $i$ -th component of  $F(x)$ ,  $e_j$  is the  $j$ -th column of the unity matrix, and  $h$  is a small perturbation of  $x$ .

If we denote with  $A_j$  as the  $j$ -th column of the matrix  $A$  which approximates the Jacobian, then

$$A_j = \frac{F(x + he_j) - F(x)}{h}. \quad (4.65)$$

**Theorem 4.11** *Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be continuously differentiable on the open and convex set  $D \subset \mathbb{R}^n$  with  $J(x)$  Lipschitz continuous with the constant  $L$  for any  $x \in D$ . Then, in the norm  $\|\cdot\|$  which satisfies  $\|e_j\| = 1$ , for any  $j = 1, \dots, n$  it follows that*

$$\left\| A_j - J(x)_j \right\| \leq \frac{L}{2} |h|, \quad (4.66)$$

and in the norm  $l_1$ ,

$$\|A - J(x)\|_1 \leq \frac{L}{2} |h|. \quad (4.67)$$

**Proof** As we know (see Theorem 2.7, (2.18)), for any  $x, x + d \in D$

$$\|F(x + d) - F(x) - J(x)d\| \leq \frac{L}{2} \|d\|^2.$$

Considering  $d = he_j$ , from the above inequality, it follows that

$$\|F(x + he_j) - F(x) - J(x)he_j\| \leq \frac{L}{2} \|he_j\|^2 = \frac{L}{2} |h|^2.$$

Dividing by  $h$ , we get (4.66). For proving (4.67) we remember that  $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|$ , thus immediately getting (4.67).  $\blacklozenge$

An efficient calculation of the sparse Jacobian matrix for nonlinear systems of equations using finite differences was presented by Andrei (1983).

Let us now consider the function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . An approximation to the gradient  $\nabla f(x)$  in  $x$  can be obtained by the *forward finite-difference formula*

$$\frac{\partial f(x)}{\partial x_i} \cong \frac{f(x + he_i) - f(x)}{h}. \quad (4.68)$$

Obviously, for the evaluation of the gradient, the function  $f$  has to be evaluated in  $n + 1$  points:  $x$  and  $x + he_i$ ,  $i = 1, \dots, n$ . Therefore, from (2.18),

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x + he_i) - f(x)}{h} + \delta_h, \quad (4.69)$$

where

$$|\delta_h| \leq \frac{L}{2}h. \quad (4.70)$$

Hence, the error in approximation by the forward finite-difference is of order  $O(h)$ .

A highly accurate approximation is obtained by using the *central finite-difference*

$$\frac{\partial f(x)}{\partial x_i} \cong \frac{f(x + he_i) - f(x - he_i)}{2h}. \quad (4.71)$$

The following theorems give bounds on the errors in the approximation of the gradient and of the Hessian by central finite-difference.

**Theorem 4.12** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuous differentiable function on the open and convex set  $D \subset \mathbb{R}^n$  for which the Hessian  $\nabla^2 f(x)$  is Lipschitz continuous with the constant  $L$  for any  $x \in D$ . Consider the norm  $\|\cdot\|$  which satisfies  $\|e_i\| = 1$  for all  $i = 1, \dots, n$ . Suppose that for any  $i = 1, \dots, n$ ,  $x + he_i, x - he_i \in D$ . Let  $a \in \mathbb{R}^n$  be a vector with the components  $a_i$ ,  $i = 1, \dots, n$ , defined as

$$a_i = \frac{f(x + he_i) - f(x - he_i)}{2h}. \quad (4.72)$$

Then

$$|a_i - |\nabla f(x)|_i| \leq \frac{L}{6}h^2. \quad (4.73)$$

If the infinite norm  $l_\infty$  is used, then

$$\|a - \nabla f(x)\|_\infty \leq \frac{L}{6}h^2. \quad (4.74)$$

**Proof** Consider Theorem 2.6 and define  $\alpha$  and  $\beta$  as

$$\alpha = f(x + he_i) - f(x) - h[\nabla f(x)]_i - \frac{1}{2}h^2[\nabla^2 f(x)]_{ii}, \quad (4.75)$$

$$\beta = f(x - he_i) - f(x) + h[\nabla f(x)]_i - \frac{1}{2}h^2[\nabla^2 f(x)]_{ii}. \quad (4.76)$$

Then, taking  $d = \pm he_i$  in (2.16), we get

$$|\alpha| \leq \frac{L}{6}h^3, |\beta| \leq \frac{L}{6}h^3$$

From the triangle inequality,

$$|\alpha - \beta| \leq \frac{L}{3}h^3.$$

With this, from (4.75), (4.76), and (4.72), it follows that  $\alpha - \beta = 2h(a_i - [\nabla f(x)]_i)$ , which proves (4.73). Next, using the definition of  $l_\infty$  from (4.73), we get (4.74) at once, which proves the theorem.  $\blacklozenge$

**Theorem 4.13** *Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the convex and open set  $D \subset \mathbb{R}^n$  for which the Hessian  $\nabla^2 f(x)$  is Lipschitz continuous with the constant  $L$  for any  $x \in D$ . Consider the norm  $\|\cdot\|$  which satisfies  $\|e_i\| = 1$  for all  $i = 1, \dots, n$ . Suppose that  $x, x + he_i, x + he_j, x + he_i + he_j \in D$  for any  $i, j = 1, \dots, n$ . Let  $A \in \mathbb{R}^{n \times n}$  be a matrix with the elements*

$$a_{ij} = \frac{f(x + he_i + he_j) - f(x + he_i) - f(x + he_j) + f(x)}{h^2} \quad (4.77)$$

Then,

$$\left| a_{ij} - [\nabla^2 f(x)]_{ij} \right| \leq \frac{5}{3}Lh \quad (4.78)$$

If  $l_1$ ,  $l_\infty$ , or the Frobenius norm are used, then

$$\|A - \nabla^2 f(x)\| \leq \frac{5}{3}Lhn. \quad (4.79)$$

**Proof** Like in the proof of the above theorem, define

$$\begin{aligned} \alpha &= f(x + he_i + he_j) - f(x) - (he_i + he_j)^T \nabla f(x) - \frac{1}{2}(he_i + he_j)^T \nabla^2 f(x)(he_i + he_j), \\ \beta &= f(x + he_i) - f(x) - (he_i)^T \nabla f(x) - \frac{1}{2}(he_i)^T \nabla^2 f(x)(he_i), \\ \gamma &= f(x + he_j) - f(x) - (he_j)^T \nabla f(x) - \frac{1}{2}(he_j)^T \nabla^2 f(x)(he_j). \end{aligned}$$

Then,

$$\alpha - \beta - \gamma = h^2 \left( a_{ij} - [\nabla^2 f(x)]_{ij} \right) \quad (4.80)$$

On the other hand,

$$|\alpha - \beta - \gamma| \leq |\alpha| + |\beta| + |\gamma| \leq \frac{L}{6} \|he_i + he_j\|^3 + \frac{L}{6} \|he_i\|^3 + \frac{L}{6} \|he_j\|^3 \leq \frac{5}{3}Lh^3.$$

This inequality together with (4.80) proves (4.78). The inequality (4.79) follows from (4.78) and from the definition of the norms.  $\blacklozenge$

The Newton method with finite-difference for solving nonlinear algebraic systems of equations  $F(x) = 0$ , where  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is continuous differentiable, may be presented as follows.

**Algorithm 4.5** *Newton with finite-difference for  $F(x) = 0$*

1. *Initialization.* Consider an initial point  $x_0 \in \mathbb{R}^n$  as well as the convergence tolerance  $\varepsilon > 0$  sufficiently small for stopping the iterations. Set  $k = 0$
2. Compute  $F(x_k)$  and for  $j = 1, \dots, n$ , compute the  $j$ -th column  

$$(A_k)_j = \frac{F(x_k + h_k e_j) - F(x_k)}{h_k}$$
of matrix  $A_k$ , where  $h_k$  is a small perturbation of  $x_k$
3. Solve the linear algebraic system  $A_k d_k = -F(x_k)$
4. If  $\|d_k\|_2 \leq \varepsilon$ , then stop; otherwise continue with step 5
5. Set  $x_{k+1} = x_k + d_k$ ,  $k = k + 1$  and go to step 2

◆

### Applications Solved by the Newton Method with Finite-Difference

In the following, let us illustrate the running of Algorithm 4.5 for solving the applications S5, S6, S9, and S14 from the SMUNO collection by means of the Newton method with finite-difference, where at every iteration  $h_k = 10^{-6}$ .

**Application S5** (*Stationary solution of a chemical reactor*) Table 4.3 presents the initial point, the functions values in the initial point  $F(x_0)$ , solution of this application obtained by the Newton method with  $\varepsilon = 10^{-6}$  in 5 iterations, and the functions values in  $x^*$ . On the other hand, Table 4.10 shows the initial point, solution of this application given by the Newton method with finite-difference with  $\varepsilon = 10^{-6}$  in 5 iterations, and the functions values in these points.

**Application S6** (*Robot kinematics problem*) Table 4.4 shows the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method with  $\varepsilon = 10^{-6}$  in 3 iterations, and the functions values in  $x^*$ . Table 4.11 shows the initial point  $x_0$ , the solution  $x^*$  of this application given by the Newton method with finite-difference with  $\varepsilon = 10^{-6}$  in 4 iterations, and the functions values in these points.

**Table 4.10** Stationary solution of a chemical reactor. Initial point, solution, functions values in these points. Newton with finite-difference method. 5 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	1.09	-35.164864	0.97424361	0.176600e-07
2	1.05	-0.2403040	0.98282907	0.102566e-09
3	3.05	3626.94637	0.05151277	-0.771388e-04
4	0.99	-1779.9233	0.93567119	0.385517e-04
5	6.05	-1814.7127	0.00009070	0.385692e-04
6	1.09	-7.13	0.06423810	0.949240e-14

**Table 4.11** Robot kinematics problem. Initial point, solution, functions values in these points. Newton with finite-difference method. 4 iterations.  $\epsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	0.164	0.0026823	0.16443166	0.0
2	-0.98	-0.0098180	-0.98638847	0.1110223e-15
3	-0.94	0.1050028	-0.94706369	0.3469446e-17
4	-0.32	0.0017588	-0.32104573	0.0
5	-0.99	-0.0127040	-0.99823316	-0.4440892e-15
6	-0.056	-0.014	0.05941842	-0.4440892e-15
7	0.41	-0.016764	0.41103315	0.499000e-09
8	-0.91	-0.0038	-0.91162039	0

**Table 4.12** Propan combustion in air – reduced variant. Initial point, solution, functions values in these points. Newton with finite-difference method. 10 iterations.  $\epsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	10	109.85000	0.0031141022	-0.1600708e-09
2	10	209.54267	34.597924281	-0.4749359e-09
3	0,05	-0.3487418	0.0650417788	-0.2873626e-09
4	50,5	5098.5172	0.8593780968	0.15903885e-06
5	0,05	2659.2930	0.0369518591	0.79210239e-07

**Application S9** (*Propan combustion in air – reduced variant*) Table 4.5 contains the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method with  $\epsilon = 10^{-6}$  in 9 iterations, and the functions values in  $x^*$ . Table 4.12 shows the initial point, the solution  $x^*$  of this application obtained by the Newton method with finite-difference with  $\epsilon = 10^{-6}$  in 10 iterations, and the functions values in these points.

**Application S14** (*Circuit design*) Table 4.6 contains the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the Newton method with  $\epsilon = 10^{-6}$  in 4 iterations, and the functions values in  $x^*$ . Table 4.13 shows the initial point  $x_0$ , the solution  $x^*$  of this application given by the Newton method with finite-difference with  $\epsilon = 10^{-6}$  in 4 iterations, and the functions values in these points.

Observe that the performances of the Newton method with finite-difference are similar to those of the Newton method.

**Theorem 4.14** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be continuously differentiable on the convex and open set  $D \subset \mathbb{R}^n$ . Suppose that there are  $x^* \in \mathbb{R}^n$  and the constants  $r, \beta > 0$  such that  $B(x^*, r) \subset D$ ,  $F(x^*) = 0$ ,  $J(x^*)^{-1}$  exists and  $\|J(x^*)^{-1}\| \leq \beta$ , where  $J$  is Lipschitz continuous on  $B(x^*, r) = \{x \in \mathbb{R}^n : \|x - x^*\| < r\}$  centered in  $x^*$  of radius  $r$ . Then, there are  $\epsilon, h > 0$  such that if  $x_0 \in B(x^*, r)$  and  $\{h_k\}$  is a sequence of real numbers with  $0 < |h_k| \leq h$ , then the sequence  $\{x_k\}$  generated by Algorithm 4.5 is well defined and

**Table 4.13** Circuit design. Initial point, solution, functions values in these points. Newton with finite-difference method. 4 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	0.7	0.95525047	0.89999995	0.427648e-06
2	0.5	3.04155543	0.44998743	0.352005e-05
3	0.9	-4.16621316	1.00000645	0.101280e-04
4	1.9	-2.11846214	2.00006862	0.144598e-04
5	8.1	5.636125856	7.99997146	-0.1617286e-06
6	8.1	22.31397917	7.99969192	0.581500e-07
7	5.9	24.95943321	5.00003074	-0.1969207e-05
8	1	42.18784331	0.99998770	0.181683e-06
9	1.9	-0.32000000	2.00005229	0.700407e-08

converges linearly to  $x^*$ . If  $\lim_{k \rightarrow \infty} h_k = 0$ , then the convergence is superlinear. If there is a constant  $c_1$  such that

$$|h_k| \leq c_1 \|x_k - x^*\|, \quad (4.81)$$

or there is another constant  $c_2$  such that

$$|h_k| \leq c_2 \|F(x_k)\|, \quad (4.82)$$

then the convergence is quadratic.

**Proof** Select  $\varepsilon$  and  $h$  such that for  $x_k \in B(x^*, \varepsilon)$ , the matrix  $A_k$  is nonsingular and  $|h_k| < h$ . Consider  $\varepsilon \leq r$  and

$$\varepsilon + h \leq \frac{1}{2\beta L}. \quad (4.83)$$

By induction we prove that

$$\|x_{k+1} - x^*\| \leq \frac{1}{2} \|x_k - x^*\|, \quad (4.84)$$

that is,

$$x_{k+1} \in B(x^*, \varepsilon).$$

For  $k = 0$ , let us prove that  $A_0$  is nonsingular. From Theorem 4.11 it follows that

$$\|A(x) - J(x)\| \leq \frac{Lh}{2}.$$

Therefore,

$$\begin{aligned} \|J(x^*)^{-1}(A_0 - J(x^*))\| &\leq \|J(x^*)^{-1}\| \|(A_0 - J(x_0)) + (J(x_0) - J(x^*))\| \\ &\leq \beta \left( \frac{Lh}{2} + L\varepsilon \right) \leq \frac{1}{2}. \end{aligned}$$

Now, from the von Neumann theorem (see Appendix A), it follows that  $A_0$  is nonsingular and

$$\|A_0^{-1}\| \leq 2\beta. \quad (4.85)$$

Therefore,  $x_1$  is well defined and

$$\begin{aligned} x_1 - x^* &= x_0 - A_0^{-1}F(x_0) - x^* \\ &= A_0^{-1}\{[F(x^*) - F(x_0) - J(x_0)(x^* - x_0)] + [(J(x_0) - A_0)(x^* - x_0)]\}. \end{aligned}$$

By using Theorem 2.7 (see (2.18)), from (4.85) and (4.83), we get

$$\begin{aligned} \|x_1 - x^*\| &\leq \|A_0^{-1}\|\{\|F(x^*) - F(x_0) - J(x_0)(x^* - x_0)\| + \|J(x_0) - A_0\|\|x^* - x_0\|\} \\ &\leq 2\beta\left\{\frac{L}{2}\|x^* - x_0\|^2 + \frac{L}{2}h\|x_0 - x^*\|\right\} \\ &\leq \beta L(\varepsilon + h)\|x^* - x_0\| \leq \frac{1}{2}\|x_0 - x^*\|, \end{aligned} \quad (4.86)$$

that is, (4.84) is true for  $k = 0$ . Now, supposing that this result is true for  $k = j$  and by using the same technique, we can prove that the result is true for  $k = j + 1$ . Therefore, (4.83), (4.84) are true, thus proving the linear convergence of the sequence  $\{x_k\}$ .

For showing the superlinear and the quadratic convergence, a finer and more improved bound for  $\|A_0 - J(x_0)\|$  should be used. When  $\lim_{k \rightarrow \infty} h_k = 0$ , then the second term in the bracket of (4.86) tends to zero, thus proving that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \rightarrow 0, \text{ when } k \rightarrow \infty,$$

that is, the superlinear convergence of  $\{x_k\}$ .

Similarly, when (4.81) is satisfied, then from (4.78) we get the quadratic convergence of  $\{x_k\}$  to  $x^*$ .  $\blacklozenge$

For the optimization problem  $\min_{x \in \mathbb{R}^n} f(x)$ , when the gradient  $\nabla f(x)$  is available, as in the case of the vast majority of applications, then a finite-difference approximation of the Hessian  $\nabla^2 f(x)$  can be obtained by using either the forward finite-difference formula or the central finite-difference formula. At iteration  $k$ , the Newton method with central-difference for minimizing function  $f$  is

$$(A)_{.j} = \frac{\nabla f(x_k + h_j e_j) - \nabla f(x_k)}{h_j}, \quad j = 1, \dots, n,$$

$$A_k = \frac{A + A^T}{2},$$

$$x_{k+1} = x_k - A_k^{-1} \nabla f(x_k),$$

where

$$h_j = \sqrt{\eta} \max \{|x_j|, \tilde{x}_j\} \operatorname{sign}(x_j);$$

$\tilde{x}_j$  is an estimation of  $x_j$  and  $\eta$  is a sufficiently small number, close to the  $\varepsilon$  machine.

If the hypotheses of Theorem 4.14 are satisfied and if  $h_j$  is of order

$$h_j = O(\|x_k - x^*\|),$$

then the Newton method with the finite-difference  $x_{k+1} = x_k - A_k^{-1} \nabla f(x_k)$  has a quadratic rate of convergence. An appropriate choice of the difference parameter  $h_j$  can be difficult. The rounding errors overwhelm the calculation if  $h_j$  is too small, while the truncation errors dominate if  $h_j$  is too large. The Newton codes rely on forward differences, since they often yield sufficient accuracy for reasonable values of  $h_j$ . The central differences are more accurate, but they require the work twice ( $2n$  gradient evaluations against  $n$  evaluations).

Many algorithms for optimization use the product of the Hessian  $\nabla^2 f(x_k)$  and a given vector  $d_k$ . In this case, an approximation of the product  $\nabla^2 f(x_k) d_k$  is given by

$$\nabla^2 f(x_k) d_k \approx \frac{\nabla f(x_k + h d_k) - \nabla f(x_k)}{h},$$

where

$$h = \frac{2\sqrt{\epsilon_m}(1 + \|x_{k+1}\|)}{\|d_k\|},$$

for which the error is of order  $O(h)$ . Observe that for obtaining this approximation, only a single evaluation of the gradient in point  $x_k + h d_k$  is necessary.

If the gradient is not available, then an approximation of the Hessian can be computed by using the values of the minimizing function. From (4.71) we get the following approximation of the Hessian:

$$(\hat{A}_k)_{ij} = \frac{[f(x_k + h_i e_i + h_j e_j) - f(x_k + h_i e_i)] - [f(x_k + h_j e_j) - f(x_k)]}{h_i h_j},$$

where

$$h_j = \sqrt[3]{\eta} \max \{|x_j|, \tilde{x}_j\} \operatorname{sign}(x_j),$$

or

$$h_j = \sqrt[3]{\tilde{\epsilon}} x_j,$$

where  $\tilde{\epsilon}$  is the accuracy of the machine. In this case, the forward or the central finite-difference formula give the following approximations to the gradient:

$$(\hat{g}_k)_j = \frac{f(x_k + h_j e_j) - f(x_k)}{h_j}, \quad j = 1, \dots, n,$$

and

$$(\hat{g}_k)_j = \frac{f(x_k + h_j e_j) - f(x_k - h_j e_j)}{2h_j}, \quad j = 1, \dots, n,$$

respectively. The errors of these approximations are of order  $O(h_j)$  and  $O(h_j^2)$ , respectively. In this case, the Newton method with finite-difference is expressed as

$$x_{k+1} = x_k - \hat{A}_k^{-1} \hat{g}_k,$$

where  $\widehat{A}_k$  and  $\widehat{g}_k$  are approximations by the finite-difference of the Hessian  $\nabla^2 f(x_k)$  and of the gradient  $\nabla f(x_k)$ , respectively. In the standard hypothesis given by Theorem 4.14, we can prove that

$$\|x_{k+1} - x^*\| \leq \left\| \widehat{A}_k^{-1} \right\| \left( \frac{L}{2} \|x_k - x^*\|^2 + \left\| \widehat{A}_k - \nabla^2 f(x_k) \right\| \|x_k - x^*\| + \|\widehat{g}_k - \nabla f(x_k)\| \right). \quad (4.87)$$

Observe that the difference between (4.87) and (4.86) is the additional term  $\|\widehat{g}_k - \nabla f(x_k)\|$ . To get the quadratic convergence, we must obviously have  $\|\widehat{g}_k - \nabla f(x_k)\| = O(\|x_k - x^*\|^2)$ , thus involving the selection  $h_j = O(\|x_k - x^*\|^2)$ . Therefore, with this selection of  $h_j$  the convergence of the Newton method with central-difference is quadratic.

It is worth mentioning that *although the error of the Newton method with central-difference is of order  $O(h_j^2)$ , which is smaller than the error corresponding to the forward-difference  $O(h_j)$ , the cost of the central-difference scheme is twice greater than the forward-difference*. This is the reason why the central-difference scheme is very rarely used in practice, only for problems for which the evaluation of the minimizing function is relatively simple or for problems requiring solutions of great accuracy. Stewart (1967) gave some practical finite-difference schemes based on commuting from forward-difference to central-difference schemes.

## 4.8 Errors in Functions, Gradients, and Hessians

The unconstrained optimization problem becomes more difficult to solve in the presence of errors in the evaluations of the function and its gradient. Suppose that we can compute only  $f$  approximately. If we compute  $\widehat{f} = f + \epsilon_f$  rather than  $f$ , then a forward-difference gradient with the difference increment  $h$

$$D_h f(x) = \frac{\widehat{f}(x+h) - \widehat{f}(x)}{h}$$

differs from  $f'$  by  $O(h + \epsilon_f/h)$ . This difference is minimized if  $h = O(\sqrt{\epsilon_f})$ . Therefore, the error in the gradient is  $\epsilon_g = O(\sqrt{\epsilon_f})$ . If a forward-difference Hessian is computed by using  $D_h$  as an approximation to the gradient, then the error in the Hessian will be

$$\Delta = O(\sqrt{\epsilon_g}) = O(\epsilon_f^{1/4}) \quad (4.88)$$

and the accuracy in  $\nabla^2 f$  will be much smaller than that of a Jacobian in the nonlinear system of equations.

If  $\epsilon_f$  is significantly larger than the machine roundoff, then (4.88) indicates that using the numerical Hessians based on a second numerical differentiation of the minimizing function will not be very accurate. Even in the best possible cases, where  $\epsilon_f$  is the same size as the machine roundoff, the finite-difference Hessians will not be very accurate and will be quite expensive to compute if the Hessian is dense. As known, if the machine roundoff is  $10^{-16}$  for most computers, then (4.88) indicates that the forward difference Hessian will be accurate to roughly four decimal digits.

Better results can be obtained with central-differences, but at a cost of twice the number of function evaluations. A central-difference approximation to  $\nabla f$  is

$$D_h f(x) = \frac{\hat{f}(x+h) - \hat{f}(x-h)}{2h}$$

and the error is  $O(h^2 + \epsilon_f/h)$ , which is minimized if  $h = O(\epsilon_f^{1/3})$  leading to an error in the gradient of  $\epsilon_g = O(\epsilon_f^{2/3})$ . Therefore, a central-difference Hessian will have an error of

$$\Delta = O((\epsilon_g)^{2/3}) = O(\epsilon_f^{4/9}),$$

which is substantially better.

*Therefore, accurate gradients are much more important than the accurate Hessian and one option is to compute the gradients with central differences and the Hessians with forward differences.* In this case, the central-difference gradient error is  $O(\epsilon_f^{2/3})$ , and therefore the forward-difference Hessian error will be  $\Delta = O(\sqrt{\epsilon_g^{1/2}}) = O(\epsilon_f^{1/3})$ .

However, in many optimization problems and applications, the accurate gradients are available. In these cases, the numerical differentiation by forward-difference for computing the Hessians is, like the numerical computation of the Jacobians for solving nonlinear algebraic systems, a reasonable idea for many problems and the less expensive forward-differences work well in practice (Kelley, 1999).

The local convergence and the evolution of the error for the Newton method are proved in the following theorem.

**Theorem 4.15** Suppose that  $f$  is twice continuously differentiable,  $\nabla f(x^*) = 0$ ,  $\nabla^2 f(x^*)$  is positive definite and  $\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq \gamma \|x - y\|$ . Then there are  $K > 0$ ,  $\delta > 0$  such that if  $x_k \in B(\delta)$ , then the Newton iterate from  $x_k$  given by  $x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$  satisfies

$$\|e_{k+1}\| \leq K \|e_k\|^2,$$

where  $e_k = x_k - x^*$  and  $B(\delta)$  is the ball of radius  $\delta$  around  $x^*$ .

**Proof** Let  $\delta$  be small enough so that the conclusions of Theorem A2.5 hold. By Theorem 2.1 it follows that

$$\begin{aligned} e_{k+1} &= e_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k) \\ &= \nabla^2 f(x_k)^{-1} \int_0^1 (\nabla^2 f(x_k) - \nabla^2 f(x^* + te_k)) e_k dt. \end{aligned}$$

By Theorem A2.5 and the Lipschitz continuity of  $\nabla^2 f(x)$ , it follows that

$$\|e_{k+1}\| \leq \left(2 \|\nabla^2 f(x^*)^{-1}\| \right) \gamma \|e_k\|^2 / 2,$$

which completes the proof with  $K = \gamma \|\nabla^2 f(x^*)^{-1}\|$ . ◆

If  $\Delta(x_k)$  and  $\epsilon_g(x_k)$  are the errors in the Hessian and in the gradient, respectively, then the following estimation of the error  $e_k$  at the iteration  $k$  in the perturbed Newton method can be obtained (Kelley, 1999).

**Theorem 4.16** Suppose that  $f$  is twice continuously differentiable,  $\nabla f(x^*) = 0$ ,  $\nabla^2 f(x^*)$  is positive definite and  $\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq \gamma \|x - y\|$ . Then there are  $K > 0$ ,  $\delta > 0$  and  $\delta_1 > 0$  such that if  $x_k \in B(\delta)$  and  $\|\Delta(x_k)\| < \delta_1$ , then

$$x_{k+1} = x_k - (\nabla^2 f(x_k) + \Delta(x_k))^{-1} (\nabla f(x_k) + \varepsilon_g(x_k))$$

is well defined, that is,  $\nabla^2 f(x_k) + \Delta(x_k)$  is nonsingular and satisfies

$$\|e_{k+1}\| \leq K \|e_k\|^2 + 16 \left\| (\nabla^2 f(x^*))^{-1} \right\|^2 \|\nabla^2 f(x^*)\| \|\Delta(x_k)\| \|e_k\| + 4 \left\| (\nabla^2 f(x^*))^{-1} \right\| \|\varepsilon_g(x_k)\|, \quad (4.89)$$

where  $e_k = x_k - x^*$  and  $B(\delta)$  is the ball of radius  $\delta$  around  $x^*$ .

**Proof** Let  $\delta$  be small enough such that the conclusions of Theorem A2.5 hold. Define  $x_{k+1}^N = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$ . Observe that

$$x_{k+1} = x_{k+1}^N + \left( (\nabla^2 f(x_k))^{-1} - (\nabla^2 f(x_k) + \Delta(x_k))^{-1} \right) \nabla f(x_k) - (\nabla^2 f(x_k) + \Delta(x_k))^{-1} \varepsilon_g(x_k).$$

Theorem A2.5 implies that

$$\begin{aligned} \|e_{k+1}\| &\leq K \|e_k\|^2 + 2 \left\| (\nabla^2 f(x_k))^{-1} - (\nabla^2 f(x_k) + \Delta(x_k))^{-1} \right\| \|\nabla^2 f(x^*)\| \|e_k\| \\ &\quad + \left\| (\nabla^2 f(x_k) + \Delta(x_k))^{-1} \right\| \|\varepsilon_g(x_k)\|. \end{aligned} \quad (4.90)$$

If

$$\|\Delta(x_k)\| \leq \frac{1}{4} \left\| (\nabla^2 f(x^*))^{-1} \right\|^{-1},$$

then from Theorem A2.5 we get

$$\|\Delta(x_k)\| \leq \frac{1}{2} \left\| (\nabla^2 f(x_k))^{-1} \right\|^{-1}.$$

The Banach lemma states that  $\nabla^2 f(x_k) + \Delta(x_k)$  is nonsingular and

$$\left\| (\nabla^2 f(x_k) + \Delta(x_k))^{-1} \right\| \leq 2 \left\| (\nabla^2 f(x_k))^{-1} \right\| \leq 4 \left\| (\nabla^2 f(x^*))^{-1} \right\|.$$

Hence,

$$\left\| (\nabla^2 f(x_k))^{-1} - (\nabla^2 f(x_k) + \Delta(x_k))^{-1} \right\| \leq 8 \left\| (\nabla^2 f(x^*))^{-1} \right\|^2 \|\Delta(x_k)\|.$$

Therefore, using these estimates, from (4.90) the estimate of the error (4.89) is obtained, that is, the conclusion of the theorem.  $\blacklozenge$

The interpretation of (4.89) is as follows. Firstly, we can see that  $\|e_{k+1}\|$  is proportional to  $\|e_k\|^2$  and the factor of proportionality is independent of  $\|\Delta(x_k)\|$  and  $\varepsilon_g(x_k)$ . Observe that in the norm of the error  $e_{k+1}$ , given by (4.89) the inaccuracy evaluation of the Hessian given by  $\|\Delta(x_k)\|$  is multiplied by

the norm of the previous error. On the other hand, the inaccuracy evaluation of the gradient given by  $\varepsilon_g(x_k)$  is not multiplied by the previous error and has a direct influence on  $\|e_{k+1}\|$ . In other words, in the norm of the error the inaccuracy in the Hessian has smaller influence than the inaccuracy of the gradient. Therefore, in this context, from (4.89), the following remarks may be emphasized:

- If both  $\|\Delta(x_k)\|$  and  $\varepsilon_g(x_k)$  are zero, then we get the quadratic convergence of the Newton method.
- If  $\varepsilon_g(x_k) \neq 0$  and  $\|\varepsilon_g(x_k)\|$  is not convergent to zero, then there is no guarantee that the error for the Newton method will converge to zero.
- If  $\|\Delta(x_k)\| \neq 0$ , then the convergence of the Newton method is slowed down from quadratic to linear or to superlinear if  $\|\Delta(x_k)\| \rightarrow 0$ .

*Therefore, we can see that the inaccuracy evaluation of the Hessian of the minimizing function is not so important. The accuracy of the evaluation of the gradient is more important.* This is the main motivation why the diagonal quasi-Newton updating method may be used for unconstrained optimization (Andrei, 2019a, 2019b, 2020c, 2020d). (See Chap. 6, Sect. 6.3.)

## 4.9 Negative Curvature Direction Methods

As we know, the Newton method is quadratic convergent when the current point is in a neighborhood of the minimum point, and the Hessian matrix in the current point is positive definite. If the Hessian in the current point is an indefinite matrix and the current point is near enough to a saddle point, then the Newton method is not defined and has to be modified. If  $\nabla f(x_k) = 0$ , then there is no direction of descent, and if  $\nabla^2 f(x_k)$  is indefinite, then  $x_k$  is a saddle point. A vector  $d$  is termed a *direction of negative curvature* if  $d^T \nabla^2 f(x_k) d < 0$ .

In this section, the modified Newton method is based on the concept of negative curvature. At the beginning the *stable Newton method* of Gill and Murray (1972) is described, and after that two methods of negative curvature developed by Fiacco and McCormick (1968) and Fletcher and Freeman (1977), respectively, are presented. A definition is now given to clarify the concepts (Sun, & Yuan, 2006).

**Definition 4.1** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the open set  $D \subset \mathbb{R}^n$ .

- If  $\nabla^2 f(x)$  has at least a negative eigenvalue, then we say that  $x$  is an *indefinite point*.
- If  $x$  is an indefinite point and  $d$  satisfies the condition  $d^T \nabla^2 f(x) d < 0$ , then we say that  $d$  is a *negative curvature direction of  $f(x)$  in  $x$* .
- If  $s^T \nabla f(x) \leq 0$ ,  $d^T \nabla f(x) \leq 0$ ,  $d^T \nabla^2 f(x) d < 0$ , then we say that the pair  $(s, d)$  is a *descent pair in the indefinite point  $x$* .  
If  $x$  is not an indefinite point and satisfies  $s^T \nabla f(x) < 0$ ,  $d^T \nabla f(x) \leq 0$ , and  $d^T \nabla^2 f(x) d = 0$ , then we say that the pair  $(s, d)$  is a *descent pair in  $x$* . ◆

An example of descent pair is as follows:

$$s = -\nabla f(x),$$

$$d = \begin{cases} 0, & \nabla^2 f(x) \geq 0, \\ -\text{sign}(u^T \nabla f(x)) u, & \text{otherwise,} \end{cases}$$

where  $u$  is the unitary eigenvector corresponding to a negative eigenvalue of  $\nabla^2 f(x)$ .

From the above definition we can see that in a stationary point, a negative curvature direction has to be a descent direction. In an arbitrary general point, if the negative curvature direction satisfies  $d^T \nabla f(x) = 0$ , then both  $d$  and  $-d$  are descent directions. On the other hand, if  $d^T \nabla f(x) \leq 0$ , then  $d$  is a descent direction. If  $d^T \nabla f(x) \geq 0$ , then  $-d$  is a descent direction.

### The Stable Newton Negative Curvature Method of Gill and Murray

If the Hessian  $G_k$  is not defined, then the modified Cholesky factorization is used in order to force  $G_k$  to be positive definite. If  $x_k$  tends to a stationary point, then a negative curvature direction is used in order to reduce the values of the minimizing function.

Now, consider the modified Cholesky factorization

$$\bar{G}_k = G_k + E_k = L_k D_k L_k^T, \quad (4.91)$$

where

$$D_k = \text{diag}(d_{11}, \dots, d_{nn}) \text{ and } E_k = \text{diag}(e_{11}, \dots, e_{nn}). \quad (4.92)$$

With these, if  $\|g_k\| \leq \varepsilon$  and  $\nabla^2 f(x_k)$  is not positive semidefinite, then the following algorithm of negative curvature direction is used.

#### Algorithm 4.6 Negative curvature direction algorithm

1.	Compute $\varphi_j = d_{jj} - e_{jj}$ , $j = 1, \dots, n$
2.	Determine an index $t$ such that $\varphi_t = \min \{\varphi_j : j = 1, \dots, n\}$
3.	If $\varphi_t \geq 0$ , stop; otherwise, solve the linear system $L_k^T d_k = e_t$ subject to $d_k$ , where $e_t$ is the $t$ -th column of the unity matrix

**Theorem 4.17** Let  $G_k$  be the Hessian of function  $f$  in point  $x_k$  and

$$\bar{G}_k = G_k + E_k = L_k D_k L_k^T,$$

the modified Cholesky factorization. If the direction  $d_k$  is generated by Algorithm 4.6, then  $d_k$  is a negative curvature direction in  $x_k$  and at least one of the directions  $d_k$  and  $-d_k$  is a descent direction in  $x_k$ .

**Proof** Since  $L_k$  is a unitary inferior triangular matrix, then the solution of the system (4.93) has the following form

$$d_k = [\rho_1, \dots, \rho_{t-1}, 1, 0, \dots, 0]^T.$$

Then,

$$\begin{aligned} d_k^T G_k d_k &= d_k^T \bar{G}_k d_k - d_k^T E_k d_k = d_k^T L_k D_k L_k^T d_k - d_k^T E_k d_k \\ &= e_t^T D_k e_t - \left( \sum_{p=1}^{t-1} \rho_p^2 e_{pp} + e_{tt} \right) = d_{tt} - e_{tt} - \sum_{p=1}^{t-1} \rho_p^2 e_{pp} \\ &= \varphi_t - \sum_{p=1}^{t-1} \rho_p^2 e_{pp}. \end{aligned}$$

From the modified Cholesky factorization, we get

$$e_{jj} = \bar{g}_{jj} - g_{jj} = d_{jj} + \sum_{p=1}^{j-1} l_{jp}^2 d_p - g_{jj} = d_{jj} - c_{jj} \geq 0,$$

that is,

$$\sum_{p=1}^{t-1} \rho_p^2 e_{pp} \geq 0.$$

Since  $\varphi_t < 0$ , it follows that  $d_k^T G_k d_k < 0$ , that is,  $d_k$  is a negative curvature direction and  $-d_k$  is also a negative curvature direction. If  $g_k^T d_k \leq 0$ , then  $d_k$  is a descent direction; otherwise  $-d_k$  is a descent direction.  $\blacklozenge$

#### Algorithm 4.7 Stable Newton algorithm – Gill and Murray

1.	Consider an initial point $x_0$ as well as the convergence tolerance $\varepsilon > 0$ sufficiently small. Set $k = 1$
2.	Compute $g_k$ and $G_k$
3.	Compute the modified Cholesky factorization by using Algorithm 4.4 $G_k + E_k = L_k D_k L_k^T$
4.	If $\ g_k\  > \varepsilon$ , then solve the system $L_k D_k L_k^T d_k = -g_k$ subject to $d_k$ and continue with step 6; otherwise, go to step 5
5.	Perform the steps of Algorithm 4.6. If the direction $d_k$ may not be generated, that is, $\varphi_t \geq 0$ , then stop; otherwise, determine the direction $d_k$ and set
	$d_k = \begin{cases} -d_k, & g_k^T d_k > 0, \\ d_k, & g_k^T d_k \leq 0 \end{cases}$
6.	Compute the stepsize $\alpha_k$ and set $x_{k+1} = x_k + \alpha_k d_k$
7.	If $f(x_{k+1}) \geq f(x_k)$ , stop; otherwise, set $k = k + 1$ and go to step 2 $\blacklozenge$

The following theorem proves the convergence of the algorithm (Gill, & Murray, 1972).

**Theorem 4.18** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a twice continuously differentiable function on the open set  $D \subset \mathbb{R}^n$ . Suppose there is  $\bar{x} \in D$  such that the level set  $S(\bar{x}) = \{x : f(x) \leq f(\bar{x})\}$  is convex, bounded, and closed. Suppose that  $x_0 \in S(\bar{x})$ . Then the sequence  $\{x_k\}$  generated by Algorithm 4.7 in which  $\varepsilon = 0$  satisfies:

- (i) If  $\{x_k\}$  is a finite sequence, then the final point of this sequence is a stationary point of function  $f(x)$ .
- (ii) If  $\{x_k\}$  is an infinite sequence, then this sequence has accumulation points and all of them are stationary points of function  $f(x)$ .  $\blacklozenge$

#### The Negative Curvature Method of Fiacco and McCormick

Fiacco and McCormick (1968) presented a modification of the Newton method for the case in which the Hessian  $G_k$  has negative eigenvalues. Their idea was to move along the negative curvature direction. Indeed, if

$$d_k^T g_k \leq 0 \text{ and } d_k^T G_k d_k < 0, \quad (4.94)$$

then

$$f(x_k + d_k) \cong f(x_k) + d_k^T g_k + \frac{1}{2} d_k^T G_k d_k$$

will be reduced. Since  $G_k$  is indefinite, the Fiacco-McCormick method uses the factorization

$$G_k = LDL^T, \quad (4.95)$$

where  $L$  is an inferior, unitary triangular matrix and  $D$  is a diagonal matrix. If  $G_k$  is positive definite, then the direction  $d_k$  generated by the factorization (4.95) is descent. However, if there is a diagonal element  $d_{ii}$  negative, then the following system is solved:

$$L^T t = a,$$

where the component  $a_i$  of the vector  $a$  is defined as

$$a_i = \begin{cases} 1, & d_{ii} \leq 0, \\ 0, & d_{ii} > 0. \end{cases}$$

Obviously, the search direction

$$d_k = \begin{cases} t, & g_k^T t \leq 0, \\ -t, & g_k^T t > 0 \end{cases}$$

is a negative curvature direction which satisfies (4.94).

The criticism of this very simple method lies in its numerical instability. The factorization (4.95) can amplify the errors or it may not exist. This is the reason why Fletcher and Freeman (1977) introduced a stable symmetric factorization, which we are going to present.

### The Negative Curvature Method of Fletcher and Freeman

Fletcher and Freeman used a symmetric, stable factorization due to Bunch and Parlett (1971). Indeed, for any symmetric matrix  $G_k$ , there is a permutation matrix  $P$  such that

$$P^T G_k P = LDL^T, \quad (4.96)$$

where  $L$  is an inferior, unitary triangular matrix and  $D$  is a block-diagonal matrix with  $1 \times 1$  or  $2 \times 2$ -dimensional blocks. The purpose of introducing the permutation matrix  $P$  is to conserve the symmetry and to maintain the numerical stability of the factorization. It is worth mentioning that the factorization (4.96) always exists and is numerically stable. This is the difference between this method and the method of Fiacco and McCormick, which uses the numerically unstable factorization (4.95).

In the case in which the pivot is an element  $1 \times 1$ -dimensional, then the matrix  $A$ ,  $n \times n$ -dimensional can be expressed as

$$A = A^{(0)} = \begin{bmatrix} a_{11} & \bar{a}_{21}^T \\ \bar{a}_{21} & A_{22} \end{bmatrix},$$

where  $\bar{a}_{21}$  is a vector  $(n - 1)$ -dimensional and  $A_{22}$  is a  $(n - 1) \times (n - 1)$  matrix. Now, by pivoting for eliminating the first row and the first column, the following matrix  $A^{(1)}$  is obtained:

$$A^{(1)} = A^{(0)} - d_{11}l_1l_1^T = \begin{bmatrix} 0 & 0^T \\ 0 & A_{22} - \bar{a}_{21}\bar{a}_{21}^T/d_{11} \end{bmatrix}, \quad (4.97)$$

where

$$d_{11} = a_{11}, l_1 = \frac{1}{d_{11}} \begin{bmatrix} a_{11} \\ \bar{a}_{21} \end{bmatrix} = \begin{bmatrix} 1 \\ \bar{a}_{21}/d_{11} \end{bmatrix}.$$

In the case in which the pivot is a  $2 \times 2$ -dimensional matrix, then

$$A^{(0)} = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix},$$

where this time  $A_{11}$  is a  $2 \times 2$ -dimensional matrix,  $A_{21}$  is  $(n-2) \times 2$ -dimensional, and  $A_{22}$  is a  $(n-2) \times (n-2)$  sub-matrix. By pivoting, the following  $A^{(2)}$  matrix is obtained:

$$\begin{aligned} A^{(2)} &= A^{(0)} - L_1 D_1 L_1^T = A^{(0)} - \begin{bmatrix} I \\ L_{21} \end{bmatrix} D_1 [I \quad L_{21}^T] \\ &= \begin{bmatrix} 0 & 0 \\ 0 & A_{22} - A_{21} D_1^{-1} A_{21}^T \end{bmatrix}, \end{aligned} \quad (4.98)$$

where

$$D_1 = A_{11}, L_1 = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} D_1^{-1} = \begin{bmatrix} I \\ A_{21} A_{11}^{-1} \end{bmatrix} \triangleq \begin{bmatrix} I \\ L_{21} \end{bmatrix}.$$

At the next step, the same procedure is performed on the  $(n-1) \times (n-1)$ -dimensional sub-matrix  $A_{22} - \bar{a}_{21}\bar{a}_{21}^T/d_{11}$  from (4.97) or on the  $(n-2) \times (n-2)$ -dimensional sub-matrix  $A_{22} - A_{21}D_1^{-1}A_{21}^T$  from (4.98). Finally, the factorization (4.96) is obtained.

The problem we face now is that of selecting the pivots, which could be a scalar or a  $2 \times 2$ -dimensional submatrix. For this, Bunch and Parlett suggested the following procedure. At first determine the maximum diagonal element in the absolute value  $\xi_d$  and the maximum non-diagonal element in the absolute value  $\xi_n$ , respectively. If the ratio  $\xi_d/\xi_n$  is acceptable, then select the maximum diagonal element in the absolute value  $\xi_d$  as pivot and perform a symmetric permutation of the rows and columns such that this element is placed on the position of  $a_{11}$ . Otherwise, select the maximum non-diagonal element in the absolute value  $\xi_n$ , let us say the element  $a_{ij}$ , and select as pivot the submatrix

$$\begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix},$$

followed by a permutation of the rows and columns in such a way that  $A_{11}$  is exactly this sub-matrix.

This factorization generates the matriceal factors  $L$  and  $D$  as an inferior, unitary triangular matrix and a block-diagonal matrix, respectively. For example, for the case in which there are 2 blocks  $1 \times 1$  and  $2 \times 2$ , then the structure of these matriceal factors is

$$L = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & 0 & 1 & & \\ & * & * & 1 & \\ & * & * & 0 & 1 \\ & * & * & * & * & 1 \end{bmatrix}, D = \begin{bmatrix} * & & & & \\ & * & * & & \\ & * & * & & \\ & & & * & * \\ & & & * & * \\ & & & & * \end{bmatrix}$$

A similar, more economical factorization was given by Bunch and Kaufman (1977).

Therefore, let the following factorization of the symmetric indefinite matrix  $G_k$  be as  $G_k = LDL^T$ . With this, solve the triangular system

$$L^T t = a, \quad (4.99)$$

where, in the case of a  $1 \times 1$  dimensional pivot, the components of  $a$  are

$$a_i = \begin{cases} 1, & d_{ii} \leq 0, \\ 0, & d_{ii} > 0; \end{cases}$$

and in the case of a  $2 \times 2$  dimensional pivot,  $[a_i \ a_{i+1}]^T$  is the unitary eigenvector corresponding to the negative eigenvalue of the sub-matrix

$$\begin{bmatrix} d_{ii} & d_{i,i+1} \\ d_{i+1,i} & d_{i+1,i+1} \end{bmatrix}.$$

With the solution of the system (4.99), consider

$$d_k = \begin{cases} t, & g_k^T t \leq 0, \\ -t, & g_k^T t > 0, \end{cases} \quad (4.100)$$

which is a negative curvature direction satisfying (4.94).

Observe that

$$d_k^T G_k d_k = d_k^T LDL^T d_k = a^T Da = \sum_{i: \lambda_i < 0} \lambda_i,$$

and

$$g_k^T d_k \leq 0.$$

If  $D$  has negative eigenvalue, then the direction  $d_k$  is computed as

$$d_k = -L^{-T} \tilde{D}^+ L^{-1} g_k, \quad (4.101)$$

where  $\tilde{D}$  is the positive part of  $D$ , that is,

$$\tilde{D}_i = \begin{cases} d_{ii}, & d_{ii} > 0, \\ 0, & \text{otherwise,} \end{cases}$$

and  $\tilde{D}^+$  is the generalized inverse of  $\tilde{D}$ .

If  $D$  has at least a zero eigenvalue, then the direction  $d_k$  is computed such that

$$G_k d_k = LDL^T d_k = 0 \text{ and } g_k^T d_k < 0. \quad (4.102)$$

If all the eigenvalues of  $D$  are positive, then all the blocks are  $1 \times 1$ -dimensional elements and in this case the generated direction is exactly the usual Newton direction

$$d_k = -L^{-T} D^{-1} L^{-1} g_k.$$

Note that the negative curvature descent direction (4.100) is limited to a certain subspace. On the other hand, the direction given by (4.101) is exactly the Newton direction limited to the subspace of the positive curvature directions.

Even if the negative curvature directions have some attractive properties, they are not efficient, particularly in situations when they are used at successive iterations. Although no solid theoretical justification is known for an alternate usage of the directions given by (4.100) or by (4.101), respectively, in practice this alternation gives better results. In the same experimental manner, if  $D$  has eigenvalues zero, then the results obtained are better if the direction is alternately computed by (4.101) or by (4.102), respectively.

What is important to notice is the Bunch-Parlett factorization of the symmetric indefinite matrices. This factorization is not only found in the context of the Newton method, but in many interior point methods from linear programming (Andrei, 2011d) or in the constrained optimization. Even if these modifications of the Newton method have theoretical justification, they are not used in solving real application. Some other unconstrained optimization methods are more efficient and more robust (conjugate gradient, quasi-Newton, truncated Newton, limited memory BFGS, etc.).

## 4.10 The Composite Newton Method

Consider the nonlinear algebraic system  $F(x) = 0$ , where  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is continuous differentiable. Then the Newton method with line-search for solving this system is defined as

$$J(x_k) d_k = -F(x_k), \quad (4.103)$$

$$x_{k+1} = x_k + \alpha_k d_k, k = 0, 1, \dots,$$

where  $\alpha_k$  is the stepsize. Choosing  $\alpha_k$  less than one is important for the global convergence. As we know, under standard assumptions the Newton method is quadratic convergent. Since the dominant work of this method is the factorization of the  $n \times n$ -dimensional Jacobian matrix, then without taking into account the work needed to evaluate the function  $F$  and its Jacobian  $J$ , the algebra required per iteration is  $O(n^3)$ . If  $n$  is large, the factorization of the Jacobian matrix can be a very serious provocation.

In order to reduce the computational effort at each iteration, the *simplified Newton method* can be defined as

$$J(x_0) d_k = -F(x_k), \quad (4.104)$$

$$x_{k+1} = x_k + \alpha_k d_k, k = 0, 1, \dots$$

The simplified Newton method requires the factorization of  $J(x_0)$  and its usage along *all* the iterations. In this case, the computational effort is of order  $O(n^2)$ . Obviously, the convergence is only linear, thus forcing the method to perform a larger number of iterations.

Between these two possibilities given by the Newton method (4.103) and the simplified Newton method (4.104), we can imagine another method, in which a number of  $m$  simplified Newton iterations is performed between any Newton iterations. In other words, for these  $m$  iterations, the same factorization of the Jacobian is used. This is known as the *composite Newton method of order  $m$* , defined as

**Algorithm 4.8** *Composite Newton of order  $m$*

1.	Solve the systems $\begin{aligned} J(x_k)d_1 &= -F(x_k), \\ J(x_k)d_2 &= -F(x_k + d_1), \\ &\vdots \\ J(x_k)d_m &= -F(x_k + d_1 + \cdots + d_{m-1}) \end{aligned}$
2.	Compute $x_{k+1} = x_k + \alpha_k(d_1 + d_2 + \cdots + d_m)$ <span style="float: right;">◆</span>

We can see at once that exactly  $m$  linear algebraic systems with the same matrix  $J(x_k)$  have to be solved at the  $k$ -th iteration. The average computational effort on iteration is of order  $O((n^3 + mn^2)/m)$ , which for  $m$  large is of order  $O(n^2)$ . Therefore, for  $m$  large, the behavior of the composite Newton method of order  $m$  is like the simplified Newton method. Consequently, advanced implementations of the composite Newton method of order  $m$  not only modify the parameter  $m$  at every Newton iteration, but would also keep  $m$  relatively as small as possible. Under the standard Newton method assumptions, the composite Newton method of order  $m$  has a convergence rate of  $m + 2$ . A proof can be found in Ortega and Rheinboldt (2000).

A *composite Newton method of order 1* is of particular interest, where one Newton iteration is composed with one simplified Newton iteration (Traub, 1964). This is defined as

**Algorithm 4.9** *Composite Newton of order 1*

1.	Solve the system subject to the Newton direction $d_N$ $J(x_k)d_N = -F(x_k),$
2.	Solve the system subject to the simplified Newton direction $d_S$ $J(x_k)d_S = -F(x_k + d_N),$
3.	Compute $x_{k+1} = x_k + \alpha_k(d_N + d_S)$ <span style="float: right;">◆</span>

The following theorem shows the cubic convergence of the composite Newton method of order 1 (Ortega, & Rheinboldt, 2000).

**Theorem 4.19** *Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a continuously differentiable function on  $B(x^*, \delta) \subset \mathbb{R}^n$ , for which the Jacobian is Lipschitz continuous with the constant  $L$ , that is, for any  $x \in B(x^*, \delta)$ ,  $\|J(x) - J(x^*)\| \leq L\|x - x^*\|$ . Suppose that  $J(x^*)$  is nonsingular. Then the composite Newton method of order 1 is cubic convergent.*

**Proof** From the Theorem 4.1 observe that the function  $N(x) = x - J(x)^{-1}F(x)$ , which defines the Newton method, is well defined and for any  $x \in B(x^*, \delta_1) \subset B(x^*, \delta)$  satisfies

$$\|N(x) - x^*\| \leq \eta \|x - x^*\|^2, \quad (4.105)$$

where  $\delta_1 < \delta$  and  $\eta$  is a positive constant. Therefore, the function

$$G(x) = N(x) - J(x)^{-1}F(N(x)) \quad (4.106)$$

is also well defined on  $B(x^*, \delta_2)$ , where  $\delta_2 \leq (\delta_1/\eta)^{1/2}$ . Hence, if

$$\|J(x)^{-1}\| \leq \beta \quad (4.107)$$

for all  $x \in B(x^*, \delta_2)$ , then by using the above relations (4.105), (4.106), and (4.107), it follows that

$$\begin{aligned} \|G(x) - x^*\| &= \|N(x) - J(x)^{-1}F(N(x)) - x^*\| \\ &= \|J(x)^{-1}[J(x)N(x) - F(N(x)) - J(x)x^*]\| \\ &\leq \|J(x)^{-1}\| \|J(x)N(x) - F(N(x)) - J(x)x^*\| \\ &\leq \beta \|J(x)N(x) - F(N(x)) - J(x)x^* + F(x^*)\| \\ &\quad + \|J(x^*)N(x) - J(x^*)x^* - J(x^*)N(x) + J(x^*)x^*\| \\ &\leq \beta \|F(N(x)) - F(x^*) - J(x^*)(N(x) - x^*)\| \\ &\quad + \beta \|J(x^*) - J(x)\| \|N(x) - x^*\| \\ &\leq \frac{1}{2} \beta L \|N(x) - x^*\|^2 + \beta L \|x - x^*\| \|N(x) - x^*\| \\ &\leq \frac{1}{2} \beta L \eta^2 \|x - x^*\|^4 + \beta L \eta \|x - x^*\|^3 \\ &= \beta L \eta \left[ \frac{1}{2} \eta \delta_2 + 1 \right] \|x - x^*\|^3 \end{aligned}$$

for all  $x \in B(x^*, \delta_2)$ , thus proving the cubic convergence of the composite Newton method of order 1.  $\blacklozenge$

In current practice it is credited that the composite Newton method is efficient when  $n$  is large and the function  $F$  is easy to evaluate. This is the case of the interior point methods for solving constrained optimization problems (including the linear programming) in which the advanced implementations use the composite Newton method (Mehrotra, 1989), (Zhang, Tapia, & Dennis, 1992), (Andrei, 1999a, 1999b).

In the following, let us present the performances of the composite Newton method of order 1 for solving some applications from the SMUNO collection (*Stationary solution of a chemical reactor*, *Robot kinematics problem*, *Propan combustion in air – reduced variant* and *Circuit design*). These applications were solved by the Newton method (see Tables 4.3, 4.4, 4.5, and 4.6)

### Applications Solved by the Composite Newton Method of Order 1

We now describe the running and the performances of the *composite Newton method of order 1* for solving some applications from the SMUNO collection (see Appendix B). All these applications considered in this numerical study are expressed as nonlinear algebraic systems.

**Application S5** (*Stationary solution of a chemical reactor*) Application S5 presented in the SMUNO collection was solved by the Newton method in Sect. 4.1 (see Table 4.3). Table 4.14 contains the initial point  $x_0$ , the functions values in the initial point  $F(x_0)$ , the solution  $x^*$  of this application given by the composite Newton method of order 1 with  $\epsilon = 10^{-6}$  in 4 iterations, and the functions values in  $x^*$ .

**Table 4.14** Stationary solution of a chemical reactor. Initial point, solution, functions values in these points. Composite Newton method of order 1. 4 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	1.09	-35.164864	0.97424361	0.0
2	1.05	-0.2403040	0.98282907	-0.622e-17
3	3.05	3626.94637	0.05151276	0.0
4	0.99	-1779.9233	0.93567106	-0.693e-17
5	6.05	-1814.7127	0.00009083	0.681e-12
6	1.09	-7.13	0.06423809	-0.416e-16

**Table 4.15** Robot kinematics problem. Initial point, solution, functions values in these points. Composite Newton method of order 1. 2 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	0.164	0.0026823	0.16443166	0.111e-15
2	-0.98	-0.0098180	-0.98638847	0.0
3	-0.94	0.1050028	-0.94706369	0.260e-17
4	-0.32	0.0017588	-0.32104573	0.555e-16
5	-0.99	-0.0127040	-0.99823316	0.0
6	-0.05	-0.014	0.05941842	0.0
7	0.41	-0.016764	0.41103315	0.110e-011
8	-0.91	-0.0038	-0.91162039	0.0

**Application S6** (*Robot kinematics problem*) Application S6 described in the SMUNO collection was solved by the Newton method (see Table 4.4). Table 4.15 presents the performances of the composite Newton method of order 1 with  $\varepsilon = 10^{-6}$  for solving this application.

**Application S9** (*Propan combustion in air – reduced variant*) This application from the SMUNO collection was solved by the Newton method, and the performances of this method were presented in Table 4.5. Table 4.16 shows the performances of the composite Newton method of order 1 with  $\varepsilon = 10^{-6}$  for solving this application.

**Application S14** (*Circuit design*) The performances of the Newton method for this application taken from the SMUNO collection were presented in Table 4.6. Table 4.17 shows the performances of the composite Newton method of order 1 with  $\varepsilon = 10^{-6}$ .

By comparing Tables 4.3, 4.4, 4.5, and 4.6 and Tables 4.10, 4.11, 4.12, and 4.13, we can see that the composite Newton method of order 1 is more efficient than the Newton method.

**Remark 4.1** Observe that from the local convergence point of view, the Newton method is exactly the same as the one for nonlinear equations applied to the problem of finding a root of  $\nabla f$ . Therefore, the positive definiteness of  $\nabla^2 f$  can be used in an implementation of the Newton method based on the

**Table 4.16** Propan combustion in air – reduced variant. Initial point, solution, functions values in these points. Composite Newton method of order 1. 7 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	10	109.85000	0.002757177	0.664e-10
2	10	209.54267	39.24228904	0.221e-09
3	0,05	-0.3487418	-0.061387604	0.177e-09
4	50,5	5098.5172	0.859724425	0.885e-09
5	0,05	2659.2930	0.036985043	-0.111e-15

**Table 4.17** Circuit design. Initial point, solution, functions values in these points. Composite Newton method of order 1. 3 iterations.  $\varepsilon = 10^{-6}$

Nr.	$x_0$	$F(x_0)$	$x^*$	$F(x^*)$
1	0.7	0.95525047	0.89999995	-0.285e-09
2	0.5	3.04155543	0.44998747	-0.169e-08
3	0.9	-4.16621316	1.00000648	-0.143e-08
4	1.9	-2.11846214	2.00006854	-0.313e-08
5	8.1	5.636125856	7.99997144	-0.840e-09
6	8.1	22.31397917	7.99969268	-0.365e-08
7	5.9	24.95943321	5.00003127	-0.401e-08
8	1	42.18784331	0.99998772	-0.687e-08
9	1.9	-0.32000000	2.00005248	0.110e-09

Cholesky factorization  $\nabla^2 f = LL^T$  where  $L$  is lower triangular and has positive diagonal. The iterations are terminated when  $\nabla f$  is sufficiently small. A natural criterion for a relative decrease in  $\|\nabla f\|$  and the termination of the iterations is

$$\|\nabla f(x_k)\| \leq \tau_r \|\nabla f(x_0)\|, \quad (4.108)$$

where  $\tau_r$  is the desired reduction in the gradient norm. However, if  $\|\nabla f(x_0)\|$  is very small, it may not satisfy the above criterion (4.108) in the floating point arithmetic and an algorithm based only on (4.108) might never terminate. A standard remedy is to augment the relative error criterion and terminate the iteration by using a combination of the relative and absolute measures of  $\nabla f$  i.e., when

$$\|\nabla f(x_k)\| \leq \tau_r \|\nabla f(x_0)\| + \tau_a, \quad (4.109)$$

where  $\tau_a$  is an absolute error tolerance. Another practical procedure often implemented in Newton's method is the modified Cholesky factorization. A *modified Cholesky factorization* of a symmetric matrix  $A$  is a factorization  $P(A + E)P^T = LDL^T$  where  $P$  is a permutation matrix,  $L$  is unit lower triangular, and  $D$  is diagonal or block diagonal and positive definite matrix. It follows that  $A + E$  is a positive definite matrix. This corresponds to making a diagonal perturbation  $E$  to  $A$  and to computing a Cholesky factorization of the modified matrix  $A+E$  (see Appendix A).

## Notes and References

Regarding the history of the Newton method, we can first mention Heron of Alexandria (10–70 AD). He described a method (called the *Babylonian method*) to iteratively approximate a square root. François Viète (1540–1603) developed a method to approximate roots of polynomials. In 1669 Sir Isaac Newton (1643–1727) improved the Viète's method in a paper published in 1711. A simplified version of Newton's method was published by Joseph Raphson (1648–1715) in 1690. Further information about Raphson can be found in a paper of Bicanic and Johnson (1979). The modern treatment and presentation of the Newton method is due to Thomas Simpson (1710–1761). (See (Kollerstrom, 1992).) A more detailed history of this method can be found in a paper by Ypma (1995).

The first sufficient conditions for the monotone convergence of the Newton method were given by Joseph Fourier (1768–1830) in 1818. His result was reconsidered in modern terms by Gaston Darboux (1842–1917) in 1869. In 1952 Baluev extended the theorem of Fourier-Darboux to nonlinear operators in partial ordered topological spaces. The conditions of Baluev's theorem are relatively powerful and difficult to verify in practice. Moreover, there are simple examples for which Baluev's result does not apply, even if the Newton method converges monotonically for these examples. Starting with 1937, Ovstrowski (1893–1986) published a number of papers in which the local convergence of the Newton method is proved for functions of a real variable. The proof that the Newton method is indeed monotonously convergent in more general conditions than those of Baluev was given by Potra and Rheinboldt (1986). However, the proof of the quadratic convergence of the Newton method in general Banach spaces was established by Kantorovich in 1948. The result of Kantorovich is fundamental in the sense that, in addition to the fact that he established a very ingenious and elegant technique of proving, he introduced the fundamental problems associated to any iterative method: *convergence*, *ordin of convergence*, and *estimation of error*, problems which have become classical and continued to dominate the numerical analysis up to the present.

Newton's method is only locally convergent. Moreover, as in the most cases of algorithms that are only locally convergent, it is usually not known apriori whether an initial estimate is indeed close enough to the solution. Ideal algorithms should be globally convergent (converge from an arbitrary initial point). However, the proofs of global convergence do need assumptions about the properties of the problem that may not be verifiable for specific problems.

The theoretical developments of this chapter are based on the books of Sun and Yuan (2006) and Kelley (1999). The most important is Theorem 4.1 which shows that in some specific conditions (the Jacobian in the minimum point is nonsingular and in a neighborhood of the minimum point the Jacobian is Lipschitz continuous), the Newton method for solving nonlinear algebraic systems is quadratic convergent. For the functions minimization, Theorem 4.3 (based on the Lipschitz continuity of the Hessian) and Theorem 4.4 (based on the bounds of the Hessian) are the most important ones, showing the quadratic convergence of the Newton method. All these results are based on the assumption that the initial point is near enough to the minimum point.

The analysis of the complexity of the Newton method with backtracking gives only a conceptual result on the bound of the number of iterations. Initialized in a point near enough to the minimum point, the Newton method with backtracking accepts the stepsize  $\alpha_k = 1$ . The difficulty is given by the requirement to evaluate the Hessian matrix of the minimizing function and by the solving, using the factorization of the Newton system.

The modifications of the Newton method may be classified in two classes: computation of the entries of the Hessian by finite-difference and modifications of the Hessian when this matrix is indefinite and the current iterations are near a saddle point. All these modifications try to ameliorate the behavior of the method. The composite Newton method of order 1 is a proper modification which substantially improves the rate of convergence of the Newton method.



# Conjugate Gradient Methods

# 5

These methods are characterized by very strong convergence properties and modest storage requirements. They are dedicated to solving large-scale unconstrained optimization problems and applications. The history of these methods starts with the researches of Cornelius Lanczos (1950, 1952), Magnus Hestenes (1951, 1955, 1956a, 1956b), Rosser (1953), Forsythe, Hestenes and Rosser (1951), (Stein), and others from the *Institute for Numerical Analysis – National Bureau of Standards*, Los Angeles, as well as with the researches of Eduard Stiefel (1958) from *Eidgenössische Technische Hochschule Zürich*. During over 70 years of researches in this area, an impressive number of developments, variants, and algorithms of these methods have appeared. A thorough and detailed presentation of these methods was given by Andrei (2020a). The search direction of these methods is a linear combination of the negative gradient and the previous search direction. The conjugate gradient methods require only the first order derivatives. As we will see, these methods may include the second order information given by an approximation of the Hessian of the minimizing function, thus increasing their convergence properties.

In this chapter we will discuss the linear and nonlinear conjugate gradient methods by presenting their derivation, the convergence properties, and their performances for solving unconstrained optimization problems and real optimization applications.

---

## 5.1 The Concept of Nonlinear Conjugate Gradient

Let us consider the quadratic function

$$f(x) = \frac{1}{2}x^T Ax - b^T x, \quad (5.1)$$

where  $A \in \mathbb{R}^{n \times n}$  is a symmetric and positive definite matrix and  $b \in \mathbb{R}^n$  is a known vector. From (5.1) we get

$$\nabla f(x) = Ax - b, \quad \nabla^2 f(x) = A. \quad (5.2)$$

Observe that the Hessian of function (5.1) is independent of  $x$ . Since the Hessian  $A$  is symmetric and positive definite, from the optimality conditions for a minimum of a differentiable function, it follows that there is a unique minimizer  $x^*$  of (5.1). From (5.2) observe that  $x^*$  is the solution of the linear system  $Ax = b$ .

The idea behind the conjugate gradient approach for minimizing quadratic functions (5.1) is to use search directions which do not interfere with one another. Given a symmetric and positive definite matrix  $A$ , the set of vectors  $\{d_0, \dots, d_{k-1}\}$  is a set of *conjugate directions* (or “*A conjugate*,” or even “*A orthogonal*”) if and only if  $d_i^T A d_j = 0$  for all  $i \neq j$ , where  $i = 0, \dots, k-1$  and  $j = 0, \dots, k-1$ . Note that  $d_i^T A d_i > 0$  for all  $i$ , since  $A$  is positive definite.

In the economy of a conjugate gradient algorithm, the conjugacy condition is an important ingredient. The main characteristic of conjugate directions is the minimization of a convex quadratic function in a subspace spanned by a set of mutually conjugate directions is equivalent to the minimization of the function along each conjugate direction in turn. However, the performances of algorithms satisfying the conjugacy condition are strongly dependent on the accuracy of the line-search. If the line-search procedure for the stepsize computation is highly accurate, then the convergence of the corresponding optimization algorithm is fast.

*In the conjugate gradient algorithm, the search direction  $d_{k+1}$  is computed as a linear combination*

$$d_{k+1} = -g_{k+1} + \beta_k d_k \quad (5.3)$$

*of the current negative gradient and the last search direction in such a way that  $d_{k+1}$  is A conjugate to all the previous search directions.*

For quadratic functions, the conjugate gradient parameter  $\beta_k$  is chosen so that  $d_{k+1}^T A d_k = 0$ , which determines

$$\beta_k = \frac{g_{k+1}^T A d_k}{d_k^T A d_k}. \quad (5.4)$$

The next approximation to the minimum is  $x_{k+1}$ . This is the unique minimum of  $f$  along the line  $l(\alpha) = x_k + \alpha d_k$ , which is given by

$$x_{k+1} = x_k + \alpha_k d_k, \quad (5.5)$$

where

$$\alpha_k = -\frac{d_k^T g_k}{d_k^T A d_k}. \quad (5.6)$$

This is the *traditional conjugate gradient algorithm* which works for quadratic functions, since it explicitly uses the matrix  $A$ , both in (5.4) and (5.6).

Now, the problem is *how the algorithm could be modified so that the quadratic nature of the minimizing function f should not explicitly appear in the algorithm*, i.e., in (5.4) and (5.6), and remains unchanged iff  $f$  is a quadratic function. As we know, the initial search direction is  $d_0 = -g_0$ . Observe that the initial search direction does not involve the matrix  $A$ . Therefore, for nonlinear functions, the initial search direction could be very well considered as the gradient of the minimizing function at the initial point. As we can see, in this context two crucial elements have to be computed: the stepsize  $\alpha_k$  and the parameter  $\beta_k$ . To determine  $\alpha_k$ , a line-search from  $x_k$  in the direction  $d_k$  is executed. For the convex quadratic functions, this line-search determines an explicit stepsize (5.6), which is the unique minimum of function  $f$  in the direction  $d_k$ . This is the advantage of quadratic functions: there is an explicit formula for the stepsize computation. For nonlinear functions, this formula for the stepsize computation can be replaced by a general line-search procedure (Armijo, Goldstein, Wolfe, etc.). This will change nothing in the quadratic case, but will generalize the algorithm to nonlinear functions. For the computation of the conjugate gradient parameter  $\beta_k$ , from

(5.4) we need to compute  $Ad_k$ . There are several possibilities to modify the algorithm in order to eliminate the explicit mention of matrix  $A$  in (5.4).

One way is to see from (5.5) that  $x_{k+1} - x_k = \alpha_k d_k$ . Therefore, for quadratic functions,

$$g_{k+1} - g_k = (Ax_{k+1} - b) - (Ax_k - b) = \alpha_k Ad_k.$$

With this, since  $Ad_k = (g_{k+1} - g_k)/\alpha_k$ , from (5.4) it follows that

$$\beta_k = \frac{g_{k+1}^T y_k}{d_k^T y_k}, \quad (5.7)$$

where  $y_k = g_{k+1} - g_k$ .

If  $f$  is a quadratic function, then the definitions (5.4) and (5.7) are equivalent, but the new definition (5.7) can be used for *any* differential function. Observe that (5.7) is *exactly the Hestenes and Stiefel formula* (Hestenes, & Stiefel, 1952) for the conjugate gradient parameter computation.

Another way to eliminate  $A$  from (5.4) is as follows. Suppose that  $f$  is quadratic. From (5.7) it follows that  $d_k^T y_k = d_k^T g_{k+1} - d_k^T g_k$ . But,  $d_k^T g_{k+1} = 0$ . Therefore,  $d_k^T y_k = -d_k^T g_k$ . Now, observe that  $d_k = -g_k + \beta_{k-1} d_{k-1}$ , so that, since  $d_{k-1}^T g_k = 0$ , it results that

$$d_k^T y_k = -(-g_k^T + \beta_{k-1} d_{k-1}^T) g_k = g_k^T g_k.$$

Therefore, from (5.7) another formula for  $\beta_k$  computation is

$$\beta_k = \frac{g_{k+1}^T y_k}{g_k^T g_k}. \quad (5.8)$$

Note that if  $f$  is quadratic, (5.8) is equivalent to (5.4). Hence, (5.8) generalizes the computation of the conjugate gradient parameter to the non-quadratic case. Formula (5.8) is *exactly the Polak-Ribière-Polyak formula* (Polak, & Ribiére, 1969; Polyak, 1969) for the conjugate gradient parameter computation.

Furthermore, it is very easy to see that if  $f$  is a quadratic function, then  $g_{k+1}^T g_k = 0$ , and in this case, from (5.8),

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}, \quad (5.9)$$

which generalizes the computation of the conjugate gradient parameter to the non-quadratic case. Observe that (5.9) is *exactly the Fletcher-Reeves formula* (Fletcher, & Reeves, 1964) for the conjugate gradient parameter computation.

Formulae (5.7), (5.8), and (5.9) generalize the conjugate gradient algorithm to *any* nonlinear differential function. They require only the gradient evaluations. For quadratic functions they are equivalent, but they are not equivalent for nonlinear functions.

For general nonlinear functions, the matrix  $A$  should be replaced by the Hessian. Since  $y_k = g_{k+1} - g_k$ , from the mean value theorem it follows that

$$d_{k+1}^T y_k = \alpha_k d_{k+1}^T \nabla^2 f(x_k + \tau \alpha_k d_k) d_k, \quad (5.10)$$

where  $\tau \in (0, 1)$ . Therefore, from (5.10), for nonlinear functions it is reasonable to consider the conjugacy condition as

$$d_{k+1}^T y_k = 0, \quad (5.11)$$

or as  $d_{k+1}^T y_k = t(g_{k+1}^T s_k)$ , where  $t > 0$  is a parameter and  $s_k = x_{k+1} - x_k$ , (Dai, & Liao, 2001). For example, the Hestenes-Stiefel method has the property that the conjugacy condition (5.11) always holds, independent of the line-search.

In general, not all the nonlinear conjugate gradient algorithms satisfy the conjugacy condition (5.11). However, what is characteristic of nonlinear conjugate gradient algorithms is that the search direction is computed as a linear combination between the negative gradient and the last search direction, where the conjugate gradient parameter  $\beta_k$  is determined in such a way that the descent condition or the sufficient descent condition holds. Of course, as it is to be seen, to improve the convergence of the algorithms, this linear combination between the negative gradient and the last search direction can be modified in such a way as to take into account, for example, the second order information of the minimizing function, or some other ingredients which accelerate the convergence. The descent or the sufficient descent conditions are crucial in the conjugate gradient methods. The descent condition  $d_k^T g_k < 0$ , (if  $g_k \neq 0$ ) is more general and leads to general convergence results for line-search algorithms under the exact or inexact line-search (and not only for conjugate gradient algorithms). The sufficient descent condition  $d_k^T g_k \leq -c \|g_k\|^2$ , for some positive constant  $c$ , is fundamental in conjugate gradient methods.

## 5.2 The Linear Conjugate Gradient Method

The linear conjugate gradient algorithm is dedicated to minimizing the convex quadratic functions (or to solving linear algebraic systems of equations with positive definite matrices). This algorithm was introduced by Hestenes and Stiefel (1952). Let us consider the quadratic function (5.1) where  $A \in \mathbb{R}^{n \times n}$  is a symmetric and positive definite matrix and  $b \in \mathbb{R}^n$  is a known vector. From (5.1) we get  $\nabla f(x) = Ax - b$  and  $\nabla^2 f(x) = A$ . Observe that the Hessian of function  $f$  is independent of  $x$ . Since the Hessian  $A$  is symmetric and positive definite, from the optimality conditions for a minimum of a differentiable function, it follows that there is a unique minimizer  $x^*$  of  $f$ . From (5.2) observe that  $x^*$  is the solution of the linear system  $Ax = b$ . Having in view that function  $f$  is quadratic, from the Taylor theorem, for all  $t \in \mathbb{R}$  and all  $y, z \in \mathbb{R}^n$  the following identity is obtained:

$$f(y + tz) = f(y) + t \nabla f(y)^T z + \frac{t^2}{2} z^T A z. \quad (5.12)$$

The linear conjugate gradient algorithm is exactly a line-search with a special choice of directions. Given the current approximation  $x_j$  to the minimum  $x^*$  as well as a direction vector  $d_j$ , a line-search algorithm computes the next approximation  $x_{j+1}$  by using the following two steps:

### Algorithm 5.1 Line-search

- |    |   |
|----|---|
| 1. | Find the stepsize $\alpha_j = \arg \min_{\alpha > 0} f(x_j + \alpha d_j)$ |
| 2. | Set $x_{j+1} = x_j + \alpha_j d_j$  |

◆

Assuming that an initial point  $x_0$  is given, then, applying  $k$  steps of the above line-search method,  $k$  iterates are obtained:  $\{x_0, x_1, \dots, x_{k-1}\}$ . From (5.12) the stepsize  $\alpha_j$  is computed as

$$\alpha_j = \frac{-d_j^T r_j}{d_j^T A d_j}, \quad (5.13)$$

where  $r_j = Ax_j - b$  is the *residual* at  $x_j$ .

**Definition 5.1** The set of directions  $\{d_0, \dots, d_{k-1}\}$  is a set of conjugate directions if and only if  $d_i^T A d_j = 0$  for all  $i = 0, \dots, k-1, j = 0, \dots, k-1$  and  $i \neq j$ .  $\blacklozenge$

Now, for all  $k = 1, \dots$ , the following vector space and affine space are introduced:

$$W_k = \text{span}\{d_0, \dots, d_{k-1}\}, \quad (5.14)$$

$$U_k = x_0 + W_k = \{z \in \mathbb{R}^n : z = x_0 + w_k, w_k \in W_k\}. \quad (5.15)$$

Denote  $W_0 = \{0\}$  and  $U_0 = \{x_0\}$ .

**Proposition 5.1** Assume that  $d_i^T A d_j = 0$  for all  $0 \leq j < i$ , where  $i$  is a fixed integer, and that  $\{x_0, \dots, x_i\}$  are computed by the above line-search algorithm. Then

$$d_i^T r_i = d_i^T \nabla f(y), \quad (5.16)$$

for all  $y \in U_i$ .

**Proof** Firstly, observe that, since  $\{x_0, \dots, x_i\}$  are computed by the line-search algorithm, it follows that  $x_i \in U_i$ . If  $y \in U_i$ , then, from the definition of  $U_i$ , it follows that  $x_i - y \in W_i$  and hence  $d_i^T A(x_i - y) = 0$ . Therefore,

$$d_i^T(r_i - \nabla f(y)) = d_i^T(Ax_i - b - Ay + b) = d_i^T A(x_i - y) = 0,$$

which proves (5.16).  $\blacklozenge$

In the following, the fundamental property of the line-search method with conjugate directions is presented. Obviously, at every step, the line-search algorithm minimizes  $f(x)$  only in a fixed direction. However, if the directions are conjugate according to Definition 5.1, then a stronger result can be proved, as in Theorem 5.1 below: *a choice of conjugate directions in the line-search method leads to obtaining a minimizer  $x_k$  for the whole space  $U_k$* .

**Theorem 5.1** If the directions in the line-search algorithm are conjugate and  $\{x_0, \dots, x_k\}$  are the iterates generated after  $k$  steps of the line-search algorithm, then

$$x_j = \arg \min_{x \in U_j} f(x),$$

for all  $1 \leq j \leq k$ .

**Proof** The theorem is proved by induction. For  $k = 1$  the result is obtained from the definition of  $x_1$  as a minimizer on  $U_1$ . Assume that for  $k = i$ ,

$$x_j = \arg \min_{y \in U_j} f(y),$$

for all  $1 \leq j \leq i$ . We need to show that if  $x_{i+1} = x_i + \alpha_i d_i$ , then

$$x_{i+1} = \arg \min_{x \in U_{i+1}} f(x).$$

By the definition of  $U_{i+1}$ , any  $x \in U_{i+1}$  can be written as  $x = y + \alpha d_i$ , where  $\alpha \in \mathbb{R}$  and  $y \in U_i$ . Now, using (5.12) and Proposition 5.1, it results

$$\begin{aligned} f(x) &= f(y + \alpha d_i) = f(y) + \alpha d_i^T \nabla f(y) + \frac{\alpha^2}{2} d_i^T A d_i \\ &= f(y) + \left[ \alpha d_i^T \nabla f(x_i) + \frac{\alpha^2}{2} d_i^T A d_i \right]. \end{aligned} \quad (5.17)$$

Observe that (5.17) is a decoupled function. The first term of the right hand side of (5.17) does not depend on  $\alpha$ , and the second term does not depend on  $y$ . Therefore,

$$\min_{x \in U_{i+1}} f(x) = \min_{y \in U_i} f(y) + \min_{\alpha \in \mathbb{R}} \left[ \alpha d_i^T r_i + \frac{\alpha^2}{2} d_i^T A d_i \right]. \quad (5.18)$$

But, the right hand side of (5.18) is minimized when  $y = x_i$  and

$$\alpha = \alpha_i = \frac{-d_i^T r_i}{d_i^T A d_i},$$

that is, the left hand side of (5.18) is minimized exactly for  $x_{i+1} = x_i + \alpha_i d_i$ . In other words,  $x_{i+1}$  is the minimizer of  $f$  over the set  $\{x : x = x_0 + \text{span}\{d_0, \dots, d_i\}\}$ .  $\blacklozenge$

To show the significance of making the search directions mutually conjugate with respect to  $A$ , let us first state and prove a technical result involving only the first two iterations of the conjugate gradient algorithm.

**Proposition 5.2** *After two iterations of the conjugate gradient method, the gradient  $g_2 = Ax_2 - b$  satisfies*

$$d_1^T g_2 = d_0^T g_2 = 0.$$

**Proof** After the first iteration, the new point is  $x_1$ . Therefore,  $g_1 = Ax_1 - b$ . Since the line-search is exact, we also have  $d_0^T g_1 = 0$ .

Now, consider the second iteration. At this iteration, the algorithm will generate a point  $x_2 = x_1 + \alpha d_1$  where  $g_2 = Ax_2 - b$  and  $d_1^T g_2 = 0$ . But

$$d_0^T g_2 = d_0^T (Ax_1 + \alpha Ad_1 - b) = d_0^T g_1 + \alpha d_0^T Ad_1.$$

The first term,  $d_0^T g_1$ , in the right hand side of the above equality, is zero because of the line-search on the first iteration. The second term,  $\alpha d_0^T Ad_1$ , is zero because  $d_0$  and  $d_1$  are conjugate with respect to  $A$ .  $\blacklozenge$

This result shows that after two iterations the gradient is orthogonal to both search directions  $d_0$  and  $d_1$ . Similarly, the above result can be generalized to prove the following proposition.

**Proposition 5.3** After  $k$  iterations of the conjugate gradient method, the gradient  $g_k = Ax_k - b$  satisfies

$$d_j^T g_k = 0 \text{ for } j = 0, 1, \dots, k-1.$$

◆

This proposition implies that after  $k$  iterations, the gradient  $g_k$  is restricted to the  $(n-k)$ -dimensional subspace orthogonal to the vectors  $d_0, \dots, d_{k-1}$ . From this, the important *finite termination property* of the conjugate gradient method can be obtained.

**Proposition 5.4** The conjugate gradient method solves an  $n \times n$  linear algebraic system  $Ax = b$  in  $n$  iterations at the most.

**Proof** Proposition 5.3 implies that, after  $n$  iterations,  $g_n$  is orthogonal to the  $n$  vectors  $d_0, \dots, d_{n-1}$ . But this means that  $g_n$  must lie in a subspace of dimension zero and so  $g_n = 0$ , which proves that  $Ax_n = b$ . ◆

The finite termination property is only guaranteed when the calculations are exact. In practice, the conclusion of Proposition 5.4 may not be exactly satisfied when the iterations are performed in real arithmetic, being subject to rounding errors. Hence, for solving some  $n \times n$  linear algebraic systems, the conjugate gradient method needs a few more than  $n$  iterations.

## The Linear Conjugate Gradient Algorithm

The result of Theorem 5.1 may now be used to present the linear conjugate gradient algorithm for generating conjugate directions. To start with, a general recurrence relation that generates a set of conjugate directions is presented. Next, it is shown that this recurrence relation can be reduced to a simple expression. Finally, the linear conjugate gradient algorithm is described.

**Proposition 5.5** Consider  $d_0 = -r_0$  and for  $k = 1, 2, \dots$  set

$$d_k = -r_k + \sum_{j=0}^{k-1} \frac{d_j^T A r_k}{d_j^T A d_j} d_j. \quad (5.19)$$

Then,  $d_j^T A d_m = 0$  for all  $0 \leq m < j \leq k$ .

**Proof** By induction, it is shown that (5.19) generates conjugate directions. For  $k = 1$ ,  $d_1^T A d_0 = 0$ . Assume that for  $k = i$  the vectors  $\{d_0, \dots, d_i\}$  are pairwise conjugate. We must show that  $d_{i+1}^T A d_m = 0$  for all  $m \leq i$ . Consider  $m \leq i$ . Then

$$\begin{aligned} d_{i+1}^T A d_m &= -r_{i+1}^T A d_m + \sum_{j=0}^i \frac{d_j^T A r_{i+1}}{d_j^T A d_j} d_j^T A d_m \\ &= -r_{i+1}^T A d_m + \frac{d_m^T A r_{i+1}}{d_m^T A d_m} d_m^T A d_m = 0, \end{aligned}$$

which proves the proposition. ◆

**Proposition 5.6** Let  $\{d_0, \dots, d_k\}$  be the directions generated by (5.19). Then:

- (i)  $W_k = \text{span}\{r_0, \dots, r_{k-1}\}$ ,
- (ii)  $r_m^T r_j = 0$ , for all  $0 \leq j < m \leq k$ ,
- (iii)  $d_k^T r_j = -r_k^T r_k$ , for all  $0 \leq j \leq k$ ,
- (iv) The direction  $d_k$  satisfies

$$d_k = -r_k + \beta_{k-1} d_{k-1}, \quad (5.20)$$

where

$$\beta_{k-1} = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}. \quad (5.21)$$

**Proof** Since  $d_0 = -r_0$ , (i) follows directly from (5.19).

To prove (ii) observe that, for  $0 \leq j < m \leq k$  and any  $t \in \mathbb{R}$ ,  $r_j \in W_{j+1} \subset W_m$  and hence  $x_m + tr_j \in U_m$ . Now, from Theorem 5.1, since  $x_m$  is the unique minimizer of function  $f$  over  $U_m$ , it follows that  $t = 0$  is the unique minimizer of  $f(x_m + tr_j)$ . Therefore,

$$0 = \frac{df(x_m + tr_j)}{dt} \Big|_{t=0} = \nabla f(x_m)^T r_j = r_m^T r_j$$

for  $0 \leq j < m \leq k$ , which proves (ii).

To prove (iii), the identity in (iii) is first proved for  $j = k$ . Now, from (i), it follows that  $r_k$  is orthogonal to each  $d_l$  for  $l < k$ . Therefore, if we take the inner product with  $r_k$ , the second term in the right side of (5.19) will vanish. But this is exactly the identity in (iii) for  $j = k$ . If  $j < k$ , then  $(x_k - x_j) \in W_k$ , and hence  $d_k^T A(x_k - x_j) = 0$ . Therefore,

$$d_k^T (r_k - r_j) = d_k^T A(x_k - x_j) = 0.$$

To prove (iv), note first that  $\{r_0, \dots, r_k\}$  form an orthogonal basis of  $W_{k+1}$ . Hence, we can write  $d_k \in W_{k+1}$  as a linear combination of  $\{r_0, \dots, r_k\}$  and then apply (iii). Indeed

$$\begin{aligned} d_k &= \sum_{j=0}^k \frac{d_k^T r_j}{r_j^T r_j} r_j = - \sum_{j=0}^k \frac{r_k^T r_k}{r_j^T r_j} r_j = -r_k - \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}} \sum_{j=0}^{k-1} \frac{r_{k-1}^T r_{k-1}}{r_j^T r_j} r_j \\ &= -r_k + \beta_{k-1} \sum_{j=0}^{k-1} \frac{d_{k-1}^T r_j}{r_j^T r_j} r_j = -r_k + \beta_{k-1} d_{k-1}. \end{aligned}$$
◆

**Remark 5.1** From (5.13) and from Proposition 5.6 (iii), it follows that

$$\alpha_k = \frac{-d_k^T r_k}{d_k^T A d_k} = \frac{r_k^T r_k}{d_k^T A d_k}. \quad (5.22)$$

On the other hand, the residual vector  $r_{k+1}$  can be written as

$$r_{k+1} = Ax_{k+1} - b = Ax_k - b + \alpha_k Ad_k = r_k + \alpha_k Ad_k. \quad (5.23)$$

With these, using Proposition 5.3 and Remark 5.1, the following linear conjugate gradient algorithm can be presented.

**Algorithm 5.2** *Linear Conjugate Gradient*

1.	Select an initial point $x_0$ and $\epsilon > 0$ sufficiently small
2.	Set $r_0 = Ax_0 - b$ , $d_0 = -r_0$ and $k = 0$
3.	If $\ r_k\  \leq \epsilon$ , then stop. Otherwise, continue with step 4
4.	Compute: $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k}, \quad x_{k+1} = x_k + \alpha_k d_k, \quad r_{k+1} = r_k + \alpha_k A d_k, \quad \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}, \quad d_{k+1} = -r_{k+1} + \beta_k d_k$
5.	Set $k = k + 1$ and continue with step 3

◆

The algorithm is simple and very easy to implement. The initial residual is the same as the first gradient search direction. If the initial solution  $x_0$  is zero, then  $r_0 = -b$  and  $d_0 = b$ . The major computational efforts at each iteration are the computation of the matrix-vector product  $A d_k$  and the computation of the inner products  $d_k^T (A d_k)$  and  $r_{k+1}^T r_{k+1}$ . Observe that if  $A$  is not symmetric and positive definite, then the denominator in  $\alpha_k$  may vanish, which leads to the breakdown of the iterations.

The linear conjugate gradient algorithm is recommended only for solving large-scale systems of equations with a symmetric and positive definite matrix. Otherwise, the Gaussian elimination, the factorization methods (the product form of the inverse or the elimination form of the inverse), and the singular value decomposition are preferred, since they are less sensitive to rounding errors.

**Remark 5.2** Recall that the basic Newton step  $d_k^N$  is obtained by solving the Newton system  $\nabla^2 f(x_k) d_k^N = -\nabla f(x_k)$ . The *Newton-CG method* computes the search direction  $d_k^N$  by applying the linear conjugate gradient Algorithm 5.2 to the Newton system. ◆

### Convergence Rate of the Linear Conjugate Gradient Algorithm

In the following, an estimate of the convergence rate of the linear conjugate gradient algorithm is presented. For this, the error reduction in the linear conjugate gradient algorithm is discussed, followed by a convergence rate estimate based on the Chebyshev polynomials.

**Proposition 5.7** *The following relation holds*

$$W_k = \text{span}\{r_0, \dots, A^{k-1} r_0\}. \quad (5.24)$$

**Proof** For  $k = 1$ , (5.24) is true. Assume that (5.24) holds for  $k = i$ , and let us show that it holds for  $k = i + 1$ . From Proposition 5.6 (i), this would be equivalent to showing that  $r_i \in \text{span}\{r_0, \dots, A^i r_0\}$ . Observe that  $r_{i-1} \in W_i$  and  $d_{i-1} \in W_i$ . By inductive assumption, we can write  $r_{i-1} = R_{i-1}(A)r_0$  and  $d_{i-1} = P_{i-1}(A)r_0$ , where  $R_{i-1}(\cdot)$  and  $P_{i-1}(\cdot)$  are polynomials depending on the matrix  $A$  of degree less than or equal to  $i - 1$ . Hence,

$$\begin{aligned} r_i &= r_{i-1} + \alpha_{i-1} A d_{i-1} \\ &= R_{i-1}(A)r_0 + \alpha_{i-1} A P_{i-1}(A)r_0 \in \text{span}\{r_0, \dots, A^i r_0\}, \end{aligned}$$

thus proving the proposition. ◆

In the following, let us present a general error estimate which connects  $\|x^* - x_k\|_A$  and  $\|x^* - x_0\|_A$ , where, for any  $y \in \mathbb{R}^n$ ,  $\|y\|_A^2 = y^T A y$ . For this, denote  $\bar{P}_k$  as the set of polynomials of degree less than or equal to  $k$ .

**Proposition 5.8** *The following estimate holds*

$$\|x^* - x_k\|_A = \inf_{P \in \bar{P}_k, P(0)=1} \|P(A)(x^* - x_0)\|_A. \quad (5.25)$$

**Proof** Since  $r_k$  is orthogonal to  $W_k$ , it follows that for all  $y \in W_k$ ,

$$(x^* - x_k)^T A y = r_k^T y = 0. \quad (5.26)$$

Denoting  $w_k = x_k - x_0 \in W_k$  and  $e_0 = x^* - x_0$ , from (5.26) we get

$$0 = (x^* - x_k)^T A y = (e_0 - w_k)^T A y$$

for all  $y \in W_k$ . Therefore,  $w_k = x_k - x_0$  is an  $A$ -orthogonal projection of  $e_0$  on  $W_k$ . Thus,

$$\|e_0 - w_k\|_A = \min_{w \in W_k} \|e_0 - w\|_A.$$

But, from Proposition 5.7 it is known that  $w = Q_{k-1}(A)r_0$ , for a polynomial  $Q_{k-1} \in \bar{P}_{k-1}$ , where  $\bar{P}_{k-1}$  is the set of polynomials of degree less than or equal to  $k-1$ . Also,  $Ae_0 = -r_0$  and  $e_0 - w = (I + Q_{k-1}(A)A)e_0$  and hence

$$\|x^* - x_k\|_A = \|e_0 - w_k\|_A = \min_{P_k \in \bar{P}_k, P_k(0)=1} \|P_k(A)e_0\|_A, \quad (5.27)$$

which completes the proof.  $\diamond$

This convergence rate is rather general and does not take into account knowledge of the distribution of the eigenvalues of  $A$ . In order to refine the above results and to obtain a qualitative estimate on the right hand side of (5.27), observe that for  $A$  symmetric and positive definite the following spectral decomposition can be written (see Appendix A):

$$A = U \Lambda U^T,$$

where  $U$  is an orthogonal matrix whose columns are the eigenvectors of  $A$  and  $\Lambda$  is a diagonal matrix with the positive eigenvalues of  $A$ ,  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$  on the diagonal. Since  $UU^T = U^T U = I$ , by the orthogonality of  $U$  it follows that for any  $j$ ,

$$A^j = U \Lambda^j U^T.$$

Therefore,

$$P_k(A) = U P_k(\Lambda) U^T.$$

Define  $A^{1/2} = U \Lambda^{1/2} U^T$ . Observe that  $\|x\|_A^2 = x^T A x = \|A^{1/2} x\|_2^2$ . Hence, for any  $x \in \mathbb{R}^n$ ,

$$\|P_k(A)x\|_A = \|A^{1/2} P_k(A)x\|_2 \leq \|P_k(A)\|_2 \|A^{1/2} x\|_2 \leq \|P_k(A)\|_2 \|x\|_A.$$

This, together with (5.27), implies that for any polynomial  $P_k(\lambda)$ ,

$$\|x^* - x_k\|_A = \min_{P_k \in \bar{P}_k, P_k(0)=1} \|P_k(A)e_0\|_A \leq \min_{P_k \in \bar{P}_k, P_k(0)=1} \rho(P_k(A))\|e_0\|_A,$$

where  $\rho(P_k(A))$  is the spectral radius of  $P_k(A)$ . Since both  $A$  and  $P_k(A)$  have the same eigenvectors, it follows that

$$\|x^* - x_k\|_A \leq \min_{P_k \in \bar{P}_k, P_k(0)=1} \max_{1 \leq j \leq n} |P_k(\lambda_j)| \|e_0\|_A, \quad (5.28)$$

where  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$  are the eigenvalues of  $A$ . The above inequality shows that the minimization of the error in the linear conjugate gradient corresponds to the minimization of the polynomial  $P_k(\lambda)$  over the entire range of eigenvalues  $[\lambda_1, \lambda_n]$ . This can be accomplished via the Chebyshev polynomials.

The Chebyshev polynomials of the first kind on the interval  $[-1, 1]$  are defined as

$$T_k(\xi) = \cos(k \arccos(\xi)), \quad k = 0, 1, \dots$$

It is easy to see that  $T_k(\xi)$  is a polynomial if the following trigonometric identities are used:

$$\begin{aligned} \cos(\alpha + \beta) &= \cos \alpha \cos \beta - \sin \alpha \sin \beta, \\ \cos(\alpha + \beta) + \cos(\alpha - \beta) &= 2 \cos \alpha \cos \beta. \end{aligned}$$

Let us denote  $\theta = \arccos(\xi)$ , then

$$\begin{aligned} T_0(\xi) &= \cos(0\theta) = 1, \\ T_1(\xi) &= \cos(1\theta) = \xi, \\ T_2(\xi) &= \cos(2\theta) = \cos^2 \theta - \sin^2 \theta = 2 \cos^2 \theta - 1 = 2\xi^2 - 1, \\ T_{k+1}(\xi) + T_{k-1}(\xi) &= \cos((k+1)\theta) + \cos((k-1)\theta) \\ &= 2 \cos(k\theta) \cos(\theta) = 2\xi T_k(\xi). \end{aligned}$$

Therefore,

$$T_0(\xi) = 1, \quad T_1(\xi) = \xi, \quad (5.29)$$

$$T_{k+1}(\xi) = 2\xi T_k(\xi) - T_{k-1}(\xi) \quad (5.30)$$

for any  $\xi \in \mathbb{R}$ .

From (5.30) for fixed  $\xi$  it follows that

$$T_k(\xi) = c_1(\eta_1(\xi))^k + c_2(\eta_2(\xi))^k, \quad k = 0, 1, \dots,$$

where  $\eta_1(\xi)$  and  $\eta_2(\xi)$  are the roots of the characteristic equation  $\eta^2 - 2\xi\eta + 1 = 0$ . The constants  $c_1$  and  $c_2$  are determined from the initial conditions (5.29). Therefore,

$$T_k(\xi) = \frac{1}{2} \left[ \left( \xi + \sqrt{\xi^2 - 1} \right)^k + \left( \xi - \sqrt{\xi^2 - 1} \right)^k \right]. \quad (5.31)$$

Observe that  $|T_k(\xi)| \leq 1$  for all  $\xi \in [-1, 1]$ . The polynomial that minimizes (5.28) over the interval  $[\lambda_1, \lambda_n]$  is

$$S_k(\lambda) = \left[ T_k \left( \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1} \right) \right]^{-1} \left[ T_k \left( \frac{\lambda_n + \lambda_1 - 2\lambda}{\lambda_n - \lambda_1} \right) \right]. \quad (5.32)$$

To prove this, assume that there exists another polynomial of degree  $k$ ,  $Q_k$ , which is better at minimizing (5.28) on the appropriate interval  $[\lambda_1, \lambda_n]$ , so that  $Q_k(0) = 1$ ,

$$Q_k(\lambda) < \left[ T_k \left( \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1} \right) \right]^{-1}.$$

The polynomial  $P_k - Q_k$  must have a zero at  $\lambda = 0$  and at  $k$  zeros of the polynomials. This means that this polynomial must have  $k + 1$  zeros, which is a contradiction. Therefore,  $S_k$  from (5.32) must be the minimizing polynomial on the interval  $[\lambda_1, \lambda_n]$ . Therefore, from (5.28) it results that

$$\|x^* - x_k\|_A \leq \left[ T_k \left( \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1} \right) \right]^{-1} \|x^* - x_0\|_A. \quad (5.33)$$

**Theorem 5.2** *The error after  $k$  iterations of the linear conjugate gradient algorithm can be bounded as follows:*

$$\|x^* - x_k\|_A \leq \frac{2}{\left( \frac{\sqrt{\kappa+1}}{\sqrt{\kappa-1}} \right)^k + \left( \frac{\sqrt{\kappa-1}}{\sqrt{\kappa+1}} \right)^k} \|x^* - x_0\|_A \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|x^* - x_0\|_A, \quad (5.34)$$

where  $\kappa = \kappa(A) = \lambda_n/\lambda_1$  is the condition number of  $A$ .

**Proof** The purpose is to calculate

$$\left[ T_k \left( \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1} \right) \right]^{-1}.$$

From (5.31), for  $\xi = (\lambda_n + \lambda_1)/(\lambda_n - \lambda_1) = (\kappa + 1)/(\kappa - 1)$ , we obtain

$$\xi \pm \sqrt{\xi^2 - 1} = \frac{\kappa + 1}{\kappa - 1} \pm \frac{2\sqrt{\kappa}}{\kappa - 1} = \frac{\kappa + 1 \pm 2\sqrt{\kappa}}{\kappa - 1} = \frac{(\sqrt{\kappa} \pm 1)^2}{(\sqrt{\kappa} - 1)(\sqrt{\kappa} + 1)} = \frac{\sqrt{\kappa} \pm 1}{\sqrt{\kappa} \mp 1}.$$

Therefore,

$$T_k \left( \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1} \right) = \frac{1}{2} \left[ \left( \frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1} \right)^k + \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \right].$$

Hence,

$$\left[ T_k \left( \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1} \right) \right]^{-1} = \frac{2}{\left( \frac{\sqrt{\kappa+1}}{\sqrt{\kappa-1}} \right)^k + \left( \frac{\sqrt{\kappa-1}}{\sqrt{\kappa+1}} \right)^k} \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k. \quad (5.35)$$

The proof is completed by substituting (5.35) in (5.33).  $\blacklozenge$

Knowing only the largest and the smallest eigenvalues of  $A$ , the bound (5.34) is the best possible. Theorem 5.2 shows that the error  $\|x^* - x_k\|_A$  is upper bounded by a sequence which is convergent to

zero. Besides, the convergence is monotone, and this explains why the linear conjugate gradient algorithm is viewed as an iterative method. Its performance depends on both  $b$  and the spectrum of  $A$  (see Kelley (1995), Greenbaum (1997)). The linear conjugate gradient will perform well if  $\kappa$  is near 1, and it may perform very poorly if  $\kappa$  is large. Geometrically,  $\kappa$  is large if the ellipsoidal level surfaces of the quadratic function  $f$  are far from spherical.

If additional information about the eigenvalues of  $A$  in the interval  $[\lambda_1, \lambda_n]$  is available, then the estimate (5.34) can be improved. Suppose, for example, that  $A$  has one eigenvalue much larger than the others, i.e.,  $\lambda_1 \leq \dots \leq \lambda_{n-1} \ll \lambda_n$ , that is,  $\lambda_n/\lambda_{n-1} \gg 1$ . Consider a polynomial  $P_k$  that is the product of a linear factor which is zero at  $\lambda_n$  and of the  $(k - 1)$  degree scaled and shifted Chebyshev polynomial on the interval  $[\lambda_1, \lambda_{n-1}]$ :

$$P_k(\lambda) = \frac{T_{k-1}\left(\frac{\lambda_{n-1} + \lambda_1 - 2\lambda}{\lambda_{n-1} - \lambda_1}\right)}{T_{k-1}\left(\frac{\lambda_{n-1} + \lambda_1}{\lambda_{n-1} - \lambda_1}\right)} \frac{\lambda_n - \lambda}{\lambda_n}. \quad (5.36)$$

Since the second factor in (5.36) is zero at  $\lambda_n$  and less than one in absolute value at each of the other eigenvalues, then the maximum absolute value of this polynomial in the entire spectrum  $\{\lambda_1, \dots, \lambda_n\}$  of  $A$  is less than the maximum absolute value of the first factor on  $\{\lambda_1, \dots, \lambda_{n-1}\}$ . Therefore, using similar arguments as those in Theorem 5.2, it follows that

$$\|x^* - x_k\|_A \leq 2 \left( \frac{\sqrt{\kappa_{n-1}} - 1}{\sqrt{\kappa_{n-1}} + 1} \right)^{k-1} \|x^* - x_0\|_A, \quad (5.37)$$

where,  $\kappa_{n-1} = \frac{\lambda_{n-1}}{\lambda_1}$ . A detailed study of the case of isolated eigenvalues is given by Axelsson and Linskog (1986).

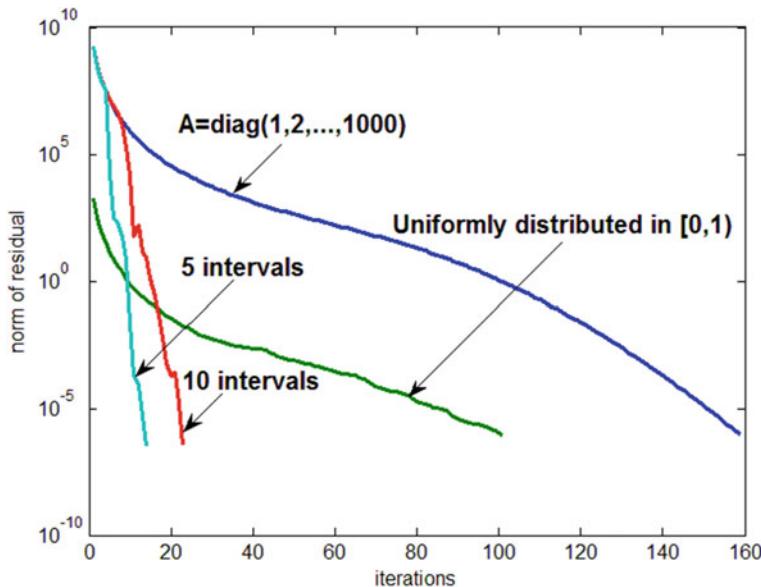
Similarly, if the matrix  $A$  has just a few large eigenvalues, say,  $\lambda_1 \leq \dots \leq \lambda_{n-m} \ll \lambda_{n-m+1} \leq \dots \leq \lambda_n$ , i.e.,  $\lambda_{n-m+1}/\lambda_{n-m} \gg 1$ , then we can consider a polynomial  $P_k$  which is the product of an  $m$ -th degree factor that is zero at each large eigenvalue and of a scaled and shifted Chebyshev polynomial of degree  $k-m$  on the interval  $[\lambda_1, \lambda_{n-m}]$ . Bounding the size of this polynomial, it results that

$$\|x^* - x_k\|_A \leq 2 \left( \frac{\sqrt{\kappa_{n-m}} - 1}{\sqrt{\kappa_{n-m}} + 1} \right)^{k-m} \|x^* - x_0\|_A, \quad (5.38)$$

where, this time,  $\kappa_{n-m} = \frac{\lambda_{n-m}}{\lambda_1}$ . (Greenbaum (1997), Van der Vorst (1993))

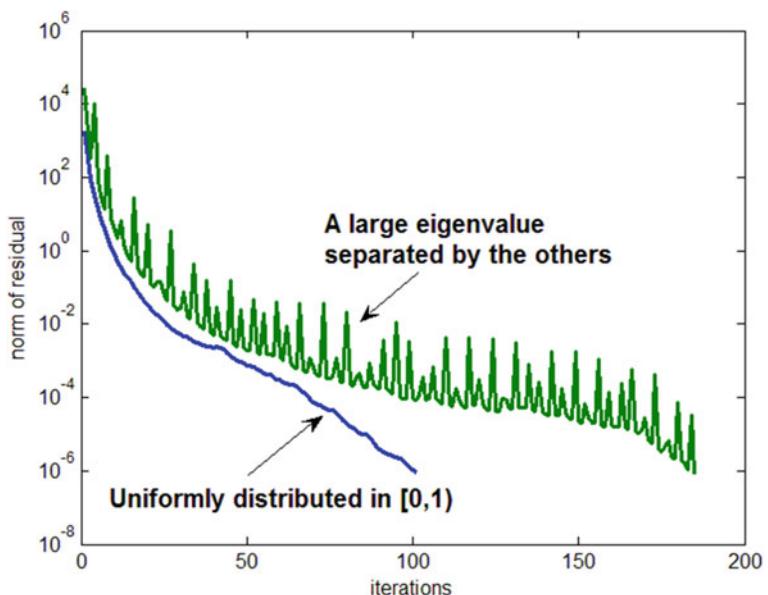
It is generally true that, if the eigenvalues of the matrix  $A$  occur in  $m$  distinct clusters, then the iterates of the linear conjugate gradient algorithm will approximately solve the problem in about  $m$  steps. This is illustrated in the following example.

**Example 5.1** In order to see the influence of the distribution of the eigenvalues on the convergence of the linear conjugate gradient algorithm, let us consider a linear system  $Ax = b$ , where  $A$  is a diagonal matrix and  $b$  is selected in such a way that the solution of the system is always  $[1, 1, \dots, 1]$ , (Andrei, 2009f). In Algorithm 5.2, let us consider  $n = 1000$  and  $\epsilon = 10^{-6}$ . In the first set of numerical experiments, the matrix  $A$  has four distributions of the eigenvalues: (a)  $A = \text{diag}(1, 2, \dots, 1000)$  for which  $\kappa(A) = 1000$ , (b) the diagonal elements are uniformly distributed in  $[0, 1]$  with  $\kappa(A) = 997.4945$ , (c) the eigenvalues of  $A$  are distributed in 10 intervals with  $\kappa(A) = 19.0198$ , and (d) the eigenvalues of  $A$  are distributed in 5 intervals with  $\kappa(A) = 9.0099$ . Figure 5.1 presents the norm of the residuals corresponding to these 4 distributions of the eigenvalues.



**Fig. 5.1** Performance of the linear conjugate gradient algorithm for solving the linear system  $Ax = b$ , where (a)  $A = \text{diag}(1, 2, \dots, 1000)$ , (b) the diagonal elements of  $A$  are uniformly distributed in  $[0,1]$ , (c) the eigenvalues of  $A$  are distributed in 10 intervals, (d) the eigenvalues of  $A$  are distributed in 5 intervals

**Fig. 5.2** Performance of the linear conjugate gradient algorithm for solving the linear system  $Ax = b$ , where the matrix  $A$  has a large eigenvalue separated from the others, which are uniformly distributed in  $[0,1]$



In the second set of numerical experiments, the matrix  $A$  has 999 eigenvalues uniformly distributed in  $[0, 1]$  and one large eigenvalue equal to 100. In this case, the condition number is 4448807.0435, and the linear conjugate gradient gives a solution in 185 iterations. Figure 5.2 presents the evolution of the norm of the residuals for this distribution of the eigenvalues.

Suppose that the eigenvalues of  $A$  consist of  $m$  large values and of  $n - m$  small eigenvalues clustered around 1. Defining  $\tau = \lambda_{n-m} - \lambda_1$ , then (5.38) tells us that after  $m + 1$  steps of the linear conjugate gradient algorithm,

$$\|x^* - x_{m+1}\|_A \approx \tau \|x^* - x_0\|_A.$$

Therefore, for a small value of  $\tau$ , i.e., for small well-clustered eigenvalues, it follows that the iterates of the algorithm provide a good estimate of the solution after only  $m + 1$  steps.

**Example 5.2** Let us consider the linear algebraic system  $Ax = b$ , where

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}, \text{ and } b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},$$

obtained from the finite difference numerical method of the one-dimensional Poisson equation (Andrei, 2000, 2009f).

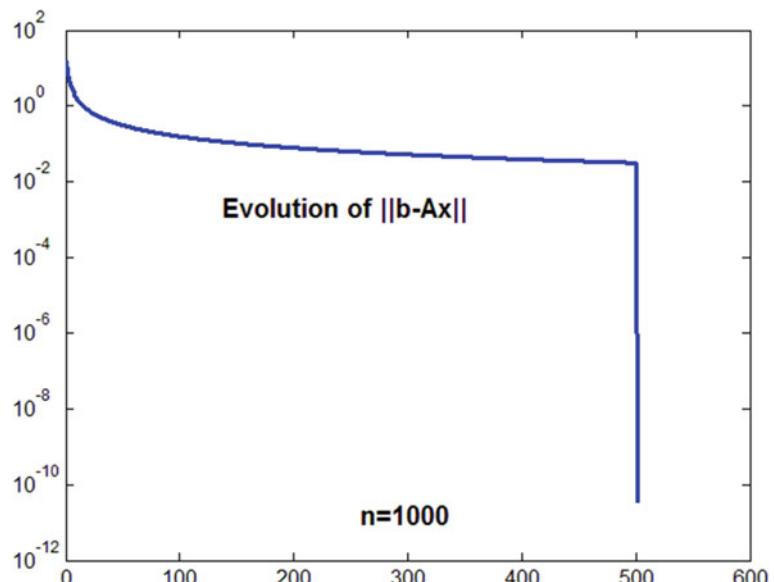
For  $n = 1000$ , the linear conjugate gradient algorithm gives a solution in 500 iterations. Figure 5.3 shows the evolution of the error  $\|b - Ax_k\|$  along the iterations for obtaining a solution with accuracy less than or equal to  $10^{-8}$ .

The eigenvalues of  $A$  are

$$\lambda_i = 2\left(1 - \cos \frac{\pi i}{n+1}\right), \quad i = 1, \dots, n.$$

Observe that the maximum eigenvalue is  $\lambda_n = 2\left(1 - \cos \frac{\pi n}{n+1}\right) \approx 4$ . On the other hand, the minimum eigenvalue is  $\lambda_1 = 2\left(1 - \cos \frac{\pi}{n+1}\right)$ . For  $i$  sufficiently small,

**Fig. 5.3** Evolution of the error  $\|b - Ax_k\|$



$$\lambda_i = 2 \left( 1 - \cos \frac{\pi i}{n+1} \right) \approx 2 \left( 1 - \left( 1 - \frac{\pi^2 i^2}{2(n+1)^2} \right) \right) = \left( \frac{\pi i}{n+1} \right)^2.$$

Therefore,  $A$  is positive definite and for  $n$  large, the condition number of  $A$  is

$$\kappa(A) = \frac{\lambda_n}{\lambda_1} \approx \frac{4(n+1)^2}{\pi^2}.$$

The behavior of the linear conjugate gradient algorithm illustrated in Fig. 5.3 has a *plateau* where, for a large number of iterations, the error  $\|b - Ax_k\|$  decreases very slowly. This behavior is typical of the linear conjugate gradient algorithm. Greenbaum and Strakoš (1992) demonstrated that finite precision conjugate gradient computations for solving a symmetric positive definite linear system  $Ax = b$  behave very similarly to the exact algorithms applied to any of a certain class of larger matrices. This class consists of matrices  $\tilde{A}$  that have lots of eigenvalues spread throughout tiny intervals around the eigenvalues of  $A$ . The width of these intervals is a modest multiple of the machine precision multiplied by the norm of  $A$ . This analogy appears to hold unless the algorithms are run for huge numbers of steps. See also Naiman, Babuska, and Elman (1997).

**Example 5.3** In the following, let us consider the linear algebraic system  $Ax = b$ , where

$$A = \begin{bmatrix} B & -I \\ -I & B & -I \\ & \ddots & \ddots & \ddots \\ & & -I & B & -I \\ & & & -I & B \end{bmatrix}, \text{ and } B = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 4 & -1 \\ & & & -1 & 4 \end{bmatrix},$$

obtained from the finite difference numerical method of the two-dimensional Poisson equation (Andrei, 2000, 2009f). The matrix  $A$  has  $n_2$  blocks on the main diagonal, where each block  $B \in \mathbb{R}^{n_1 \times n_1}$ . Hence,  $A \in \mathbb{R}^{n \times n}$ , where  $n = n_1 n_2$ . The right hand side  $b$  is chosen so that the solution of the system  $Ax = b$  is  $x^* = [1, 1, \dots, 1]$ . Considering  $n = 10000$ , the evolution of the error  $\|b - Ax_k\|$  computed by the linear conjugate gradient algorithm for five different values of  $n_1$  and  $n_2$  is presented in Fig. 5.4.

The eigenvalues of  $A$  are as follows:

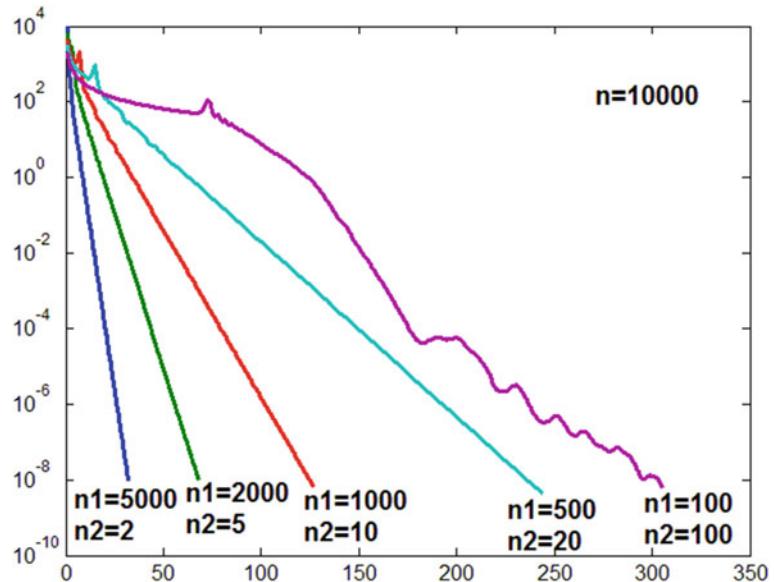
$$\lambda_{ij} = 4 \sin^2 \left( \frac{i\pi}{2(n_1 + 1)} \right) + 4 \sin^2 \left( \frac{j\pi}{2(n_2 + 1)} \right), \quad i = 1, \dots, n_1, \quad j = 1, \dots, n_2.$$

The maximum eigenvalue is  $\lambda_{\max} = 8$ , while the minimum eigenvalue is  $\lambda_{\min} = 8 \sin^2(\pi/2)$ . Therefore,  $A$  is positive definite and its condition number is

$$\frac{\lambda_{\max}}{\lambda_{\min}} \approx \frac{4}{\pi^2}.$$

From Fig. 5.4, for  $n_1 = 5000$  and  $n_2 = 2$ , that is when there are only two blocks on the main diagonal of  $A$ , the linear conjugate gradient algorithm needs only 31 iterations. Therefore, the convergence is faster. On the other hand, when  $n_2 = 100$ , i.e., there are 100 blocks on the main diagonal of the matrix  $A$ , then the algorithm needs 304 iterations. So, the smaller the number of blocks on the main diagonal of the matrix  $A$  is, the faster the convergence is.

**Fig. 5.4** Evolution of the error  $\|b - Ax_k\|$  of the linear conjugate gradient algorithm for different numbers ( $n_2$ ) of blocks on the main diagonal of matrix  $A$



## Preconditioning

Having in view the above discussion, it follows that the conjugate gradient method can be accelerated by transforming the linear system  $Ax = b$  to improve the eigenvalue distribution of  $A$ . This is achieved by a change of the variables from  $x$  to  $\hat{x}$  via a nonsingular matrix  $C$ , that is,

$$\hat{x} = Cx. \quad (5.39)$$

The quadratic (5.1) is transformed as

$$\hat{f}(\hat{x}) = \frac{1}{2}\hat{x}^T C^{-T} A C^{-1} \hat{x} - (C^{-T} b)^T \hat{x}. \quad (5.40)$$

If Algorithm 5.2 is used to minimize  $\hat{f}$  or, equivalently, to solve the linear system

$$(C^{-T} A C^{-1}) \hat{x} = C^{-T} b,$$

then the rate of the convergence will depend on the eigenvalues of the matrix  $C^{-T} A C^{-1}$  rather than on those of  $A$ . Therefore, our purpose is to choose  $C$  such that the eigenvalues of  $C^{-T} A C^{-1}$  are more favorable for the theory presented above. One approach is to choose  $C$  such that the condition number of  $C^{-T} A C^{-1}$  is much smaller than the condition number of  $A$ , so that the constant in (5.34) is smaller. Another approach is to select  $C$  for the eigenvalues of  $C^{-T} A C^{-1}$  to be clustered, which, as we have already mentioned, ensures that the number of iterates needed to find a good approximate solution is not much larger than the number of clusters.

It follows that the transformation (5.39) does not need to be carried out explicitly. Instead, we can apply Algorithm 5.2 directly to the problem (5.40) in terms of the variables  $\hat{x}$  and then invert the transformations to re-express all the equations in terms of  $x$ . Define  $M = C^T C$ .

**Algorithm 5.3** Preconditioned conjugate gradient

1.	Select an initial point $x_0$ , the preconditioner matrix $M$ and $\epsilon > 0$ sufficiently small
2.	Set $r_0 = Ax_0 - b$
3.	Solve the system $My_0 = r_0$
4.	Set $d_0 = -y_0$ and $k = 0$
5.	If $\ r_k\  \leq \epsilon$ , then stop. Otherwise, continue with step 6
6.	Compute: $\alpha_k = \frac{r_k^T y_k}{d_k^T A d_k}, \quad x_{k+1} = x_k + \alpha_k d_k, \quad r_{k+1} = r_k + \alpha_k A d_k,$ $\beta_k = \frac{r_{k+1}^T y_{k+1}}{r_k^T y_k}, \quad d_{k+1} = -y_{k+1} + \beta_k d_k$
7.	Set $k = k + 1$ and continue with step 5

◆

The main difference between the unpreconditioned conjugate gradient Algorithm 5.2 and the preconditioned conjugate gradient 5.3 is the need to solve the linear system  $My = r$  in step 6 of Algorithm 5.3.

The selection of the preconditioner matrix  $M$  is problem dependent. Knowing the structure and the origin of a problem is the key to devising an effective preconditioner. The preconditioner is often defined in such a way so that the system  $My = r$  should represent a simplified version of the original system  $Ax = b$ . The most important strategies for selecting the preconditioner matrix include symmetric successive overrelaxation, incomplete Cholesky factorization, banded preconditioners, etc. (Golub, & Van Loan, 1996). The incomplete Cholesky factorization might be the most effective.

**Incomplete Cholesky Factorization**

The idea of this factorization is as follows. Perform the Cholesky procedure, but, instead of computing the exact Cholesky factor  $L$  that satisfies  $A = LL^T$ , an approximation factor  $\bar{L}$  which is sparser than  $L$  is computed. Therefore, we have  $A \approx \bar{L}\bar{L}^T$  and by choosing  $C = \bar{L}^T$  we obtain  $M = \bar{L}\bar{L}^T$  and  $C^{-T}AC^{-1} = \bar{L}^{-1}A\bar{L}^{-T} \approx I$ , so the eigenvalues distribution of  $C^{-T}AC^{-1}$  is favorable. Obviously, the matrix  $M$  is not explicitly computed. Instead, the factor  $\bar{L}$  is stored, and the system  $My = r$  is solved by performing two triangular substitutions with  $\bar{L}$ .

**Remark 5.3** Two methods are known for the acceleration of conjugate gradient algorithms. The first one consists of modifying the stepsize in a multiplicative manner in such a way as to reduce the values of the minimizing function, as described in Sect. 3.3 (see Remark 3.1). The second one is the preconditioning. No comparisons between these acceleration methods are known. ◆

**Comparison of the Convergence Rate of the Linear Conjugate Gradient and of the Steepest Descent**

As we have already seen in Proposition 5.4, the linear conjugate gradient algorithm has a quadratic (finite) termination property, i.e., for convex quadratic functions, the linear conjugate gradient algorithm with exact line-search terminates after  $n$  iterations. In (5.33), (5.34), (5.37) and (5.38), some formulae for the convergence rates of the linear conjugate gradient algorithm have been

presented, showing that the convergence rate is not worse than the one of the steepest descent algorithm, that is, it is not worse than linear.

In the following, let us have a comparison between the linear conjugate gradient algorithm and the steepest descent algorithm subject to the reduction of the function values along the iterations (Sun, & Yuan, 2006). Consider the quadratic function

$$f(x) = \frac{1}{2}x^T Ax, \quad (5.41)$$

where  $A \in \mathbb{R}^{n \times n}$  is symmetric and positive definite. In this case, the explicit expression for the stepsize is

$$\alpha_k = -\frac{d_k^T Ax_k}{d_k^T Ad_k} = -\frac{d_k^T g_k}{d_k^T Ad_k}. \quad (5.42)$$

Therefore,

$$\begin{aligned} f(x_{k+1}) &= \frac{1}{2}x_{k+1}^T Ax_{k+1} \\ &= \frac{1}{2}(x_k + \alpha_k d_k)^T A(x_k + \alpha_k d_k) \\ &= \frac{1}{2}x_k^T Ax_k - \frac{1}{2} \frac{(g_k^T d_k)^2}{d_k^T Ad_k}. \end{aligned} \quad (5.43)$$

Now, for the steepest descent algorithm  $d_k = -g_k$  and from (5.43),

$$f(x_{k+1}^{SD}) = \frac{1}{2}x_k^T Ax_k - \frac{1}{2} \frac{\|g_k\|^4}{g_k^T Ag_k}. \quad (5.44)$$

On the other hand, for the linear conjugate gradient algorithm  $d_k = -g_k + \beta_{k-1} d_{k-1}$  and from (5.43),

$$f(x_{k+1}^{CG}) = \frac{1}{2}x_k^T Ax_k - \frac{1}{2} \frac{\|g_k\|^4}{d_k^T Ad_k}. \quad (5.45)$$

Since

$$\begin{aligned} d_k^T Ad_k &= (-g_k + \beta_{k-1} d_{k-1})^T A(-g_k + \beta_{k-1} d_{k-1}) \\ &= g_k^T Ag_k + \beta_{k-1}^2 d_{k-1}^T Ad_{k-1} \\ &\leq g_k^T Ag_k, \end{aligned}$$

it follows that

$$f(x_{k+1}^{CG}) \leq f(x_{k+1}^{SD}).$$

Therefore, the linear conjugate gradient algorithm reduces the value of the minimizing function  $f$  at least as much as the steepest descent algorithm. Since the steepest descent algorithm has a linear rate of convergence, it follows that the linear conjugate gradient algorithm has a convergence rate that is not worse than the linear rate. From (5.45) it results that, for the linear conjugate gradient algorithm, the objective function is strictly decreased along the iterations.

### 5.3 General Convergence Results for Nonlinear Conjugate Gradient Methods

For solving the nonlinear unconstrained optimization problem

$$\min f(x), \quad (5.46)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is a continuously differentiable function, any nonlinear conjugate gradient algorithm generates the sequence  $\{x_k\}$  of the form

$$x_{k+1} = x_k + \alpha_k d_k, \quad (5.47)$$

where  $\alpha_k$  is the stepsize obtained by line-search and  $d_k$  is the search direction computed by

$$d_{k+1} = -g_{k+1} + \beta_k d_k, \quad (5.48)$$

for  $k \geq 0$ , where  $\beta_k$  is the conjugate gradient parameter and  $g_k = \nabla f(x_k)$ . In conjugate gradient methods,  $d_0 = -g_0$ .

A popular strategy for the stepsize determination, which plays a key role in the efficiency of the unconstrained optimization algorithms, consists in accepting a positive stepsize  $\alpha_k$  that satisfies the *Wolfe* line-search conditions

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k d_k^T g_k, \quad (5.49)$$

$$\nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma d_k^T g_k, \quad (5.50)$$

where  $0 < \rho < \sigma < 1$ . Often, the *strong Wolfe* line-search is used in the implementation of conjugate gradient methods. These are given by (5.49) and

$$|\nabla f(x_k + \alpha_k d_k)^T d_k| \leq -\sigma d_k^T g_k, \quad (5.51)$$

where again  $0 < \rho < \sigma < 1$ . Observe that if  $\sigma = 0$ , then the strong Wolfe line-search reduces to the exact line-search. Dai and Yuan (1999, 2001) proved that the Wolfe line-search (5.49) and (5.50) ensures the convergence and therefore can be successfully used in the current implementations in computing programs of the conjugate gradient methods.

A conjugate gradient algorithm given by (5.47) and (5.48) generates a sequence  $\{x_k\}$ . The interest is to see the conditions under which this sequence converges to the solution  $x^*$  of the problem (5.46). Since the algorithm given by (5.47) and (5.48) depends only on the parameter  $\beta_k$ , it follows that the interest is to see the values of this parameter for which the algorithm is convergent.

An important requirement for the optimization methods based on line-search is that the search direction must be a descent one. The search direction  $d_k$  satisfies the descent property, i.e., it is a *descent direction* if for all  $k = 1, 2, \dots$

$$g_k^T d_k < 0. \quad (5.52)$$

For conjugate gradient methods, from (5.48) it follows that

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \beta_k g_k^T d_k. \quad (5.53)$$

Now, if the line-search is exact, i.e., if  $g_k^T d_k = 0$ , then  $g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2$ . Therefore,  $d_{k+1}$  is a descent direction if  $g_{k+1} \neq 0$ . However, for the inexact line-search this may not be true. Using the

restart with  $d_{k+1} = -g_{k+1}$  this situation can be corrected. The search direction  $d_k$  satisfies the sufficient descent property, i.e., it is a *sufficient descent direction* if

$$g_k^T d_k \leq -c \|g_k\|^2 \quad (5.54)$$

for all  $k = 1, 2, \dots$ , where  $c > 0$  is a constant.

The convergence properties of a line-search method, such as the nonlinear conjugate gradient, can be studied by measuring the effectiveness of the search direction and of the length of the step. The quality of a search direction  $d_k$  can be determined by studying the angle between the steepest descent direction  $-g_k$  and the search direction  $d_k$  defined by

$$\cos \theta_k = \frac{-g_k^T d_k}{\|g_k\| \|d_k\|}. \quad (5.55)$$

To establish the general convergence results of any method of the form (5.47) and (5.48), the following *basic assumptions* on the objective function are introduced.

### Assumption CG

- (i) *The level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  is bounded, i.e., there exists a constant  $B > 0$  so that  $\|x\| \leq B$  for all  $x$  in the level set.*
- (ii) *In some neighborhood  $N$  of the level set,  $f$  is continuously differentiable, and its gradient is Lipschitz continuous, i.e., there exists a constant  $L > 0$  so that*

$$\|g(x) - g(y)\| \leq L \|x - y\|, \text{ for all } x, y \in N. \quad (5.56)$$

Note that these assumptions imply that there is a constant  $\Gamma$  so that  $\|g(x)\| \leq \Gamma$  for all  $x$  from the level set  $S$ . The boundedness assumption of the level set is not necessary in all the situations. Only the assumption that  $f$  is bounded below on the level set can be used for the global convergence analysis.♦

Under the Assumption CG, the following theorem, due to Zoutendijk (1970) and Wolfe (1969, 1971), is essential in proving the global convergence results of the unconstrained optimization algorithms, including the conjugate gradient or the Newton one.

**Theorem 5.3** *Suppose that  $f$  is bounded below in  $\mathbb{R}^n$  and that  $f$  is continuously differentiable in a neighborhood  $N$  of the level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ . Also assume that the gradient is Lipschitz continuous, i.e., there exists a constant  $L > 0$  so that (5.56) is satisfied for all  $x, y \in N$ . Consider any iteration of the form (5.47), where  $d_k$  is a descent direction and  $\alpha_k$  satisfies the Wolfe line-search conditions (5.49) and (5.50). Then,*

$$\sum_{k=1}^{\infty} \cos^2 \theta_k \|g_k\|^2 < \infty. \quad (5.57)$$

**Proof** From (5.50) it follows that

$$(g_{k+1} - g_k)^T d_k \geq (\sigma - 1) g_k^T d_k.$$

On the other hand, the Lipschitz continuity (5.56) results in

$$(g_{k+1} - g_k)^T d_k \leq \alpha_k L \|d_k\|^2.$$

Therefore, the combination of these two relations gives

$$\alpha_k \geq \frac{(\sigma - 1)}{L} \frac{g_k^T d_k}{\|d_k\|^2}. \quad (5.58)$$

Now, using the first Wolfe condition (5.49) and (5.58), it results that

$$f_{k+1} \leq f_k + \rho \frac{(\sigma - 1)}{L} \frac{(g_k^T d_k)^2}{\|d_k\|^2}. \quad (5.59)$$

From the definition (5.55) of  $\cos\theta_k$ , it follows that (5.59) can be written as

$$f_{k+1} \leq f_k + c \cos^2 \theta_k \|g_k\|^2, \quad (5.60)$$

where  $c = \rho(\sigma - 1)/L$ . Summing (5.60) for  $k \geq 1$  and having in view that  $f$  is bounded below, (5.57) is obtained.  $\blacklozenge$

The relation (5.57) is called the *Zoutendijk condition*, and from (5.55) it can be rewritten as

$$\sum_{k=1}^{\infty} \frac{(g_k^T d_k)^2}{\|d_k\|^2} < \infty. \quad (5.61)$$

It is interesting to see how the Zoutendijk condition is used to obtain global convergence results. Suppose that the iteration (5.47) is in such a way, so that

$$\cos \theta_k \geq \delta > 0, \quad (5.62)$$

for all  $k$ . Then, from (5.57) it follows that

$$\lim_{k \rightarrow \infty} \|g_k\| = 0. \quad (5.63)$$

In other words, if the search direction generated by any unconstrained optimization method does not tend to be orthogonal to the gradient, then the corresponding sequence of gradients converges to zero. For line-search methods (5.47), the limit (5.63) is the best type of the global convergence result that can be obtained. We cannot guarantee if the method converges to the minimizers of function  $f$ , but only that it converges to stationary points. The implications of the Zoutendijk condition are as follows.

1. For the steepest descent with Wolfe line-search,  $\cos\theta_k = 1$  for all  $k$ . Thus, the steepest descent method is globally convergent only if the stepsizes are adequately computed.
2. Consider the Newton-type methods, where the search direction is computed as  $d_k = -B_k^{-1}g_k$ , where  $B_k$  is a nonsingular symmetric matrix ( $B_k = I$ ,  $B_k = \nabla^2 f(x_k)$  or  $B_k$  is a symmetric and positive definite approximation of the Hessian  $\nabla^2 f(x_k)$ ). Assuming that the condition number of the matrices  $B_k$  is uniformly bounded, i.e., for any  $k$ ,  $\|B_k\| \|B_k^{-1}\| \leq \Delta$ , where  $\Delta > 0$  is a constant, then from (5.55) it follows that

$$\begin{aligned}\cos \theta_k &= -\frac{g_k^T d_k}{\|g_k\| \|d_k\|} = \frac{g_k^T B_k^{-1} g_k}{\|g_k\| \|B_k^{-1} g_k\|} \\ &\geq \frac{1}{\|g_k\| \|B_k\|} \frac{\|g_k\|^2}{\|B_k^{-1}\| \|g_k\|} = \frac{1}{\|B_k\| \|B_k^{-1}\|} \geq \frac{1}{\Delta}.\end{aligned}$$

Hence,  $\cos \theta_k \geq 1/\Delta$ , i.e., it is bounded away from 0. Therefore, the Newton method or the quasi-Newton methods are globally convergent if the matrices  $B_k$  are positive definite (descent condition), if their condition number is bounded and if the line-search satisfies the Wolfe conditions. Observe that the condition (5.62) is crucial for obtaining these results.

3. For the conjugate gradient methods, it is not possible to show the limit (5.63), but only a weaker result, that is,

$$\liminf_{k \rightarrow \infty} \|g_k\| = 0. \quad (5.64)$$

This type of result is also obtained from the Zoutendijk condition. Indeed, suppose that (5.64) does not hold, i.e., the gradients remain bounded away from zero. In other words, suppose that there exists a constant  $\gamma > 0$  so that for any  $k$ ,

$$\|g_k\| \geq \gamma. \quad (5.65)$$

In this case, from the Zoutendijk condition (5.57) again, it follows that

$$\cos \theta_k \rightarrow 0. \quad (5.66)$$

Therefore, the algorithm can only fail in the sense of (5.65) if the sequence  $\{\cos \theta_k\}$  converges to zero. Hence, to establish (5.64) suffice it to show that a subsequence  $\{\cos \theta_{k_j}\}$  of the sequence  $\{\cos \theta_k\}$  is bounded away from zero.

Let us now present some conditions on  $\beta_k$  which determine the convergence of the conjugate gradient algorithms. Suppose that  $\beta_k \geq 0$  and the search direction  $d_k$  is a descent direction, i.e.,  $g_k^T d_k < 0$ . At this moment, we are interested in finding a  $\beta_k$  which produces a descent direction  $d_{k+1}$ , i.e., a direction which satisfies

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k < 0. \quad (5.67)$$

**Proposition 5.9** *Suppose that  $\beta_k \geq 0$ . If*

$$\beta_k \leq \frac{\|g_{k+1}\|^2}{d_k^T y_k}, \quad (5.68)$$

then  $d_{k+1}$  is a descent direction for function  $f$ .

**Proof** Since  $g_k^T d_k < 0$  and  $\beta_k \geq 0$  for all  $k \geq 1$ , we can write

$$\begin{aligned}
-\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k &= -\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k - \beta_k g_k^T d_k + \beta_k g_k^T d_k \\
&= -\|g_{k+1}\|^2 + \beta_k d_k^T y_k + \beta_k g_k^T d_k \\
&\leq -\|g_{k+1}\|^2 + \beta_k d_k^T y_k,
\end{aligned} \tag{5.69}$$

where  $y_k = g_k - g_{k-1}$ . Therefore, the non-positivity of (5.69) is sufficient to show that the condition (5.67) holds. Hence,

$$\|g_{k+1}\|^2 \geq \beta_k d_k^T y_k, \tag{5.70}$$

proving the proposition.  $\blacklozenge$

The following theorem proved by Dai and Yuan (2001) shows how  $\beta_k$  selected to satisfy (5.68) determines the convergence of the conjugate gradient algorithm.

**Theorem 5.4** Suppose that the Assumption CG holds. Let  $\{x_k\}$  be the sequence generated by the algorithm (5.47) and (5.48), where  $\beta_k$  satisfies (5.68). Then, the algorithm either determines a stationary point or it converges in the sense that

$$\liminf_{k \rightarrow \infty} \|g_k\| = 0.$$

**Proof** From (5.48) we get  $d_{k+1} + g_{k+1} = \beta_k d_k$ . Squaring both sides of this relation results in

$$\|d_{k+1}\|^2 = \beta_k^2 \|d_k\|^2 - 2g_{k+1}^T d_{k+1} - \|g_{k+1}\|^2. \tag{5.71}$$

From (5.71), dividing both sides by  $(g_{k+1}^T d_{k+1})^2$ , it follows that

$$\begin{aligned}
\frac{\|d_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} &= \frac{(\beta_k)^2 \|d_k\|^2}{(g_{k+1}^T d_{k+1})^2} - 2 \frac{g_{k+1}^T d_{k+1}}{(g_{k+1}^T d_{k+1})^2} - \frac{\|g_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} \\
&= \frac{(\beta_k)^2 \|d_k\|^2}{(g_{k+1}^T d_{k+1})^2} - \frac{2}{g_{k+1}^T d_{k+1}} - \frac{\|g_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} \\
&= \frac{(\beta_k)^2 \|d_k\|^2}{(g_{k+1}^T d_{k+1})^2} - \left( \frac{1}{\|g_{k+1}\|} + \frac{\|g_{k+1}\|}{g_{k+1}^T d_{k+1}} \right)^2 + \frac{1}{\|g_{k+1}\|^2} \\
&\leq \frac{(\beta_k)^2 \|d_k\|^2}{(g_{k+1}^T d_{k+1})^2} + \frac{1}{\|g_{k+1}\|^2} \\
&= \left( \frac{(\beta_k)^2 (g_k^T d_k)^2}{(g_{k+1}^T d_{k+1})^2} \right) \frac{\|d_k\|^2}{(g_k^T d_k)^2} + \frac{1}{\|g_{k+1}\|^2}.
\end{aligned}$$

Now, using (5.48) again,

$$\frac{\|d_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} \leq \left( \frac{(\beta_k)^2 (g_k^T d_k)^2}{(-\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k)^2} \right) \frac{\|d_k\|^2}{(g_k^T d_k)^2} + \frac{1}{\|g_{k+1}\|^2}. \tag{5.72}$$

Therefore, from the above relations and from (5.68),

$$\left( \underbrace{-\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k}_{<0} + \underbrace{\beta_k g_k^T d_k}_{<0} \right) \left( \underbrace{-\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k - \beta_k g_k^T d_k}_{\leq 0} \right) \geq 0.$$

Hence,

$$\frac{(\beta_k)^2 (g_k^T d_k)^2}{(-\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k)^2} \leq 1. \quad (5.73)$$

From (5.72) and (5.73) the following inequality is obtained:

$$\frac{\|d_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} \leq \frac{\|d_k\|^2}{(g_k^T d_k)^2} + \frac{1}{\|g_{k+1}\|^2}. \quad (5.74)$$

Observe that

$$\frac{\|d_0\|^2}{(g_0^T d_0)^2} = \frac{1}{\|g_0\|^2}.$$

Therefore, from (5.74) it follows that

$$\frac{\|d_k\|^2}{(g_k^T d_k)^2} \leq \sum_{i=0}^k \frac{1}{\|g_i\|^2} \quad (5.75)$$

for all  $k$ . Now, if the theorem is not true, then there exists a constant  $\gamma > 0$  so that  $\|g_k\| \geq \gamma$  for all  $k$ , i.e., from (5.75),

$$\frac{(g_k^T d_k)^2}{\|d_k\|^2} \geq \frac{\gamma^2}{k+1},$$

which implies

$$\sum_{k=0}^{\infty} \frac{(g_k^T d_k)^2}{\|d_k\|^2} = \infty,$$

thus contradicting the Zoutendijk condition. ◆

### Convergence Under the Strong Wolfe Line-Search

The following theorem shows that if  $\beta_k$  is chosen to satisfy the condition (5.68) for all  $k$ , then, under the strong Wolfe line-search conditions (5.49) and (5.51), the direction (5.48) satisfies the sufficient descent condition (5.54).

**Theorem 5.5** Suppose that  $x_0$  is an initial point and the Assumption CG holds. Let  $\{x_k\}$  be the sequence generated by the conjugate gradient algorithm (5.47) and (5.48). If  $\beta_k$  is so that

$\|g_{k+1}\|^2 \geq \beta_k d_k^T y_k$  and  $\beta_k \geq 0$  and the stepsize  $\alpha_k$  satisfies the strong Wolfe conditions (5.49) and (5.51), then the conjugate gradient algorithm satisfies the sufficient descent condition (5.54) with  $c = 1/(1 + \sigma)$ .

**Proof** The proof is given by induction as follows. Observe that  $\sigma > 0$  implies that  $-1 < -1/(1 + \sigma)$ . Therefore, for  $k = 1$  the conclusion holds, since

$$g_0^T d_0 = -\|g_0\|^2 \leq -\frac{1}{1 + \sigma} \|g_0\|^2 = -c \|g_0\|^2,$$

where  $c = 1/(1 + \sigma)$ . Suppose that (5.54) holds for some  $k \geq 0$ . Then, from the second strong Wolfe condition (5.51), it follows that

$$l_k \equiv \frac{g_{k+1}^T d_k}{g_k^T d_k} \in [-\sigma, \sigma] \text{ and } d_k^T y_k > 0.$$

Hence,

$$l_k - 1 = \frac{g_{k+1}^T d_k}{g_k^T d_k} - 1 = \frac{g_{k+1}^T d_k - g_k^T d_k}{g_k^T d_k} = \frac{d_k^T y_k}{g_k^T d_k} \neq 0. \quad (5.76)$$

By (5.48) it results that

$$g_{k+1}^T d_{k+1} = g_{k+1}^T (-g_{k+1} + \beta_k d_k) = -\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k.$$

Following the sign of  $g_{k+1}^T d_k$ , the following two cases arise.

(i) The case  $g_{k+1}^T d_k \leq 0$ . Hence,

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k \leq -\|g_{k+1}\|^2 \leq -\frac{1}{1 + \sigma} \|g_{k+1}\|^2. \quad (5.77)$$

(ii) The case  $g_{k+1}^T d_k > 0$ . Using the conditions on  $\beta_k$  and (5.76), it follows that

$$\begin{aligned} g_{k+1}^T d_{k+1} &= -\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k \\ &\leq -\|g_{k+1}\|^2 + \frac{\|g_{k+1}\|^2}{d_k^T y_k} g_{k+1}^T d_k = \left( -1 + \frac{g_{k+1}^T d_k}{d_k^T y_k} \right) \|g_{k+1}\|^2 \\ &= \left( \frac{g_k^T d_k}{d_k^T y_k} \right) \|g_{k+1}\|^2 = \frac{1}{l_k - 1} \|g_{k+1}\|^2. \end{aligned}$$

But

$$\frac{1}{1 + \sigma} \leq \frac{1}{1 - l_k} \leq \frac{1}{1 - \sigma}. \quad (5.78)$$

From (5.78),

$$g_{k+1}^T d_{k+1} \leq \frac{1}{l_k - 1} \|g_{k+1}\|^2 \leq -\frac{1}{1 + \sigma} \|g_{k+1}\|^2. \quad (5.79)$$

From (5.77) and (5.79) it follows that at the iteration  $k + 1$  the sufficient descent condition is satisfied with  $c = 1/(1 + \sigma)$ .  $\blacklozenge$

The following theorem, which introduces the *Nocedal condition*, presents a general convergence result for any conjugate gradient method (5.47) and (5.48) under the strong Wolfe line-search (5.49) and (5.51). The theorem mainly says that, if  $\|d_k\|^2$  is at most linearly increasing, i.e., if  $\|d_k\|^2 \leq c_1 k + c_2$  for all  $k$ , where  $c_1$  and  $c_2$  are some constants, then a conjugate gradient method with strong Wolfe line-search is globally convergent. The theorem is proved by Dai (2011). Also see (Nocedal, 1996).

**Theorem 5.6** Suppose that the Assumption CG holds. Consider any conjugate gradient method (5.47) and (5.48) with  $d_k$  satisfying  $g_k^T d_k < 0$  and with strong Wolfe line-search (5.49) and (5.51). Then the method is globally convergent if

$$\sum_{k=1}^{\infty} \frac{1}{\|d_k\|^2} = \infty. \quad (5.80)$$

**Proof** From (5.53) and (5.51) it follows that

$$|g_{k+1}^T d_{k+1}| + \sigma |\beta_k| |g_k^T d_k| \geq \|g_{k+1}\|^2. \quad (5.81)$$

Applying the following inequality  $(a + \sigma b)^2 \leq (1 + \sigma^2)(a^2 + b^2)$  valid for all  $a, b, \sigma \geq 0$ , with  $a = |g_{k+1}^T d_{k+1}|$  and  $b = |\beta_k| |g_k^T d_k|$ , (5.81) yields to

$$(g_{k+1}^T d_{k+1})^2 + \beta_k^2 (g_k^T d_k)^2 \geq c \|g_{k+1}\|^4, \quad (5.82)$$

where  $c = (1 + \sigma^2)^{-1}$  is a positive constant.

Now, from (5.48) it follows that  $d_{k+1} + g_{k+1} = \beta_k d_k$ , for all  $k \geq 1$ . Squaring both sides of this equality, the following is obtained:

$$\|d_{k+1}\|^2 = -\|g_{k+1}\|^2 - 2g_{k+1}^T d_{k+1} + \beta_k^2 \|d_k\|^2.$$

Since  $g_{k+1}^T d_{k+1} < 0$ , it follows that

$$\|d_{k+1}\|^2 \geq -\|g_{k+1}\|^2 + \beta_k^2 \|d_k\|^2. \quad (5.83)$$

Hence, from (5.82) and (5.83),

$$\begin{aligned} \frac{(g_{k+1}^T d_{k+1})^2}{\|d_{k+1}\|^2} + \frac{(g_k^T d_k)^2}{\|d_k\|^2} &= \frac{1}{\|d_{k+1}\|^2} \left[ (g_{k+1}^T d_{k+1})^2 + \frac{\|d_{k+1}\|^2}{\|d_k\|^2} (g_k^T d_k)^2 \right] \\ &\geq \frac{1}{\|d_{k+1}\|^2} \left[ (g_{k+1}^T d_{k+1})^2 + \beta_k^2 (g_k^T d_k)^2 - \frac{(g_k^T d_k)^2}{\|d_k\|^2} \|g_{k+1}\|^2 \right] \\ &\geq \frac{1}{\|d_{k+1}\|^2} \left[ c \|g_{k+1}\|^4 - \frac{(g_k^T d_k)^2}{\|d_k\|^2} \|g_{k+1}\|^2 \right]. \end{aligned} \quad (5.84)$$

Assume that (5.64) is not true and there exists a constant  $\gamma > 0$  so that for any  $k$ ,

$$\|g_k\| \geq \gamma. \quad (5.85)$$

Observe that the Zoutendijk condition (5.61) implies that  $g_k^T d_k / \|d_k\|$  tends to zero. Therefore, by (5.84) and (5.85), for sufficiently large  $k$  it results that

$$\frac{(g_{k+1}^T d_{k+1})^2}{\|d_{k+1}\|^2} + \frac{(g_k^T d_k)^2}{\|d_k\|^2} \geq c \frac{\|g_{k+1}\|^4}{\|d_{k+1}\|^2}. \quad (5.86)$$

Thus, by the Zoutendijk condition and (5.85), we must have

$$\sum_{k=1}^{\infty} \frac{1}{\|d_k\|^2} \leq \frac{1}{\gamma^2} \sum_{k=1}^{\infty} \frac{\|g_k\|^2}{\|d_k\|^2} < \infty,$$

which is a contradiction to the assumption (5.80). Therefore, the convergence relation (5.64) holds.♦

The theorem says that the iterations of the conjugate gradient method can fail in the sense of (5.85) only if  $\|d_k\| \rightarrow \infty$  rapidly enough. More exactly, the sequence of the gradient norms  $\|g_k\|$  can be bounded away from zero only if

$$\sum_{k=1}^{\infty} \frac{1}{\|d_k\|^2} < \infty. \quad (5.87)$$

In the following, we shall provide a condition on  $\beta_k$  which is sufficient for the global convergence of the general conjugate gradient method with strong Wolfe line-search (Dai, 2010). By Theorem 5.6 we know that, if (5.80) holds, then the conjugate gradient method is convergent in the sense of (5.64). Otherwise, we have (5.87), which gives

$$\lim_{k \rightarrow \infty} \|d_k\| = \infty. \quad (5.88)$$

Therefore, from the Assumption CG, it follows that

$$\|g_k\| \leq \Gamma, \quad (5.89)$$

for some  $\Gamma > 0$  and for all  $k \geq 0$ . Now, from (5.48) and the above relations (5.88) and (5.89), it follows that

$$\|d_{k+1}\| \approx |\beta_k| \|d_k\|. \quad (5.90)$$

Therefore, if the scalars  $\beta_k$  are so that

$$\sum_{k=1}^{\infty} \prod_{j=0}^k \beta_j^{-2} = \infty, \quad (5.91)$$

it is possible to establish the Nocedal condition (5.80) and then, by Theorem 5.6 a contradiction to (5.87) is obtained. The following theorem details these discussions.

**Theorem 5.7** Suppose that the Assumption CG holds. Consider a conjugate gradient method given by (5.47) and (5.48), where the search direction  $d_k$  is descent, i.e.,  $d_k^T g_k < 0$ . Consider that the stepsize is determined by the strong Wolfe line-search conditions (5.49) and (5.51). If  $\beta_k$  satisfies (5.91), then  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ .

**Proof** Observe that (5.48) can be written as

$$d_{k+1} + g_{k+1} = \beta_k d_k. \quad (5.92)$$

Squaring both sides of (5.92) leads to

$$\|d_{k+1}\|^2 = -2g_{k+1}^T d_{k+1} - \|g_{k+1}\|^2 + \beta_k^2 \|d_k\|^2. \quad (5.93)$$

But,

$$-2g_{k+1}^T d_{k+1} - \|g_{k+1}\|^2 \leq \frac{(g_{k+1}^T d_{k+1})^2}{\|g_{k+1}\|^2}. \quad (5.94)$$

Therefore, from (5.93) and (5.94), it results that

$$\|d_{k+1}\|^2 \leq \frac{(g_{k+1}^T d_{k+1})^2}{\|g_{k+1}\|^2} + \beta_k^2 \|d_k\|^2. \quad (5.95)$$

Having in view the definition of  $\cos\theta_k$  in (5.55), from (5.95) it follows that

$$\begin{aligned} \|d_{k+1}\|^2 &\leq (1 - \cos^2\theta_{k+1})^{-1} \beta_k^2 \|d_k\|^2 \leq \dots \\ &\leq \left( \prod_{j=1}^{k+1} (1 - \cos^2\theta_j)^{-1} \right) \left( \prod_{j=0}^k \beta_j^2 \right) \|d_0\|^2. \end{aligned} \quad (5.96)$$

Suppose that  $\liminf_{k \rightarrow \infty} \|g_k\| \neq 0$ . Therefore, there exists a constant  $\gamma > 0$  so that

$$\|g_k\| \geq \gamma, \text{ for all } k \geq 0. \quad (5.97)$$

Hence, from the Zoutendijk condition (5.61), from the definition of  $\cos\theta_k$  and from (5.97), it follows that

$$\sum_{k=1}^{\infty} \cos^2\theta_k < \infty. \quad (5.98)$$

With this, (5.98) implies that

$$\prod_{j=1}^{k+1} (1 - \cos^2\theta_j) \geq c, \quad (5.99)$$

where  $c > 0$  is a constant. Hence

$$\frac{1}{\|d_{k+1}\|^2} \geq c \left( \prod_{j=0}^k \beta_j^{-2} \right) \|d_0\|^{-2}. \quad (5.100)$$

From (5.100) and (5.91) it follows that (5.80) holds. Thus, by Theorem 5.6,  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ . But this, together with (5.97), gives a contradiction, thus proving the theorem.  $\diamond$

Theorem 5.7 shows that the global convergence of any conjugate gradient algorithm is obtained if the conjugate gradient parameters  $\beta_k$  satisfy the condition (5.91) and the stepsize is determined by the strong Wolfe line-search. Mainly, it is based on the Zoutendijk condition. It should be mentioned that

in Theorem 5.7, it is the descent condition (5.52) which is used and not the sufficient descent condition (5.54).

## Convergence Under the Wolfe Line-Search

Dai (2010) proved that the conclusion of Theorem 5.7 for the global convergence of any conjugate gradient method also holds under the Wolfe line-search. This result is based on the following proposition proved by Dai and Yuan (2003).

**Proposition 5.10** *Consider any conjugate gradient method (5.47) and (5.48). Define  $\psi_k$  and  $t_k$  as follows*

$$\psi_k^2 = \begin{cases} \|g_k\|^2, & \text{for } k = 0, \\ \beta_0^2 \beta_1^2 \cdots \beta_{k-1}^2, & \text{for } k \geq 1 \end{cases} \quad (5.101)$$

and

$$t_k = \frac{\|d_k\|^2}{\psi_k^2}. \quad (5.102)$$

Then, for all  $k \geq 0$ ,

$$t_k = -2 \sum_{i=0}^k \frac{g_i^T d_i}{\psi_i^2} - \sum_{i=0}^k \frac{\|g_i\|^2}{\psi_i^2}. \quad (5.103)$$

**Proof** Since  $d_0 = -g_0$  and  $\psi_0^2 = \|g_0\|^2$ , (5.103) holds for  $k = 0$ . For  $k \geq 1$ , dividing (5.93) by  $\psi_{k+1}^2$  and using the definitions of  $\psi_k$  and  $t_k$ , it follows that

$$t_{k+1} = t_k - 2 \frac{g_{k+1}^T d_{k+1}}{\psi_{k+1}^2} - \frac{\|g_{k+1}\|^2}{\psi_{k+1}^2}. \quad (5.104)$$

Summing (5.104) results in

$$t_{k+1} = t_0 - 2 \sum_{i=1}^{k+1} \frac{g_i^T d_i}{\psi_i^2} - \sum_{i=1}^{k+1} \frac{\|g_i\|^2}{\psi_i^2}. \quad (5.105)$$

Since  $t_0 = \|g_0\|^2 / \psi_0^2$ , it follows that (5.105) is equivalent to (5.103). Therefore (5.103) holds for any  $k \geq 0$ .  $\blacklozenge$

To show the sufficiency of the condition (5.91) on  $\beta_k$ , the following proposition is needed.

**Proposition 5.11** Suppose that  $\{a_i\}$  and  $\{b_i\}$  are two real positive number sequences satisfying:

$$b_k \leq c_1 + c_2 \sum_{i=1}^k a_i, \quad \text{for all } k,$$

where  $c_1$  and  $c_2$  are positive constants. If  $\sum_{k \geq 1} a_k$  is divergent, then  $\sum_{k \geq 1} a_k/b_k$  is also divergent.

**Proof** Let  $S_k = \sum_{i=1}^k a_i$ , for any  $k \geq 1$ . Now,  $\sum_{i \geq 1} a_i = \infty$  implies that  $\lim_{k \rightarrow \infty} S_k = \infty$ . Observe that the sequence  $\{S_k\}$  is increasing. Let  $c = c_1/c_2$ . There exists  $k_0 \geq 1$  so that  $S_k \geq c$  for  $k \geq k_0$ . Note that

$$\sum_{k=k_0}^{\infty} \frac{a_k}{b_k} \geq \sum_{k=k_0}^{\infty} \frac{a_k}{c_1 + c_2 S_k} = \frac{1}{c_2} \sum_{k=k_0}^{\infty} \frac{a_k}{c + S_k} \geq \frac{1}{c_2} \sum_{k=k_0}^{\infty} \frac{a_k}{S_k + S_k} = \frac{1}{2c_2} \sum_{k=k_0}^{\infty} \frac{a_k}{S_k}.$$

Let

$$A(k, r) = \sum_{i=1}^r \frac{a_{k+i}}{S_{k+i}}, \quad \text{for } r \geq 1.$$

It is easy to see that for every  $k \geq 1$  there exists  $r \geq 1$  so that  $A(k, r) \geq \frac{1}{2}$ . Indeed,

$$A(k, r) \geq \sum_{i=1}^r \frac{a_{k+i}}{S_{k+r}} = \frac{S_{k+r} - S_k}{S_{k+r}} = 1 - \frac{S_k}{S_{k+r}}.$$

But,

$$\lim_{k \rightarrow \infty} A(k, r) \geq \lim_{k \rightarrow \infty} \left( 1 - \frac{S_k}{S_{k+r}} \right) = 1.$$

Hence, there exists  $r \geq 1$  so that  $A(k, r) \geq 1/2$ .

Therefore, there is a sequence  $\{r_k\}$  with  $r_k \geq 1$  so that  $k_1 = k_0 + r_1$ ,  $k_2 = k_1 + r_2$ , ... and  $A(k_0, r_1) \geq \frac{1}{2}$ ,  $A(k_1, r_2) \geq \frac{1}{2}$ , ... Hence,

$$\sum_{k=k_0}^{\infty} \frac{a_k}{S_k} = \sum_{i=0}^{\infty} A(k_i, r_{i+1}) \geq \sum_{i=0}^{\infty} \frac{1}{2} = \infty. \quad \blacklozenge$$

The following theorem proved by Dai (2010) shows that the condition (5.91) on  $\beta_k$  is sufficient for the global convergence of any conjugate gradient method (5.47) and (5.48).

**Theorem 5.8** Suppose that the Assumption CG holds. Consider a conjugate gradient method given by (5.47) and (5.48), where the search direction  $d_k$  is descent, i.e.,  $d_k^T g_k < 0$  and the stepsize is determined by the Wolfe line-search conditions (5.49) and (5.50). If  $\beta_k$  satisfies (5.91), then  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ .

**Proof** Define  $\psi_k^2$  as in (5.101). Then, from (5.91) it follows that

$$\sum_{k \geq 1} \frac{1}{\psi_k^2} = \infty. \quad (5.106)$$

Now, using (5.94) in (5.103) we can write

$$t_k \leq \sum_{i=0}^k \frac{(g_i^T d_i)^2}{\|g_i\|^2 \psi_i^2}. \quad (5.107)$$

Since  $t_{k+1} \geq 0$ , from (5.103) it follows that

$$-2 \sum_{i=0}^k \frac{g_i^T d_i}{\psi_i^2} \geq \sum_{i=0}^k \frac{\|g_i\|^2}{\psi_i^2}. \quad (5.108)$$

But  $(\|g_{k+1}\|^2 + 2(g_{k+1}^T d_{k+1}))^2 \geq 0$ . Hence, for any  $k$ ,

$$-4g_{k+1}^T d_{k+1} - \|g_{k+1}\|^2 \leq 4 \frac{(g_{k+1}^T d_{k+1})^2}{\|g_{k+1}\|^2}. \quad (5.109)$$

Therefore, from (5.108) and (5.109),

$$4 \sum_{i=0}^k \frac{(g_i^T d_i)^2}{\|g_i\|^2 \psi_i^2} \geq -4 \sum_{i=0}^k \frac{g_i^T d_i}{\psi_i^2} - \sum_{i=0}^k \frac{\|g_i\|^2}{\psi_i^2} \geq \sum_{i=0}^k \frac{\|g_i\|^2}{\psi_i^2}. \quad (5.110)$$

Now, let us proceed by contradiction and assume that (5.97) holds. Then, by (5.110) and (5.106) it follows that

$$\sum_{k \geq 1} \frac{(g_k^T d_k)^2}{\|g_k\|^2 \psi_k^2} = \infty. \quad (5.111)$$

From (5.111), (5.102) and from Proposition 5.11,

$$\sum_{k=1}^{\infty} \frac{(g_k^T d_k)^2}{\|g_k\|^2 \psi_k^2} \frac{1}{t_k} = \sum_{k=1}^{\infty} \frac{(g_k^T d_k)^2}{\|g_k\|^2 \|d_k\|^2} = \sum_{k=1}^{\infty} \cos^2 \theta_k = \infty, \quad (5.112)$$

which contradicts (5.98). This contradiction shows that (5.91) is true.  $\blacklozenge$

Observe that the proof of this theorem is based on the Zoutendijk condition. Theorem 5.8 provides the condition (5.91) on  $\beta_k$  which is sufficient for the global convergence of the general conjugate gradient method with Wolfe line-search. Besides, notice that only the descent condition  $g_k^T d_k < 0$  for any  $k$  is used in Theorem 5.8. This is an important contribution to the general theory of conjugate gradient methods, since very often, in the implementation of the conjugate gradient algorithms, only the Wolfe line-search conditions are used.

The above theorems present the necessary and sufficient condition on the conjugate gradient parameter  $\beta_k$ , namely, (5.91), for the global convergence of any general conjugate gradient method under the Wolfe line-search.

## 5.4 Standard Conjugate Gradient Methods

In the following, let us present the standard conjugate gradient algorithms as well as their convergence results for solving unconstrained optimization problems. The standard conjugate gradient methods are listed in Table 5.1. These methods have simple algebraic expressions, and their convergence results are particularizations or specializations of the general results on the convergence of conjugate gradient algorithms. In general, the convergence results of these algorithms are based on the Assumption CG and on some other natural hypotheses on the line-search.

A rapid inspection of Table 5.1 shows that, except for Daniel's method (1967) which requires the evaluation of the Hessian at each iteration, the numerator of the update parameter  $\beta_k$  is either  $\|g_{k+1}\|^2$  or  $g_{k+1}^T y_k$  and the denominator is either  $\|g_k\|^2$  or  $d_k^T y_k$  or  $-d_k^T g_k$ . Here,  $y_k = g_{k+1} - g_k$ . Mainly, these two choices for the numerator and the three choices for the denominator lead to six different choices for  $\beta_k$ .

If function  $f$  is strongly convex quadratic and the line-search is exact, then, in theory all the above choices for the update parameter  $\beta_k$  presented in Table 5.1 are equivalent. For non-quadratic objective functions, each choice for  $\beta_k$  leads to algorithms with different numerical performances (number of iterations, number of function and its gradient evaluations, or CPU time). Therefore, in the following, the global convergence properties of the standard conjugate gradient methods with the numerator  $\|g_{k+1}\|^2$  for the update parameter  $\beta_k$  (FR, CD, and DY) and with  $g_{k+1}^T y_k$  in the numerator of  $\beta_k$  (HS, PRP, and LS) will be separately presented. As a general remark, the convergence theory for the methods with numerator  $\|g_{k+1}\|^2$  is better developed than the theory for the methods with numerator  $g_{k+1}^T y_k$  of  $\beta_k$ . However, the methods with  $g_{k+1}^T y_k$  in the numerator of  $\beta_k$  perform better in practice than

**Table 5.1** Choices of  $\beta_k$  in standard conjugate gradient methods

$\beta_k^{HS} = \frac{g_{k+1}^T y_k}{d_k^T y_k}$	Original <i>linear</i> conjugate gradient method of Hestenes and Stiefel (1952)
$\beta_k^{FR} = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$	First <i>nonlinear</i> conjugate gradient method by Fletcher and Reeves (1964)
$\beta_k^D = \frac{g_{k+1}^T \nabla^2 f(x_k) d_k}{d_k^T \nabla^2 f(x_k) d_k}$	Proposed by Daniel (1967). This conjugate gradient method requires evaluation of the Hessian
$\beta_k^{PRP} = \frac{g_{k+1}^T y_k}{g_k^T g_k}$	Proposed by Polak and Ribi��re (1969) and by Polyak (1969)
$\beta_k^{PRP+} = \max \left\{ 0, \frac{g_{k+1}^T y_k}{g_k^T g_k} \right\}$	Proposed by Powell (1984) and analyzed by Gilbert and Nocedal (1992)
$\beta_k^{CD} = \frac{g_{k+1}^T g_{k+1}}{-d_k^T g_k}$	Proposed by Fletcher (1987), known as CD (conjugate descent) method
$\beta_k^{LS} = \frac{g_{k+1}^T y_k}{-d_k^T g_k}$	Proposed by Liu and Storey (1991)
$\beta_k^{DY} = \frac{g_{k+1}^T g_{k+1}}{d_k^T y_k}$	Proposed by Dai and Yuan (1999)

the methods with  $\|g_{k+1}\|^2$  in the numerator of  $\beta_k$ . The general algorithm for the standard conjugate gradient methods is as follows.

**Algorithm 5.4** General nonlinear conjugate gradient

1. Choose an initial point  $x_0$  and a convergence tolerance  $\varepsilon > 0$  sufficiently small. Set  $d_0 = -g_0$  and  $k = 0$
2. Test a criterion for stopping the iterations. If this test is satisfied, then stop; otherwise, continue with step 3
3. Using the Wolfe line-search conditions, determine the stepsize  $\alpha_k$
4. Compute  $x_{k+1} = x_k + \alpha_k d_k$ ,  $f_{k+1}$ ,  $g_{k+1}$  and  $y_k = g_{k+1} - g_k$
5. Compute the conjugate gradient parameter  $\beta_k$
6. Compute the search direction  $d_{k+1} = -g_{k+1} + \beta_k d_k$
7. Restarting criterion. If  $|g_{k+1}^T g_k| > 0.2 \|g_{k+1}\|^2$  then set  $d_{k+1} = -g_{k+1}$
8. Set  $k = k + 1$  and continue with step 2

◆

Observe that a restarting criterion is used in step 6 of Algorithm 5.4. This is known as the Powell restart criterion, introduced by Powell (1977). Crowder and Wolfe (1969) proved that the standard conjugate gradient method without restart is at most linearly convergent. Yuan (1993) showed that the convergence rate of the conjugate gradient method without restart is exactly linear for uniformly convex quadratics. Cohen (1972) and McCormick and Ritter (1974) proved that the convergence rate of the conjugate gradient method may be improved from linear to  $n$ -step quadratic if the method is restarted with the negative gradient. Powell (1977) reported some numerical results with conjugate gradient methods, showing that the immediate reduction in the objective function with restart is usually smaller than the reduction without restart. Therefore, the current implementations of the conjugate gradient methods use this restart criterion. Some restart procedures for conjugate gradient methods can be found in Dai, Liao, and Li (2004).

### Conjugate Gradient Methods with $\|g_{k+1}\|^2$ in the Numerator of $\beta_k$

In this section, let us discuss the methods Fletcher-Reeves (FR), Conjugate Descent by Fletcher (CD), and Dai-Yuan (DY). A general characterization of these methods versus some other choices for the update parameter  $\beta_k$  is that the global convergence theorems require only the Lipschitz assumption, not the boundedness assumption.

#### The Fletcher-Reeves Method

This conjugate gradient method is defined by

$$x_{k+1} = x_k + \alpha_k d_k, \quad (5.113)$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k, \quad (5.114)$$

where the conjugate gradient parameter is computed as

$$\beta_k^{FR} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}. \quad (5.115)$$

The first global convergence result for the FR method under the exact line-search was given by Zoutendijk (1970). In other words, when the stepsize  $\alpha_k$  is the exact solution of the problem  $\min_{\alpha \geq 0} f(x_k + \alpha d_k)$ , then the FR method is global convergent. As usual, let  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  be the level set.

**Theorem 5.9** Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable on the level set  $S$  and consider that the FR method is implemented with the exact line-search. Then, the sequence  $\{x_k\}$  generated by the algorithm has at least an accumulation point which is a stationary point of function  $f$ , that is:

- (i) If  $\{x_k\}$  is a finite sequence, then  $x^*$  is a stationary point of  $f$ .
- (ii) If  $\{x_k\}$  is an infinite sequence, then this sequence has a limit point, and any limit point is a stationary point.

**Proof**

- (i) If  $\{x_k\}$  is a finite sequence, then, from the condition of the termination of iterations, it follows that the final point  $x^*$  satisfies  $\nabla f(x^*) = 0$ , i.e.,  $x^*$  is a stationary point of  $f$ .
- (ii) If  $\{x_k\}$  is an infinite sequence, then for any  $k$ ,  $\nabla f(x_k) \neq 0$ . Since  $d_{k+1} = -g_{k+1} + \beta_k d_k$  and the line-search is exact, that is,  $g_{k+1}^T d_k = 0$ , it follows that

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k = -\|g_{k+1}\|^2 < 0, \quad (5.116)$$

i.e.,  $d_{k+1}$  is a descent direction,  $\{f(x_k)\}$  is a monotone decreasing sequence, and consequently,  $\{x_k\} \subset S$ . Hence,  $\{x_k\}$  is bounded and has a limit. Let  $x^*$  be the limit of  $\{x_k\}$ . Then there exists a subsequence  $\{x_k\}_{K_1}$  convergent to  $x^*$ , where  $K_1$  is the set of indices corresponding to the subsequence from  $\{x_k\}$ . Since  $\{x_k\}_{K_1} \subset \{x_k\}$ , it follows that  $\{f(x_k)\}_{K_1} \subset \{f(x_k)\}$ . From the continuity of  $f$  we know that for  $k \in K_1$  (see Appendix A),

$$f(x^*) = f\left(\lim_{k \rightarrow \infty} x_k\right) = \lim_{k \rightarrow \infty} f(x_k) = f^*. \quad (5.117)$$

Similarly,  $\{x_{k+1}\}$  is a bounded sequence. Therefore, there exists a subsequence  $\{x_{k+1}\}_{K_2}$  convergent to  $\bar{x}^*$ , where  $K_2$  is the set of indices corresponding to the subsequence from  $\{x_{k+1}\}$ . In this case,

$$f(\bar{x}^*) = f\left(\lim_{k \rightarrow \infty} x_{k+1}\right) = \lim_{k \rightarrow \infty} f(x_{k+1}) = f^*. \quad (5.118)$$

Hence,

$$f(\bar{x}^*) = f(x^*) = f^*. \quad (5.119)$$

Now, let us prove by contradiction that  $\nabla f(x^*) = 0$ . Suppose that  $\nabla f(x^*) \neq 0$ . Then, for any  $\alpha$  small enough it follows that

$$f(x^* + \alpha d^*) < f(x^*). \quad (5.120)$$

Since for any  $\alpha > 0$ ,

$$f(x_{k+1}) = f(x_k + \alpha_k d_k) \leq f(x_k + \alpha d_k),$$

then, for  $k \in K_2$ , at limit for  $k \rightarrow \infty$  from (5.120), it results that

$$f(\bar{x}^*) \leq f(x^* + \alpha d^*) < f(x^*), \quad (5.121)$$

which contradicts (5.119). Therefore, this shows that  $\nabla f(x^*) = 0$ , that is,  $x^*$  is a stationary point of  $f$ . ♦

Powell (1977) showed that the FR method with the exact line-search is susceptible of jamming, i.e., along the iterations the algorithm could take many short steps without any significant progress to the minimum. The modest performances of the FR method can be explained by this jamming phenomenon, as it is detailed by Nocedal and Wright (2006).

The first global convergence result of the FR method under the inexact line-search was given by Al-Baali (1985). Using the strong Wolfe line-search

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k d_k^T g_k, \quad (5.122)$$

$$|\nabla f(x_k + \alpha_k d_k)^T d_k| \leq -\sigma d_k^T g_k, \quad (5.123)$$

with  $\sigma < 1/2$ , Al-Baali proved that the FR method generates sufficient descent directions.

**Theorem 5.10** Suppose that the conjugate gradient FR is implemented with the strong Wolfe line-search (5.122) and (5.123), where  $0 < \sigma < 1/2$ . Then the FR method generates descent directions  $d_k$  satisfying the following inequalities:

$$-\frac{1}{1-\sigma} \leq \frac{g_k^T d_k}{\|g_k\|^2} \leq \frac{2\sigma-1}{1-\sigma}, \quad (5.124)$$

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

**Proof** At first, notice that the function  $u(t) = (2t-1)/(1-t)$  on the interval  $[0, 1/2]$  is monotonically increasing and  $u(0) = -1$  and  $u(1/2) = 0$ . Therefore, since  $\sigma \in (0, 1/2)$ , it follows that

$$-1 < \frac{2\sigma-1}{1-\sigma} < 0. \quad (5.125)$$

The proof is given by induction as follows. For  $k = 0$  the middle term in (5.124) is  $-1$ , so, by using (5.125) both inequalities in (5.124) are satisfied. Now, assume that (5.124) holds for some  $k \geq 1$ . From the definition of  $\beta_k^{FR}$  given in (5.115) and from (5.114), it follows that

$$\frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} = -1 + \frac{g_{k+1}^T d_k}{\|g_k\|^2}. \quad (5.126)$$

From the second Wolfe condition (5.123) and from (5.126),

$$-1 + \sigma \frac{g_k^T d_k}{\|g_k\|^2} \leq \frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} \leq -1 - \sigma \frac{g_k^T d_k}{\|g_k\|^2}.$$

Taking into account the left hand side of the induction hypothesis (5.124) and substituting it in the above inequality, it follows that

$$-1 - \frac{\sigma}{1-\sigma} \leq \frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} \leq -1 + \frac{\sigma}{1-\sigma},$$

which shows that (5.124) holds for  $k + 1$  as well.  $\diamond$

The theorem shows that for the FR conjugate gradient method, the sufficient descent condition  $g_k^T d_k \leq -c \|g_k\|^2$  is satisfied. Therefore, from the Zoutendijk condition, it follows the global convergence of the FR method. Besides, the bounds on  $g_k^T d_k$  in (5.124) impose a limit on how fast  $\|d_k\|$  can grow along the iterations when the gradients are not small. For  $\sigma = 1/2$ ,  $d_k$  is a descent direction; however the analysis did not establish the sufficient descent.

The main difficulty in proving the global convergence of a conjugate gradient algorithm is to show that the search direction  $d_k$  is descent under the mild assumption on the line-search conditions. If the line-search is exact, i.e.,  $g_{k+1}^T d_k = 0$ , then  $d_{k+1}$  is exactly the steepest descent, since  $g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2$ . Using this property and supposing that the Assumption CG holds, Powell showed that the FR algorithm is globally convergent, i.e.,  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ . Al-Baali extended this result, showing that the descent property holds for all  $k$  if  $\alpha_k$  is determined by the strong Wolfe line-search.

The following theorem shows that under the inexact line-search, the search directions  $d_k$  generated by the FR method satisfy the descent condition  $g_k^T d_k < 0$ .

**Theorem 5.11** *If for all  $k$  the stepsize  $\alpha_k$  is determined by the strong Wolfe line-search (5.122) and (5.123), where  $\sigma \in (0, 1/2)$ , then for the FR method the following inequalities hold:*

$$-\sum_{j=0}^k \sigma^j \leq \frac{g_k^T d_k}{\|g_k\|^2} \leq -2 + \sum_{j=0}^k \sigma^j \quad (5.127)$$

for any  $k$ . As soon as  $g_k \neq 0$  for all  $k$ , the descent property of  $d_k$  is satisfied, i.e.,

$$g_k^T d_k < 0. \quad (5.128)$$

**Proof** The theorem is proved by induction. For  $k = 0$ ,  $d_0 = -g_0$ ,  $\sigma^0 = 1$  and therefore (5.127) and (5.128) are true. Now, suppose that (5.127) and (5.128) are true for any  $k \geq 0$ . From  $d_{k+1} = -g_{k+1} + \beta_k d_k$  and  $\beta_k = g_{k+1}^T g_{k+1} / g_k^T g_k$  (Fletcher-Reeves updating formula), it follows that

$$\frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} = -1 + \frac{g_{k+1}^T d_k}{\|g_k\|^2}. \quad (5.129)$$

From the second strong Wolfe condition  $|g_{k+1}^T d_k| \leq -\sigma g_k^T d_k$  and from (5.128),

$$-1 + \sigma \frac{g_k^T d_k}{\|g_k\|^2} \leq \frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} \leq -1 - \sigma \frac{g_k^T d_k}{\|g_k\|^2}. \quad (5.130)$$

From (5.127) it follows that

$$-\sum_{j=0}^{k+1} \sigma^j = -1 - \sigma \sum_{j=0}^k \sigma^j \leq \frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} \leq -2 + \sum_{j=0}^{k+1} \sigma^j,$$

showing that (5.127) holds for  $k + 1$ . On the other hand, since

$$\frac{g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} \leq -2 + \sum_{j=0}^{k+1} \sigma^j \quad (5.131)$$

and

$$\sum_{j=0}^{k+1} \sigma^j < \sum_{j=0}^{\infty} \sigma^j = \frac{1}{1-\sigma}, \quad (5.132)$$

where  $\sigma \in [0, 1/2]$ , from  $1 - \sigma > 1/2$  it results that

$$-2 + \sum_{j=0}^{k+1} \sigma^j < 0.$$

Therefore, from (5.128),  $g_{k+1}^T d_{k+1} < 0$  is obtained, proving the theorem.  $\blacklozenge$

With this result, the global convergence of the FR method with the inexact line-search can be proved as in the following theorem (Al-Baali, 1985).

**Theorem 5.12** *Assume that the function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable and the level set  $S$  is bounded. Suppose that  $\alpha_k$  is determined by the strong Wolfe conditions (5.122) and (5.123), where  $\rho < \sigma < 1/2$ . Then the sequence  $\{x_k\}$  generated by the Fletcher-Reeves method is global convergent, i.e.,*

$$\liminf_{k \rightarrow \infty} \|g_k\| = 0. \quad (5.133)$$

**Proof** From the second strong Wolfe condition (5.123) and from the relations (5.127) and (5.132), it follows that

$$|g_{k+1}^T d_k| \leq -\sigma g_k^T d_k \leq \sigma \|g_k\|^2 \sum_{j=0}^k \sigma^j \leq \frac{\sigma}{1-\sigma} \|g_k\|^2.$$

From the relations  $d_{k+1} = -g_{k+1} + \beta_k d_k$  and  $\beta_k = g_{k+1}^T g_{k+1} / g_k^T g_k$  (Fletcher-Reeves formula), it results that

$$\begin{aligned} \|d_{k+1}\|^2 &= \|g_{k+1}\|^2 - 2\beta_k^F g_{k+1}^T d_k + (\beta_k^F)^2 \|d_k\|^2 \\ &\leq \|g_{k+1}\|^2 + \frac{2\sigma}{1-\sigma} \|g_{k+1}\|^2 + (\beta_k^F)^2 \|d_k\|^2 \\ &= \left(\frac{1+\sigma}{1-\sigma}\right) \|g_{k+1}\|^2 + (\beta_k^F)^2 \|d_k\|^2. \end{aligned}$$

But, from the Fletcher-Reeves updating formula (5.115), it is clear that

$$(\beta_k^{FR})^2 (\beta_{k-1}^{FR})^2 \cdots (\beta_{k-i}^{FR})^2 = \frac{\|g_{k+1}\|^4}{\|g_{k-i}\|^4}.$$

Therefore, by recurrence, the following is obtained:

$$\|d_{k+1}\|^2 \leq \left(\frac{1+\sigma}{1-\sigma}\right) \|g_{k+1}\|^4 \left( \sum_{j=0}^{k+1} \|g_j\|^2 \right). \quad (5.134)$$

Now, let us prove (5.133) by contradiction. Suppose that (5.133) is not true. Then there exists a positive constant  $\epsilon > 0$  so that for all  $k$  sufficiently large,

$$\|g_k\| \geq \epsilon > 0. \quad (5.135)$$

Since on the level set  $S$  the gradient  $g_k$  is upper bounded, from (5.134) it follows that

$$\|d_k\|^2 \leq c_1 k, \quad (5.136)$$

where  $c_1$  is a positive constant. Hence, from (5.127) and (5.132),

$$\begin{aligned} \cos \theta_k &= -\frac{g_k^T d_k}{\|g_k\| \|d_k\|} \geq \left(2 - \sum_{j=0}^k \sigma^j\right) \frac{\|g_k\|}{\|d_k\|} \\ &\geq \left(\frac{1-2\sigma}{1-\sigma}\right) \frac{\|g_k\|}{\|d_k\|}. \end{aligned} \quad (5.137)$$

Since  $\sigma < 1/2$ , introducing (5.135) and (5.136) in (5.137), it results that

$$\sum_k \cos^2 \theta_k \geq \left(\frac{1-2\sigma}{1-\sigma}\right)^2 \sum_k \frac{\|g_k\|^2}{\|d_k\|^2} \geq c_2 \sum_k \frac{1}{k}, \quad (5.138)$$

where  $c_2$  is a positive constant. Therefore, the series  $\sum_k \cos^2 \theta_k$  is divergent. Let  $M$  be an upper bound of  $\|\nabla^2 f(x)\|$  on the level set  $S$ . Then,

$$g_{k+1}^T d_k = (g_k + \alpha_k \nabla^2 f(x_k) d_k)^T d_k \leq g_k^T d_k + \alpha_k M \|d_k\|^2.$$

From the second strong Wolfe condition (5.123) written as

$$\sigma g_k^T d_k \leq g_{k+1}^T d_k \leq -\sigma g_k^T d_k$$

it follows that

$$\alpha_k \geq -\frac{1-\sigma}{M \|d_k\|^2} g_k^T d_k. \quad (5.139)$$

Introducing this value of  $\alpha_k$  from (5.139) in the first Wolfe condition (5.122), it results that

$$f(x_{k+1}) \leq f(x_k) - \frac{(1-\sigma)\rho}{M} \left( \frac{g_k^T d_k}{\|d_k\|} \right)^2 = f(x_k) - c_3 \|g_k\|^2 \cos^2 \theta_k,$$

where  $c_3 = \frac{(1-\sigma)\rho}{M} > 0$ . Since  $f$  is lower bounded, it follows that the series  $\sum_k \|g_k\|^2 \cos^2 \theta_k$  is convergent. Hence, from (5.135), the series  $\sum_k \cos^2 \theta_k$  is convergent. But this contradicts (5.138), thus proving the theorem.  $\blacklozenge$

### The CD Method

The CD (Conjugate Descent) method elaborated by Fletcher (1987) is very close to the FR method. The conjugate gradient parameter  $\beta_k$  is computed as

$$\beta_k^{CD} = \frac{\|g_{k+1}\|^2}{-d_k^T g_k}. \quad (5.140)$$

Under the exact line-search,  $\beta_k^{CD} = \beta_k^{FR}$ . The difference between CD and FR is that in CD with strong Wolfe line-search, the sufficient descent condition  $g_k^T d_k \leq -c \|g_k\|^2$  holds. In this case, the constraint  $\sigma \leq 1/2$  that arose in the FR method is not necessary for the CD method. For a line-search that satisfies the generalized Wolfe conditions  $\sigma_1 d_k^T g_k \leq d_k^T g_{k+1} \leq -\sigma_2 d_k^T g_k$ , with  $\sigma_1 < 1$  and  $\sigma_2 = 0$ , it can be shown that  $0 \leq \beta_k^{CD} \leq \beta_k^{FR}$ . Therefore, from the analysis given by Al-Baali (1985) the global convergence is achieved. On the other hand, if  $\sigma_1 \geq 1$  or  $\sigma_2 > 0$ , then Dai and Yuan (1996) constructed numerical examples where  $\|d_k\|^2$  increases exponentially, and therefore the CD method converges to a point where the gradient does not vanish. In particular, the CD method may not converge to a stationary point under the strong Wolfe line-search. Details are given by Hager and Zhang (2006b).

### The Dai-Yuan Method

In the DY method elaborated by Dai and Yuan (1999), the conjugate gradient parameter  $\beta_k$  is computed as

$$\beta_k^{DY} = \frac{\|g_{k+1}\|^2}{d_k^T y_k}. \quad (5.141)$$

The DY method always generates descent directions under the Wolfe line-search. Besides, when the Lipschitz assumption holds, the DY is globally convergent.

To get (5.141), let us suppose that the current search direction  $d_k$  is descent, i.e.,  $d_k^T g_k < 0$  and we are interested in finding a  $\beta_k$  for which the new search direction  $d_{k+1}$  is also descent. This requires that

$$-\|g_{k+1}\|^2 + \beta_k g_{k+1}^T d_k < 0.$$

Assume that  $\beta_k > 0$  and denote  $\tau_k = \|g_{k+1}\|^2 / \beta_k$ . Then the above inequality is equivalent to  $\tau_k > g_{k+1}^T d_k$ . Therefore, we can consider  $\tau_k = d_k^T y_k$ , which gives the DY updating formula (5.141).

A new representation of  $\beta_k^{DY}$  can immediately be obtained. From (5.114) and (5.141), the following can be written:

$$g_{k+1}^T d_{k+1} = \frac{\|g_{k+1}\|^2}{d_k^T y_k} g_k^T d_k = \beta_k^{DY} g_k^T d_k. \quad (5.142)$$

Therefore,  $\beta_k^{DY}$  is

$$\beta_k^{DY} = \frac{g_{k+1}^T d_{k+1}}{g_k^T d_k}. \quad (5.143)$$

Observe that (5.141) is well defined because the Wolfe line-search implies that  $d_k^T y_k > 0$ . If the line-search in this method is exact, then the DY method is the same as the FR method. The convergence of this method is given by the following theorem.

**Theorem 5.13** Suppose that the initial point  $x_0$  satisfies the Assumption CG and let  $\{x_k\}$  be the sequence generated by the general algorithm (5.113) and (5.114), where  $\beta_k$  is computed as in (5.143). Then the algorithm either terminates at a stationary point off or converges in the sense

$$\liminf_{k \rightarrow \infty} \|g_k\| = 0. \quad (5.144)$$

**Proof** If the algorithm does not terminate after a finite number of iterations, then  $\|g_k\| > 0$  for all  $k$ . Firstly, let us show that the search directions are descent, i.e.,

$$g_k^T d_k < 0 \quad (5.145)$$

for all  $k$ . For  $k = 1$  the above inequality (5.145) is satisfied. Now, let us prove it for all  $k > 1$  by induction. Assume that (5.145) holds for  $k$ . From the second Wolfe line-search condition  $\nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma d_k^T g_k$ , it follows that

$$d_k^T y_k \geq (\sigma - 1) d_k^T g_k > 0. \quad (5.146)$$

From (5.142) it follows that (5.145) holds for  $k + 1$ . Hence, the search direction given by  $\beta_k^{DY}$  is descent for all the values of  $k$ .

Now, (5.114) can be written as

$$d_{k+1} + g_{k+1} = \beta_k d_k. \quad (5.147)$$

Squaring both sides of (5.147), it follows that

$$\|d_{k+1}\|^2 = \beta_k^2 \|d_k\|^2 - 2g_{k+1}^T d_{k+1} - \|g_{k+1}\|^2. \quad (5.148)$$

Dividing both sides of (5.148) by  $(g_{k+1}^T d_{k+1})^2$  and applying (5.143),

$$\begin{aligned} \frac{\|d_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} &= \frac{\|d_k\|^2}{(g_k^T d_k)^2} - \frac{2}{g_{k+1}^T d_{k+1}} - \frac{\|g_{k+1}\|^2}{(g_{k+1}^T d_{k+1})^2} \\ &= \frac{\|d_k\|^2}{(g_k^T d_k)^2} - \left( \frac{1}{\|g_{k+1}\|} + \frac{\|g_{k+1}\|}{g_{k+1}^T d_{k+1}} \right)^2 + \frac{1}{\|g_{k+1}\|^2} \\ &\leq \frac{\|d_k\|^2}{(g_k^T d_k)^2} + \frac{1}{\|g_{k+1}\|^2}. \end{aligned} \quad (5.149)$$

But,  $\|d_0\|^2 / (g_0^T d_0) = 1 / \|g_0\|^2$ , then (5.149) shows that

$$\frac{\|d_k\|^2}{(g_k^T d_k)^2} \leq \sum_{i=0}^k \frac{1}{\|g_i\|^2}, \quad (5.150)$$

for all  $k$ . If the theorem is not true, then there exists a positive constant  $c > 0$  so that

$$\|g_k\| \geq c \quad (5.151)$$

for all  $k$ . Therefore, from (5.150) and (5.151), it follows that

$$\frac{\|d_k\|^2}{(g_k^T d_k)^2} \leq \frac{k}{c^2},$$

which implies that

$$\sum_{k \geq 0} \frac{(g_k^T d_k)^2}{\|d_k\|^2} = \infty. \quad (5.152)$$

Observe that the relation (5.152) contradicts the condition (5.61). This contradiction proves the theorem.  $\diamond$

### Conjugate Gradient Methods with $g_{k+1}^T y_k$ in the Numerator of $\beta_k$

In the following, let us consider the conjugate gradient methods Hestenes-Stiefel (HS), Polak-Ribi  re-Polyak (PRP), and Liu-Storey (LS). Even if for the methods with  $\|g_{k+1}\|^2$  in the numerator of  $\beta_k$  a strong convergence theory has been developed, these methods have modest performances, mainly because of the jamming phenomenon. Namely, these methods begin to take small steps without making significant progress to the minimum. On the other hand, the HS, PRP, and LS methods, having  $g_{k+1}^T y_k$  in the numerator of  $\beta_k$ , possess a built-in restart feature that addresses the jamming. When the step  $s_k = x_{k+1} - x_k$  is small (i.e., close to the minimum), then the factor  $y_k = g_{k+1} - g_k$  in the numerator of  $\beta_k$  tends to zero. Hence,  $\beta_k$  becomes small, and the new search direction  $d_{k+1}$  is essentially the steepest descent direction  $-g_{k+1}$ . This property of the PRP method is important for its analysis. Such a method with this property is said to have Property (\*). These methods automatically adjust  $\beta_k$  in order to avoid jamming. Therefore, the numerical performance of these methods is better than the numerical performance of the methods with  $\|g_{k+1}\|^2$  in the numerator of  $\beta_k$ .

#### The Polak-Ribi  re-Polyak Method

The Polak-Ribi  re-Polyak conjugate gradient method is defined by (5.113) and (5.114), where the conjugate gradient parameter is computed as

$$\beta_k^{PRP} = \frac{g_{k+1}^T y_k}{g_k^T g_k}. \quad (5.153)$$

If  $f$  is strongly convex and the line-search is exact, then Polak and Ribi  re (1969) and Polyak (1969) established the global convergence of the PRP method. Powell (1977) proved that for a general nonlinear function  $f$ , if (i) the stepsize  $s_k = x_{k+1} - x_k$  tends to zero, (ii) the line-search is exact and (iii) the Lipschitz continuity holds, then the PRP method is globally convergent. On the other hand, in a laborious paper Powell (1984) constructed a counter-example with three variables and showed that the PRP method may cycle infinitely without approaching any solution. Therefore, the assumption that the stepsize tends to zero is needed for convergence.

Later on, under the assumption that the search direction is a descent direction, Yuan (1993) established the global convergence of the PRP method for strongly convex functions and the Wolfe line-search. However, Dai (1997), in his Ph.D. Thesis, presented an example which showed that even when the objective function is strongly convex and  $\sigma \in (0, 1)$  is sufficiently small, the PRP method may still fail by generating an ascent search direction. Dai, Han, Liu, Sun, Yin, and Yuan (1999) constructed an example showing that the boundedness of the level set is necessary for the

convergence of the PRP method even if the line-search is exact. Therefore, the convergence of the PRP method is not certain. However, this method proved to be one of the most efficient for solving large-scale unconstrained optimization problems.

In order to prove the convergence of the PRP method, we need a technical result presented in the following proposition.

**Proposition 5.12** *Let  $\nabla f(x)$  be uniformly continuous on the level set  $S$ . Consider the angle  $\theta_k$  between  $d_k$  and  $-\nabla f(x_k)$ , where  $d_k$  is a descent direction. If*

$$\theta_k \leq \frac{\pi}{2} - \mu, \text{ for such } \mu > 0, \quad (5.154)$$

then  $\nabla f(x_k) = 0$  for a certain  $k$ , or  $f(x_k) \rightarrow -\infty$ , or  $\nabla f(x_k) \rightarrow 0$ .

**Proof** Suppose that for all  $k$ ,  $\nabla f(x_k) \neq 0$  and that  $f(x_k)$  is lower bounded. Since  $\{f(x_k)\}$  is monotone decreasing, this sequence has a limit. Hence

$$f(x_k) - f(x_{k+1}) \rightarrow 0. \quad (5.155)$$

Suppose, by contradiction, that  $\nabla f(x_k) \rightarrow 0$  is not true. Then, there exists an  $\varepsilon > 0$  so that  $\|\nabla f(x_k)\| \geq \varepsilon$ . Therefore,

$$-\frac{\nabla f(x_k)^T d_k}{\|d_k\|} = \|\nabla f(x_k)\| \cos \theta_k \geq \varepsilon \sin \mu \equiv \varepsilon_1 \quad (5.156)$$

Observe that

$$\begin{aligned} f(x_k + \alpha d_k) &= f(x_k) + \alpha \nabla f(\xi_k)^T d_k \\ &= f(x_k) + \alpha \nabla f(x_k)^T d_k + \alpha [\nabla f(\xi_k) - \nabla f(x_k)]^T d_k \\ &\leq f(x_k) + \alpha \|d_k\| \left( \frac{\nabla f(x_k)^T d_k}{\|d_k\|} + \|\nabla f(\xi_k) - \nabla f(x_k)\| \right), \end{aligned} \quad (5.157)$$

where  $\xi_k$  is on the line segment connecting  $x_k$  and  $x_k + \alpha d_k$ . Since  $\nabla f(x)$  is uniformly continuous on  $S$ , there exists  $\bar{\alpha}$  so that, when  $0 \leq \alpha \|d_k\| \leq \bar{\alpha}$  we have

$$\|\nabla f(\xi_k) - \nabla f(x_k)\| \leq \frac{1}{2} \varepsilon_1. \quad (5.158)$$

From (5.155)–(5.158) it follows that

$$f\left(x_k + \bar{\alpha} \frac{d_k}{\|d_k\|}\right) \leq f(x_k) + \bar{\alpha} \left( \frac{\nabla f(x_k)^T d_k}{\|d_k\|} + \frac{1}{2} \varepsilon_1 \right) \leq f(x_k) - \frac{1}{2} \bar{\alpha} \varepsilon_1.$$

Hence,

$$f(x_{k+1}) \leq f\left(x_k + \bar{\alpha} \frac{d_k}{\|d_k\|}\right) \leq f(x_k) - \frac{1}{2} \bar{\alpha} \varepsilon_1,$$

which contradicts (5.155). Therefore,  $\nabla f(x_k) \rightarrow 0$ , thus completing the proof.  $\blacklozenge$

With this, for strongly convex functions and under the exact line-search, the following result on the convergence of the PRP method can be proved.

**Theorem 5.14** Consider  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  twice continuously differentiable and suppose that the level set  $S$  is bounded. Assume that there exists a positive constant  $m > 0$  so that for  $x \in S$  and any  $y \in \mathbb{R}^n$ ,

$$m\|y\|^2 \leq y^T \nabla^2 f(x)y. \quad (5.159)$$

Then, the sequence  $\{x_k\}$  generated by the PRP method with exact line-search converges to the unique minimum  $x^*$  of function  $f$ .

**Proof** From Proposition 5.12 it follows that it is enough to prove (5.154), that is, there exists a positive constant  $\omega > 0$ , so that

$$-g_{k+1}^T d_{k+1} \geq \omega \|g_{k+1}\| \|d_{k+1}\|, \quad (5.160)$$

i.e.,  $\cos \theta_k \geq \omega > 0$ . Observe that  $g_k \rightarrow 0$  and  $g(x^*) = 0$ . From (5.159) it follows that  $\{x_k\} \rightarrow x^*$ , which is the unique minimum of function  $f$ . Since the line-search is exact, from  $d_{k+1} = -g_{k+1} + \beta_k d_k$  and from the fact that  $g_{k+1}^T d_k = 0$ , it follows that  $g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2$ . Then (5.160) is equivalent to

$$\frac{\|g_{k+1}\|}{\|d_{k+1}\|} \geq \omega. \quad (5.161)$$

Having in view that  $d_{k+1} = -g_{k+1} + \beta_k d_k$  and the line-search is exact, from (5.6) the following value for  $\alpha_k$  is obtained

$$\alpha_k = -\frac{g_k^T d_k}{d_k^T A_k d_k} = \frac{\|g_k\|^2}{d_k^T A_k d_k}, \quad (5.162)$$

where

$$A_k = \int_0^1 \nabla^2 f(x_k + t\alpha_k d_k) dt. \quad (5.163)$$

Using the mean value theorem (see Appendix A), from (5.163),

$$g_{k+1} - g_k = \nabla f(x_k + \alpha_k d_k) - \nabla f(x_k) = \alpha_k A_k d_k. \quad (5.164)$$

Therefore,  $\beta_k^{PRP}$  can be expressed as

$$\begin{aligned} \beta_k^{PRP} &= \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k} = \alpha_k \frac{g_{k+1}^T A_k d_k}{\|g_k\|^2} \\ &= \frac{g_{k+1}^T A_k d_k}{d_k^T A_k d_k}. \end{aligned} \quad (5.165)$$

Since the level set  $S$  is bounded, there exists a positive constant  $M > 0$ , so that for  $x \in S$  and any  $y \in \mathbb{R}^n$ ,

$$y^T A(x) y \leq M \|y\|^2 \quad (5.166)$$

Using the above relations, the following bound for  $\beta_k^{PRP}$  can be established:

$$|\beta_k^{PRP}| \leq \frac{\|g_{k+1}\| \|A_k d_k\|}{m \|d_k\|^2} \leq \frac{M}{m} \frac{\|g_{k+1}\|}{\|d_k\|}. \quad (5.167)$$

Hence,

$$\begin{aligned} \|d_{k+1}\| &\leq \|g_{k+1}\| + |\beta_k^{PRP}| \|d_k\| \\ &\leq \|g_{k+1}\| + \frac{M}{m} \|g_{k+1}\| = \left(1 + \frac{M}{m}\right) \|g_{k+1}\|, \end{aligned}$$

that is,

$$\frac{\|g_{k+1}\|}{\|d_{k+1}\|} \geq \left(1 + \frac{M}{m}\right)^{-1}, \quad (5.168)$$

showing that (5.161) holds, where  $\omega = m/(m + M)$ .  $\diamond$

Powell (1984, 1986b) introduced the PRP+ method as

$$\beta_k^{PRP+} = \max \{0, \beta_k^{PRP}\}. \quad (5.169)$$

Later on, Gilbert and Nocedal (1992) proved the global convergence of the conjugate gradient methods with nonnegative  $\beta_k$  under inexact line-search. The PRP+ method was introduced to rectify the convergence failure of the PRP method when implemented with Wolfe line-search. A modified Polak-Ribiére-Polyak conjugate gradient algorithm for unconstrained optimization was given by Andrei (2011a).

### The Hestenes-Stiefel Method

The HS conjugate gradient method is defined by (5.113) and (5.114), where the conjugate gradient parameter is computed as

$$\beta_k^{HS} = \frac{g_{k+1}^T y_k}{d_k^T y_k}. \quad (5.170)$$

The HS method has the property that the conjugacy condition

$$d_{k+1}^T y_k = 0 \quad (5.171)$$

is always satisfied, independent of the line-search used. For the exact line-search,  $\beta_k^{HS} = \beta_k^{PRP}$ . Therefore, the global convergence properties of the HS method are similar to the global convergence properties of the PRP method. In particular, by Powell's example (1984), the HS method with exact line-search may not converge for general nonlinear functions.

### The Liu-Storey Method

This conjugate gradient method is defined by (5.113) and (5.114), where the conjugate gradient parameter is computed as

$$\beta_k^{LS} = \frac{g_{k+1}^T y_k}{-d_k^T g_k}. \quad (5.172)$$

For the exact line-search, the LS method is identical to the PRP method. Liu and Storey (1991) studied this method, proving its global convergence. The techniques developed for the analysis of the PRP method may be applied to the LS method.

## Numerical Study: Standard Conjugate Gradient Methods

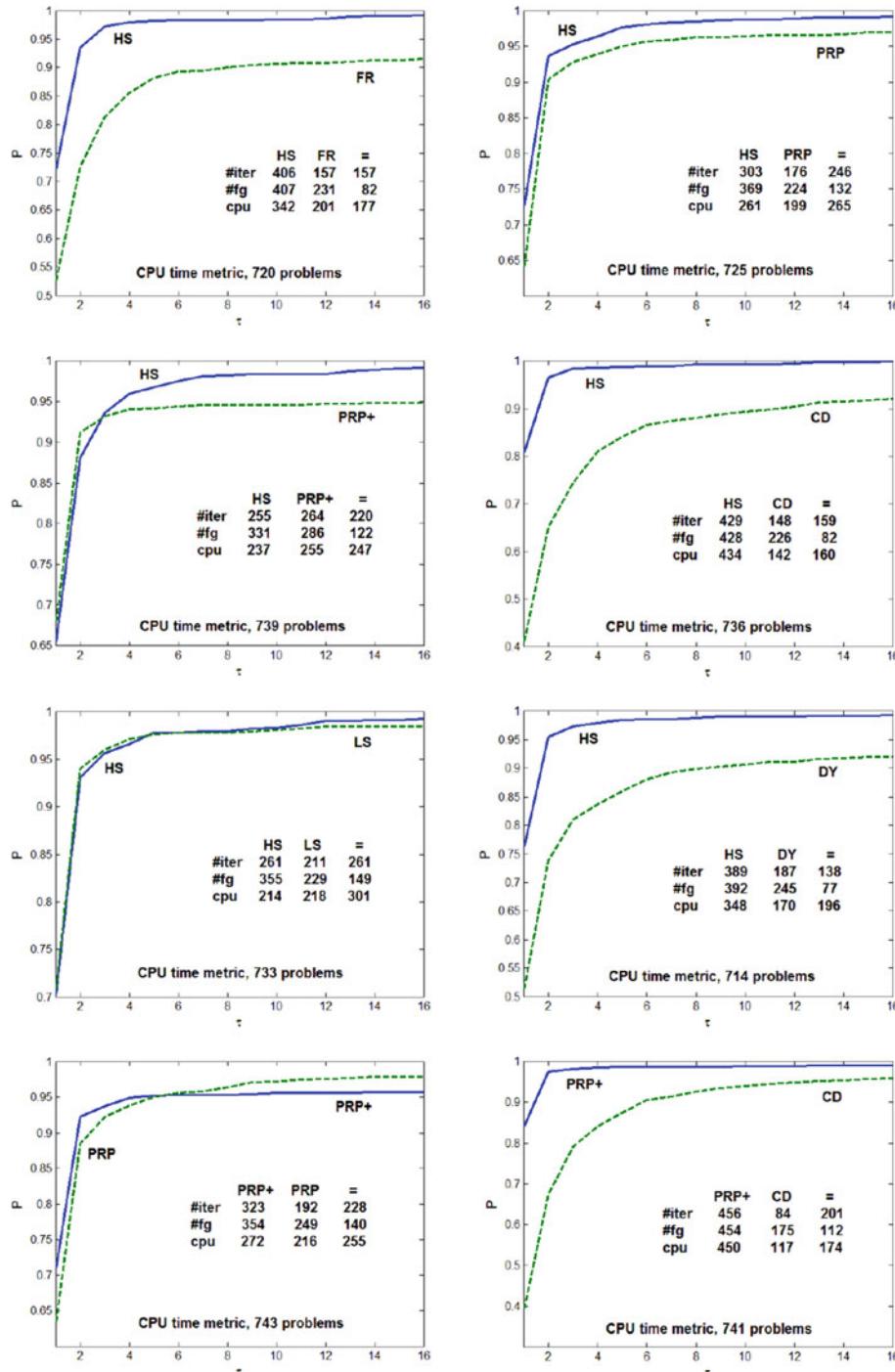
In this section, some numerical results with the standard conjugate gradient methods are presented. For this, consider 80 unconstrained optimization problems from the UOP collection (Andrei, 2020a), each with the number of variables  $n = 1000, 2000, \dots, 10000$ . The comparisons of algorithms are given in the context of Remark 1.1. The maximum number of iterations was limited to 2000.

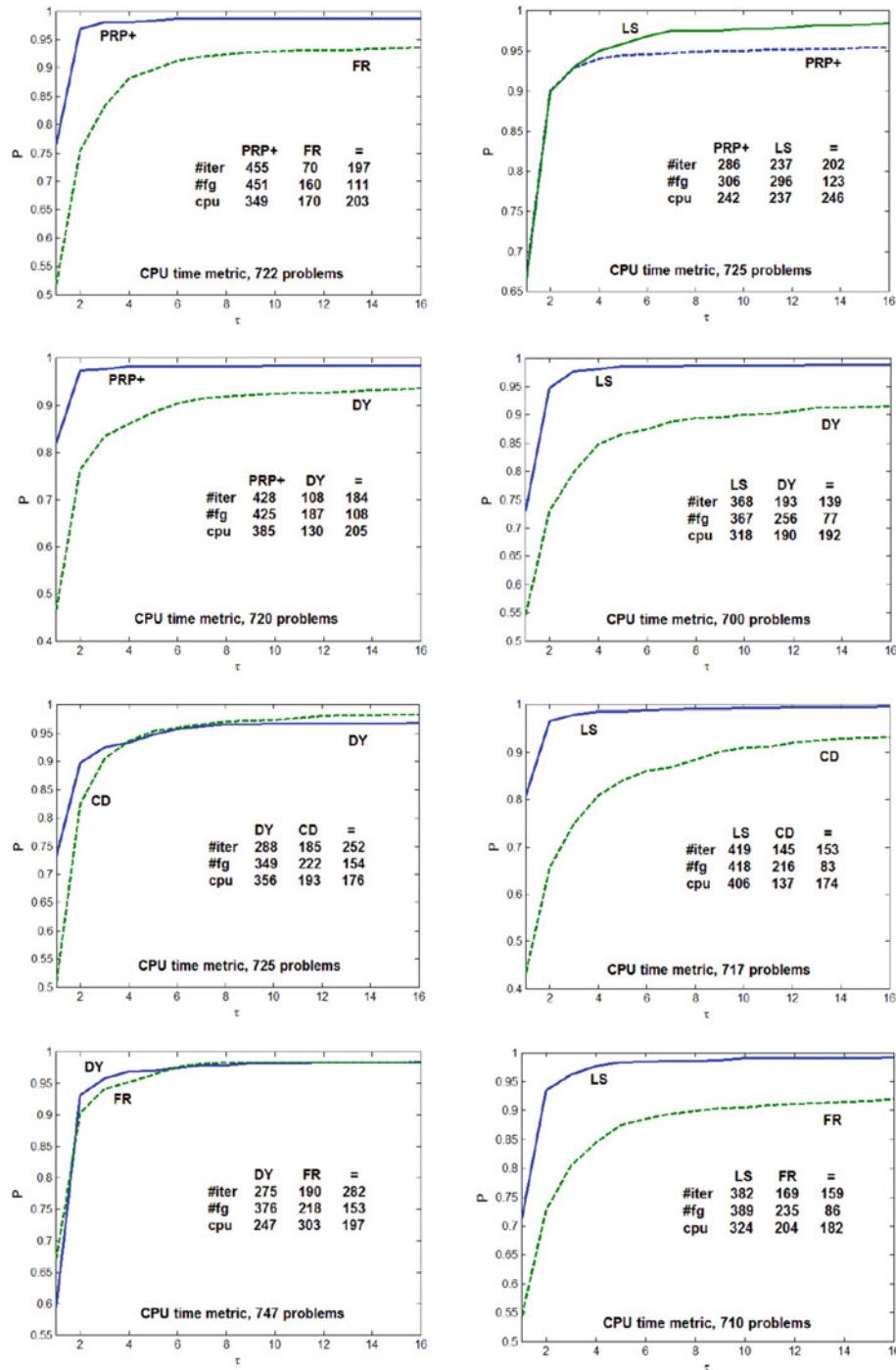
Figures 5.5 and 5.6 present the Dolan and Moré's performance profiles of the standard conjugate gradient methods. The tables inside the plots show the performances of the algorithms subject to the number of iterations (#iter), the number of function and its gradient evaluations (#fg), and subject to the CPU time metric (cpu) in seconds. When comparing HS versus FR (see Fig. 5.5) subject to the number of iterations, we can see that HS was better in 406 problems (i.e., it achieved the minimum number of iterations in 406 problems). FR was better in 157 problems. Both methods achieved the same number of iterations in 157 problems, etc. Out of 800 problems considered in this numerical experiment, only for 720 problems does the criterion (1.3) hold. From Fig. 5.5 we can see that, subject to the CPU time metric, HS is the fastest. Close to HS are PRP+ and LS. Observe that HS is much better than FR, CD, and DY. One explanation is that the HS method satisfies the conjugacy condition (5.171), independent of the line-search. From Figs. 5.5 and 5.6, we notice that PRP+ is more efficient than FR, PRP, CD, and DY. Close to PRP+ is LS, but LS is slightly more robust. From Fig. 5.6 we can see that LS is better than FR, CD, and DY. Close to CD is DY. At least, for this set of 800 unconstrained optimization problems, HS and PRP+, are the best methods. HS, PRP+ and LS automatically adjust  $\beta_k$  to avoid jamming. Figure 5.7 presents a global comparison of the standard conjugate gradient methods. Note that *there is an experimental confirmation of the classification of conjugate gradient methods in two classes according to the formula for  $\beta_k$  computation: with  $\|g_{k+1}\|^2$  or with  $g_{k+1}^T y_k$  at the numerator of  $\beta_k$ .*

We can see that HS, PRP, PRP+, and LS (all with  $g_{k+1}^T y_k$  at the numerator of  $\beta_k$ ) are more efficient and more robust subject to the CPU time metric than FR, CD, and DY (all with  $\|g_{k+1}\|^2$  at the numerator of  $\beta_k$ ). Despite the strong convergence theory that has been developed for methods with  $\|g_{k+1}\|^2$  in the numerator of  $\beta_k$ , these methods have modest numerical performances in comparison with the methods with  $g_{k+1}^T y_k$  at the numerator of  $\beta_k$ . Some more details on the performance profiles of conjugate gradient algorithms for unconstrained optimization are presented by Andrei (2008e).

In the second set of numerical experiments, let us present the performances of the standard conjugate gradient methods for solving five applications from the MINPACK-2 collection described in Appendix D, where  $nx = 200$  and  $ny = 200$ . The conjugate gradient algorithms implement the same stopping criterion  $\|g_k\|_\infty \leq \varepsilon_g$ , where  $\varepsilon_g = 10^{-6}$  and the stepsize is computed by the Wolfe line-search (5.49) and (5.50) with  $\rho = 0.0001$  and  $\sigma = 0.9$ .

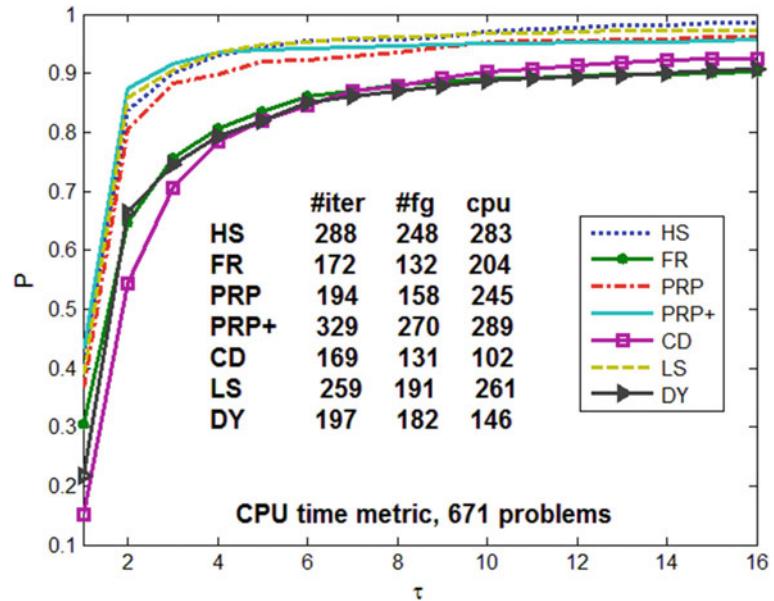
Tables 5.2, 5.3, and 5.4 present the performances of the standard conjugate gradient methods for solving these applications. In these tables,  $n$  is the number of variables of the application, #iter is the number of iterations, #fg the number of function and its gradient evaluations, and cpu is the CPU time (in seconds) for solving the application. The last line in these tables contains the total number of iterations, the total number of function and its gradient evaluations, and the total CPU time for solving these five applications for each conjugate gradient method. Observe that for solving them, CD, HS, and PRP require the minimum time. For example, CD needs 137.09 seconds, HS 139.65, and PRP

**Fig. 5.5** Performance profiles of the standard conjugate gradient methods



**Fig. 5.6** Performance profiles of the standard conjugate gradient methods

**Fig. 5.7** Performance profiles of seven standard conjugate gradient methods



**Table 5.2** Performances of HS, FR, and PRP for solving five applications from the MINPACK-2 collection

	<i>n</i>	HS			FR			PRP		
		#iter	#fg	cpu	#iter	#fg	cpu	#iter	#fg	cpu
A1	40,000	359	457	8.89	1082	1164	24.05	601	857	5.73
A2	40,000	1286	1650	34.65	2698	2885	35.40	1036	1477	12.84
A3	40,000	1510	1730	45.55	20001	25625	411.04	3001	4035	65.31
A4	40,000	841	1069	41.32	2070	2167	83.06	809	1148	43.91
A5	40,000	583	714	9.24	1830	1931	24.65	668	927	11.63
Total	—	4579	5620	139.65	27681	33772	578.20	6115	8444	139.42

**Table 5.3** Performances of PRP+ and CD for solving five applications from the MINPACK-2 collection

	<i>n</i>	PRP+			CD		
		#iter	#fg	cpu	#iter	#fg	cpu
A1	40,000	604	862	6.67	944	1097	12.80
A2	40,000	2335	1901	21.05	1887	2138	32.17
A3	40,000	3219	4306	69.73	2148	2287	37.52
A4	40,000	915	1314	50.17	1032	1148	43.99
A5	40,000	810	1125	14.09	759	836	10.61
Total	—	7883	9508	161.71	6770	7506	137.09

139.42 seconds. PRP+ and LS have comparable performances. However, FR and DY need the longest time for solving all these five applications. Details on the performances of conjugate gradient algorithms for solving the applications from the MINPACK-2 collection are found in (Andrei, 2006b).

**Table 5.4** Performances of LS and DY for solving five applications from the MINPACK-2 collection

	<i>n</i>	LS			DY		
		#iter	#fg	cpu	#iter	#fg	cpu
A1	40,000	642	886	10.51	464	488	6.08
A2	40,000	1085	1526	19.01	1031	1062	14.72
A3	40,000	2398	3120	50.76	8393	8423	141.57
A4	40,000	1455	2033	77.50	886	909	35.22
A5	40,000	591	821	10.25	2382	2410	31.61
Total	—	6171	8386	168.03	13156	13292	229.20

## 5.5 Hybrid Conjugate Gradient Methods

The idea behind the hybrid conjugate gradient methods is to combine the standard conjugate gradient methods in order to exploit the attractive features of each of them. The standard conjugate gradient methods may be combined in two distinct ways, thus obtaining two classes of hybrid conjugate gradient methods. The first class is based on the *projection concept*. They consider a pair of standard conjugate gradient methods and use one of them as soon as a certain criterion has been satisfied. When the criterion is violated, then the other conjugate gradient method from the pair is used. The hybrid conjugate gradient methods based on the projection concept have a simple algebraic expression. As soon as a conjugate gradient method has entered a jamming phase, the hybrid scheme triggers another conjugate gradient method in the pair, thus trying to improve the numerical performances of the hybrid one. The second class is based on the *convex combination* of the standard methods. In this case, some standard conjugate gradient methods are combined in a convex manner, thus obtaining a hybrid conjugate gradient method. In general, the hybrid methods are more efficient and more robust than the standard ones (Andrei, 2020a).

### Hybrid Conjugate Gradient Methods Based on the Projection Concept

Table 5.5 presents some hybrid selections of the parameter  $\beta_k$  based on the projection of the standard conjugate gradient methods presented in Table 5.1.

As it can be seen, there is a large variety of hybrid conjugate gradient methods. Their purpose is to combine the properties of the standard ones in order to get new ones, rapidly convergent to the solution. The idea is to avoid jamming. As we know, the FR method has strong convergence properties, but it may not perform well in computational experiments. On the other hand, although the PRP method and the HS method may not generally converge, they often perform better than FR. Therefore, the combination of these methods tries to exploit the attractive features of each one. Thus, in the hybrid conjugate gradient method TAS proposed by Touati-Ahmed and Storey (1990), if the iterations are affected by jamming, the method commutes from FR to PRP.

The same motivation is for the hybrid computational scheme PRP-FR proposed by Hu and Storey (1991). Indeed, the PRP method possesses a built-in restart feature that directly addresses the jamming. When the step  $s_k$  is small, the factor  $y_k$  in the numerator of  $\beta_k^{PRP}$  tends to zero. Hence, in this case, the search direction  $d_{k+1}$  computed as in (5.3) with  $\beta_k = \beta_k^{PRP}$  is essentially the steepest descent direction  $-g_{k+1}$ . The DY method has even better global convergence properties than the FR method. Consequently, Dai and Yuan (2001) combined their algorithm with the HS algorithm and

**Table 5.5** Hybrid selection of  $\beta_k$  based on the projection concept

$\beta_k^{TAS} = \begin{cases} \beta_k^{PRP}, & 0 \leq \beta_k^{PRP} \leq \beta_k^{FR}, \\ \beta_k^{FR}, & \text{otherwise.} \end{cases}$	Proposed by Touati-Ahmed and Storey (1990)
$\beta_k^{PRP-FR} = \max \left\{ 0, \min \left\{ \beta_k^{PRP}, \beta_k^{FR} \right\} \right\}$	Proposed by Hu and Storey (1991)
$\beta_k^{GN} = \max \left\{ -\beta_k^{FR}, \min \left\{ \beta_k^{PRP}, \beta_k^{FR} \right\} \right\}$	Proposed by Gilbert and Nocedal (1992)
$\beta_k^{HS-DY} = \max \left\{ 0, \min \left\{ \beta_k^{HS}, \beta_k^{DY} \right\} \right\}$	Proposed by Dai and Yuan (2001) and Dai and Ni (2003)
$\beta_k^{hDY} = \max \left\{ -\left( \frac{1-\sigma}{1+\sigma} \right) \beta_k^{DY}, \min \left\{ \beta_k^{HS}, \beta_k^{DY} \right\} \right\}$	Proposed by Dai and Yuan (2001). $\sigma$ is the parameter from the second Wolfe line-search condition
$\beta_k^{DDF} = \frac{\ g_{k+1}\ ^2}{\max \{-d_k^T g_k, d_k^T y_k\}}.$	Proposed by Dai (2002a)
$\beta_k^{VFR} = \frac{\mu_1 \ g_{k+1}\ ^2}{\mu_2  g_{k+1}^T d_k  + \mu_3 \ g_k\ ^2}$	Proposed by Wei, Li, and Qi (2006). $\mu_1 > 0$ , $\mu_3 > 0$ , and $\mu_2 > \mu_1$ are parameters
$\beta_k^{VPRP} = \frac{\ g_{k+1}\ ^2 - \ g_{k+1}\ (g_{k+1}^T g_k) / \ g_k\ }{\ g_k\ ^2}$	Proposed by Wei, Yao and Liu (2006)
$\beta_k^{YWH} = \frac{\ g_{k+1}\ ^2 - \ g_{k+1}\ (g_{k+1}^T g_k) / \ g_k\ }{d_k^T y_k}$	Proposed by Yao, Wei, and Huang (2007)
$\beta_k^{JCP} = \begin{cases} \beta_k^{DY}, & \text{if } g_{k+1}^T d_k \geq \ g_{k+1}\ ^2, \\ \beta_k^{FR}, & \text{else.} \end{cases}$	Proposed by Jiao, Chen, and Pan (2007)
$\beta_k^{LS-CD} = \max \left\{ 0, \min \left\{ \beta_k^{LS}, \beta_k^{CD} \right\} \right\}$	Proposed by Andrei (2008g)
$\beta_k^{DPRP} = \frac{\ g_{k+1}\ ^2 - \ g_{k+1}\ (g_{k+1}^T g_k) / \ g_k\ }{\mu  g_{k+1}^T d_k  + \ g_k\ ^2}$	Proposed by Dai and Wen (2012). $\mu > 1$ is a parameter
$\beta_k^{JHJ} = \frac{\ g_{k+1}\ ^2 - \ g_{k+1}\  \max \{0, (g_{k+1}^T g_k)\} / \ g_k\ }{\max \{\ g_k\ ^2, d_k^T y_k\}}$	Proposed by Jian, Han, and Jiang (2015)

proposed the hybrid scheme with  $\beta_k^{HS-DY}$  or  $\beta_k^{hDY}$  in (5.3). The HS-DY hybrid conjugate gradient method was also discussed by Andrei (2008f). Gilbert and Nocedal (1992) showed that even for strongly convex functions, it is quite possible for  $\beta_k^{PRP}$  to be negative. Therefore, in order to extend the number of iterations in which  $\beta_k^{PRP}$  is used, they suggested the hybrid method with  $\beta_k^{GN}$  in (5.3) to maintain the global convergence.

The hybrid conjugate gradient method DDF, which employs either the DY or the FR standard conjugate gradient algorithms, was proposed by Dai (2002a). He established that this hybrid scheme generates descent directions, independent of the line-search.

The VFR hybrid conjugate gradient, which is a variation of the FR method proposed by Wei, Li, and Qi (2006a), satisfies the sufficient descent condition  $g_k^T d_k \leq -(1 - \mu_1/\mu_2) \|g_k\|^2$  independent of the line-search.

The VPRP hybrid conjugate gradient method proposed by Wei, Yao, and Liu (2006b) is a variant of the PRP method. This hybrid conjugate gradient method inherits the properties of PRP. Under the strong Wolfe line-search with  $0 < \sigma < 1/4$ , Huang, Wei, and Yao (2007) showed that the search direction of the VPRP method satisfies the sufficient descent condition and the algorithm is globally convergent. An interesting property of VPRP is that it has the Property (\*), first introduced by Gilbert and Nocedal (1992).

The YWH hybrid method proposed by Yao, Wei, and Huang (2007) is a variant of the HS method. They established that under the strong Wolfe line-search with  $\sigma \in (0, 1/3)$ , the YWH method generates sufficient descent directions.

Jiao, Chen, and Pan (2007) proposed a hybrid conjugate gradient method JCP and established the global convergence under the Goldstein line-search (2.56).

The hybrid conjugate gradient method LS-CD was proposed by Andrei (2008g). Its global convergence under the Wolfe line-search was proved by Yang, Luo, and Dai (2013). Numerical experiments showed that the hybrid LS-CD method outperforms both the LS and the CD methods.

Dai and Wen (2012) proposed the hybrid DPRP method, where  $\mu > 1$ . They proved that the corresponding search direction satisfies the sufficient descent condition for any line-search. They also proved that the DPRP method is globally convergent under the Wolfe line-search.

A hybridization of the HS and DY conjugate gradient methods was proposed by Babaie-Kafaki and Ghanbari (2015). In their algorithm, the hybridization parameter is computed by solving the least-squares problem of minimizing the distance between the search direction of the hybrid method and the search direction corresponding to the three-term conjugate gradient method of Zhang, Zhou, and Li (2007), which possesses the sufficient descent property.

The hybrid method JHJ proposed by Jian, Han, and Jiang (2015) generates sufficient descent directions for any line-search and is globally convergent under the Wolfe line-search.

Other hybrid conjugate gradient methods were proposed by Zhang (2009a, 2009b), Han, Zhang, and Chen (2017). Two descent hybrid conjugate gradient algorithms as a projection of PRP and HS, and also of HS and DY, respectively, were developed by Zhang and Zhou (2008).

The following general result proved by Dai, Han, Liu, Sun, Yin, and Yuan (1999) shows that there are a lot of possibilities to generate hybrid conjugate gradient methods by taking into consideration the FR method.

**Theorem 5.15** Suppose that the Assumption CG holds and consider any general method of the form (5.47)–(5.48) where the stepsize is determined by the strong Wolfe line-search (5.49) and (5.51) with  $0 < \rho < \sigma < 1$  and  $\beta_k$  satisfying

$$\sigma |\beta_k| \leq \bar{\sigma} \beta_k^{FR}, \quad (5.173)$$

with  $\bar{\sigma} \in (0, 1/2]$  and

$$\|g_{k+1}\|^2 \sum_{j=0}^k \prod_{i=j}^k \left( \frac{\beta_i}{\beta_i^{FR}} \right)^2 \leq c_2 k, \quad (5.174)$$

for some constant  $c_2 > 0$ . Then,

$$\liminf_{k \rightarrow \infty} \|g_k\| = 0. \quad (5.175)$$

**Proof** As known,  $\beta_k^{FR} = g_{k+1}^T g_{k+1} / g_k^T g_k$ . Then, from (5.48), from the strong Wolfe line-search  $|g(x_k + \alpha_k d_k)^T d_k| \leq -\sigma g_k^T d_k$  and from (5.173), it results that

$$\begin{aligned} \frac{-g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} &= 1 - \beta_k \frac{g_{k+1}^T d_k}{\|g_{k+1}\|^2} = 1 - \left( \frac{\beta_k}{\beta_k^{FR}} \right) \frac{g_{k+1}^T d_k}{\|g_k\|^2} \\ &\leq 1 + \left| \frac{\beta_k}{\beta_k^{FR}} \right| \frac{-\sigma g_k^T d_k}{\|g_k\|^2} \leq 1 + \bar{\sigma} \left( \frac{-g_k^T d_k}{\|g_k\|^2} \right) \\ &\leq \dots \leq \sum_{j=0}^k \bar{\sigma}^j + \bar{\sigma}^{k+1} \left( \frac{-g_0^T d_0}{\|g_0\|^2} \right) = \frac{1 - \bar{\sigma}^{k+1}}{1 - \bar{\sigma}} < \frac{1}{1 - \bar{\sigma}}. \end{aligned} \quad (5.176)$$

Similarly, since  $\bar{\sigma} \leq 1/2$ ,

$$\frac{-g_{k+1}^T d_{k+1}}{\|g_{k+1}\|^2} \geq 1 - \bar{\sigma} \frac{1 - \bar{\sigma}^k}{1 - \bar{\sigma}} > 0. \quad (5.177)$$

Therefore,  $d_k$  is a descent direction. Since  $d_{k+1} + g_{k+1} = \beta_k d_k$ , it follows that

$$\|d_{k+1}\|^2 = -\|g_{k+1}\|^2 - 2g_{k+1}^T d_{k+1} + \beta_k^2 \|d_k\|^2.$$

Thus,

$$\|d_{k+1}\|^2 \leq -2g_{k+1}^T d_{k+1} + \beta_k^2 \|d_k\|^2. \quad (5.178)$$

Using (5.178) recursively and noting that  $d_0 = -g_0$ , it follows that

$$\begin{aligned} \|d_{k+1}\|^2 &\leq -2g_{k+1}^T d_{k+1} - 2 \sum_{j=0}^k \prod_{i=j}^k \beta_i^2 (g_j^T d_j) \\ &= -2g_{k+1}^T d_{k+1} - 2\|g_{k+1}\|^4 \sum_{j=0}^k \prod_{i=j}^k \left( \frac{\beta_i}{\beta_i^{FR}} \right)^2 \left( \frac{g_j^T d_j}{\|g_j\|^4} \right). \end{aligned} \quad (5.179)$$

If the theorem is not true, then (5.174) holds and there exists a positive constant  $\gamma$  so that  $\|g_k\| \geq \gamma$  for all  $k$ . Thus, from (5.176) and (5.179), it results that

$$\frac{\|d_{k+1}\|^2}{\|g_{k+1}\|^2} \leq \frac{2}{1 - \bar{\sigma}} \left( 1 + \frac{\|g_{k+1}\|^2}{\gamma^2} \sum_{j=0}^k \prod_{i=j}^k \left( \frac{\beta_i}{\beta_i^{FR}} \right)^2 \right). \quad (5.180)$$

From (5.180) and (5.174), it follows that

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^2}{\|d_k\|^2} \leq +\infty.$$

This implies that  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ . ◆

The theorem shows that for  $|\beta_k| \leq (\bar{\sigma}/\sigma)\beta_k^{FR}$  the algorithm (5.113)–(5.114) with  $d_0 = -g_0$  is globally convergent. In other words, to use  $\beta_k^{FR}$  in the hybrid schemes presented in Table 5.5 is quite natural. The theorem extends the result of Gilbert and Nocedal (1992) and Hu and Storey (1991) to

the case when  $\bar{\sigma} = 1/2$ , i.e., when  $2\sigma|\beta_k| < \beta_k^{FR}$ . If  $\bar{\sigma} \in (0, 1/2)$ , then from (5.177) it follows that the sufficient descent  $-g_{k+1}^T d_{k+1} \geq c \|g_{k+1}\|^2$  holds for any positive constant  $c$ . However, if  $\bar{\sigma} = 1/2$ , then only

$$\frac{-g_k^T d_k}{\|g_k\|^2} \geq \frac{1}{2^k}$$

holds, which does not imply the sufficient descent condition.

## Numerical Study: Hybrid Conjugate Gradient Methods

In the following, let us present the performances of some hybrid conjugate gradient methods based on the projection concept: TAS, PRP-FR, GN, hDY, HS-DY, and LS-CD. For this, the set of 800 unconstrained optimization problems from the UOP collection is used (Andrei, 2020a).

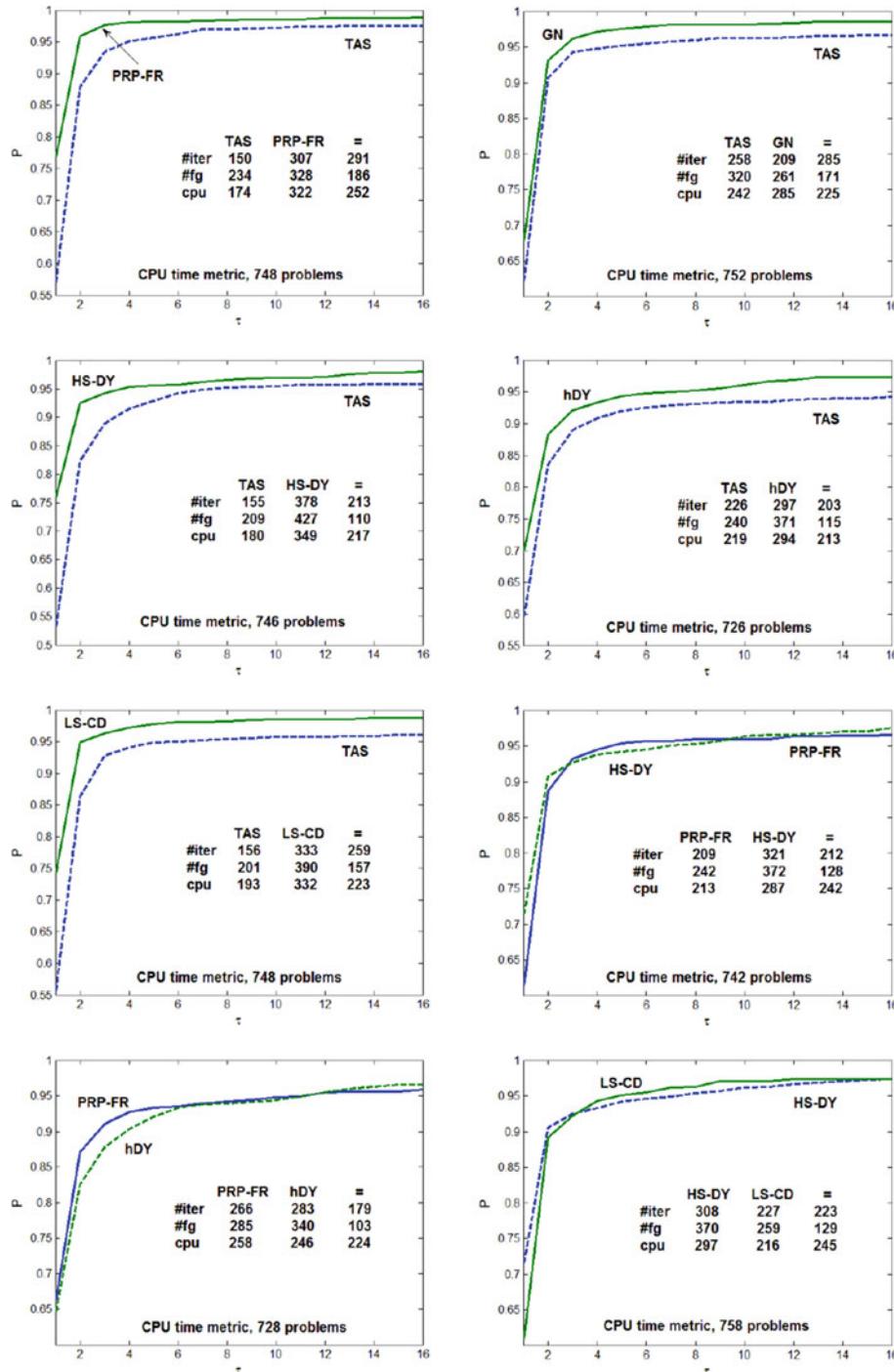
Figure 5.8 presents the Dolan and Moré performance profiles of some hybrid conjugate gradient methods subject to the CPU time metric. The tables inside the plots show the performances of the algorithms subject to the number of iterations (#iter), the number of function and its gradient evaluations (#fg), and the CPU time in seconds (cpu).

For example, when comparing TAS versus PRP-FR in Fig. 5.8, we can see that TAS was better in 150 problems (i.e., TAS achieved the minimum number of iterations in 150 problems). PRP-FR was better in 307 problems, and they achieved the same number of iterations in 291 problems, etc. Out of 800 problems considered in this numerical experiment, only for 748 problems does the criterion (1.3) hold. From Fig. 5.8, subject to the CPU time metric, it is clear that the TAS method is less efficient and less robust than PRP-FR, GN, HS-DY, hDY, and LS-CD. On the other hand, the PRP-FR hybrid conjugate gradient method is faster than hDY. Notice that HS-DY is more efficient than LS-CD, etc.

Figure 5.9 presents a global comparison of the hybrid conjugate gradient methods. Concerning their robustness, for this set of unconstrained optimization problems, subject to the CPU time metric, observe that the LS-CD algorithm is top performer, being more robust than the hybrid conjugate gradient methods considered in this study. On the other hand, HS-DY is the most efficient. TAS and GN are less efficient. Anyway, the performance profiles of the compared hybrid methods are grouped, one method being slightly more efficient or more robust than the other one. Out of 800 problems in this numerical experiment, only for 714 problems does the criterion (1.3) hold. The table inside Fig. 5.9 shows that, out of 714 problems, HS-DY was faster in 328 problems, followed by hDY, which was faster in 316 problems, etc.

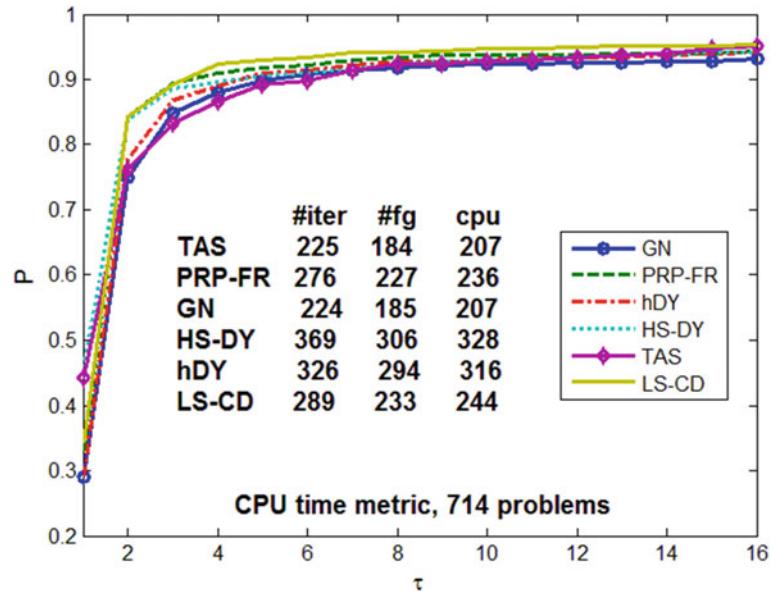
From all these computational experiments with a large set of nonlinear unconstrained optimization problems with different structures and complexities, we can see that the hybrid conjugate gradient methods do not have spectacular performances versus the standard conjugate gradient methods, subject to the CPU time metric. The first hybrid conjugate gradient method based on the projection concept was given by Touati-Ahmed and Storey (1990) and by Hu and Storey (1991). The motivation of introducing the hybrid conjugate gradient methods was to use the PRP update parameter when the iterations jam. However, the PRP-FR method is less efficient and less robust than LS, HS, and PRP+.

In the following, the performances of the hybrid conjugate gradient methods based on the projection concept for solving some applications from the MINPACK-2 applications are presented. Tables 5.6 and 5.7 show the performances of the hybrid conjugate gradient methods for solving five MINPACK-2 applications, each of them having 40,000 variables.



**Fig. 5.8** Performance profiles of some hybrid conjugate gradient methods based on the projection concept

**Fig. 5.9** Global performance profiles of six hybrid conjugate gradient methods



**Table 5.6** Performances of TAS, PRP-FR, and GN for solving five applications from the MINPACK-2 collection

	<i>n</i>	TAS			PRP-FR			GN		
		#iter	#fg	cpu	#iter	#fg	cpu	#iter	#fg	cpu
A1	40,000	402	576	6.80	385	540	6.65	460	650	7.97
A2	40,000	602	1179	15.46	602	1179	15.46	602	1179	15.45
A3	40,000	806	1615	36.76	806	1615	34.86	806	1615	34.55
A4	40,000	294	595	30.21	294	595	30.29	294	595	30.21
A5	40,000	368	701	11.45	368	701	11.42	368	701	11.44
Total	—	2472	4666	100.68	2455	4630	98.68	2530	4740	99.62

**Table 5.7** Performances of HS-DY, hDY, and LS-CD for solving five applications from the MINPACK-2 collection

	<i>n</i>	HS-DY			hDY			LS-CD		
		#iter	#fg	cpu	#iter	#fg	cpu	#iter	#fg	cpu
A1	40,000	533	684	6.97	591	753	6.89	402	563	6.44
A2	40,000	935	1771	23.67	935	1771	23.73	649	1242	16.36
A3	40,000	1034	1939	42.53	1034	1939	41.90	1159	2073	45.28
A4	40,000	577	1148	58.39	577	1148	58.32	294	594	30.30
A5	40,000	400	760	12.62	400	760	12.64	375	702	11.53
Total	—	3479	6302	144.18	3537	6371	143.48	2879	5174	109.91

It appears that the hybrid PRP-FR is the fastest, followed by GN, etc. The most time consuming is HS-DY. Observe that Table 5.3 shows that for solving all five applications, CD needs only 137.09 seconds, the minimum time among all the methods considered in the numerical study. Clearly, the hybrid methods are top performers versus the standard ones.

## Hybrid Conjugate Gradient Methods as Convex Combinations of the Standard Conjugate Gradient Methods

The hybrid conjugate gradient methods based on the convex combination of the standard conjugate gradient methods are defined by (5.113) and (5.114), where the conjugate gradient parameter  $\beta_k$  is computed as

$$\beta_k = (1 - \theta_k)\beta_k^{M1} + \theta_k\beta_k^{M2}, \quad (5.181)$$

where  $\beta_k^{M1}$  and  $\beta_k^{M2}$  are the conjugate gradient parameters of the standard conjugate gradient methods which we want to hybridize (see Table 5.1) and  $0 \leq \theta_k \leq 1$  is a parameter. The idea is to combine the standard algorithms in a convex way in order to get algorithms with better performances. The parameter  $\theta_k$  in the convex combination (5.181) can be determined by means of two procedures. The *first one* is given by the conjugacy condition  $y_k^T d_{k+1} = 0$ , where

$$d_{k+1} = -g_{k+1} + ((1 - \theta_k)\beta_k^{M1} + \theta_k\beta_k^{M2})d_k. \quad (5.182)$$

From the equality  $y_k^T d_{k+1} = 0$ , where  $d_{k+1}$  is given by (5.182), the following value for the parameter  $\theta_k$  is obtained:

$$\theta_k = \frac{y_k^T g_{k+1} - \beta_k^{M1}(y_k^T d_k)}{(\beta_k^{M2} - \beta_k^{M1})(y_k^T d_k)}. \quad (5.183)$$

Obviously, instead of the standard conjugacy condition  $y_k^T d_{k+1} = 0$ , it is quite possible to use the Dai and Liao conjugacy condition

$$y_k^T d_{k+1} = -ts_k^T g_{k+1}, \quad (5.184)$$

where  $t \geq 0$  is a scalar parameter. In this case, from (5.184) where  $d_{k+1}$  is given by (5.182), it results that

$$\theta_k = \frac{y_k^T g_{k+1} - ts_k^T g_{k+1} - \beta_k^{M1}(y_k^T d_k)}{(\beta_k^{M2} - \beta_k^{M1})(y_k^T d_k)}. \quad (5.185)$$

This is another value for the parameter  $\theta_k$  obtained from the Dai and Liao conjugacy condition (5.184). However, a value for  $t$  must be chosen in this case, which is rather difficult (see Andrei (2011b)).

On the other hand, if the point  $x_{k+1}$  is close enough to a local minimizer  $x^*$ , then a good direction to follow is the one given by the Newton direction, that is,  $d_{k+1} = -\nabla^2 f(x_{k+1})^{-1} g_{k+1}$ . Therefore, the *second procedure* to determine  $\theta_k$  in (5.181) considers the formal equality between the search direction of the hybrid algorithm and the Newton direction, i.e.,

$$-g_{k+1} + ((1 - \theta_k)\beta_k^{M1} + \theta_k\beta_k^{M2})d_k = -\nabla^2 f(x_{k+1})^{-1} g_{k+1}. \quad (5.186)$$

Observe that (5.186) is only a technical argument to get a value for  $\theta_k$ . With some simple algebraic manipulations, from (5.186),

$$\theta_k = \frac{s_k^T \nabla^2 f(x_{k+1}) g_{k+1} - s_k^T g_{k+1} - \beta_k^{M1} (s_k^T \nabla^2 f(x_{k+1}) d_k)}{(\beta_k^{M2} - \beta_k^{M1}) (s_k^T \nabla^2 f(x_{k+1}) d_k)}. \quad (5.187)$$

Both these procedures for the  $\theta_k$  computation are operational and can be used to generate hybrid conjugate gradient methods. The salient point in (5.187) for the  $\theta_k$  computation is the presence of the Hessian  $\nabla^2 f(x_{k+1})$ . For large-scale problems, in practice, choices for the update parameters that do not require the evaluation of the Hessian are preferred over the methods that require the Hessian at each iteration. In order to avoid the exact computation of  $\nabla^2 f(x_{k+1})$ , a solution is to use the secant equation. In quasi-Newton methods, the secant equation plays an essential role for the approximation of the Hessian of the objective function at each iteration. Therefore, in the second procedure for the  $\theta_k$  computation, the search direction  $d_k$  can be calculated as solution of the following algebraic linear system  $B_k d_k = -g_k$ , where  $B_k$  is an approximation of  $\nabla^2 f(x_k)$ . In the quasi-Newton methods, the matrix  $B_k$ , which is symmetric and positive definite, is effectively updated to obtain a new matrix,  $B_{k+1}$ , also symmetric and positive definite, as an approximation of  $\nabla^2 f(x_{k+1})$ . The matrix  $B_{k+1}$  needs to satisfy some suitable equations, namely, the secant equations, which include the second order information. The most popular is the standard secant equation

$$B_{k+1} s_k = y_k. \quad (5.188)$$

By using the standard secant equation in (5.187), the following value for  $\theta_k$  could be obtained:

$$\theta_k = \frac{y_k^T g_{k+1} - s_k^T g_{k+1} - \beta_k^{M1} (y_k^T d_k)}{(\beta_k^{M2} - \beta_k^{M1}) (y_k^T d_k)}. \quad (5.189)$$

The parameter  $\theta_k$ , computed as in (5.183), (5.185), (5.187), or (5.189), may be outside the interval  $[0,1]$ . To get a convex combination in (5.181), a simple procedure is followed: if  $\theta_k \leq 0$ , then in (5.181)  $\theta_k = 0$ , that is,  $\beta_k = \beta_k^{M1}$ ; if  $\theta_k \geq 1$ , then  $\theta_k = 1$  in (5.181), that is,  $\beta_k = \beta_k^{M2}$ .

Hence, the general hybrid conjugate gradient method based on the convex combination of the standard conjugate gradient methods is as follows:

**Algorithm 5.5** General hybrid conjugate gradient algorithm by using the convex combination of standard schemes

1.	Choose an initial point $x_0 \in \mathbb{R}^n$ , $\varepsilon \geq 0$ sufficiently small and $\varepsilon_b \geq 0$ . Compute $f(x_0)$ and $g_0$ . Set $d_0 = -g_0$ and the initial guess $\alpha_0 = 1/\ g_0\ $ . Set $k = 0$
2.	Test a criterion for stopping the iterations. For example, if $\ g_k\ _\infty \leq \varepsilon$ , then stop; otherwise, continue with step 3
3.	Compute the stepsize $\alpha_k$ satisfying the Wolfe line-search conditions
4.	Set $x_{k+1} = x_k + \alpha_k d_k$ . Compute $f(x_{k+1})$ and $g_{k+1}$ . Compute $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$
5.	If $ (\beta_k^{M2} - \beta_k^{M1}) (y_k^T d_k)  \leq \varepsilon_b$ , then set $\theta_k = 0$ . Otherwise, compute $\theta_k$ by (5.183), (5.185) or by (5.189) according to the selected procedure
6.	If $0 < \theta_k < 1$ , then compute $\beta_k$ by (5.181). If $\theta_k \leq 0$ , then set $\beta_k = \beta_k^{M1}$ . If $\theta_k \geq 1$ , then set $\beta_k = \beta_k^{M2}$
7.	Compute $d = -g_{k+1} + \beta_k d_k$
8.	If the Powell restart criterion $ g_{k+1}^T g_k  \geq 0.2 \ g_{k+1}\ ^2$ is satisfied, then set $d_{k+1} = -g_{k+1}$ , otherwise set $d_{k+1} = d$ .
9.	Compute the initial guess $\alpha_k = \alpha_{k-1} \ d_{k-1}\  / \ d_k\ $
10.	Set $k = k + 1$ and go to step 2

◆

Algorithm 5.5 is general. In step 5 it can be particularized by combining in a convex manner different standard conjugate gradient methods, where the parameter  $\theta_k$  in the convex combination (5.181) may be selected by means of the conjugacy or the Newton direction procedures.

As it can be seen, there is a large variety of possibilities to combine the standard conjugate gradient methods in a convex way by using the conjugacy conditions or the standard or the modified secant equations. In the following, let us present a hybrid conjugate gradient method based on the convex combination of the standard conjugate gradient algorithms.

### The Hybrid Convex Combination of LS and DY

The method is based on a convex combination of LS and DY (see Table 5.1) (Liu and Li, 2014). In it,

$$\beta_k^{M1} = \beta_k^{LS} = -\frac{y_k^T g_{k+1}}{d_k^T g_k} \quad \text{and} \quad \beta_k^{M2} = \beta_k^{DY} = \frac{g_{k+1}^T g_{k+1}}{d_k^T y_k}. \quad (5.190)$$

Using the Dai and Liao conjugacy condition (5.184), from (5.185), where  $\beta_k^{M1}$  and  $\beta_k^{M2}$  are given as in (5.190), the following value for the parameter  $\theta_k$  is obtained:

$$\theta_k^{DL} = \frac{(y_k^T g_{k+1})(d_k^T g_{k+1}) - t(s_k^T g_{k+1})(d_k^T g_k)}{\|g_{k+1}\|^2(d_k^T g_k) + (y_k^T g_{k+1})(d_k^T y_k)}. \quad (5.191)$$

It is interesting to see the value of  $\theta_k$  obtained from the second procedure based on the equality of the Newton direction and the direction corresponding to the hybrid conjugate gradient algorithm. Indeed, from (5.189), where  $\beta_k^{M1}$  and  $\beta_k^{M2}$  are given as in (5.190), after some algebraic manipulations, it results that

$$\theta_k^{NT} = \frac{(y_k^T g_{k+1})(d_k^T g_{k+1}) - (s_k^T g_{k+1})(d_k^T g_k)}{\|g_{k+1}\|^2(d_k^T g_k) + (y_k^T g_{k+1})(d_k^T y_k)}. \quad (5.192)$$

Observe that  $\theta_k^{DL} = \theta_k^{NT}$  when  $t = 1$ . Therefore, an interesting property of this hybrid conjugate gradient algorithm is that the search direction  $d_{k+1}$  not only satisfies the Dai and Liao conjugacy condition, but it is also the Newton direction when  $t = 1$ . Hence, in this hybrid algorithm  $t = 1$  is considered.

The algorithm corresponding to this convex combination of LS and DY, which we call NDLSDY, is a particularization of Algorithm 5.5, where step 5 is modified as follows: “If  $\left| \|g_{k+1}\|^2(d_k^T g_k) + (y_k^T g_{k+1})(d_k^T y_k) \right| \leq \varepsilon_b$ , then set  $\theta_k = 0$ . Otherwise, compute  $\theta_k$  by (5.192)” and set  $\beta_k^{M1} = \beta_k^{LS}$  and  $\beta_k^{M2} = \beta_k^{DY}$  in step 6.

The following theorem shows that the search direction  $d_k$  generated by the algorithm NDLSDY satisfies the sufficient descent condition.

**Theorem 5.16** *Let  $\{g_k\}$  and  $\{d_k\}$  be the sequences generated by the algorithm NDLSDY with strong Wolfe line-search. Then, the search direction  $d_k$  satisfies the sufficient descent condition*

$$d_k^T g_k \leq -c \|g_k\|^2 \quad (5.193)$$

for any  $k \geq 0$ , where  $c = (1 - 1.2\sigma)/(1 - \sigma)$ ,  $\sigma < 0.5$ .

**Proof** Assume that the Powell restart criterion does not hold, i.e.,

$$|g_{k+1}^T g_k| < 0.2 \|g_{k+1}\|^2. \quad (5.194)$$

The proof is given by induction. For  $k = 0$ ,  $g_0^T d_0 = -\|g_0\|^2$ . Since  $c < 1$ , it follows that (5.193) is satisfied. Now, assume that (5.193) holds for some  $k \geq 1$ .

From the second strong Wolfe condition  $|\nabla f(x_k + \alpha_k d_k)^T d_k| \leq -\sigma d_k^T g_k$ , it results that

$$d_k^T y_k = d_k^T g_{k+1} - d_k^T g_k \geq -(1 - \sigma) d_k^T g_k \geq 0. \quad (5.195)$$

From (5.114), we get

$$d_{k+1}^T g_{k+1} = -\|g_{k+1}\|^2 + \beta_k d_k^T g_{k+1}. \quad (5.196)$$

Now, when  $\theta_k \geq 1$ , as seen in step 6 of the algorithm NDLSDY, it follows that  $\beta_k = \beta_k^{DY}$ . Therefore, from the above relations (5.195) and (5.196), it results that

$$d_{k+1}^T g_{k+1} \leq -\|g_{k+1}\|^2 + \frac{\|g_{k+1}\|^2}{d_k^T y_k} |d_k^T g_{k+1}| \leq -\frac{1 - 2\sigma}{1 - \sigma} \|g_{k+1}\|^2. \quad (5.197)$$

From step 6 of the algorithm NDLSDY, when  $\theta_k \leq 0$ ,  $\beta_k = \beta_k^{LS}$ . Therefore, from the second strong Wolfe line-search and from (5.194), it results that

$$d_{k+1}^T g_{k+1} \leq -\|g_{k+1}\|^2 + \frac{|g_{k+1}^T y_k|}{|d_k^T g_k|} |d_k^T g_{k+1}| \leq -(1 - 1.2\sigma) \|g_{k+1}\|^2. \quad (5.198)$$

Finally, when  $\theta_k \in (0, 1)$ , as seen in step 6 of the algorithm NDLSDY,  $\beta_k$  is computed as in (5.181). Observe that  $\beta_k^{DY}$  can be written as  $\beta_k^{DY} = d_{k+1}^T g_{k+1} / d_k^T g_k$ . Now, by using (5.194), the above relations and the definition of  $\beta_k^{LS}$ , it follows that

$$\begin{aligned} d_{k+1}^T g_{k+1} &\leq -\|g_{k+1}\|^2 + |\beta_k^{LS}| |d_k^T g_{k+1}| + |\beta_k^{DY}| |d_k^T g_{k+1}| \\ &\leq -\|g_{k+1}\|^2 + \sigma |\beta_k^{LS}| |d_k^T g_k| + \sigma |\beta_k^{DY}| |d_k^T g_k| \\ &= -\|g_{k+1}\|^2 + \sigma |g_{k+1}^T y_k| + \sigma |d_{k+1}^T g_{k+1}| \\ &\leq -\|g_{k+1}\|^2 + \sigma \|g_{k+1}\|^2 + \sigma |g_{k+1}^T g_k| + \sigma |d_{k+1}^T g_{k+1}| \\ &\leq -\|g_{k+1}\|^2 + 1.2\sigma \|g_{k+1}\|^2 + \sigma |d_{k+1}^T g_{k+1}|. \end{aligned} \quad (5.199)$$

But from (5.199),

$$d_{k+1}^T g_{k+1} - \sigma |d_{k+1}^T g_{k+1}| \leq -(1 - 1.2\sigma) \|g_{k+1}\|^2.$$

Since  $\sigma < 0.5$ , it follows that there always is a constant  $v > 0$  so that

$$d_{k+1}^T g_{k+1} - \sigma |d_{k+1}^T g_{k+1}| = v(d_{k+1}^T g_{k+1}).$$

Therefore,

$$d_{k+1}^T g_{k+1} \leq -q \|g_{k+1}\|^2, \quad (5.200)$$

where  $q = (1 - 1.2\sigma)/v$ ,  $v = 1 + \sigma$  or  $1 - \sigma$ .

In conclusion, (5.197), (5.198), and (5.200) show that (5.193) holds for  $k + 1$ .  $\blacklozenge$

Suppose that the Assumption CG holds. Then the following theorem proves the global convergence of the algorithm NDLSDY.

**Theorem 5.17** Suppose that the Assumption CG holds. Let  $\{g_k\}$  and  $\{d_k\}$  be the sequences generated by the algorithm NDLSDY with strong Wolfe line-search. Then,  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ .

**Proof** From the Assumption CG it follows that there exists a positive constant  $\Gamma > 0$  so that  $\|g(x)\| \leq \Gamma$  for all  $x \in S$ . Observe that from Proposition 2.4,  $\alpha_k \geq \lambda$ , where  $\lambda$  is a positive constant.

Suppose that  $\liminf_{k \rightarrow \infty} \|g_k\| \neq 0$ . Then there exists a positive constant  $r > 0$  so that for all  $k$  sufficiently large,

$$\|g_k\| \geq r. \quad (5.201)$$

By the second strong Wolfe condition and from (5.193),

$$d_k^T y_k = d_k^T g_{k+1} - d_k^T g_k \geq -(1-\sigma)d_k^T g_k \geq c(1-\sigma)\|g_k\|^2. \quad (5.202)$$

From the Lipschitz continuity of the gradient, it results that

$$\|y_k\| = \|g_{k+1} - g_k\| \leq L\|x_{k+1} - x_k\| \leq LD, \quad (5.203)$$

where  $D = \max \{\|x - y\| : x, y \in S\}$  is the diameter of the level set  $S$ .

Now, having in view the above inequalities, it follows that

$$\begin{aligned} |\beta_k| &= |(1-\theta_k)\beta_k^{LS} + \theta_k\beta_k^{DY}| \\ &\leq |\beta_k^{LS}| + |\beta_k^{DY}| = \frac{|g_{k+1}^T y_k|}{|d_k^T g_k|} + \frac{\|g_{k+1}\|^2}{|d_k^T y_k|} \\ &\leq \frac{\|g_{k+1}\|\|y_k\|}{c\|g_k\|^2} + \frac{\|g_{k+1}\|^2}{c(1-\sigma)\|g_k\|^2} \leq \frac{\Gamma LD}{cr^2} + \frac{\Gamma^2}{c(1-\sigma)r^2} \equiv M. \end{aligned}$$

According to the selection of the parameter  $\beta_k$ , in step 6 of the algorithm NDLSDY, when  $\theta_k \notin (0, 1)$ , it is obvious that the above inequality also holds. Therefore,

$$\|d_{k+1}\| \leq \|g_{k+1}\| + |\beta_k|\|d_k\| = \|g_{k+1}\| + \frac{|\beta_k|\|s_k\|}{\alpha_k} \leq \Gamma + \frac{MD}{\lambda} \equiv E,$$

which implies that

$$\sum_{k \geq 0} \frac{1}{\|d_k\|^2} = \infty. \quad (5.204)$$

On the other hand, from (5.193), (5.201) and from the Zoutendijk condition, it results that

$$c^2 r^4 \sum_{k \geq 0} \frac{1}{\|d_k\|^2} \leq \sum_{k \geq 0} \frac{c^2 \|g_k\|^4}{\|d_k\|^2} \leq \sum_{k \geq 0} \frac{(g_k^T d_k)^2}{\|d_k\|^2} < \infty,$$

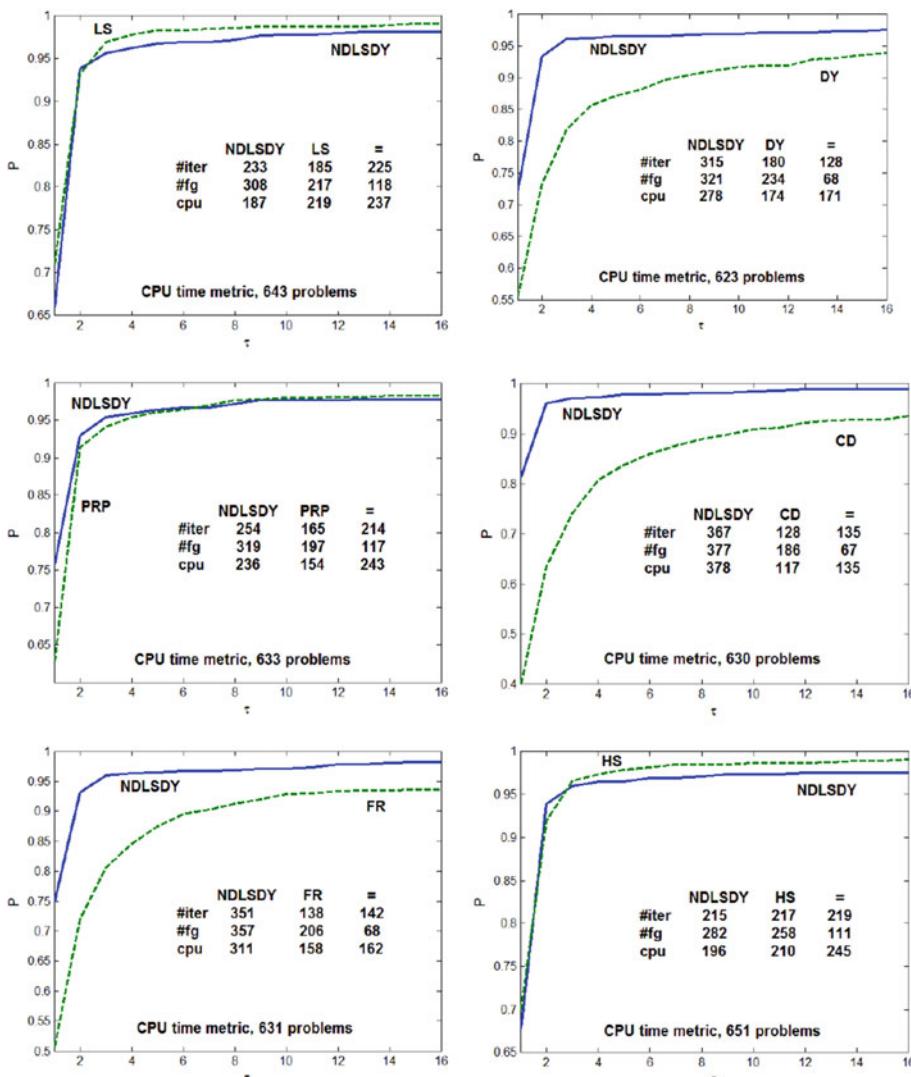
which contradicts (5.204). Hence, (5.201) does not hold and hence  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$  is proved.  $\blacklozenge$

New hybrid conjugate gradient algorithms for the unconstrained optimization were designed by Andrei (2008b, 2008d, 2008g, 2009a, 2010a).

## Numerical Study: NDLSDY

In the following, let us present some numerical results with NDLSDY for solving unconstrained optimization problems from the UOP collection, as well as some comparisons with standard conjugate gradient methods and with hybrid methods based on the projection concept.

Figure 5.10 presents Dolan and Moré's performance profiles of NDLSDY versus LS, DY, PRP, CD, FR, and HS conjugate gradient methods for solving the problems from the UOP collection. For this set of unconstrained optimization test problems, Fig. 5.10 shows that NDLSDY, as a convex combination of LS and DY, is more efficient than DY. However, LS is slightly more efficient and more robust than NDLSDY. Figure 5.6 points out the computational evidence that LS is more efficient and more robust than DY. Therefore, in this convex combination of LS and DY, the main role is played by LS. Comparisons with PRP show that NDLSDY is more efficient, but PRP is slightly



**Fig. 5.10** Performance profiles of NDLSDY versus the standard conjugate gradient methods LS, DY, PRP, CD, FR, and HS

**Table 5.8** Performances of NDLSDY for solving five applications from the MINPACK-2 collection

	<i>n</i>	NDLSDY		
		#iter	#fg	cpu
A1	40,000	451	521	13.39
A2	40,000	1411	1740	23.55
A3	40,000	1166	1272	38.44
A4	40,000	1018	1172	65.59
A5	40,000	274	301	4.04
Total	—	4320	5006	145.01

more robust. NDLSDY is seen as being more efficient and more robust than CD and than FR. Even if HS is slightly more robust than NDLSDY, NDLSDY is close to HS as regards its efficiency.

In the following, Table 5.8 presents the performances of the NDLSDY hybrid conjugate gradient method for solving five applications from the MINPACK-2 collection.

Table 5.8 shows that, subject to the CPU time metric, the hybrid convex combination of HS and DY based on the Newton direction is more efficient. Tables 5.2 and 5.4 show that HS with 139.65 seconds is top performer versus DY with 229.20 seconds. Table 5.4 reveals that LS with 168.03 seconds is again better than DY. In another realm of numerical experiments, in the above hybrid algorithms, DY with  $\|g_{k+1}\|^2$  in the numerator of  $\beta_k$ , is combined in a convex way with HS and LS, which have  $g_{k+1}^T y_k$  in the numerator of  $\beta_k$ . Notice that the hybrid convex combination with the Newton direction using HS is top performer versus the hybrid combination with LS.

---

## 5.6 Conjugate Gradient Methods as Modifications of the Standard Schemes

Due to their simplicity and low memory requirements, conjugate gradient methods represent an important contribution to the class of methods for solving unconstrained optimization problems. These methods have good convergence properties and their iterations do not involve any matrices, making them extremely attractive for solving large-scale problems.

In this section, some conjugate gradient methods obtained as modifications of the standard scheme are developed. The idea is to modify the formula of the standard conjugate gradient methods for computing the conjugate gradient parameter  $\beta_k$  in order to improve the numerical performances of the algorithm. Any standard conjugate gradient method (see Table 5.1) may be modified, but some modifications of the HS method are to be presented in the following.

A modification of the HS method is CG-DESCENT of Hager and Zhang (2005). CG-DESCENT is interpreted as a particular value of the parameter in the Dai-Liao conjugate gradient algorithm. Another interpretation of CG-DESCENT is a particularization of the Perry-Shanno self-scaling memoryless BFGS algorithm. A deeper modification of the HS method is to determine the search direction satisfying both the sufficient descent and the conjugacy conditions. Thus, the DESCON conjugate gradient algorithm is obtained, in which the stepsize is determined by a modification of the Wolfe line-search (Andrei, 2013c). Before presenting them, let us discuss the Dai-Liao conjugate gradient method.

## The Dai-Liao Conjugate Gradient Method

For general nonlinear functions, by the mean value theorem (see Appendix A), there exists  $\xi \in (0, 1)$  so that  $d_{k+1}^T g_{k+1} = d_{k+1}^T g_k + \alpha_k d_{k+1}^T \nabla^2 f(x_k + \xi \alpha_k d_k) d_k$ . Defining  $y_k = g_{k+1} - g_k$ , the following can be written  $d_{k+1}^T y_k = \alpha_k d_{k+1}^T \nabla^2 f(x_k + \xi \alpha_k d_k) d_k$ . Therefore, for the nonlinear optimization, it is reasonable to replace the conjugacy condition from the linear case with  $d_{k+1}^T y_k = 0$ .

But, for unconstrained optimization methods, the search direction  $d_{k+1}$  can be written as  $d_{k+1} = -H_{k+1}g_{k+1}$ , where  $H_{k+1}$  is an approximation to the inverse of the Hessian  $\nabla^2 f(x_{k+1})$ , symmetric and positive definite, which satisfies the secant equation  $H_{k+1}y_k = s_k$ , where  $s_k = x_{k+1} - x_k$ . Therefore,

$$d_{k+1}^T y_k = -(H_{k+1}g_{k+1})^T y_k = -g_{k+1}^T (H_{k+1}y_k) = -g_{k+1}^T s_k.$$

Hence, the conjugacy condition  $d_{k+1}^T y_k = 0$  is satisfied if the line-search is exact, since, in this case  $g_{k+1}^T s_k = 0$ . However, in practical situations, the exact line-search is not used. Therefore, it is quite natural to replace the conjugacy condition  $d_{k+1}^T y_k = 0$  with

$$d_{k+1}^T y_k = -t g_{k+1}^T s_k, \quad (5.205)$$

where  $t \geq 0$  is a scalar. To determine the parameter  $\beta_k$  in the search direction (5.48) that satisfies the conjugacy condition (5.205), let us multiply (5.48) by  $y_k$  and use (5.205), thus obtaining

$$\beta_k^{DL} = \frac{g_{k+1}^T y_k - t g_{k+1}^T s_k}{y_k^T d_k}, \quad (5.206)$$

known as the *Dai-Liao conjugate gradient parameter*. The method where the search direction is computed as in (5.48) with  $\beta_k$  computed as in (5.206) is called the *Dai-Liao conjugate gradient method* (Dai, & Liao, 2001). It is obvious that

$$\beta_k^{DL} = \beta_k^{HS} - t \frac{g_{k+1}^T s_k}{y_k^T d_k}, \quad (5.207)$$

showing that the DL method is a modification of the HS method. If the line-search is exact ( $g_{k+1}^T s_k = 0$ ), then  $\beta_k^{DL} = \beta_k^{HS}$ . Dai and Liao (2001) proved that for strongly convex functions, the norm of the search directions generated by the DL method is bounded. Therefore, the DL method with strong Wolfe line-search is globally convergent.

In order to ensure the global convergence for general nonlinear functions, similar to the PRP+ method introduced by Powell (1984), Dai and Liao restrict  $\beta_k^{DL}$  to be positive, thus suggesting  $\beta_k^{DL+}$ , where

$$\beta_k^{DL+} = \max \left\{ \frac{g_{k+1}^T y_k}{d_k^T y_k}, 0 \right\} - t \frac{g_{k+1}^T s_k}{d_k^T y_k}. \quad (5.208)$$

If the Lipschitz and boundedness assumptions hold and if  $d_k$  satisfies the sufficient descent condition  $g_k^T d_k \leq -c \|g_k\|^2$  where  $c > 0$  is a constant, then Dai and Liao (2001) showed that DL+ with strong Wolfe line-search is globally convergent.

## The Conjugate Gradient with Guaranteed Descent (CG-DESCENT)

For solving the problem (5.46), Hager and Zhang (2005, 2006a) proposed one of the most respected conjugate gradient algorithms:

$$x_{k+1} = x_k + \alpha_k d_k, \quad (5.209)$$

$$d_{k+1} = -g_{k+1} + \bar{\beta}_k^N d_k, \quad d_0 = -g_0, \quad (5.210)$$

$$\bar{\beta}_k^N = \max \{ \beta_k^N, \eta_k \}, \quad (5.211)$$

$$\eta_k = \frac{-1}{\|d_k\| \min \{ \eta, \|g_k\| \}}, \quad (5.212)$$

$$\beta_k^N = \frac{1}{y_k^T d_k} \left( y_k - 2 \frac{\|y_k\|^2}{y_k^T d_k} d_k \right)^T g_{k+1}, \quad (5.213)$$

where  $\eta > 0$  is a constant ( $\eta = 0.01$ ). In order to achieve the global convergence for general nonlinear functions, similar to the PRP+ method, the truncation (5.211) is introduced. Observe that in the restricted scheme (5.211), the lower bound on  $\bar{\beta}_k^N$  is dynamically adjusted in order to make the lower bound smaller as the iterates converge. Observe that (5.213) is a modification of the HS method.

Hager and Zhang obtained this scheme by deleting a term from the direction of the memoryless quasi-Newton method of Perry (1976, 1977) and Shanno (1978b). Indeed, the search direction of Hager and Zhang may be written as

$$d_{k+1}^{HZ} = -g_{k+1} - \frac{y_k^T y_k}{d_k^T y_k} \left[ 2 \frac{d_k^T g_{k+1}}{d_k^T y_k} - \frac{y_k^T g_{k+1}}{y_k^T y_k} \right] d_k. \quad (5.214)$$

On the other hand, the search direction of Perry/Shanno  $d_k^{PS}$  is

$$d_{k+1}^{PS} = -\frac{y_k^T s_k}{y_k^T y_k} g_{k+1} - \left[ 2 \frac{s_k^T g_{k+1}}{y_k^T s_k} - \frac{y_k^T g_{k+1}}{y_k^T y_k} \right] s_k + \frac{s_k^T g_{k+1}}{y_k^T y_k} y_k. \quad (5.215)$$

(The Perry-Shanno direction is to be presented in the next section.) Observe that the relationship between the search direction of Perry/Shanno and that of Hager/Zhang is as follows:

$$d_{k+1}^{PS} = \frac{s_k^T y_k}{y_k^T y_k} \left( d_{k+1}^{HZ} + \frac{d_k^T g_{k+1}}{y_k^T d_k} y_k \right), \quad (5.216)$$

or

$$d_{k+1}^{HZ} = \frac{y_k^T y_k}{y_k^T s_k} d_{k+1}^{PS} - \frac{d_k^T g_{k+1}}{y_k^T d_k} y_k. \quad (5.217)$$

Obviously,  $d_{k+1}^{HZ}$  may be written as

$$d_{k+1}^{HZ} = - \left[ I - \frac{d_k y_k^T}{y_k^T d_k} + 2 \frac{y_k^T y_k}{(y_k^T d_k)^2} d_k d_k^T \right] g_{k+1} \equiv -Q_{k+1}^{HZ} g_{k+1}. \quad (5.218)$$

Observe that  $Q_{k+1}^{HZ}$  is not symmetric and does not satisfy the quasi-Newton equation, properties satisfied by  $d_{k+1}^{PS}$ . If in a canonical manner  $Q_{k+1}^{HZ}$  is symmetrized and imposed to satisfy the quasi-Newton equation, the Perry/Shanno direction (5.215) is obtained. From (5.216), when the angle between  $d_k$  and  $g_{k+1}$  is sufficiently small and  $f$  is strongly convex, we can see that the term  $d_{k+1}^{HZ}$  dominates the  $y_k$  term. In this case, the directions  $d_{k+1}^{HZ}$  are approximately multiples of  $d_{k+1}^{PS}$ . The Perry/Shanno scheme, analyzed by Shanno and Phua (1980) and by Shanno (1985), has global convergence for convex functions and an inexact line-search (Shanno, 1978b), but in general it does not necessarily converge, even when the line-search is exact (Powell, 1984). However, the Perry/Shanno scheme is convergent if the restarts are employed, but in this case the speed of convergence can decrease. Han, Liu, and Yin (1997) proved that if the Wolfe line-search is employed, then the convergence to a stationary point of the minimizing function  $f$  is achieved when  $\lim_{k \rightarrow \infty} \|y_k\| = 0$  and the gradient of  $f$  is Lipschitz continuous.

The following theorem shows that if  $y_k^T d_k \neq 0$ , a condition which is satisfied when  $f$  is strongly convex or the line-search satisfies the Wolfe conditions, then the computational method given by (5.209)–(5.213) always generates descent directions (Hager, & Zhang, 2005).

**Theorem 5.18** *If  $y_k^T d_k \neq 0$  and*

$$d_{k+1} = -g_{k+1} + \tau d_k, \quad (d_0 = -g_0) \quad (5.219)$$

for any  $\tau \in [\beta_k^N, \max\{0, \beta_k^N\}]$ , then

$$g_k^T d_k \leq -\frac{7}{8} \|g_k\|^2. \quad (5.220)$$

**Proof** Since  $d_0 = -g_0$ , it follows that  $g_0^T d_0 = -\|g_0\|^2$ , which satisfies (5.220). Suppose that  $\tau = \beta_k^N$ . Multiplying (5.219) by  $g_{k+1}^T$  it results that

$$\begin{aligned} g_{k+1}^T d_{k+1} &= -\|g_{k+1}\|^2 + \beta_k^N g_{k+1}^T d_k \\ &= -\|g_{k+1}\|^2 + \left( \frac{y_k^T g_{k+1}}{y_k^T d_k} - 2 \frac{\|y_k\|^2 d_k^T g_{k+1}}{(y_k^T d_k)^2} \right) g_{k+1}^T d_k \\ &= \frac{(y_k^T g_{k+1})(y_k^T d_k)(g_{k+1}^T d_k) - \|g_{k+1}\|^2 (y_k^T d_k)^2 - 2\|y_k\|^2 (g_{k+1}^T d_k)^2}{(y_k^T d_k)^2} \end{aligned}$$

Now, for the first term  $(y_k^T g_{k+1})(y_k^T d_k)(g_{k+1}^T d_k)$  from the above equality, let us apply the classical inequality  $|u^T v| \leq \frac{1}{2} (\|u\|^2 + \|v\|^2)$ , where  $u = \frac{1}{2} (y_k^T d_k) g_{k+1}$  and  $v = 2(g_{k+1}^T d_k) y_k$ , to get (5.220). On the other hand, if  $\tau \neq \beta_k^N$ , then  $\beta_k^N \leq \tau \leq 0$ . After multiplying (5.219) by  $g_{k+1}^T$ , it follows that

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \tau g_{k+1}^T d_k.$$

If  $g_{k+1}^T d_k \geq 0$ , then (5.220) immediately follows, since  $\tau \leq 0$ . If  $g_{k+1}^T d_k < 0$ , then

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \tau g_{k+1}^T d_k \leq -\|g_{k+1}\|^2 + \beta_k^N g_{k+1}^T d_k$$

since  $\beta_k^N \leq \tau \leq 0$ . Therefore, (5.220) is true by the above analysis.  $\blacklozenge$

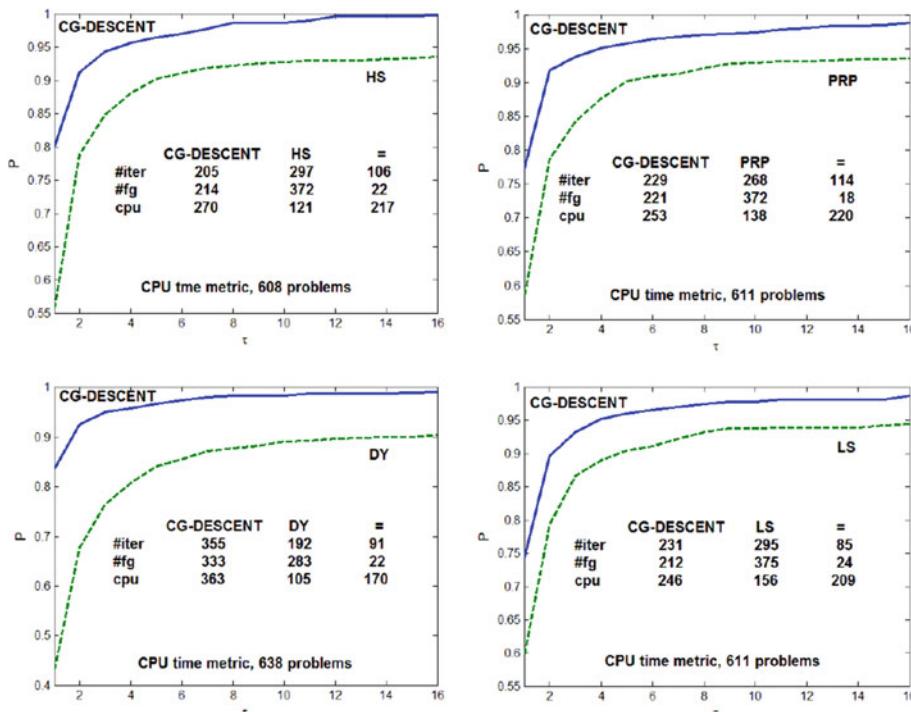
## Numerical Study: CG-DESCENT

In the following, let us see the performances of CG-DESCENT (version 1.4) of Hager and Zhang for solving 80 unconstrained optimization problems from the UOP collection (Andrei, 2020a), where the number of variables is  $n = 1000, 2000, \dots, 10000$ . The numerical experiments are given in the context of Remark 1.1. The maximum number of iterations was limited to 2000.

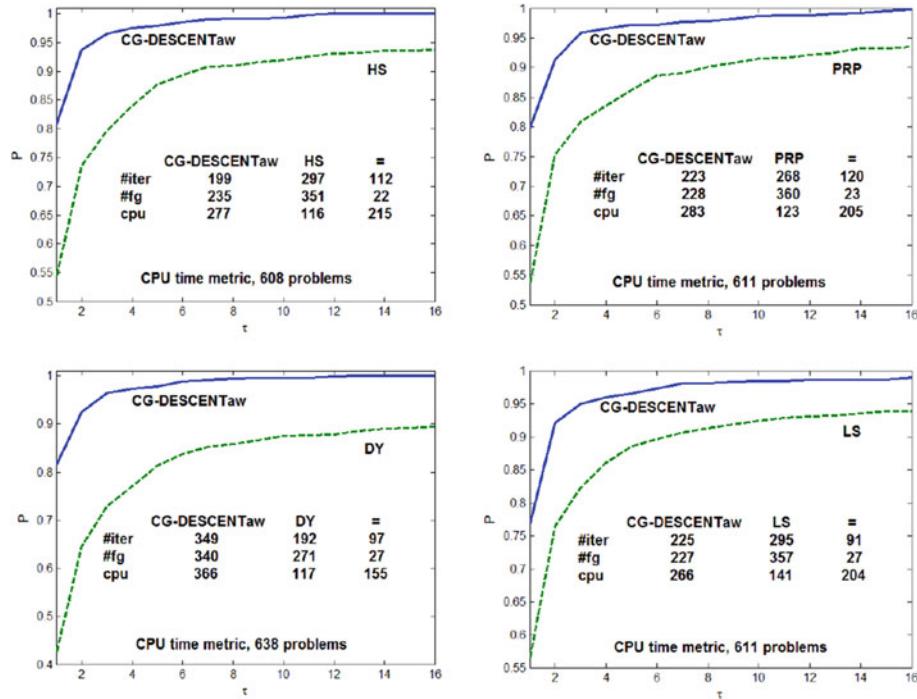
Figure 5.11 presents the Dolan and Moré performance profiles of CG-DESCENT (version 1.4) with Wolfe line-search versus the standard conjugate gradient methods HS, PRP, DY, and LS. On the other hand, Fig. 5.12 presents the performance profiles of CG-DESCENT with approximate Wolfe line-search (CG-DESCENTaw) versus the standard conjugate gradient methods HS, PRP, DY, and LS.

From Figs. 5.11 and 5.12, it is clear that CG-DESCENT and CG-DESCENTaw are more efficient and more robust than the standard conjugate gradient methods considered in this numerical study. CG-DESCENT is a modification of the self-scaling BFGS method of Perry and Shanno. Therefore, it is able to capture the curvature of the minimizing function better. Besides, CG-DESCENTaw implements the approximate Wolfe line-search (2.67). The difference between these two versions of CG-DESCENT is important. In contrast to the Wolfe conditions, the approximate Wolfe conditions (2.67) are satisfied at a minimizer of  $\varphi_k(\alpha) = f(x_k + ad_k)$ . Therefore, when trying to satisfy the approximate Wolfe conditions, we focus on minimizing  $\varphi_k$ ; when trying to satisfy the usual Wolfe conditions, we focus on minimizing  $\psi_k(\alpha) = \varphi_k(\alpha) - \varphi_k(0) - \alpha\rho\dot{\varphi}_k(0)$ .

Since  $\psi_k(0) = 0$ , it is required that the local minimizer  $\alpha^*$  satisfy  $\psi_k(\alpha^*) < 0$  and  $\psi'_k(\alpha^*) = 0$ . But, these two relations together imply that the Wolfe conditions hold in a neighborhood of  $\alpha^*$  when  $\rho < \sigma$ . Although there is no theory to guarantee the convergence of the algorithm with approximate Wolfe conditions, however, Hager and Zhang pointed out that there is a numerical advantage in using the



**Fig. 5.11** Performance profiles of CG-DESCENT versus HS, PRP, DY, and LS



**Fig. 5.12** Performance profiles of CG-DESCENTaw (CG-DESCENT with approximate Wolfe conditions) versus HS, PRP, DY, and LS

approximate Wolfe conditions: with approximate Wolfe conditions the local minimizers are computed with the accuracy on the order of the machine epsilon rather than with the accuracy on the order of the square root of the machine epsilon.

We must emphasize that the conjugate gradient method has an  $n$ -step quadratic convergence property when  $\alpha_k$  is the minimum of  $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ .

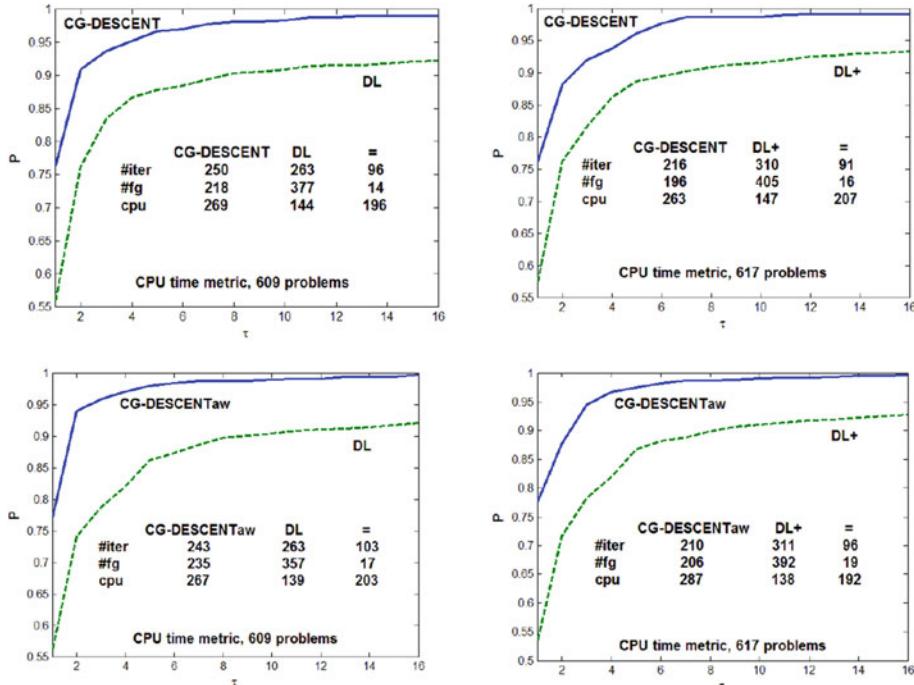
Figure 5.13 shows the performance profiles of CG-DESCENT and CG-DESCENTaw versus DL ( $t = 1$ ) and DL+ ( $t = 1$ ). Both CG-DESCENT and CG-DESCENTaw are more efficient and more robust than DL or DL+.

The performances of CG-DESCENT with Wolfe line-search (CG-DESCENT) and of CG-DESCENT with approximate Wolfe line-search (CG-DESCENTaw) for solving the MINPACK-2 applications are presented in Table 5.9. (#f represents the number of function calls.)

For solving the applications from the MINPACK-2, the performances of CG-DESCENTaw are similar to those of CG-DESCENT.

### The Conjugate Gradient with Guaranteed Descent and Conjugacy Conditions and a Modified Wolfe Line-Search (DESCON)

For solving the unconstrained optimization problem (5.46), Andrei (2013c) developed the following conjugate gradient algorithm:



**Fig. 5.13** Performance profiles of CG-DESCENT and CG-DESCENTaw (CG-DESCENT with approximate Wolfe conditions) versus DL ( $t = 1$ ) and DL+ ( $t = 1$ )

**Table 5.9** Performances of CG-DESCENT and CG-DESCENTaw for solving five applications from the MINPACK-2 collection

	<i>n</i>	CG-DESCENT			CG-DESCENTaw		
		#iter	#f	cpu	#iter	#f	cpu
A1	40,000	323	647	9.67	323	647	7.77
A2	40,000	788	1577	31.35	788	1577	27.05
A3	40,000	1043	2088	64.96	1043	2088	66.03
A4	40,000	435	871	81.40	435	871	72.24
A5	40,000	286	573	9.89	286	573	13.25
Total	—	2875	5756	197.27	2875	5756	186.34

$$x_{k+1} = x_k + \alpha_k d_k, \quad (5.221)$$

where  $\alpha_k > 0$  is obtained by a variant of the Wolfe line-search discussed below and the directions  $d_k$  are generated as

$$d_{k+1} = -\theta_k g_{k+1} + \beta_k s_k, \quad (5.222)$$

$$\beta_k = \frac{y_k^T g_{k+1} - t_k s_k^T g_{k+1}}{y_k^T s_k}, \quad (5.223)$$

$d_0 = -g_0$ , where  $\theta_k$  and  $t_k$  are scalar parameters which are to be determined. Observe that in  $d_{k+1}$  given by (5.222),  $g_{k+1}$  is scaled by parameter  $\theta_k$  while parameter  $t_k$  in (5.223) is changed at every iteration. Algorithms of this form or their variations were studied in many papers. For example, Andrei (2007a, 2007b, 2007c, 2008c) considered a preconditioned conjugate gradient algorithm

where the preconditioner is a scaled memoryless BFGS matrix and the parameter scaling the gradient is selected as the spectral gradient. Andrei (2012) developed another conjugate gradient algorithm, in which the search direction satisfies both the descent and the conjugacy condition at every iteration. Stoer and Yuan (1995) studied the conjugate gradient algorithm on a subspace, where the search direction  $d_{k+1}$  is taken from the subspace  $\text{span}\{-g_{k+1}, d_k\}$ . Observe that, if for every  $k \geq 1$ ,  $\theta_k = 1$  and  $t_k = t$ , then (5.222) reduces to the Dai and Liao search direction.

Suppose that the Assumption CG holds. Therefore, there exists a constant  $\Gamma \geq 0$  so that  $\|\nabla f(x)\| \leq \Gamma$  for all  $x \in S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ . Besides, it is easy to see that  $\|s_k\| = \|x_{k+1} - x_k\| \leq \|x_{k+1}\| + \|x_k\| \leq 2B$ , where  $B \geq 0$  is a scalar.

In our algorithm, for all  $k \geq 0$  the scalar parameters  $\theta_k$  and  $t_k$  in (5.222) and (5.223), respectively, are determined in such a way so that both the sufficient descent and the conjugacy conditions are satisfied. Therefore, from the *sufficient descent condition*

$$g_{k+1}^T d_{k+1} \leq -w \|g_{k+1}\|^2 \quad (5.224)$$

it follows that

$$-\theta_k \|g_{k+1}\|^2 + \frac{(y_k^T g_{k+1})(s_k^T g_{k+1})}{y_k^T s_k} - t_k \frac{(s_k^T g_{k+1})^2}{y_k^T s_k} = -w \|g_{k+1}\|^2 \quad (5.225)$$

and from the Dai-Liao *conjugacy condition*

$$d_{k+1}^T y_k = -v (g_{k+1}^T s_k) \quad (5.226)$$

it results that

$$-\theta_k y_k^T g_{k+1} + y_k^T g_{k+1} - t_k s_k^T g_{k+1} = -v (s_k^T g_{k+1}), \quad (5.227)$$

where  $v > 0$  and  $w > 0$  are *known* scalar parameters. Observe that in (5.225) the classical sufficient descent condition (5.224) is modified with equality. As known, the main condition in any conjugate gradient algorithm is the descent condition  $g_k^T d_k < 0$  or the sufficient descent condition (5.224). In (5.224),  $w$  is selected close to 1. This is quite a reasonable value. For example, Hager and Zhang (2005, 2006a) showed that in their CG-DESCENT algorithm,  $w = 7/8$ . On the other hand, the conjugacy condition is not so strict. In fact, very few conjugate gradient algorithms satisfy this condition. For example, the Hestenes and Stiefel algorithm has the property that the pure conjugacy condition always holds, independent of the line-search.

If  $v = 0$ , then (5.227) is the “pure” conjugacy condition. However, in order to improve the algorithm and to incorporate the second order information,  $v > 0$  is taken.

Now, let us define

$$\bar{\Delta}_k \equiv (y_k^T g_{k+1})(s_k^T g_{k+1}) - \|g_{k+1}\|^2 (y_k^T s_k), \quad (5.228)$$

$$\Delta_k \equiv (s_k^T g_{k+1}) \bar{\Delta}_k, \quad (5.229)$$

$$a_k \equiv v (s_k^T g_{k+1}) + y_k^T g_{k+1}, \quad (5.230)$$

$$b_k \equiv w \|g_{k+1}\|^2 (y_k^T s_k) + (y_k^T g_{k+1})(s_k^T g_{k+1}). \quad (5.231)$$

Supposing that  $\Delta_k \neq 0$  and  $y_k^T g_{k+1} \neq 0$ , then, from the linear algebraic system given by (5.225) and (5.227),  $t_k$  and  $\theta_k$  can be determined as

$$t_k = \frac{b_k(y_k^T g_{k+1}) - a_k(y_k^T s_k) \|g_{k+1}\|^2}{\Delta_k}, \quad (5.232)$$

$$\theta_k = \frac{a_k - t_k(s_k^T g_{k+1})}{y_k^T g_{k+1}}, \quad (5.233)$$

with which the parameter  $\beta_k$  and the direction  $d_{k+1}$  can immediately be computed. Observe that by using (5.232) in (5.233),  $\theta_k$  can be expressed as

$$\theta_k = \frac{a_k}{y_k^T g_{k+1}} \left[ 1 + \frac{(y_k^T s_k) \|g_{k+1}\|^2}{\bar{\Delta}_k} \right] - \frac{b_k}{\bar{\Delta}_k}. \quad (5.234)$$

Again, by using (5.232) in (5.223),  $\beta_k$  can be determined as

$$\beta_k = \frac{y_k^T g_{k+1}}{y_k^T s_k} \left( 1 - \frac{b_k}{\bar{\Delta}_k} \right) + a_k \frac{\|g_{k+1}\|^2}{\bar{\Delta}_k}. \quad (5.235)$$

Note that the crucial element in our algorithm is  $\bar{\Delta}_k$ . Therefore, the proposed conjugate gradient algorithm with guaranteed descent and conjugacy conditions is defined by (5.221) and (5.222), where the scalar parameters  $\theta_k$  and  $\beta_k$  are given by (5.234) and (5.235), respectively, and  $\alpha_k$  is computed by a variant of the Wolfe line-search, which is discussed in the following.

*Modified Wolfe line-search conditions.* In order to define the algorithm, a small modification of the second Wolfe line-search condition (5.50) is considered as

$$g(x_k + \alpha_k d_k)^T d_k \geq \sigma_k g_k^T d_k, \quad (5.236)$$

where  $\sigma_k$  is a sequence of the parameters satisfying the condition  $0 < \rho < \sigma_k < 1$  for all  $k$ . The interpretation of (5.236) is that the rate of decrease of  $f$  in the direction  $d_k$  at  $x_{k+1}$  is larger than a fraction  $\sigma_k$  of the rate of decrease of  $f$  in the direction  $d_k$  at  $x_k$ . Note that  $\sigma_k$  is modified at every iteration. The condition  $\rho < \sigma_k$  for all  $k \geq 0$  guarantees that the Wolfe line-search (5.49) and (5.236) can be satisfied simultaneously. Relations (5.49) and (5.236) are called *the modified Wolfe line-search conditions*.

**Proposition 5.13** *If*

$$\frac{1}{2} < \sigma_k \leq \frac{\|g_{k+1}\|^2}{|y_k^T g_{k+1}| + \|g_{k+1}\|^2}, \quad (5.237)$$

*then, for all  $k \geq 1$ ,  $\bar{\Delta}_k < 0$ .*

**Proof** Observe that

$$s_k^T g_{k+1} = s_k^T y_k + s_k^T g_k < s_k^T y_k. \quad (5.238)$$

The modified Wolfe condition (5.236) gives

$$g_{k+1}^T s_k \geq \sigma_k g_k^T s_k = -\sigma_k y_k^T s_k + \sigma_k g_{k+1}^T s_k. \quad (5.239)$$

Since  $\sigma_k < 1$ , (5.239) can be rearranged to obtain

$$g_{k+1}^T s_k \geq \frac{-\sigma_k}{1 - \sigma_k} y_k^T s_k. \quad (5.240)$$

Since  $y_k^T s_k > 0$  (if  $\|g_k\| \neq 0$ ), the combination of this lower bound for  $g_{k+1}^T s_k$  with the upper bound (5.238) gives

$$|g_{k+1}^T s_k| \leq |y_k^T s_k| \max \left\{ 1, \frac{\sigma_k}{1 - \sigma_k} \right\}. \quad (5.241)$$

But, since  $\sigma_k > 1/2$ , from (5.241) it follows that

$$|g_{k+1}^T s_k| < \frac{\sigma_k}{1 - \sigma_k} |y_k^T s_k|. \quad (5.242)$$

If (5.237) is true, then

$$\frac{\sigma_k}{1 - \sigma_k} |y_k^T g_{k+1}| \leq \|g_{k+1}\|^2. \quad (5.243)$$

From (5.243), since  $y_k^T s_k > 0$ , it follows that

$$\frac{\sigma_k}{1 - \sigma_k} |y_k^T s_k| |g_{k+1}^T y_k| \leq |y_k^T s_k| \|g_{k+1}\|^2. \quad (5.244)$$

Now, from (5.242) and (5.244) it results that

$$|s_k^T g_{k+1}| |y_k^T g_{k+1}| < \frac{\sigma_k}{1 - \sigma_k} |y_k^T s_k| |y_k^T g_{k+1}| \leq |y_k^T s_k| \|g_{k+1}\|^2, \quad (5.245)$$

i.e.,  $\bar{\Delta}_k < 0$  for all  $k \geq 1$ . ◆

Therefore, in our algorithm,  $\sigma_k$  is computed as

$$\sigma_k = \frac{\|g_{k+1}\|^2}{|y_k^T g_{k+1}| + \|g_{k+1}\|^2}. \quad (5.246)$$

If  $g_k \neq 0$  for all  $k \geq 0$ , then  $0 < \sigma_k < 1$  for all  $k \geq 0$ .

Taking into account the acceleration Scheme (3.32), see Remark 3.1, where the acceleration factor  $\eta_k$  is computed as in (3.33), according to the value of the parameter “acceleration” (true or false), the following algorithms DESCON and DESCONa can be presented. DESCONa is the accelerated version of DESCON.

**Algorithm 5.6** *Guaranteed descent and conjugacy conditions with a modified Wolfe line-search: DESCON / DESCONa*

- |    |  |
|----|--|
| 1. | Select a starting point $x_0 \in \text{dom } f$ and compute: $f_0 = f(x_0)$ and $g_0 = \nabla f(x_0)$ . Select some positive values for $\rho$ and $\sigma_0$ as well as for $v$ and $w$ . Set $d_0 = -g_0$ and $k = 0$ . Select the small positive constants $\varepsilon_A$ and $\varepsilon_m$ sufficiently small |
| 2. | Test a criterion for stopping the iterations. If the test is satisfied, then stop; otherwise, continue with step 3   |
| 3. | Determine the stepsize $\alpha_k$ by the modified Wolfe line-search conditions (5.49) and (5.246). Update the variables as $x_{k+1} = x_k + \alpha_k d_k$ . Compute $f_{k+1}$ and $g_{k+1}$ . Compute $y_k = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$  |

4.	If <i>acceleration</i> is true, then:
	(a) Compute: $z = x_k + \alpha_k d_k$ , $g_z = \nabla f(z)$ and $y_k = g_k - g_z$
	(b) Compute: $\bar{a}_k = \alpha_k g_k^T d_k$ , and $\bar{b}_k = -\alpha_k y_k^T d_k$
	(c) If $ \bar{b}_k  \geq \varepsilon_A$ , then compute $\eta_k = -\bar{a}_k / \bar{b}_k$ and update the variables as $x_{k+1} = x_k + \eta_k \alpha_k d_k$ . Compute $f_{k+1}$ and $g_{k+1}$ . Compute $y_k = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$
5.	Compute $\bar{\Delta}_k$ as in (5.228)
6.	If $ \bar{\Delta}_k  \geq \varepsilon_m$ , then determine $\theta_k$ and $\beta_k$ as in (5.234) and (5.235), respectively; else set $\theta_k = 1$ and $\beta_k = 0$
7.	Compute the search direction as: $d_{k+1} = -\theta_k g_{k+1} + \beta_k s_k$
8.	Compute $\sigma_k = \ g_{k+1}\ ^2 / (\ y_k^T g_{k+1}\  + \ g_{k+1}\ ^2)$
9.	Restart criterion. If $ g_{k+1}^T g_k  > 0.2 \ g_{k+1}\ ^2$ then set $d_{k+1} = -g_{k+1}$
10.	Consider $k = k + 1$ and go to step 2

◆

Under reasonable assumptions, the modified Wolfe line-search conditions and the Powell restart criterion are sufficient to prove the global convergence of the algorithm. The first trial of the stepsize crucially affects the practical behavior of the algorithm. At every iteration  $k \geq 1$ , the starting guess for the step  $\alpha_k$  in the line-search is computed as  $\alpha_k = 1/\|d_{k-1}\|/\|d_k\|$ . Observe that in the line-search procedure (step 3 of the algorithm), the stepsize  $\alpha_k$  is computed by using the updated value of the parameter  $\sigma_k$  computed as in step 8. For strongly convex functions, the linear convergence of the acceleration scheme is proved as in Andrei (2009c).

*The convergence for general nonlinear functions.* In order to prove the global convergence of DESCON, assume that the stepsize  $\alpha_k$  satisfies the strong Wolfe line-search conditions (5.49) and

$$|g(x_k + \alpha_k d_k)^T d_k| \leq -\sigma_k g_k^T d_k, \quad (5.247)$$

where  $\rho$  and  $\sigma_k$  are arbitrary positive constants so that  $0 < \rho < \sigma_k < 1$ . Observe that, since  $\rho$  is small enough, the parameter  $\sigma_k$  in (5.247) can be selected at each iteration as in (5.246), thus satisfying the above condition  $0 < \rho < \sigma_k < 1$ .

In the following, let us prove that in very mild conditions the direction  $d_k$  generated by (5.222), where  $\theta_k$  and  $\beta_k$  are given by (5.234) and (5.235), respectively, is bounded.

**Theorem 5.19** Suppose that the Assumption CG holds and  $\|g_k\| \geq \gamma > 0$  for all  $k \geq 0$ . Consider the conjugate gradient algorithm (5.221), where the direction  $d_{k+1}$  given by (5.222) and (5.223) satisfies the descent condition  $g_k^T d_k = -w \|g_k\|^2$ , where  $w > 1$  and the stepsize  $\alpha_k$  is obtained by the strong Wolfe line-search (5.49) and (5.247), where  $1/2 \leq \sigma_k < 1$ . Then,  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ .

**Proof** From (5.223), by using (5.232), after some algebraic manipulations, we have

$$\beta_k = \frac{y_k^T g_{k+1}}{y_k^T s_k} \left( 1 - \frac{b_k}{\bar{\Delta}_k} \right) + a_k \frac{\|g_{k+1}\|^2}{\bar{\Delta}_k}. \quad (5.248)$$

From: the definition of  $\omega_k$ , the modified Wolfe condition (5.236) and from the descent condition  $g_k^T d_k = -w \|g_k\|^2$ , since  $\|g_k\| \geq \gamma > 0$  and  $\sigma_k < 1$  for all  $k \geq 0$ , it follows that

$$y_k^T s_k \geq w \omega_k (1 - \sigma_k) \gamma^2 > w \omega (1 - \sigma_k) \gamma^2 > 0.$$

However, from the Assumption CG,

$$|y_k^T g_{k+1}| \|s_k\| \leq \|y_k\| \|g_{k+1}\| \|s_k\| \leq L \|s_k\|^2 \Gamma \leq L \Gamma (2B)^2.$$

Therefore,

$$\frac{|y_k^T g_{k+1}|}{|y_k^T s_k|} \leq \frac{L \Gamma (2B)^2}{w \omega (1 - \sigma_k) \gamma^2} \frac{1}{\|s_k\|} = \frac{\bar{c}}{\|s_k\|}, \quad (5.249)$$

where

$$\bar{c} = \frac{L \Gamma (2B)^2}{w \omega (1 - \sigma_k) \gamma^2}.$$

Now, observe that, since for all  $k \geq 0$ ,  $\bar{\Delta}_k < 0$  and  $b_k > 0$ , it follows that  $-b_k/\bar{\Delta}_k > 0$ . Besides, from (5.228) and (5.231), it follows that

$$-\frac{b_k}{\bar{\Delta}_k} = w + (1 + w) \frac{(y_k^T g_{k+1})(s_k^T g_{k+1})}{-\bar{\Delta}_k}. \quad (5.250)$$

Since  $-\bar{\Delta}_k > 0$  and  $s_k^T g_{k+1}$  tends to zero along the iterations, it follows that  $-b_k/\bar{\Delta}_k$  tends to  $w > 0$ . Hence  $1 - b_k/\bar{\Delta}_k$  tends to  $1 + w$ . Therefore, there exists a positive constant  $c_4 > 1$  so that  $1 < 1 - b_k/\bar{\Delta}_k \leq c_4$ .

Again, from the Assumption CG,

$$|y_k^T s_k| \|s_k\| \leq \|y_k\| \|s_k\|^2 \leq L \|s_k\|^3 \leq L (2B)^3.$$

Therefore,  $|y_k^T s_k| \leq L (2B)^3 / \|s_k\|$ . Now, from (5.230) and (5.241), the following estimation is obtained

$$\begin{aligned} |a_k| &= |v(s_k^T g_{k+1}) + (y_k^T g_{k+1})| \leq v|s_k^T g_{k+1}| + |y_k^T g_{k+1}| \\ &\leq v|y_k^T s_k| \max \left\{ 1, \frac{\sigma_k}{1 - \sigma_k} \right\} + |y_k^T g_{k+1}| \\ &\leq v \frac{L (2B)^3}{\|s_k\|} \max \left\{ 1, \frac{\sigma_k}{1 - \sigma_k} \right\} + \frac{L \Gamma (2B)^2}{\|s_k\|}. \end{aligned} \quad (5.251)$$

Since  $1/2 \leq \sigma_k < 1$ , there exists a positive constant  $c_5 > 0$  so that  $\max\{1, \sigma_k/(1 - \sigma_k)\} \leq c_5$ . Hence,

$$|a_k| \leq \left( v L c_5 (2B)^3 + L \Gamma (2B)^2 \right) \frac{1}{\|s_k\|} = \frac{\hat{c}}{\|s_k\|}, \quad (5.252)$$

where  $\hat{c} = v L c_5 (2B)^3 + L \Gamma (2B)^2$ . With this, from (5.248) the following estimation is obtained:

$$\begin{aligned} |\beta_k| &\leq \left| \frac{y_k^T g_{k+1}}{y_k^T s_k} \right| \left| 1 - \frac{b_k}{\bar{\Delta}_k} \right| + |a_k| \frac{\|g_{k+1}\|^2}{|\bar{\Delta}_k|} \leq \frac{\bar{c} c_4}{\|s_k\|} + \frac{\bar{c} \Gamma^2}{c_3} \frac{1}{\|s_k\|} \\ &= \left[ \bar{c} c_4 + \frac{\bar{c} \Gamma^2}{c_3} \right] \frac{1}{\|s_k\|}. \end{aligned} \quad (5.253)$$

Therefore, from (5.222) it follows that

$$\begin{aligned}\|d_{k+1}\| &\leq |\theta_k| \|g_{k+1}\| + |\beta_k| \|s_k\| \\ &\leq c_2 \Gamma + \left[ \frac{\widehat{c} \Gamma^2}{cc_4 + \frac{\widehat{c} \Gamma^2}{c_3}} \right] \frac{1}{\|s_k\|} \|s_k\| \equiv E,\end{aligned}\quad (5.254)$$

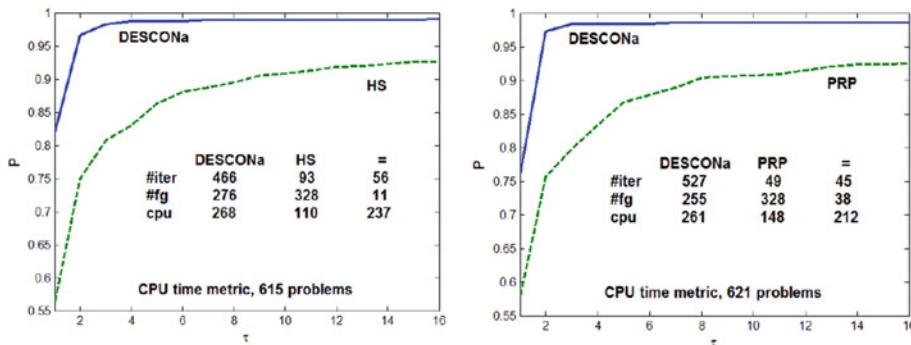
where  $E$  is a positive constant. Hence, for all  $k \geq 0$ ,  $\|d_k\| \leq E$  implies  $\sum_{k \geq 1} 1/\|d_k\| = \infty$ . Since  $d_k$  is a descent direction, it follows that  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ .  $\blacklozenge$

## Numerical Study: DESCON

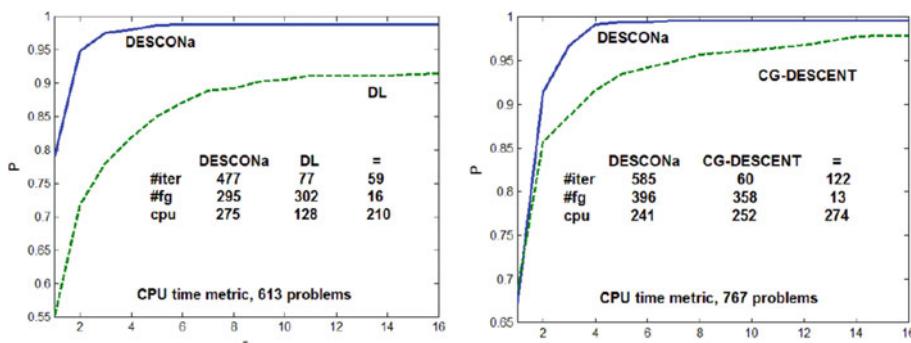
Let us solve the problems from the UOP collection with the number of variables  $n = 1000, 2000, \dots, 10000$ . The DESCON algorithm implements the Wolfe line-search conditions with  $\rho = 0.0001$ ,  $\sigma = \|g_{k+1}\|^2 / (|y_k^T g_{k+1}| + \|g_{k+1}\|^2)$ . In DESCON we take  $w = 7/8$  and  $v = 0.05$ . The numerical experiments are given in the context of Remark 1.1. Figure 5.14 shows the performance profiles of the accelerated variant of DESCON (DESCONa) versus the standard conjugate gradient algorithms HS and PRP.

Figure 5.15 shows the performance profiles of DESCONa versus DL ( $t = 1$ ) and versus CG-DESCENT (version 1.4) with Wolfe line-search.

The performance profiles of accelerated DESCON (DESCONa) versus CG-DESCENT with approximate Wolfe line-search (CG-DESCENTaw) are illustrated in Fig. 5.16.

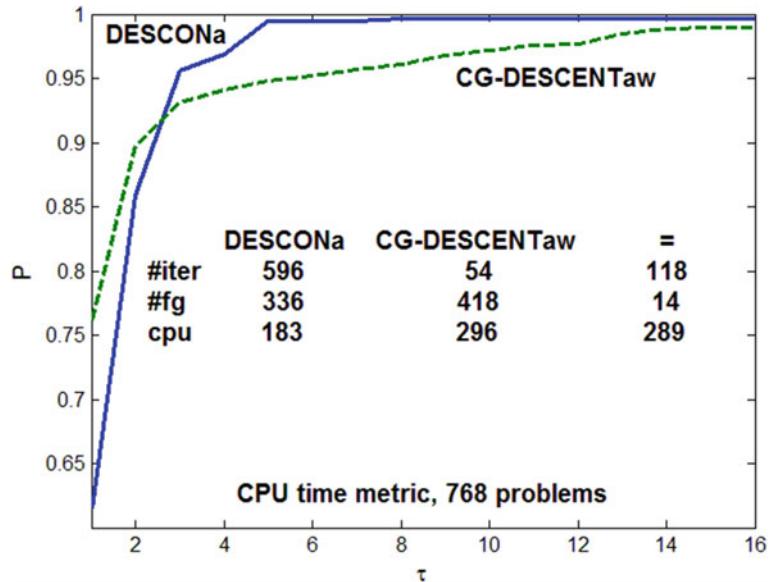


**Fig. 5.14** Performance profile of DESCONa versus HS and versus PRP



**Fig. 5.15** Performance profile of DESCONa versus DL ( $t = 1$ ) and versus CG-DESCENT

**Fig. 5.16** Performances of DESCONa versus CG-DESCENTaw



**Table 5.10** Performances of DESCONa for solving five applications from the MINPACK-2 collection

		DESCONa		
	<i>n</i>	#iter	#fg	cpu
A1	40,000	241	510	4.12
A2	40,000	631	1297	12.11
A3	40,000	1036	2100	32.25
A4	40,000	299	632	23.80
A5	40,000	278	576	6.71
Total	—	2485	5115	78.99

Figure 5.16 points out that DESCONa is top performer in comparison with CG-DESCENTaw. However, CG-DESCENT with approximate Wolfe line-search is more efficient than DESCONa. Figure 5.15 shows that both DESCONa and CG-DESCENT with Wolfe line-search practically have the same efficiency. From the table inside Fig. 5.15 notice that DESCONa was faster in 241 problems, while CG-DESCENT in 252 problems. Once again, this illustrates the importance of the line-search in conjugate gradient algorithms. An accurate line-search increases the performances of the algorithm.

Table 5.10 presents the performances of DESCONa for solving the applications from the MINPACK-2 collection, where  $nx = 200$  and  $ny = 200$ .

## 5.7 Conjugate Gradient Methods Memoryless BFGS Preconditioned

The purpose of this section is to see how the second order information of the minimizing function may be used in the formula for the search direction computation. Thus, new conjugate gradient algorithms with better convergence properties are obtained. In fact, these methods include a special preconditioning of conjugate gradient algorithms by using the BFGS updating. Memoryless quasi-Newton methods were first introduced by Perry (1977) and Shanno (1978a). These methods can be

considered as the quasi-Newton methods, for which the approximation to the inverse of the Hessian is taken as the identity matrix at every iteration.

Four such BFGS preconditioned conjugate gradient methods are known. The first preconditioned conjugate gradient method called CONMIN, developed by Shanno (1983), is based on the idea that the conjugate gradient methods are exactly the BFGS quasi-Newton method, where the approximation to the inverse Hessian of the minimizing function is restarted as the identity matrix at every iteration. The second one, developed by Andrei (2007a, 2007b, 2007c, 2010b), is using scaling in the frame of the memoryless BFGS method. In the third method, called DK/CGOPT, the search direction is chosen in one-dimensional manifold closest to the Perry-Shanno self-scaling memoryless BFGS method, developed by Dai and Kou (2013). Finally, using the trace and the determinant or a combination of these operators of the self-scaling memoryless BFGS iteration matrix, a new conjugate gradient method is developed (Andrei, 2019c). Only CONMIN and DK/CGOPT are presented in this section (Andrei, 2020a).

### The Memoryless BFGS Preconditioned Conjugate Gradient (CONMIN)

Perry (1976) noted that the search direction  $d_{k+1} = -g_{k+1} + \beta_k d_k$ , where  $\beta_k$  is given by  $\beta_k = g_{k+1}^T y_k / d_k^T y_k$ , can be written as

$$d_{k+1} = -\left[I - \frac{d_k y_k^T}{y_k^T d_k}\right] g_{k+1}. \quad (5.255)$$

Defining  $s_k = \alpha_k d_k = x_{k+1} - x_k$  and since  $d_k y_k^T / y_k^T d_k = s_k y_k^T / y_k^T s_k$ , (5.255) can be rewritten as

$$d_{k+1} = -\left[I - \frac{s_k y_k^T}{y_k^T s_k}\right] g_{k+1}. \quad (5.256)$$

Perry relaxed the assumption that  $\alpha_k$  is chosen to minimize  $f(x_k + \alpha d_k)$  and added the correction term  $s_k s_k^T / y_k^T s_k$  to the matrix from (5.256) and proposed the modified search direction

$$d_{k+1} = -\left[I - \frac{s_k y_k^T}{y_k^T s_k} + \frac{s_k s_k^T}{y_k^T s_k}\right] g_{k+1} \equiv -Q_{k+1} g_{k+1}, \quad (5.257)$$

where  $Q_{k+1}$  defined in (5.257) satisfies the equation

$$y_k^T Q_{k+1} = s_k^T, \quad (5.258)$$

which is similar, but not identical to the secant equation

$$H_{k+1} y_k = s_k, \quad (5.259)$$

where  $H_{k+1}$  is an approximation to the inverse Hessian. If the line-search is exact, then  $s_k^T g_{k+1} = 0$  and (5.257) is identical to the HS method.

The major difficulty with this approach, not corrected by Perry, is that the matrix  $Q_{k+1}$  is not symmetric and therefore it is not positive definite. Thus, the search directions  $d_{k+1}$  given by (5.257) are not necessarily descent directions and hence numerical instabilities can result, ruining the convergence of the algorithm. To overcome this difficulty, Shanno (1978a, 1978b) investigated the relationship between the conjugate gradient and the BFGS methods.

As known, the best quasi-Newton method is BFGS, where the updating to the inverse Hessian of the minimizing function is given by

$$H_{k+1} = H_k - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k} + \left( 1 + \frac{y_k^T H_k y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}. \quad (5.260)$$

The major difference between the conjugate gradient and the quasi-Newton methods is the presence of the matrix  $H_k$  in the updating formula for the search direction  $d_{k+1}$ . For problems with a large number of variables, it is impossible to store an approximation to the inverse Hessian, and therefore the conjugate gradient methods are preferred.

As in Shanno (1978a), to see the relationship between the conjugate gradient and the BFGS methods, the matrix  $Q_{k+1}$  defined by (5.257) is first symmetrized to get

$$Q_{k+1} = I - \frac{s_k y_k^T}{y_k^T s_k} - \frac{y_k s_k^T}{y_k^T s_k} + \frac{s_k s_k^T}{y_k^T s_k}. \quad (5.261)$$

Afterwards, let us force  $Q_{k+1}$  defined by (5.261) to satisfy the quasi-Newton eq. (5.259), yielding the symmetric update

$$Q_{k+1}^* = I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + \left( 1 + \frac{y_k^T y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}. \quad (5.262)$$

Observe that, if in (5.260)  $H_k = I$  then it results precisely  $Q_{k+1}^*$  from (5.262). Therefore, the important result noticed for the first time by Shanno was that *the conjugate gradient methods are precisely the BFGS quasi-Newton methods, where the approximation to the inverse Hessian is restarted as the identity matrix at every iteration*. Hence, the conjugate gradient methods are often called *memoryless quasi-Newton methods*.

**Proposition 5.14** *If  $x, y, z \in \mathbb{R}^n$ , then*

$$(x^T y)z = (zy^T)x \text{ and } x^T(yz^T) = (x^T y)z^T.$$

**Proof** The above equalities are obtained by direct calculation. ◆

By using Proposition 5.14, a conjugate gradient method in which the search directions are computed as

$$d_{k+1} = -Q_{k+1}^* g_{k+1} \quad (5.263)$$

does not explicitly require the construction of  $Q_{k+1}^*$ . Indeed, from (5.263) and (5.262), it is easy to get

$$d_{k+1} = -g_{k+1} - \left[ \left( 1 + \frac{y_k^T y_k}{y_k^T s_k} \right) \frac{s_k^T g_{k+1}}{y_k^T s_k} - \frac{y_k^T g_{k+1}}{y_k^T s_k} \right] s_k + \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k, \quad (5.264)$$

which shows that no additional information is needed more than that required by the known standard conjugate gradient methods.

To get a superlinear convergence of a conjugate gradient method in which the search direction is computed as  $d_{k+1} = -g_{k+1} + \beta_k d_k$  with  $\beta_k$  given by the HS method, Beale (1972) suggested a modification of the search direction as

$$d_{t+1} = -g_{t+1} + \beta_t d_t, \quad (5.265)$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k + \mu_k d_t, \quad (5.266)$$

where

$$\mu_k = \frac{y_t^T g_{k+1}}{y_t^T d_t}, \quad \text{with } t < k < t+n \quad (5.267)$$

In particular, starting with  $t = 0$ , at every  $n$  iterations a new pair of vectors  $d_t$  and  $y_t$  are stored, known as *the restart vectors of Beale*, and the generated search directions are conjugate subject to these restart vectors as well as to the previous ones. After  $n$  iterations, the restart direction is replaced by the current one.

Powell (1977) made a modification of the restarting of the conjugate gradient methods. Powell's criterion was to use (5.266) whenever

(a)  $k$  (or  $k - t$ ) is a multiple of  $n$ , or if

$$|g_{k+1}^T g_k| \geq 0.2 \|g_{k+1}\|^2.$$

Obviously, (5.266) may be written as

$$d_{k+1} = -\left[ I - \frac{d_k y_k^T}{y_k^T d_k} - \frac{d_t y_t^T}{y_t^T d_t} \right] g_{k+1} \equiv -P_k g_{k+1}, \quad (5.268)$$

where  $P_k$ , which modifies the gradient  $g_{k+1}$ , is an update using the information from two prior points. It is easy to show that if  $f(x)$  is quadratic and the line-search is exact, then  $P_k$  is a projection matrix of rank  $n - 2$  rather than rank  $n - 1$  of the matrix from (5.255).

Using this double update with information in two points and keeping the philosophy behind the matrix  $Q_{k+1}^*$  from (5.262), based on the fact that the conjugate gradient method is exactly the memoryless BFGS method, Shanno (1978a) suggested the following search direction:

$$d_{k+1} = -H_{k+1} g_{k+1}, \quad (5.269)$$

where

$$H_{k+1} = \widehat{H}_k - \frac{\widehat{H}_k y_k s_k^T + s_k y_k^T \widehat{H}_k}{y_k^T s_k} + \left( 1 + \frac{y_k^T \widehat{H}_k y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}, \quad (5.270)$$

with

$$\widehat{H}_k = I - \frac{y_t s_t^T + s_t y_t^T}{y_t^T s_t} + \left( 1 + \frac{y_t^T y_t}{y_t^T s_t} \right) \frac{s_t s_t^T}{y_t^T s_t}. \quad (5.271)$$

Observe that, if in (5.270)  $\widehat{H}_k = I$  is set, then exactly the matrix  $Q_{k+1}^*$  is obtained. Besides, observe that the conjugate gradient method (5.269)–(5.271) does not imply additional memory requirements. From Proposition 5.14 it is clear that the search direction may be written as

$$\begin{aligned} d_{k+1} &= -\hat{H}_k g_{k+1} + \frac{s_k^T g_{k+1}}{y_k^T s_k} \hat{H}_k y_k \\ &\quad - \left[ \left( 1 + \frac{y_k^T \hat{H}_k y_k}{y_k^T s_k} \right) \frac{s_k^T g_{k+1}}{y_k^T s_k} - \frac{y_k^T \hat{H}_k g_{k+1}}{y_k^T s_k} \right] s_k, \end{aligned} \quad (5.272)$$

where the vectors  $\hat{H}_k g_{k+1}$  and  $\hat{H}_k y_k$  are computed as

$$\hat{H}_k g_{k+1} = g_{k+1} - \frac{s_t^T g_{k+1}}{y_t^T s_t} y_t + \left[ \left( 1 + \frac{y_t^T y_t}{y_t^T s_t} \right) \frac{s_t^T g_{k+1}}{y_t^T s_t} - \frac{y_t^T g_{k+1}}{y_t^T s_t} \right] s_t, \quad (5.273)$$

$$\hat{H}_k y_k = y_k - \frac{s_t^T y_k}{y_t^T s_t} y_t + \left[ \left( 1 + \frac{y_t^T y_t}{y_t^T s_t} \right) \frac{s_t^T y_k}{y_t^T s_t} - \frac{y_t^T y_k}{y_t^T s_t} \right] s_t. \quad (5.274)$$

It should be emphasized that the implementation of this method requires only seven vectors to store:  $x_k$ ,  $x_{k+1}$ ,  $g_k$ ,  $g_{k+1}$ ,  $d_k$ ,  $d_t$  and  $y_t$ . By the time the update has been accomplished, the information in  $x_k$  is no longer required and so,  $\hat{H}_k y_k$  from (5.274) may be stored in  $x_k$ . Once  $\hat{H}_k y_k$  and the scalars  $y_k^T s_k$  and  $y_k^T (\hat{H}_k y_k)$  have been computed,  $g_k$  is no longer needed, so  $\hat{H}_k g_{k+1}$  from (5.273) can be explicitly computed and stored in  $g_k$ . This double update scheme was first proposed by Perry (1976, 1977), but with  $y_t$  and  $s_t$  replaced by  $y_{k-1}$  and  $s_{k-1}$ , respectively.

Another modified conjugate gradient method can be obtained as

$$d_{k+1} = -\gamma g_{k+1} - \left[ \left( 1 + \gamma \frac{y_k^T y_k}{y_k^T s_k} \right) \frac{s_k^T g_{k+1}}{y_k^T s_k} - \gamma \frac{y_k^T g_{k+1}}{y_k^T s_k} \right] s_k + \gamma \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k, \quad (5.275)$$

where

$$\gamma = \frac{y_k^T s_k}{y_k^T y_k}, \quad (5.276)$$

where  $\gamma$  is a scaling parameter used only at the initial step. Now, substituting (5.276) in (5.275), after some simple algebraic manipulations, it results that

$$d_{k+1} = -\frac{y_k^T s_k}{y_k^T y_k} g_{k+1} - \left[ 2 \frac{s_k^T g_{k+1}}{y_k^T s_k} - \frac{y_k^T g_{k+1}}{y_k^T y_k} \right] s_k + \frac{s_k^T g_{k+1}}{y_k^T y_k} y_k, \quad (5.277)$$

which is the *Perry/Shanno ( $d_{k+1}^{PS}$ ) search direction*. Again, observe that if  $s_k^T g_{k+1} = 0$ , that is, if the line-search is exact, then

$$d_{k+1} = -\frac{y_k^T s_k}{y_k^T y_k} g_{k+1} + \frac{y_k^T g_{k+1}}{y_k^T y_k} s_k = \gamma \left[ -g_{k+1} + \frac{y_k^T g_{k+1}}{y_k^T s_k} s_k \right], \quad (5.278)$$

so the effect is one of multiplying the search direction (5.256) with a scalar, thus the  $n$ -step convergence to the minimum of a quadratic function being maintained.

Therefore, using this scaling technique in (5.271), it follows that

$$\hat{H}_k = \gamma_t \left[ I - \frac{s_t y_t^T + y_t s_t^T}{y_t^T s_t} + \frac{y_t^T y_t}{y_t^T s_t} \frac{s_t s_t^T}{y_t^T s_t} \right] + \frac{s_t s_t^T}{y_t^T s_t}, \quad (5.279)$$

where

$$\gamma_t = \frac{\mathbf{y}_t^T \mathbf{s}_t}{\mathbf{y}_t^T \mathbf{y}_t}.$$

With these developments, the conjugate gradient memoryless BFGS preconditioned algorithm is defined by (5.272), where, this time, the vectors  $\widehat{H}_k g_{k+1}$  and  $\widehat{H}_k y_k$  are computed as

$$\widehat{H}_k g_{k+1} = \frac{\mathbf{y}_t^T \mathbf{s}_t}{\mathbf{y}_t^T \mathbf{y}_t} g_{k+1} - \frac{\mathbf{s}_t^T g_{k+1}}{\mathbf{y}_t^T \mathbf{y}_t} \mathbf{y}_t + \left[ 2 \frac{\mathbf{s}_t^T g_{k+1}}{\mathbf{y}_t^T \mathbf{s}_t} - \frac{\mathbf{y}_t^T g_{k+1}}{\mathbf{y}_t^T \mathbf{y}_t} \right] \mathbf{s}_t, \quad (5.280)$$

$$\widehat{H}_k y_k = \frac{\mathbf{y}_t^T \mathbf{s}_t}{\mathbf{y}_t^T \mathbf{y}_t} y_k - \frac{\mathbf{s}_t^T y_k}{\mathbf{y}_t^T \mathbf{y}_t} \mathbf{y}_t + \left[ 2 \frac{\mathbf{s}_t^T y_k}{\mathbf{y}_t^T \mathbf{s}_t} - \frac{\mathbf{y}_t^T y_k}{\mathbf{y}_t^T \mathbf{y}_t} \right] \mathbf{s}_t. \quad (5.281)$$

As a final remark of these developments, the fact that, under the exact line-search the conjugate gradient methods are simple projections of the gradient, they make their initial length a poor approximation to the desired stepsize. Fletcher (1987) proposed scaling  $d_{k+1}$  by

$$\widehat{d}_{k+1} = \frac{2(f(x_{k+1}) - f(x_k))}{d_{k+1}^T g_{k+1}} d_{k+1}.$$

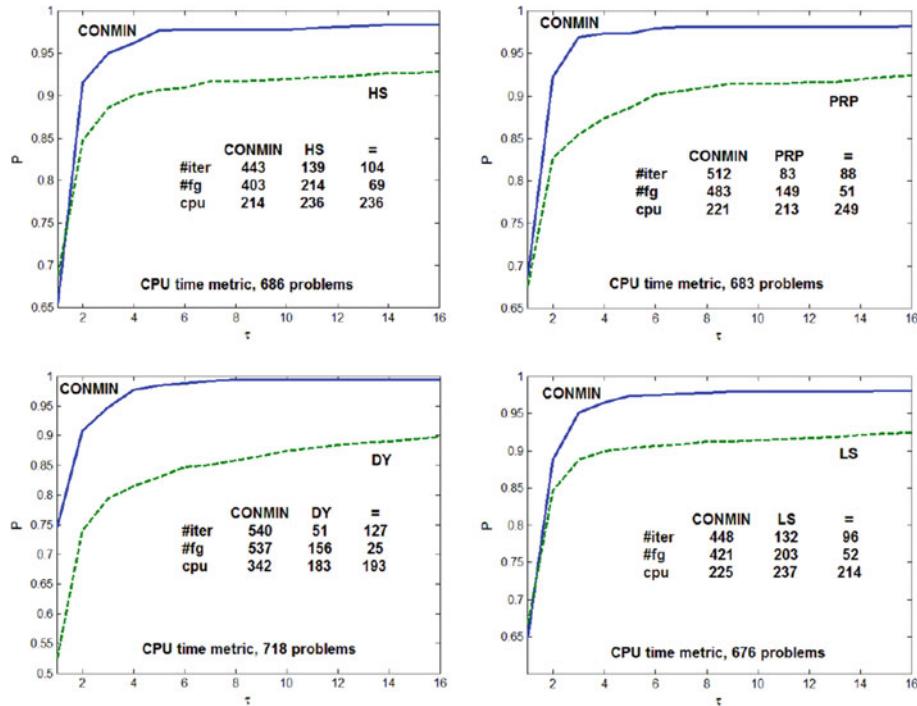
Observe that this scaling of the search direction includes the function values in two successive steps.

**Algorithm 5.7** *Conjugate gradient memoryless BFGS preconditioned: CONMIN*

1.	Choose an initial point $x_0 \in \mathbb{R}^n$ and $\epsilon > 0$ sufficiently small. Set $k = 0$ . Compute $f(x_k)$ , $g_k = \nabla f(x_k)$ and set $d_k = -g_k$
2.	Compute the stepsize $\alpha_k > 0$ satisfying the Wolfe line-search conditions (5.49) and (5.50)
3.	Compute $x_{k+1} = x_k + \alpha_k d_k$ , $f(x_{k+1})$ , $g_{k+1} = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$
4.	Test a criterion for stopping the iterations. If this criterion is satisfied, then stop; otherwise continue with step 5
5.	<p>Test the criterion for restarting the iterations. If:</p> <ul style="list-style-type: none"> <li>(a) Iteration <math>k</math> is a multiple of <math>n</math>, or</li> <li>(b) <math> g_{k+1}^T g_k  \geq 0.2 \ g_{k+1}\ ^2</math>,</li> </ul> <p>then compute the search direction <math>d_{k+1}</math> as in (5.275). Set <math>s_t = d_k</math>, <math>y_t = y_k</math>, <math>k = k + 1</math> and continue with step 2. Otherwise, continue with step 6</p>
6.	Compute the search direction $d_{k+1}$ as in (5.272), where the vectors $\widehat{H}_k g_{k+1}$ and $\widehat{H}_k y_k$ are computed as in (5.280) and (5.281), respectively
7.	<p>Scale the search direction <math>d_{k+1}</math> as</p> $d_{k+1} = [2(f(x_{k+1}) - f(x_k)) / d_{k+1}^T g_{k+1}] d_{k+1},$ <p>set <math>k = k + 1</math> and continue with step 2</p>

◆

For convex functions, using the exact line-search, Shanno (1978b) proved the convergence of Algorithm 5.7 when the Hessian of the minimizing function is strictly bounded. However, since the search direction is computed using the BFGS updating strategy and since the line-search is based on the Wolfe conditions and the Beale restart is implemented, the convergence of the algorithm is ensured. For general nonlinear functions bounded below with the level set bounded, Shanno showed that the algorithm is not possible to converge to a point where the gradient is bounded away from zero. Cycling in the form defined by Powell is the only way in which the conjugate gradient memoryless BFGS preconditioned can possibly fail to converge on general nonlinear functions.



**Fig. 5.17** Performance profiles of CONMIN versus HS, PRP, DY, and LS

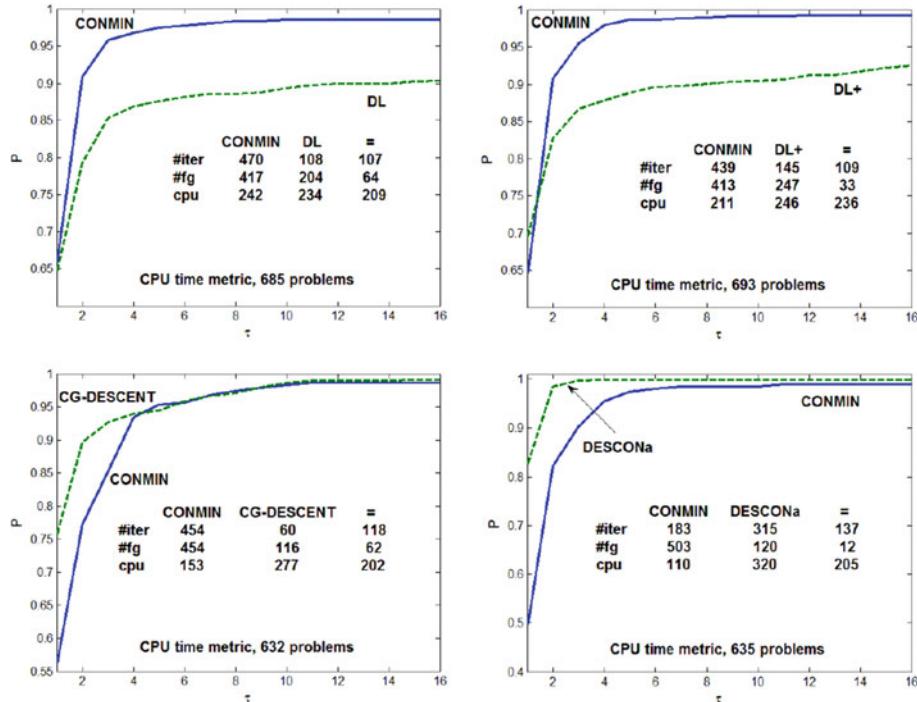
### Numerical Study: CONMIN

CONMIN is one of the oldest and most respectable conjugate gradient methods, implemented by Shanno (1983). In the following, let us present some results with this method and its comparisons versus some other methods for solving the unconstrained optimization test problems from the UOP collection (Andrei, 2020a).

In the first set of numerical experiments, CONMIN is compared versus the standard conjugate gradient methods HS, PRP, DY, and LS. Figure 5.17 shows the Dolan and Moré performance profiles of these methods. We can see that CONMIN is top performer, being more robust versus all these methods.

Comparisons of CONMIN versus the modified conjugate gradient methods DL ( $t = 1$ ), DL+ ( $t = 1$ ), CG-DESCENT (version 1.4), and DESCONa are presented in Fig. 5.18. Observe that CONMIN is more robust than DL and DL+. However, both CG-DESCENT and DESCONa are more efficient and slightly more robust than CONMIN. CG-DESCENT and CONMIN are based on the memoryless BFGS quasi-Newton method. The search direction in CG-DESCENT is obtained from the Perry/Shanno search direction by deleting a term. On the other hand, DESCON is a modification of the HS method with guaranteed sufficient descent and conjugacy conditions and a modified Wolfe line-search.

In the second set of numerical experiments, let us present the performances of CONMIN for solving the applications from the MINPACK-2 collection, each of them with 40,000 variables. Table 5.11 shows the performances of CONMIN for solving these applications.



**Fig. 5.18** Performance profiles of CONMIN versus DL ( $t = 1$ ), DL+ ( $t = 1$ ), CG-DESCENT and DESCONA

**Table 5.11** Performances of CONMIN for solving five applications from the MINPACK-2 collection

	<i>n</i>	CONMIN		
		#iter	#fg	cpu
A1	40,000	241	484	5.67
A2	40,000	827	1674	21.25
A3	40,000	1094	2217	41.65
A4	40,000	486	985	40.21
A5	40,000	374	757	11.39
Total	—	3022	6117	120.17

### The Conjugate Gradient Method Closest to the Scaled Memoryless BFGS Search Direction (DK / CGOPT)

The search directions in the quasi-Newton methods are computed as  $d_k = -\bar{H}_k g_k$ , where  $\bar{H}_k \in \mathbb{R}^{n \times n}$  is an approximation to the inverse Hessian of the minimizing function (see Chap. 6). At the iteration  $k$ , the approximation  $\bar{H}_k$  to the inverse Hessian is updated to achieve  $\bar{H}_{k+1}$  as a new approximation to the inverse Hessian in such a way, so that  $\bar{H}_{k+1}$  satisfies a particular equation, namely, the secant equation, which includes the second order information. The most used is the standard secant equation  $\bar{H}_{k+1}y_k = s_k$ , where  $s_k = x_{k+1} - x_k$  and  $y_k = g_{k+1} - g_k$ .

Given the initial approximation  $\bar{H}_0$  to the inverse Hessian as an arbitrary symmetric and positive definite matrix, the most known quasi-Newton updating formula is the BFGS update

$$\bar{H}_{k+1} = \bar{H}_k - \frac{s_k y_k^T \bar{H}_k + \bar{H}_k y_k s_k^T}{y_k^T s_k} + \left( 1 + \frac{y_k^T \bar{H}_k y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}. \quad (5.282)$$

The self-scaling memoryless BFGS method of Perry (1977) and Shanno (1978a) is obtained by updating the scaled identity matrix  $(1/\tau_k)I$  by the BFGS updating formula (5.282), i.e., by considering  $\bar{H}_k = (1/\tau_k)I$  in (5.282), where  $I$  is the  $n \times n$  identity matrix and  $\tau_k$  is the scaling parameter. Therefore, the search direction in the self-scaling memoryless BFGS method is computed as

$$d_{k+1} = -H_{k+1} g_{k+1}, \quad (5.283)$$

where

$$H_{k+1} = \frac{1}{\tau_k} \left( I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} \right) + \left( 1 + \frac{1}{\tau_k} \frac{\|y_k\|^2}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}, \quad (5.284)$$

and  $\tau_k$  is the scaling parameter, known as the *SSML-BFGS updating*. Now, substituting (5.284) in (5.283), the SSML-BFGS search direction, i.e., *the self-scaling memoryless BFGS search direction of Perry and Shanno* (with a multiplier difference) is obtained as

$$d_{k+1}^{PS} = -g_{k+1} + \left[ \frac{g_{k+1}^T y_k}{y_k^T s_k} - \left( \tau_k + \frac{\|y_k\|^2}{y_k^T s_k} \right) \frac{g_{k+1}^T s_k}{y_k^T s_k} \right] s_k + \frac{g_{k+1}^T s_k}{y_k^T s_k} y_k. \quad (5.285)$$

(Observe that the Perry/Shanno search direction (5.277) is a scaling of (5.285), with a particular value for the scaling parameter  $\tau_k$ .) Thus, subject to the parameter  $\tau_k$ , a family of the Perry-Shanno self-scaling memoryless BFGS quasi-Newton methods is obtained. Now, the following particularizations of the search direction  $d_{k+1}^{PS}$  may be considered.

- Having in view that  $s_k = \alpha_k d_k$ , by deletion of the last term in (5.285) the following search direction is obtained:

$$d_{k+1} = -g_{k+1} + \left[ \frac{y_k^T g_{k+1}}{y_k^T d_k} - \left( \tau_k + \frac{\|y_k\|^2}{y_k^T s_k} \right) \frac{s_k^T g_{k+1}}{y_k^T d_k} \right] d_k. \quad (5.286)$$

As suggested by Oren and Spedicato (1976), if  $\tau_k$  in (5.286) is chosen as

$$\tau_k^{OS} = \frac{\|y_k\|^2}{y_k^T s_k}, \quad (5.287)$$

then (5.286) reduces to the well-known conjugate gradient algorithm CG-DESCENT, proposed by Hager and Zhang (2005)

$$d_{k+1} = -g_{k+1} + \beta_k^{CG-DESCENT} d_k, \quad (5.288)$$

where

$$\beta_k^{CG-DESCENT} = \frac{g_{k+1}^T y_k}{y_k^T d_k} - 2 \frac{\|y_k\|^2}{y_k^T s_k} \frac{g_{k+1}^T s_k}{y_k^T d_k}. \quad (5.289)$$

To establish the global convergence for general nonlinear functions, the conjugate gradient parameter is truncated as

$$\beta_k^{CG-DESCENT+} = \max \left\{ \beta_k^{CG-DESCENT}, \frac{-1}{\|d_k\| \min \{\eta, \|g_k\|\}} \right\}, \quad (5.290)$$

where  $\eta > 0$  is a constant ( $\eta = 0.01$ ). The numerical experiments showed that CG-DESCENT is more efficient and more robust than the self-scaling memoryless BFGS method given by (5.285) (Dai and Kou, 2013).

Other values for the scaling parameter  $\tau_k$  in (5.286) were proposed as follows. Oren (1974) and Oren and Luenberger (1984) proposed for  $\tau_k$  the value  $y_k^T s_k / (s_k^T B_k s_k)$  with  $B_k = H_k^{-1}$ . If  $H_k$  is the identity matrix, then this value reduces to

$$\tau_k^{OL} = \frac{y_k^T s_k}{\|s_k\|^2}. \quad (5.291)$$

Al-Baali (1998) suggested the following two choices:

$$\bar{\tau}^H = \min \left\{ 1, \frac{\|y_k\|^2}{y_k^T s_k} \right\} \text{ and } \bar{\tau}^B = \min \left\{ 1, \frac{y_k^T s_k}{\|s_k\|^2} \right\}.$$

For a general nonlinear convex objective function, Nocedal and Yuan (1993) proved the global convergence of the self-scaling BFGS method with  $\tau_k$  given by (5.291) and with Wolfe line-search. They also presented numerical results indicating that the unscaled BFGS method is in general superior to the self-scaling BFGS with  $\tau_k$  given by (5.291).

2. Observe that the self-scaling memoryless BFGS search direction of Perry and Shanno (5.285) is a three-term conjugate gradient algorithm. A more reasonable way to deal with the last term in (5.285) was suggested by Dai and Kou (2013), who proposed to seek the search direction closest to  $d_{k+1}^{PS}$  as a vector on the manifold  $S_{k+1} = \{-g_{k+1} + \beta d_k : \beta \in \mathbb{R}\}$ . This approach is not new. For example, Andrei (2017b) presented accelerated adaptive Perry conjugate gradient algorithms based on the minimization of the Frobenius norm of the difference between the symmetrical scaled Perry conjugate gradient direction and the self-scaling memoryless BFGS update. Also, Livieris, Tampakas, and Pintelas (2018) developed a convex hybridization of the conjugate gradient algorithms DY and HS, in which the hybridization parameter is computed by minimizing the distance between the hybrid direction and the self-scaling memoryless BFGS direction.

The search direction in  $S_{k+1}$  closest to  $d_{k+1}^{PS}$  is obtained as solution of the following least-squares problem

$$d_{k+1} = \arg \min \left\{ \|d - d_{k+1}^{PS}\|_2 : d \in S_{k+1} \right\}, \quad (5.292)$$

which is

$$d_{k+1}^{DK} = -g_{k+1} + \beta_k^{DK}(\tau_k)s_k, \quad (5.293)$$

where

$$\beta_k^{DK}(\tau_k) = \frac{g_{k+1}^T y_k}{y_k^T s_k} - \left( \tau_k + \frac{\|y_k\|^2}{y_k^T s_k} - \frac{s_k^T y_k}{\|s_k\|^2} \right) \frac{g_{k+1}^T s_k}{y_k^T s_k}. \quad (5.294)$$

If the line-search is exact,  $s_k^T g_{k+1} = 0$ , then the second term in (5.294) is missing and the search direction reduces to the HS formula.

In order to avoid the non-convergence of the algorithm, similar to Gilbert and Nocedal (1992), who proved the global convergence of the PRP methods for general nonlinear functions by restricting  $\beta_k \geq 0$ , (5.294) is truncated, being replaced by

$$\beta_k^{DK+}(\tau_k) = \max \left\{ \beta_k^{DK}(\tau_k), \eta \frac{g_{k+1}^T d_k}{\|d_k\|^2} \right\}, \quad (5.295)$$

where  $\eta \in [0, 1]$  is a parameter. ( $\eta = 0.5$ ).

Hence, the Dai-Kou conjugate gradient algorithm belongs to the same family of conjugate gradient methods obtained from the memoryless BFGS method by Perry and Shanno. Under the Assumption CG, the family of conjugate gradient methods (5.293) and (5.294) generate sufficient descent directions. Numerical experiments with this family of conjugate gradient algorithms showed that the most efficient is the one where the parameter  $\tau_k$  is given by (5.291). Therefore, substituting this choice of  $\tau_k$  in (5.294) and (5.295), it results that

$$\beta_k^{DK} = \frac{y_k^T g_{k+1}}{d_k^T y_k} - \frac{\|y_k\|^2}{d_k^T y_k} \frac{d_k^T g_{k+1}}{d_k^T y_k} \quad (5.296)$$

and

$$\beta_k^{DK+} = \max \left\{ \beta_k^{DK}, \eta \frac{d_k^T g_{k+1}}{\|d_k\|^2} \right\}, \quad (5.297)$$

where  $\eta \in [0, 1]$ . Observe that (5.296) is exactly the Dai-Liao conjugate gradient algorithm with  $t = \|y_k\|^2 / (s_k^T y_k)$ . Again, note that (5.296) differs from the Hager and Zhang algorithm only in a constant coefficient in the second term. Thus, the family of Dai-Kou self-scaling memoryless BFGS quasi-Newton methods is obtained.

If  $\tau_k$  in (5.294) is selected as  $\tau_k^{OL}$ , then the CGOPT conjugate gradient algorithm of Dai and Kou (2013) is obtained, where the search direction is computed as

$$d_{k+1}^{CGOPT} = -g_{k+1} + \beta_k^{CGOPT} d_k, \quad (5.298)$$

where

$$\beta_k^{CGOPT} = \frac{g_{k+1}^T y_k}{y_k^T d_k} - \frac{\|y_k\|^2}{y_k^T s_k} \frac{g_{k+1}^T s_k}{y_k^T d_k}, \quad (5.299)$$

which is identical to  $\beta_k^{DK}$  given by (5.296). Observe that the difference between the conjugate gradient parameters of CG-DESCENT given by (5.289) and of the CGOPT given by (5.299) is the absence of the constant factor 2 in the second term of the parameter from (5.299). Again, the numerical experiments showed that CGOPT performs more efficiently than the self-scaling memoryless BFGS method given by (5.285) (Dai and Kou, 2013). If the line-search is exact, i.e.,  $g_{k+1}^T s_k = 0$ ,

then the second term in (5.294) or in (5.289) or in (5.299) is missing and the search direction reduces to that of the HS algorithm.

Dai and Kou (2013) (see Lemma 2.1) proved that if  $y_k^T s_k > 0$ , then the search direction given by (5.293) and (5.294) satisfies

$$g_{k+1}^T d_{k+1} \leq -\min \left\{ \tau_k \frac{\|s_k\|^2}{y_k^T s_k}, \frac{3}{4} \right\} \|g_{k+1}\|^2. \quad (5.300)$$

More generally, if function  $f$  is continuously differentiable and bounded below and its gradient  $g$  is Lipschitz continuous, then Dai and Kou (2013) (see Lemma 2.2) proved that the search direction (5.293), where  $\tau_k$  in (5.294) is chosen to be any of  $\tau_k^{OS}$ ,  $\tau_k^{OL}$ ,  $\bar{\tau}_k^H$  or  $\bar{\tau}_k^B$  and  $y_k^T s_k > 0$ , satisfies  $g_{k+1}^T d_{k+1} \leq -c \|g_{k+1}\|^2$  for some positive constant  $c > 0$ .

Dai and Kou (2013) implemented the algorithm (5.2) with (5.298) and (5.299) endowed with two ingredients which improve its performances. The first ingredient is an improved Wolfe line-search, which avoids the numerical drawback of the first Wolfe line-search condition and guarantees the global convergence of the algorithm. The second one is an adaptive restart of the algorithm along the negative gradient based on how the minimizing function is close to some quadratic function.

The *improved Wolfe line-search* consists in the modified first Wolfe condition as

$$\psi_k(\alpha) \leq \psi_k(0) + \min \{ \varepsilon |\psi'_k(0)|, \rho \alpha \psi'_k(0) + \eta_k \}, \quad (5.301)$$

where  $\psi_k(\alpha) = f(x_k + \alpha d_k)$ ,  $\varepsilon > 0$  is a given constant and  $\{\eta_k\}$  is a positive sequence satisfying  $\sum_{k \geq 1} \eta_k < \infty$ , as well as in the second Wolfe line-search

$$\psi'_k(\alpha) \geq \sigma \psi'_k(0), \quad (5.302)$$

where  $0 < \rho < \sigma < 1$ .

The *dynamic restart strategy* is based on measuring how the minimizing function is close to some quadratic function. The deviation from quadratic can be measured by using an approximation of the size of the third derivative, as given by Nash and Nocedal (1991). Instead, similar to the ratio used for adjusting the trust-region radius, Dai and Kou (2013) evaluated the following quantity:

$$r_k = \frac{2(f_{k+1} - f_k)}{\alpha_k (g_k^T d_k + g_{k+1}^T d_k)}, \quad (5.303)$$

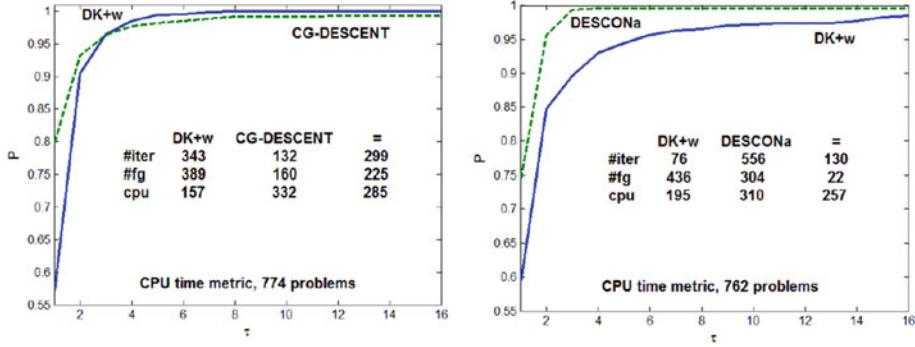
$k = 0, \dots$ . If  $r_k$  is close to 1, then it can be supposed that  $\psi_k(\alpha) = f(x_k + \alpha d_k)$  is close to some quadratic function. If there are continuously a number of iterations so that  $r_k$  is close to 1, then the algorithm is restarted with the steepest descent direction.

## Numerical Study: DK/CGOPT

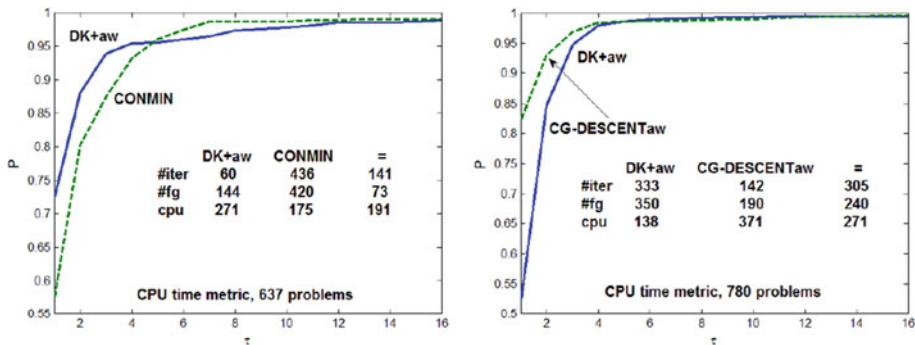
In the following, let us present the performances of the conjugate gradient algorithm denoted as DK+w, given by (5.47), where the search direction is computed as

$$d_{k+1} = -g_{k+1} + \beta_k^{DK+} s_k \quad (5.304)$$

and  $\beta_k^{DK+}$  is given by (5.297) with Wolfe line-search (5.49) and (5.50), for solving all the problems from the UOP collection. Even if  $\beta_k^{DK+} \equiv \beta_k^{CGOPT+}$ , where  $\beta_k^{CGOPT+} =$



**Fig. 5.19** Performance profiles of DK+w versus CG-DESCENT and DESCONa



**Fig. 5.20** Performance profiles of DK+aw versus CONMIN and CG-DESCENTaw

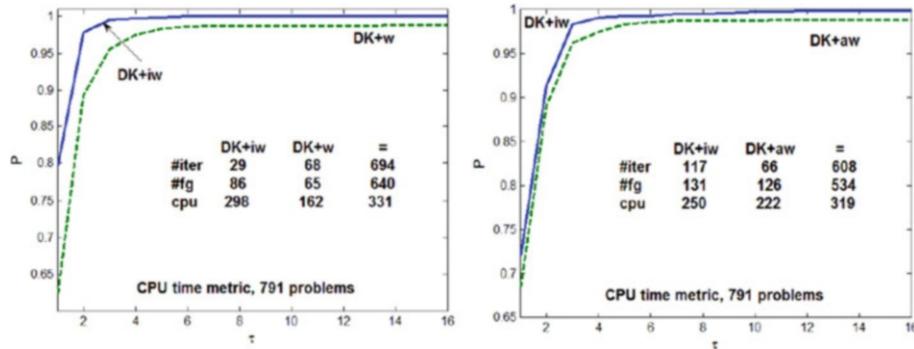
$\max \left\{ \beta_k^{CGOPT}, \eta(s_k^T g_{k+1}) / \|s_k\|^2 \right\}$ , the algorithm given by (5.304) and (5.297) with Wolfe line-search is called DK+w because we did not implement in it the dynamic restart strategy used in CGOPT or any other ingredients specific to CGOPT. For each test problem from the UOP collection (Andrei, 2020a), 10 numerical experiments were run, with the number of variables  $n = 1000, \dots, 10000$ . The initial value for the stepsize is computed as  $\alpha_k^0 = \alpha_{k-1} \|d_{k-1}\| / \|d_k\|$ . All the numerical experiments were run in the context of Remark 1.1.

Figure 5.19 presents the performance profiles of DK+w with Wolfe line-search versus CG-DESCENT (version 1.4) and versus DESCONa. CG-DESCENT is more efficient than DK+w, but DK+w is slightly more robust. DESCONa is clearly more efficient and more robust than DK+w.

Figure 5.20 shows the performance profiles of DK+aw (DK+ with approximate Wolfe line-search) versus CONMIN and versus CG-DESCENTaw (CG-DESCENT with approximate Wolfe line-search).

To see the performances of the DK+ algorithm with the improved Wolfe line-search, in (Andrei, 2019c), we manufactured a new code implementing (5.304) with (5.297), where this time the stepsize is determined by the improved Wolfe line-search (5.301) and (5.302) (without the dynamic restart strategy).

The algorithm using (5.304), (5.297), with improved Wolfe line-search (5.301) and (5.302) is called DK+iw (DK+ with improved Wolfe line-search). The improved Wolfe line-search is implemented with  $\epsilon = 10^{-6}$  and  $\eta_k = 1/(k^2)$ , where  $k$  is the iteration number.



**Fig. 5.21** Performance profiles of DK+iw versus DK+w and DK+aw

**Table 5.12** Performances of DK+w and DK+aw for solving five applications from the MINPACK-2 collection

	<i>n</i>	DK+w			DK+aw		
		#iter	#fg	cpu	#iter	#fg	cpu
A1	40,000	406	637	7.92	323	647	7.17
A2	40,000	940	1508	20.54	791	1583	32.64
A3	40,000	4001	6260	182.63	987	1976	65.87
A4	40,000	670	1065	84.22	435	871	80.42
A5	40,000	417	654	8.80	289	580	9.54
Total	—	6434	10124	304.11	2825	5657	195.64

Figure 5.21 presents the performance profiles of DK+iw (DK+ with improved Wolfe line-search) versus DK+w (DK+ with Wolfe line-search) and DK+aw (DK+ with approximate Wolfe line-search). Observe that DK+iw is more efficient and more robust than DK+w.

In the following, let us present the performances of DK+w and DK+aw for solving the applications from the MINPACK-2 collection as in Table 5.12.

Table 5.12 points out that DK+ with the approximate Wolfe line-search (DK+aw) is top performer versus DK+ with the Wolfe line-search (DK+w).

## 5.8 Solving Large-Scale Applications

Nonlinear conjugate gradient algorithms are used in solving large-scale unconstrained optimization problems. They have the significant advantage of a low storage requirement over most of the other unconstrained optimization algorithms discussed in this book, only the steepest descent being an exception. A practical conjugate gradient algorithm uses restart procedures. The most important theoretical result about restarting is the *n*-step quadratic convergence, that is,  $\|x_{k+n} - x^*\| = O(\|x_k - x^*\|^2)$ . The critical point with these algorithms is the procedure for stepsize computation. The conjugate gradient algorithms implement the weak Wolfe line-search (5.49) and (5.50) or the strong Wolfe line-search (5.49) and (5.51). In the advanced conjugate gradient algorithms, the stepsize is computed by means of the approximate Wolfe line-search (2.67) of Hager and Zhang, the modified Wolfe line-search (5.49) and (5.236) of Andrei, or the improved Wolfe line-search (2.72) and (5.50) of Dai and Kou. Tables 5.13, 5.14, and 5.15 show the performances of CG-DESCENT with Wolfe and with approximate Wolfe line-searches (CG-DESCENTaw), the performances of DESCON with modified Wolfe line-search and the

**Table 5.13** Performances of CG-DESCENT and of CG-DESCENTaw for solving five large-scale applications from the MINPACK-2 collection

	<i>n</i>	CG-DESCENT			CG-DESCENTaw		
		#iter	#f	cpu	#iter	#f	cpu
A1	250,000	610	1221	92.42	610	1221	136.47
A2	250,000	1752	3505	382.78	1752	3505	448.94
A3	250,000	2370	4742	878.06	2370	4742	943.39
A4	250,000	925	1851	902.03	925	1851	961.31
A5	250,000	635	1271	145.72	635	1271	196.56
Total	—	6292	12590	2401.01	6292	12590	2686.67

**Table 5.14** Performances of DESCON and of DESCONa for solving five large-scale applications from the MINPACK-2 collection

	<i>n</i>	DESCON			DESCONa		
		#iter	#fg	cpu	#iter	#fg	cpu
A1	250,000	602	950	53.01	591	1209	69.58
A2	250,000	2578	4056	309.55	1495	3021	261.97
A3	250,000	5001	7626	1134.73	2342	4727	607.68
A4	250,000	1644	2577	868.62	727	1489	468.01
A5	250,000	1070	1674	216.98	655	1334	130.31
Total	—	10895	16883	2582.89	5810	11780	1537.55

**Table 5.15** Performances of DK+w and of DK+iw for solving five large-scale applications from the MINPACK-2 collection

	<i>n</i>	DK+w			DK+iw		
		#iter	#fg	cpu	#iter	#fg	cpu
A1	250,000	693	1093	107.58	613	1227	145.21
A2	250,000	2299	3650	413.70	1762	3525	457.22
A3	250,000	4001	6257	1048.29	2354	4710	1007.74
A4	250,000	1396	2211	846.27	923	1847	937.49
A5	250,000	931	1455	227.40	622	1245	202.24
Total	—	9320	14666	2643.24	6274	12554	2749.90

accelerated DESCON with modified Wolfe line-search (DESCONa), and the performances of DK/CGOPT with Wolfe line-search (DK+w) and DK/CGOPT with improved Wolfe line-search (DK+iw), respectively, for solving 5 applications from the MINPACK2 collection described in Appendix D, each of them with 250,000 variables.

Firstly, observe that the conjugate gradient algorithms can solve large-scale unconstrained optimization applications. Besides, from the above tables, we can see that the accelerated DESCON with modified Wolfe line-search is the fastest. From the practical point of view, both the acceleration and the modified Wolfe line-search play a crucial role in establishing the performances of DESCON. Even if the acceleration at each iteration involves one additional evaluation of the gradient of the minimizing function, the efficiency of the algorithm endowed with the acceleration scheme is significantly improved.

## Notes and References

The idea of the methods described in this chapter is to include the approximations to the Hessian of the minimizing function into the formula for computing the conjugate gradient parameter  $\beta_k$ . This was first considered by Perry (1976, 1977). Actually, the foundation of the self-scaling memoryless BFGS algorithm was first presented by Perry as a technique for developing a nonlinear conjugate gradient algorithm with memory, i.e., with stored information from the previous iterations, as an alternative to the quasi-Newton methods for large-scale problems, where to store and handle the Hessian matrix is impractical. This method was the first effort for solving large-scale problems, preceding the introduction by Nocedal (1980) of the limited memory BFGS method. Shanno (1978a) reinterpreted Perry's algorithm and showed that the conjugate gradient methods are exactly the BFGS quasi-Newton method, where the approximation to the inverse Hessian is restarted as the identity matrix at every step. He introduced a scaling term, thus improving the final form of the self-scaling memoryless BFGS method, i.e., the SSML-BFGS method. A modification of the self-scaling memoryless BFGS method was given by Kou and Dai (2015). They multiplied the third term in (5.285) by some nonnegative parameter, thus obtaining a new self-scaling BFGS algorithm with better convergence properties.

The SSML-BFGS method provided a very good understanding of the relationship between nonlinear conjugate gradient methods and quasi-Newton methods. For convex quadratic functions, if the line-search is exact and the identity matrix is used as the initial approximation to the Hessian, then both BFGS and SSML-BFGS methods generate the same iterations as the conjugate gradient method. This was the starting point for the conjugate gradient methods memoryless BFGS preconditioned.

Using this approach, Shanno and Phua (1976, 1978, 1980) and Shanno (1983) developed the CONMIN algorithm, one of the best conjugate gradient algorithms and codes. Comparisons among different conjugate gradient methods (ZXCGR (Powell, 1977), E04DBF (NAG – The Numerical Algorithms Group), CONMIN) were given by Navon and Legler (1987).

Hager and Zhang (2005) presented the CG-DESCENT, one of the most reliable conjugate gradient algorithms. In CG-DESCENT, Hager and Zhang introduced an approximate Wolfe line-search. Later on, Dai and Kou (2013) proposed the CGOPT algorithm, where the search direction is closest to the direction of the scaled memoryless BFGS method. Similar to Hager and Zhang, Dai and Kou developed an improved Wolfe line-search. In this way, a family of conjugate gradient algorithms was obtained, where the stepsize was computed by an improved Wolfe line-search. Andrei (2013c) introduced DESCON, a new conjugate gradient algorithm with guaranteed sufficient descent and conjugacy conditions, where the stepsize is computed by a modified weak Wolfe line-search. Intensive numerical experiments proved that, for solving large-scale unconstrained optimization problems and applications, DESCON is top performer versus CONMIN, CG-DESCENT, and CGOPT. Further on, Andrei (2018d, 2020a), by using the determinant, the trace or a combination of these operators known as the measure function of Byrd and Nocedal, developed new efficient self-scaling memoryless BFGS conjugate gradient methods.

An adaptive conjugate gradient algorithm for the large-scale unconstrained optimization was presented by Andrei (2016). In this algorithm, the search direction is computed as the sum of the negative gradient and a vector determined by minimizing the quadratic approximation of the objective function at the current point. Using a special approximation to the inverse Hessian of the objective function, which depends on a positive parameter, a new search direction satisfying both the sufficient descent condition and the Dai–Liao conjugacy is obtained.

Two different approaches based on the eigenvalues and the singular values of the matrix representing the search direction in conjugate gradient algorithms were considered by Andrei

(2017a). Using a special approximation of the inverse Hessian of the objective function, which depends on a positive parameter, a search direction is obtained, satisfying both the sufficient descent condition and the Dai-Liao conjugacy condition. In the first approach, the parameter in the search direction is determined by clustering the eigenvalues of the matrix defining it. The second approach uses the minimization of the condition number of the matrix representing the search direction. The conclusion of these two approaches is that basically they represent two different ways to pursue similar ideas based on eigenvalues or on singular values of the iteration matrix, respectively. See also Andrei (2018b), where a Dai-Liao conjugate gradient algorithm with the clustering of the eigenvalues is presented. A modification of the Dai-Yuan conjugate computational scheme with sufficient descent condition is presented in (Andrei, 2009b, 2010c). An accelerated conjugate gradient algorithm with modified secant condition for unconstrained optimization is developed by Andrei (2009g). In this algorithm, the modified secant equation of Zhang, Deng, and Chen (1999) is used.

Another class is represented by the three-term conjugate gradient algorithms introduced by Beale (1972) and developed by Powell (1984), Nazareth (1977, 1986), Dai and Yuan (1999), Narushima, Yabe, and Ford (2011), and Andrei (2013a, 2013b, 2013d, 2020a). A family of three-term conjugate gradient methods with sufficient descent property was presented by Al-Baali, Narushima, and Yabe (2015). Even if these three-term conjugate gradient algorithms are based on different concepts which include satisfying the descent and conjugacy conditions (Andrei, 2013b), the subspace minimization (Andrei, 2014), and the minimization of one-parameter quadratic model of the minimizing function (Andrei, 2015b), they have similar performances. A numerical comparison of conjugate gradient algorithms for the unconstrained optimization is given in (Andrei, 2007d).

The most important modern packages implementing different variants of the conjugate gradient algorithms for solving large-scale unconstrained optimization problems are CONMIN (Shanno, 1983), CG-DESCENT (Hager, & Zhang 2005, 2006a), CGOPT (Dai, & Kou, 2013), and DESCON (Andrei, 2013c).

In Andrei (2011b), some open questions regarding the conjugate gradient algorithms are discussed: Why is the initial search direction  $d_0 = -g_0$  critical? Can we take advantage of the problem structure to design more effective nonlinear conjugate gradient algorithms? Which is the best conjugacy condition? Which is the best hybrid conjugate gradient algorithm? What is the most convenient restart procedure of the conjugate gradient algorithms? etc.



# Quasi-Newton Methods

# 6

The idea of these methods is not to use the Hessian  $\nabla^2 f(x_k)$  of the minimizing function in the current point at every iteration, but instead to use an approximation of it. In this chapter, consider  $B_k$  as an approximation to  $\nabla^2 f(x_k)$  and  $H_k$  an approximation to the inverse Hessian  $\nabla^2 f(x_k)^{-1}$ . Obviously, both these approximations  $B_k$  or  $H_k$  may be used, but the use of  $B_k$  involves the solving of a linear algebraic system. The quasi-Newton method requires only the gradient of the minimizing function, gradient which must be supplied by the user at every iteration. *The purpose of this chapter is to present these methods together with the theoretical aspects concerning their convergence to solution, as well as their performances for solving large-scale complex optimization problems and applications.* In quasi-Newton methods, the approximations to the Hessian (or to the inverse Hessian) may be achieved through matrices of rank one or through matrices of rank two. From the multitude of quasi-Newton methods, this chapter will present only the rank-two updating methods, that is, Davidon-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS), and the rank one, that is, Symmetric Rank 1 (SR1), together with some modifications of them.

Since the quasi-Newton methods use only the gradient of the minimizing function, they are very attractive for solving large classes of complex, large-scale unconstrained optimization problems and applications. Moreover, since the second derivatives are not required, sometimes quasi-Newton methods are more efficient and more robust than the Newton method. It is worth mentioning that the development of the automatic differentiation techniques gives the user the possibility not to supply second derivatives in the Newton method. The automatic differentiation is not applicable in many situations, and it is more difficult to work with second derivatives in it than with the gradient. This is the reason why quasi-Newton methods represent an important class of methods for the unconstrained optimization.

## 6.1 DFP and BFGS Methods

Consider the minimizing problem

$$\min_{x \in \mathbb{R}^n} f(x), \quad (6.1)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable and bounded below. In the current point  $x_k$ , let us consider the following quadratic model of the minimizing function  $f$

$$m_k(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d, \quad (6.2)$$

where  $B_k \in \mathbb{R}^{n \times n}$  is a symmetric and positive definite matrix which, as we said, will be updated at every iteration. Observe that the minimizer  $d_k$  of (6.2) can be explicitly written as

$$d_k = -B_k^{-1} g_k, \quad (6.3)$$

where  $g_k = \nabla f(x_k)$ . The next approximation to the minimum point is computed as

$$x_{k+1} = x_k + \alpha_k d_k, \quad (6.4)$$

where  $\alpha_k$  is the stepsize computed by line-search, for example, the Wolfe line-search:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k \nabla f(x_k)^T d_k \quad (6.5)$$

$$\nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma \nabla f(x_k)^T d_k, \quad (6.6)$$

If in (6.3)  $B_k = \nabla^2 f(x_k)$ , then the Newton method is obtained.

In the quasi-Newton methods, the matrix  $B_k$  is updated in a simple manner in order to take into account the curvature of the minimizing function during the most recent iterates. Suppose that we generated a new iteration  $x_{k+1}$  as in (6.4) and then construct a new quadratic model:

$$m_{k+1}(d) = f(x_{k+1}) + \nabla f(x_{k+1})^T d + \frac{1}{2} d^T B_{k+1} d, \quad (6.7)$$

where  $B_{k+1}$  is an update of  $B_k$ . At this moment, we are facing the following problems. How to generate the approximations  $B_{k+1}$  symmetric and positive definite, for which  $d_{k+1} = -B_{k+1}^{-1} g_{k+1}$  is a descent direction? Computationally, how to generate  $B_{k+1}$  as easily as possible? Finally, what are the conditions the approximation  $B_{k+1}$  has to satisfy for the convergence of the corresponding algorithms?

Since we have access to the gradient, it is quite reasonable to require that the gradient of  $m_{k+1}$  should match the gradient of the minimizing function at the latest two iterates  $x_k$  and  $x_{k+1}$ . However, as we can see,  $\nabla m_{k+1}(0) = \nabla f(x_{k+1})$ . Therefore, the second requirement is automatically satisfied. The first requirement can be written as

$$\nabla f(x_k) = \nabla m_{k+1}(-\alpha_k d_k) = \nabla f(x_{k+1}) - \alpha_k B_{k+1} d_k.$$

Therefore,

$$B_{k+1} \alpha_k d_k = \nabla f(x_{k+1}) - \nabla f(x_k). \quad (6.8)$$

Let us define

$$s_k = x_{k+1} - x_k = \alpha_k d_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k). \quad (6.9)$$

With this, (6.8) becomes

$$B_{k+1} s_k = y_k, \quad (6.10)$$

known as the *secant equation*.

Observe that the secant equation requires that the symmetric and positive definite matrix  $B_{k+1}$  map the displacement  $s_k$  into the change of gradients  $y_k$ . Since  $s_k^T B_{k+1} s_k > 0$  ( $B_{k+1}$  is positive definite), it follows that the secant equation is satisfied only if  $s_k$  and  $y_k$  satisfy the *curvature condition*

$$s_k^T y_k > 0. \quad (6.11)$$

If  $f$  is strongly convex, the curvature condition (6.11) will be satisfied for any two points  $x_k$  and  $x_{k+1}$ . However, this condition will not always hold for nonconvex functions. In these cases, the curvature condition (6.11) has to be enforced by imposing restrictions on the line-search procedure that computes the stepsize  $\alpha_k$ . In fact, the curvature condition (6.11) is guaranteed to hold if the stepsize is computed by the weak Wolfe conditions (6.5) and (6.6) or by the strong Wolfe conditions (6.5) and (6.12)

$$|\nabla f(x_k + \alpha_k d_k)^T d_k| \leq -\sigma \nabla f(x_k)^T d_k. \quad (6.12)$$

Indeed, from (6.9) and (6.12), we get

$$y_k^T s_k \geq (\sigma - 1) \alpha_k \nabla f(x_k)^T d_k \geq 0,$$

since  $\sigma < 1$  and  $d_k$  is a descent direction.

If the curvature condition (6.11) is satisfied, then the secant equation (6.10) always has a solution  $B_{k+1}$ . Actually, the secant equation admits an infinite number of solutions. This is because the symmetric matrix  $B_{k+1}$  has a number of  $n(n + 1)/2$  elements which is greater than  $n$  conditions imposed by the secant equation plus  $n$  additional conditions (inequalities) imposed by the positive definiteness of  $B_{k+1}$  (all the principal minors must be positive). In order to determine  $B_{k+1}$  uniquely, a reasonable additional condition is imposed: *among all the symmetric matrices satisfying the secant equation,  $B_{k+1}$  is in a certain sense closest to the current matrix  $B_k$ .* In other words,  $B_{k+1}$  is the solution of the following problem:

$$\begin{aligned} & \min_B \|B - B_k\| \\ & \text{subject to} \\ & B = B^T, B s_k = y_k, \end{aligned} \quad (6.13)$$

where  $s_k$  and  $y_k$  satisfy the curvature condition (6.11) and  $B_k$  is symmetric and positive definite. Obviously, different norms in (6.13) may be used, each of them determining different quasi-Newton methods. A norm that allows an easy solution of the minimization problem (6.13) is the weighted Frobenius norm defined as  $\|A\|_W \equiv \|W^{1/2} A W^{1/2}\|_F$ , where  $\|C\|_F = \sum_{i=1}^n \sum_{j=1}^n c_{ij}^2$  (see Appendix A). Now, selecting  $W = \bar{G}_k^{-1}$ , where  $\bar{G}_k$  is the average Hessian

$$\bar{G}_k = \int_0^1 \nabla^2 f(x_k + t \alpha_k d_k) dt$$

observe that  $y_k = \bar{G}_k s_k$ . This choice of the weighting matrix  $W$  has an important property that is non-dimensional, i.e., the solution of (6.13) is not dependent on the units of the problem.

With the selection of the weighting matrix and of the norm, the unique solution of (6.13) is

$$(DFP \text{ direct}) \quad B_{k+1} = (I - \rho_k y_k s_k^T) B_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T, \quad (6.14)$$

where

$$\rho_k = \frac{1}{y_k^T s_k}. \quad (6.15)$$

The update (6.14) is called the *DFP update*, since it was proposed by Davidon (1959, 1980) and studied and popularized by Fletcher and Powell (1963). After simple algebraic manipulations, from (6.14) and (6.15), another expression of  $B_{k+1}$  is obtained as

$$(DFP\ direct) B_{k+1} = B_k - \frac{B_k s_k y_k^T + y_k s_k^T B_k}{y_k^T s_k} + \left(1 + \frac{s_k^T B_k s_k}{y_k^T s_k}\right) \frac{y_k y_k^T}{y_k^T s_k}. \quad (6.16)$$

As we said, the inverse of  $B_k$  is  $H_k$ , i.e.,  $H_k = B_k^{-1}$ . Using the matrix  $H_k$  in the implementations of the DFP method is quite appealing, since it allows to compute the search direction  $d_k = -H_k \nabla f(x_k)$  by a simple matrix-vector multiplication. Using the Sherman-Morrison-Woodbury formula twice, from (6.16) (see Appendix A), the following expression for the update of the inverse Hessian approximation  $H_k$  that corresponds to the DFP update is obtained

$$(DFP\ inverse) H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}. \quad (6.17)$$

Observe that the  $H_k$  matrix in (6.17) is modified by two terms, each of them of rank one. Also, (6.16) is a rank-two modification of  $B_k$ .

Even if the DFP updating is efficient and robust for solving unconstrained optimization problems, it was superseded by the BFGS updating. The BFGS introduced by Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970) follows the same methodology as above, but on the inverse Hessian  $H_k$ . Obviously, the updated Hessian  $H_{k+1}$  must be a symmetric and positive definite matrix which satisfies the *secant equation*

$$H_{k+1} y_k = s_k. \quad (6.18)$$

As above, the updated Hessian  $H_{k+1}$  closests to  $H_k$  is determined as the unique solution of the minimizing problem

$$\begin{aligned} & \min_H \|H - H_k\| \\ & \text{subject to} \\ & H = H^T, Hy_k = s_k. \end{aligned} \quad (6.19)$$

Using the same weighted Frobenius norm, where this time the weight matrix  $W$  satisfies  $Ws_k = y_k$ , the unique solution  $H_{k+1}$  of the problem (6.19) is

$$(BFGS\ inverse) H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad (6.20)$$

where  $\rho_k$  is defined in (6.15). After some simple algebraic manipulation, from (6.20) and (6.15), we get

$$(BFGS\ inverse) H_{k+1} = H_k - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k} + \left(1 + \frac{y_k^T H_k y_k}{y_k^T s_k}\right) \frac{s_k s_k^T}{y_k^T s_k}. \quad (6.21)$$

Now, applying twice the Sherman-Morrison-Woodbury formula (see Appendix A) to (6.21), we get a version of the BFGS algorithm that works with the Hessian approximation  $B_k$  rather than with the inverse  $H_k$ . Therefore, from (6.21), we get

$$(BFGS \text{ direct}) \quad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \quad (6.22)$$

Note that the DFP and the BFGS updating formulae are *duals* of each other in the sense that one can be obtained from the other one by making the interchanges  $s_k \leftrightarrow y_k$  and  $B_k \leftrightarrow H_k$ .

In the current implementations of the BFGS algorithm, the inverse update  $H_{k+1}$  is used for the search direction computation  $d_{k+1} = -H_{k+1} \nabla f(x_{k+1})$ . This is very appealing because it can be performed at a cost of  $O(n^2)$  arithmetic operations.

The implementation of the BFGS algorithm requires the initial matrix  $H_0$ . This is often taken as the identity matrix or the identity matrix multiplied by a parameter  $\tau$ . If  $\tau$  is too large, then the first step  $d_0 = -\tau g_0$  is too long, and many function evaluations may be required for determining a suitable stepsize  $\alpha_0$ . Thus, to obtain  $H_1$ , Nocedal and Wright recommend selecting  $H_0$  as

$$H_0 = \frac{y_0^T s_0}{y_0^T y_0} I$$

before applying (6.17) and (6.21). The BFGS algorithm can be presented as follows.

**Algorithm 6.1** *DFP and BFGS methods*

- |    |   |
|----|---|
| 1. | Initialization. Consider an initial point $x_0 \in \mathbb{R}^n$ , the initial approximation $H_0$ to the inverse Hessian as well as the convergence tolerance $\epsilon > 0$ sufficiently small. Set $k = 0$ |
| 2. | Test a criterion for stopping the iterations. For example, if $\ \nabla f(x_k)\  \leq \epsilon$ , then stop, otherwise set $k = k + 1$ and go to step 3   |
| 3. | Compute the descent direction $d_k = -H_k \nabla f(x_k)$  |
| 4. | Compute the stepsize $\alpha_k$ which satisfies the Wolfe line-search conditions  |
| 5. | Compute $x_{k+1} = x_k + \alpha_k d_k$  |
| 6. | Compute $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$   |
| 7. | For DFP compute $H_{k+1}$ by (6.17). For BFGS compute $H_{k+1}$ by (6.21)   |
| 8. | Set $k = k + 1$ and go to step 2  |

◆

In step 4 of the algorithm, the stepsize is determined by the weak Wolfe conditions (6.5) and (6.6) (or by the strong Wolfe conditions (6.5) and (6.12)), which should always try the initial stepsize  $\alpha_k = 1$ . Numerical experiments showed that, eventually, this stepsize will always be accepted, thus producing the superlinear convergence of the algorithm.

The following numerical example illustrates the importance of the initial approximation  $H_0$  on the performances of the algorithms DFP and BFGS.

**Example 6.1** Let us minimize the function

$$f(x) = (x_1 - 2)^4 + (x_1 - 2)^2 x_2^2 + (x_2 + 1)^2,$$

which has the minimum  $x^* = [2, -1]^T$  with  $f(x^*) = 0$ . In the minimum point, we have

$$\nabla f(x^*) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \nabla^2 f(x^*) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}.$$

Consider  $x_0 = [1.1, 0]^T$  and  $H_0 = I$ . Then, the DFP (6.17) and the BFGS (6.21) methods with weak Wolfe line-search (6.5) and (6.6) implemented in subroutine *wolfeLS* with cubic interpolation (see Fig. 2.3) give the following results.

DFP	BFGS
$x_{15} = [2 \ -0.999999]^T$	$x_{13} = [2 \ -1]^T$
$f(x_{15}) = 0.1022e - 24$	$f(x_{13}) = 0.7674e - 24$
$H_{15} = \begin{bmatrix} 0.500732 & 0.000125 \\ 0.000125 & 0.5002158 \end{bmatrix}$	$H_{13} = \begin{bmatrix} 0.500089 & 0.000135 \\ 0.000135 & 0.500195 \end{bmatrix}$

Now, from the initial point  $x_0 = [1.1, 0]^T$  and with  $H_0 = \nabla^2 f(x_0)$ , then the DFP and the BFGS methods give the following results.

DFP	BFGS
$x_{25} = [2 \ -0.999999]^T$	$x_{32} = [2 \ -1]^T$
$f(x_{25}) = 0.2317e - 23$	$f(x_{32}) = 0.141342e - 14$
$H_{25} = \begin{bmatrix} 0.5 & 0.408111e - 6 \\ 0.408111e - 6 & 0.5 \end{bmatrix}$	$H_{32} = \begin{bmatrix} 0.4995756 & -0.423871e - 3 \\ -0.423871e - 3 & 0.4995739 \end{bmatrix}$

From the initial point  $x_0 = [1.1, 0]^T$  and with  $H_0 = (s_0^T y_0 / y_0^T y_0) I$ , then the DFP and the BFGS methods give the following results.

DFP	BFGS
$x_{14} = [2 \ -1]^T$	$x_{14} = [1.999999 \ -0.999999]^T$
$f(x_{14}) = 0.6751e - 13$	$f(x_{14}) = 0.6006455e - 13$
$H_{14} = \begin{bmatrix} 0.500222 & 0.113729e - 3 \\ 0.113729e - 3 & 0.50005778 \end{bmatrix}$	$H_{14} = \begin{bmatrix} 0.5002339 & 0.115678e - 3 \\ 0.1156781e - 3 & 0.50005478 \end{bmatrix}$

◆

**Proposition 6.1** Let  $B_k$  be a symmetric and positive definite matrix, and assume that  $B_{k+1}$  is obtained from  $B_k$  using the BFGS update formula. Then,  $B_{k+1}$  is positive definite if and only if  $y_k^T s_k > 0$ .

**Proof** If  $B_k$  is positive definite, then it can be factored as  $B_k = L_k L_k^T$ , where  $L_k$  is nonsingular. (This is exactly the Cholesky factorization of  $B_k$ .) Now, if this factorization is introduced into the BFGS formula for the  $B_{k+1}$  computation, then

$$B_{k+1} = L_k W_k L_k^T,$$

where

$$W_k = I - \frac{\hat{s}_k \hat{s}_k^T}{\hat{s}_k^T \hat{s}_k} + \frac{\hat{y}_k \hat{y}_k^T}{\hat{y}_k^T \hat{s}_k}, \quad \hat{s}_k = L_k^T s_k, \quad \hat{y}_k = L_k^{-1} y_k.$$

Observe that  $B_{k+1}$  will be positive definite if and only if  $W_k$  is positive definite. To see that  $W_k$  is positive definite, let us test if  $v^T W_k v > 0$  for all  $v \neq 0$ . Let  $\theta_1$  be the angle between  $v$  and  $\hat{s}_k$ ,  $\theta_2$  the angle between  $v$  and  $\hat{y}_k$ , and  $\theta_3$  the angle between  $\hat{s}_k$  and  $\hat{y}_k$ . Then,

$$\begin{aligned} v^T W_k v &= v^T v - \frac{(v^T \hat{s}_k)^2}{\hat{s}_k^T \hat{s}_k} + \frac{(v^T \hat{y}_k)^2}{\hat{y}_k^T \hat{s}_k} \\ &= \|v\|^2 - \frac{\|v\|^2 \|\hat{s}_k\|^2 \cos^2 \theta_1}{\|\hat{s}_k\|^2} + \frac{\|v\|^2 \|\hat{y}_k\|^2 \cos^2 \theta_2}{\|\hat{y}_k\| \|\hat{s}_k\| \cos \theta_3} \\ &= \|v\|^2 \left[ 1 - \cos^2 \theta_1 + \frac{\|\hat{y}_k\| \cos^2 \theta_2}{\|\hat{s}_k\| \cos \theta_3} \right] \\ &= \|v\|^2 \left[ \sin^2 \theta_1 + \frac{\|\hat{y}_k\| \cos^2 \theta_2}{\|\hat{s}_k\| \cos \theta_3} \right]. \end{aligned}$$

If  $y_k^T s_k > 0$ , then  $\hat{y}_k^T \hat{s}_k > 0$  and  $\cos \theta_3 > 0$ . Hence,  $v^T W_k v > 0$ , that is,  $W_k$  is positive definite. If  $y_k^T s_k < 0$ , then  $\cos \theta_3 < 0$ . In this case,  $v$  can be chosen so that  $v^T W_k v < 0$ , that is,  $W_k$  is not positive definite, which completes the proof.  $\blacklozenge$

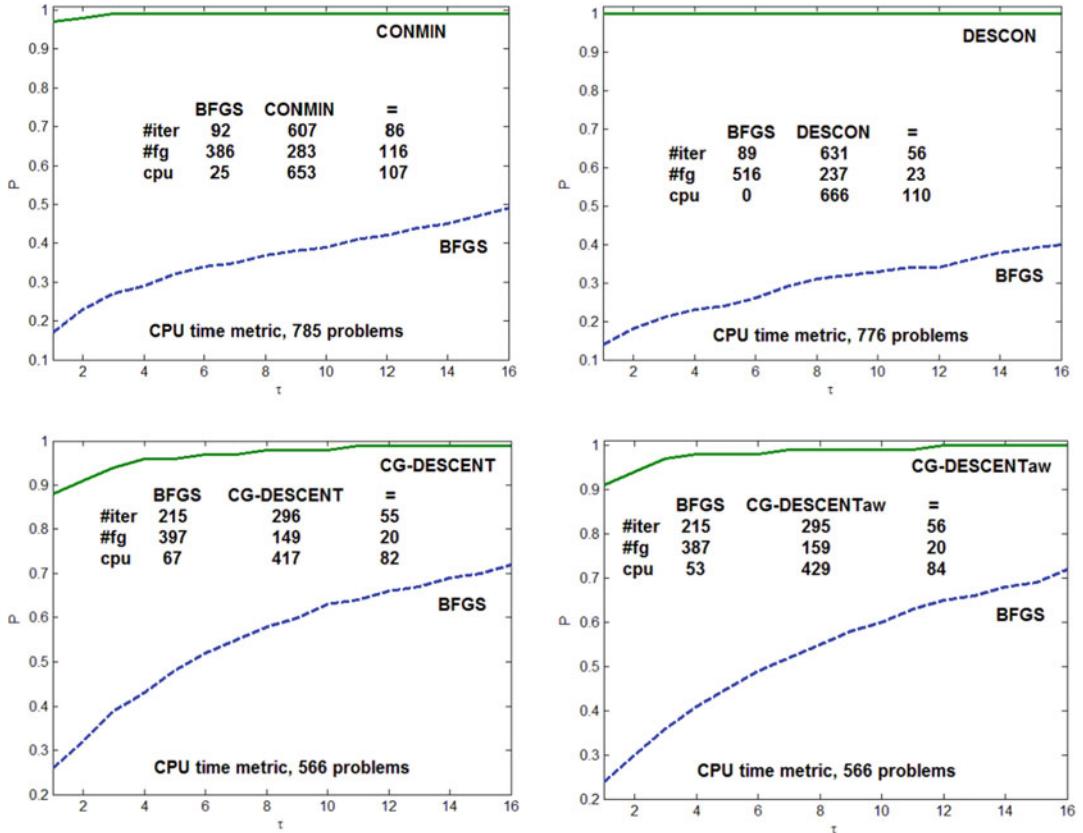
The new update  $B_{k+1}$  matrix will be positive definite only if  $y_k^T s_k > 0$ . This property can be guaranteed by performing an appropriate line-search, for example, the Wolfe line-search. Therefore, the quasi-Newton method based on the BFGS updating formula is often implemented with the (weak) Wolfe line-search (6.5) and (6.6).

### Numerical Study: BFGS

In the first set of the numerical experiments that follow, let us see the performances of the BFGS algorithm versus the conjugate gradient algorithm CONMIN for solving 800 unconstrained optimization problems from the UOP collection (Andrei, 2020a) with the number of variables in the range [100, 1000]. In the second set of numerical experiments, the performance profile of BFGS versus the conjugate gradient algorithm DESCN is presented. The third set of numerical experiments presents the performances of BFGS versus the conjugate gradient CG-DESCENT with the Wolfe line-search and versus CG-DESCENTaw with the approximate Wolfe line-search. Figure 6.1 shows the performance profiles of these algorithms. The conjugate gradient algorithms CONMIN, DESCN, CG-DESCENT, and CG-DESCENTaw are way more efficient and more robust than BFGS. In Fig. 6.1, we can see that CONMIN was the fastest in solving 653 problems from this collection, while BFGS was the fastest only in 25 problems. Out of 800 problems considered in this numerical study, only for 785 problems does the criterion (1.3) hold. Similarly, DESCN was the fastest in solving 666 problems, but BFGS was the fastest only in 0 problems, etc. All these algorithms implement the weak Wolfe line-search with cubic interpolation and safeguarded stepsizes (see subroutine wolfeLS in Fig. 2.3). Even if the BFGS algorithm has some very interesting convergence properties which will be discussed in the following lines, the conjugate gradient methods are way more efficient and more robust.

### The Broyden Class

There are plenty of quasi-Newton updating formulae, but we have described only DFP and BFGS. The Broyden class of the quasi-Newton methods is a family of updates specified by the following general formula:



**Fig. 6.1** Performance profiles of BFGS versus CONMIN, DESCON, CG-DESCENT, and CG-DESCENTaw

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} + \Phi_k (s_k^T B_k s_k) v_k v_k^T, \quad (6.23)$$

where  $\Phi_k$  is a scalar parameter and

$$v_k = \frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}. \quad (6.24)$$

The BFGS and DFP algorithms are members of this family. By setting  $\Phi_k = 0$ , the BFGS update is obtained, while by setting  $\Phi = 1$ , the DFP is obtained.

Observe that the last term in (6.23) is a rank-one correction, which, by the interlacing eigenvalue theorem, increases the eigenvalues of the matrix  $B_{k+1}$  when  $\Phi_k$  is positive. Therefore,  $B_{k+1}$  is positive definite for all  $\Phi_k \geq 0$ . On the other hand, when  $\Phi_k$  is negative, the last term in (6.23) decreases the eigenvalues of  $B_{k+1}$ . Obviously, when  $\Phi_k$  is decreasing,  $B_{k+1}$  becomes singular and then indefinite. By simple algebraic manipulations, we obtain that  $B_{k+1}$  is singular for  $\Phi_k = \Phi_k^c$ , where

$$\Phi_k^c = \frac{1}{1 - \mu_k}, \text{ and } \mu_k = \frac{(y_k^T B_k^{-1} y_k)(s_k^T B_k s_k)}{(y_k^T s_k)^2}. \quad (6.25)$$

By the Cauchy-Schwarz inequality, we see that  $\mu_k \geq 1$ , and therefore,  $\Phi_k^c \leq 0$ . If the initial Hessian approximation  $B_0$  is symmetric and positive definite and if  $y_k^T s_k > 0$  and  $\Phi_k > \Phi_k^c$  for each  $k$ , then all the matrices  $B_k$  generated by the Broyden update (6.23) remain symmetric and positive definite.

Moreover, for general nonlinear functions, when the line-search is exact, *all the methods* in the Broyden class with  $\Phi_k \geq \Phi_k^c$  generate the same sequence of iterates because, when all the line-searches are exact, the directions generated by the Broyden class of updates differ only in their lengths.

Applied with exact line-searches to quadratic functions, the Broyden class of methods has some remarkable properties, as it is stated in the following theorem.

**Theorem 6.1** Suppose that a method in the Broyden class is applied to minimize the strongly convex quadratic function  $f(x) = \frac{1}{2}x^T Ax + b^T x$ , where  $x_0$  is the initial point and  $B_0$  is any symmetric and positive definite matrix. Assume that  $\alpha_k$  is the exact stepsize and that  $\Phi_k \geq \Phi_k^c$  for all  $k$ , where  $\Phi_k^c$  is given by (6.25). Then, the following statements are true:

1. The iterates are independent of  $\Phi_k$  and converge to the minimum point in at most  $n$  iterations.
2. The secant equation is satisfied for all previous search directions, that is,  $B_k s_j = y_j$ ,  $j = k - 1, k - 2, \dots, 1$ .
3. If the initial matrix  $B_0 = I$ , then the iterations are identical to those generated by the conjugate gradient method. In particular, the search directions are conjugate, that is,  $s_i^T A s_j = 0$ , for  $i \neq j$ .
4. If  $n$  iterations are performed, then  $B_n = A$ . ◆

Theorem 6.1 can be generalized in the sense that it continues to hold if the Hessian approximations remain nonsingular, but not necessarily positive definite. More than that, if the initial matrix  $B_0$  is not the identity matrix, then the Broyden class of methods is identical to the preconditioned conjugate gradient method that uses  $B_0$  as preconditioner.

Since BFGS and DFP conserve the positive definiteness of the Hessian approximations when  $y_k^T s_k > 0$ , it follows that the same property will hold for the Broyden family if  $0 \leq \Phi_k \leq 1$ . The restricted Broyden class is obtained by restricting  $\Phi_k$  to the interval  $[0,1]$ .

The following result shows the convergence of the restricted Broyden class of update formulae (Byrd, Nocedal, & Yuan, 1987a). The theorem assumes that  $\nabla^2 f(x)$  is always positive definite, that is, the objective function  $f$  is strictly convex. It excludes the DFP formula.

**Theorem 6.2** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a real-valued function. Let  $x_0$  be an initial point and  $\{x_k\}$  a sequence where  $x_{k+1} = x_k + \alpha_k d_k$ ,  $\alpha_k \geq 0$  is a scalar, and  $d_k \in \mathbb{R}^n$ . Assume that the level set  $S = \{x : f(x) \leq f(x_0)\}$  is bounded,  $f(x)$ ,  $\nabla f(x)$ , and  $\nabla^2 f(x)$  are continuous for all  $x \in S$ ,  $\nabla^2 f(x)$  is positive definite for all  $x$ , the search directions  $\{d_k\}$  are computed as solutions of the linear systems  $B_k d_k = -\nabla f(x_k)$ , where  $B_0 = I$ , and the matrices  $\{B_k\}$  are updated using the restricted Broyden class, with  $0 \leq \Phi_k < 1$ , the stepsizes  $\{\alpha_k\}$  satisfy the Wolfe line-search (6.5) and (6.6) with  $0 < \rho < \sigma < 1$ , and the line-search uses  $\alpha_k = 1$  whenever possible. Then,

$$\lim_{k \rightarrow \infty} x_k = x^*,$$

where  $x^*$  is the unique global minimizer of  $f$  and the rate of convergence of  $\{x_k\}$  is superlinear. ◆

### The Global Convergence of BFGS

In the convergence analysis, a key requirement for a line-search algorithm is that the search direction  $d_k$  is a direction of *sufficient descent*, which is defined as

$$\frac{g_k^T d_k}{\|g_k\| \|d_k\|} \leq -\varepsilon, \quad (6.26)$$

where  $\varepsilon > 0$ . This condition bounds the elements of the sequence  $\{d_k\}$  of the search directions from being arbitrarily close to the orthogonality to the gradient. Often, the line-search methods are so that  $d_k$  is defined in a way that satisfies the sufficient descent condition (6.26), even though an explicit value for  $\varepsilon > 0$  is not known.

**Theorem 6.3** Suppose that  $\{B_k\}$  is a sequence of bounded and positive definite symmetric matrices whose condition number is also bounded, i.e., the smallest eigenvalue is bounded away from zero. If  $d_k$  is defined to be the solution of the system  $B_k d_k = -g_k$ , then  $\{d_k\}$  is a sequence of sufficient descent directions.

**Proof** Let  $B_k$  be a symmetric positive definite matrix with eigenvalues  $0 < \lambda_1^k \leq \lambda_2^k \leq \dots \leq \lambda_n^k$ . Therefore, from  $B_k d_k = -g_k$ , it follows that

$$\|g_k\| = \|B_k d_k\| \leq \|B_k\| \|d_k\| = \lambda_n^k \|d_k\|. \quad (6.27)$$

From the system  $B_k d_k = -g_k$ , using (6.27), we have

$$-\frac{g_k^T d_k}{\|g_k\| \|d_k\|} = \frac{d_k^T B_k d_k}{\|g_k\| \|d_k\|} \geq \lambda_1^k \frac{\|d_k\|^2}{\|g_k\| \|d_k\|} = \lambda_1^k \frac{\|d_k\|}{\|g_k\|} \geq \lambda_1^k \frac{\|d_k\|}{\lambda_n^k \|d_k\|} = \frac{\lambda_1^k}{\lambda_n^k} > 0.$$

The quality of the search direction  $d_k$  can be determined by studying the angle  $\theta_k$  between the steepest descent direction  $-g_k$  and the search direction  $d_k$ . Hence, by applying this result to each matrix in the sequence  $\{B_k\}$ , we get

$$\cos \theta_k = -\frac{g_k^T d_k}{\|g_k\| \|d_k\|} \geq \frac{\lambda_1^k}{\lambda_n^k} \geq \frac{1}{M}, \quad (6.28)$$

where  $M$  is a positive constant. Observe that  $M$  is a positive constant and it is well-defined since the smallest eigenvalue of the matrices  $B_k$  in the sequence  $\{B_k\}$  generated by the algorithm is bounded away from zero. Therefore, the search directions  $\{d_k\}$  generated as solutions of  $B_k d_k = -g_k$  form a sequence of sufficient descent directions.  $\blacklozenge$

The main consequence of this theorem on how to modify the quasi-Newton system defining the search direction  $d_k$  is to ensure that  $d_k$  is a solution of a system that has the same properties as  $B_k$ .

A global convergence result for the BFGS method was given by Powell (1976). Using the trace and the determinant to measure the effect of the two rank-one corrections on  $B_k$  in (6.22), he proved that if  $f$  is convex, then for any starting point  $x_0$  and any positive definite starting matrix  $B_0$ , the BFGS method gives  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ . In addition, if the sequence  $\{x_k\}$  converges to a solution point at which the Hessian matrix is positive definite, then the rate of convergence is superlinear. The analysis of Powell was extended by Byrd, Nocedal, and Yuan (1987a) to the Broyden class of quasi-Newton methods (see Theorem 6.2).

With the Wolfe line-search, the BFGS approximation is always positive definite, so the line-search works very well. It behaves “almost” like the Newton method in the limit (the convergence is superlinear). DFP has the interesting property that, for a quadratic objective, it simultaneously generates the directions of the conjugate gradient method while constructing the inverse Hessian. However, DFP is highly sensitive to the inaccuracies in the line-searches.

Consider the BFGS algorithm with the Wolfe line-search for minimizing a twice continuously differentiable function from an arbitrary initial point  $x_0$  and from any initial approximation of the Hessian  $B_0$ , that is, a symmetric and positive definite matrix. The following assumptions are supposed to be satisfied by the minimizing function  $f$ :

**Assumption A**

- (i) *The minimizing function  $f$  is twice continuously differentiable.*
- (ii) *The level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  is convex, and there exist the positive constants  $m$  and  $M$  such that*

$$m\|z\|^2 \leq z^T G(x) z \leq M\|z\|^2, \quad (6.29)$$

for all  $z \in \mathbb{R}^n$  and  $x \in S$ , where  $G(x) = \nabla^2 f(x)$ .

Observe that (ii) implies that the Hessian  $G(x)$  is positive definite on  $S$  and, therefore,  $f$  has a unique minimum.

Now, since  $y_k = \bar{G}_k s_k$ , where  $\bar{G}_k$  is the average Hessian, from (6.29), it follows that

$$\frac{y_k^T s_k}{s_k^T s_k} = \frac{s_k^T \bar{G}_k s_k}{s_k^T s_k} \geq m. \quad (6.30)$$

On the other hand, by Assumption A,  $\bar{G}_k$  is positive definite, and defining  $z_k = \bar{G}_k^{1/2} s_k$ , it follows that

$$\frac{y_k^T y_k}{y_k^T s_k} = \frac{s_k^T \bar{G}_k^2 s_k}{s_k^T \bar{G}_k s_k} = \frac{z_k^T \bar{G}_k z_k}{z_k^T z_k} \leq M. \quad (6.31)$$

For proving the global convergence of the BFGS method, instead of establishing a bound on the condition number of the Hessian approximations  $B_k$  as it was considered for proving the convergence of the descent direction algorithms (see Chap. 2), Byrd, Nocedal, and Yuan (1987a) introduced two new tools based on the trace and determinant in order to estimate the size of the largest and the smallest eigenvalues of the Hessian approximations.

**Theorem 6.4** *Let  $B_0$  be any symmetric and positive definite initial matrix, and consider  $x_0$  as the starting point for which Assumption A is satisfied. Then, the sequence  $\{x_k\}$  generated by the BFGS Algorithm 6.1 is convergent to the minimizer  $x^*$  of  $f$ .*

**Proof** Define

$$m_k = \frac{y_k^T s_k}{s_k^T s_k}, M_k = \frac{y_k^T y_k}{y_k^T s_k}. \quad (6.32)$$

From (6.30) and (6.31), it follows that

$$m_k \geq m, M_k \leq M. \quad (6.33)$$

Now, from (6.22) (see Appendix A), we have

$$\text{trace}(B_{k+1}) = \text{trace}(B_k) - \frac{\|B_k s_k\|^2}{s_k^T B_k s_k} + \frac{\|y_k\|^2}{y_k^T s_k} \quad (6.34)$$

and

$$\det(B_{k+1}) = \det(B_k) \frac{y_k^T s_k}{s_k^T B_k s_k}. \quad (6.35)$$

Now, let us define

$$\cos \theta_k = \frac{s_k^T B_k s_k}{\|s_k\| \|B_k s_k\|}, q_k = \frac{s_k^T B_k s_k}{s_k^T s_k}, \quad (6.36)$$

where, as we can see,  $\theta_k$  is the angle between  $s_k$  and  $B_k s_k$ .

With this,

$$\frac{\|B_k s_k\|^2}{s_k^T B_k s_k} = \frac{\|B_k s_k\|^2 \|s_k\|^2}{(s_k^T B_k s_k)^2} \frac{(s_k^T B_k s_k)}{\|s_k\|^2} = \frac{q_k}{\cos^2 \theta_k}. \quad (6.37)$$

Now, a combination of the trace and determinant is introduced by the following function of a positive definite matrix  $B$ :

$$\varphi(B) = \text{trace}(B) - \ln(\det(B)), \quad (6.38)$$

where  $\ln(\cdot)$  is the natural logarithm. Fletcher (1991) observed that both the BFGS and the DFP formulae can be derived by a variational argument using function  $\varphi$ . This is an elegant and efficient tool for analyzing the global behavior of quasi-Newton methods. Observe that function  $\varphi$  works simultaneously with trace and determinant, thus simplifying the analysis of the quasi-Newton methods. In fact, this function is a measure of the matrices involving all the eigenvalues of  $B$ , not only the smallest and the largest, as it is traditionally used in the analysis of the quasi-Newton methods based on the condition number of matrices (Andrei, 2015c, 2016, 2017a). Observe that this function is strictly convex on the set of symmetric and positive definite matrices and is minimized by  $B = I$ . Besides, it becomes unbounded as  $B$  becomes singular or infinite, and therefore, it works as a barrier function that keeps  $B$  positive definite.

With this, we can write

$$\begin{aligned} \varphi(B_{k+1}) &= \text{trace}(B_k) + M_k - \frac{q_k}{\cos^2 \theta_k} - \ln(\det(B_k)) - \ln(m_k) + \ln(q_k) \\ &= \varphi(B_k) + (M_k - \ln(m_k) - 1) + \left[ 1 - \frac{q_k}{\cos^2 \theta_k} + \ln \frac{q_k}{\cos^2 \theta_k} \right] + \ln \cos^2 \theta_k. \end{aligned} \quad (6.39)$$

Since the function  $h(t) = 1 - t + \ln(t)$  is nonpositive for all  $t > 0$ , it follows that the term inside the brackets is nonpositive, and thus, from (6.33) and (6.39), it follows that

$$0 < \varphi(B_{k+1}) \leq \varphi(B_0) + c(k+1) + \sum_{j=0}^k \ln \cos^2 \theta_j, \quad (6.40)$$

where without the loss of the generality, we can assume that the constant  $c = M - \ln(m) - 1$  is positive.

Observe that  $\cos\theta_k$  defined by (6.36) is the angle between the steepest descent direction and the search direction, which is crucial in the global convergence of the descent direction algorithms. We know that the sequence  $\{\|g_k\|\}$  generated by the line-search algorithm is bounded away from zero only if  $\cos\theta_j \rightarrow 0$  (see Corollary 2.1).

Now, let us proceed by contradiction, and assume that  $\cos\theta_j \rightarrow 0$ . Then, there exists  $k_1 > 0$  such that for all  $j > k_1$  we have  $\ln(\cos^2\theta_j) < -2c$ , where  $c$  is the constant defined above. Therefore, from (6.40), it follows that for all  $k > k_1$  we have

$$\begin{aligned} 0 &< \varphi(B_0) + c(k+1) + \sum_{j=0}^{k_1} \ln \cos^2\theta_j + \sum_{j=k_1+1}^k (-2c) \\ &= \varphi(B_0) + \sum_{j=0}^{k_1} \ln \cos^2\theta_j + 2ck_1 + c - ck. \end{aligned}$$

However, the right-hand side is negative for large  $k$ , thus giving a contradiction. Therefore, there exists a subsequence of indices  $\{j_k\}_{k=1, 2, \dots}$  such that  $\cos\theta_{j_k} \geq \delta > 0$ . By the Zoutendijk condition, it follows that  $\liminf \|\nabla f(x_k)\| \rightarrow 0$ . Since the problem is strongly convex, this limit is sufficient to prove that  $x_k \rightarrow x^*$ .  $\diamond$

Theorem 6.4 has been generalized to the entire restricted Broyden class, *except for the DFP method*. That is, Theorem 6.4 holds for all  $\Phi_k \in [0, 1]$ .

The BFGS method has many interesting properties. For example, the BFGS updating formula has effective *self-correcting properties*. If the matrix  $B_k$  incorrectly estimates the curvature of the minimizing function, and if this bad estimate slows down the iteration, then the Hessian approximation will tend to correct itself within a few steps. The DFP method is less effective in correcting bad Hessian approximations, and it is believed that this is the reason for its poorer practical performances. The self-correcting property of BFGS holds only when the Wolfe line-search is used for the stepsize determination.

The BFGS updating formula is known to have the so-called *bounded deterioration* property (Broyden, Dennis, & Moré, 1973), namely, the approximate Hessians do not drift too far from the exact Hessian if the initial data are good. An updating formula for the Hessian approximation  $B_k$  satisfies the bounded deterioration property if there exists a constant  $c > 0$  so that for all  $x_k$  and  $B_k$  the new  $x_{k+1}$  and  $B_{k+1}$  satisfy

$$\|B_{k+1} - \nabla^2 f(x^*)\| \leq \|B_k - \nabla^2 f(x^*)\|(1 + \sigma) + c\sigma,$$

where  $\sigma = \max\{\|x_k - x^*\|, \|x_{k+1} - x^*\|\}$ . If a quasi-Newton method satisfies the bounded deterioration property and there exist the positive constants  $\epsilon$  and  $\delta$  so that  $\|x_0 - x^*\| \leq \epsilon$  and  $\|B_0 - \nabla^2 f(x^*)\| \leq \delta$ , then the sequence  $\{x_k\}$  generated by the algorithm is well-defined and converges  $q$ -linearly to  $x^*$ . In order to have  $q$ -superlinearly convergence of the algorithm, it is necessary to have the *consistency* of it. A quasi-Newton method is consistent if  $\{x_k\}$  converges to  $x^*$ , then  $\{B_k\}$  converges to  $\nabla^2 f(x^*)$ . The consistency condition is sufficient but not necessary.

Assuming that the initial point  $x_0$  is sufficiently close to the solution  $x^*$  and that the initial Hessian approximation is sufficiently close to  $\nabla^2 f(x^*)$ , then using the bounded deterioration property we can prove that the iteration cannot stay away from the solution.

Another interesting property is the *Dennis-Moré condition*. If  $d_k = -B_k^{-1}\nabla f(x_k)$  is a BFGS search direction or a quasi-Newton search direction, then

$$\lim_{k \rightarrow \infty} \frac{\|(B_k - \nabla^2 f(x^*))d_k\|}{\|d_k\|} = 0. \quad (6.41)$$

Therefore, a superlinear convergence rate can be attained even if the sequence of the quasi-Newton matrices  $B_k$  does not converge to  $\nabla^2 f(x^*)$ ; suffice it to say that the  $B_k$  become increasingly accurate approximations to  $\nabla^2 f(x^*)$  along the search directions  $d_k$ . Specifically, we have

**Theorem 6.5** Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable, and consider the iteration  $x_{k+1} = x_k + \alpha_k d_k$ , where  $d_k$  is a descent direction and  $\alpha_k$  satisfies the Wolfe line-search conditions (6.5) and (6.6) with  $\rho \leq 1/2$ . If the sequence  $\{x_k\}$  converges to a point  $x^*$  such that  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite, and if the search direction satisfies

$$\lim_{k \rightarrow \infty} \frac{\|\nabla f(x_k) - \nabla^2 f(x_k)d_k\|}{\|d_k\|} = 0,$$

then the stepsize  $\alpha_k = 1$  is admissible for all  $k$  greater than a certain index  $k_0$ ; if  $\alpha_k = 1$  for all  $k > k_0$ , then  $\{x_k\}$  converges to  $x^*$  superlinearly.

The following theorem emphasizes that the Dennis-Moré condition (6.41) is both *necessary* and *sufficient* for the superlinear convergence of the quasi-Newton methods.

**Theorem 6.6** Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable. Consider the iteration  $x_{k+1} = x_k + d_k$ , that is, the stepsize  $\alpha_k$  is uniformly one and  $d_k = -B_k^{-1}\nabla f(x_k)$ . Let us assume that the sequence  $\{x_k\}$  converges to a point  $x^*$  such that  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite. Then, the sequence  $\{x_k\}$  converges superlinearly if and only if the Dennis-Moré condition (6.41) holds.

**Proof** Let  $d_k^N = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$  be the Newton step. Let us show that (6.41) is equivalent to

$$d_k - d_k^N = o(\|d_k\|). \quad (6.42)$$

Assuming that (6.41) holds and since  $\|\nabla^2 f(x_k)^{-1}\|$  is bounded above for  $x_k$  sufficiently close to  $x^*$  and since  $\nabla^2 f(x^*)$  is positive definite, then it follows that

$$\begin{aligned} d_k - d_k^N &= \nabla^2 f(x_k)^{-1}(\nabla^2 f(x_k)d_k + \nabla f(x_k)) \\ &= \nabla^2 f(x_k)^{-1}(\nabla^2 f(x_k) - B_k)d_k \\ &= O(\|\nabla^2 f(x_k) - B_k\|) = o(\|d_k\|). \end{aligned}$$

The converse follows if (6.42) is multiplied by  $\nabla^2 f(x_k)$ . We have

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

Therefore,  $\|d_k\| = O(\|x_k - x^*\|)$ , that is,  $\|x_k + d_k - x^*\| \leq o(\|x_k - x^*\|)$ , proving the superlinear convergence of  $\{x_k\}$ .  $\blacklozenge$

An extension of this analysis shows that the rate of convergence of the BFGS algorithm is linear. More exactly, we can show that the sequence  $\{\|x_k - x^*\|\}$  converges to zero rapidly enough, so that

$$\sum_{k=1}^{\infty} \|x_k - x^*\| < \infty. \quad (6.43)$$

If the minimizing function is twice continuously differentiable and the iterates generated by the BFGS algorithm converge to a minimizer  $x^*$  at which the Hessian  $\nabla^2 f(x)$  is Lipschitz continuous and (6.43) holds, then  $\{x_k\}$  converges to  $x^*$  at a superlinear rate. More exactly, the local superlinear convergence of the BFGS algorithm can be established by the following arguments, presented for completeness.

Consider that the Hessian  $\nabla^2 f(x)$  is Lipschitz continuous, i.e.,  $\|\nabla^2 f(x) - \nabla^2 f(x^*)\| \leq L\|x - x^*\|$  for all  $x$  near enough  $x^*$  and  $L$ , a positive constant. Let us introduce the following quantities:

$$\tilde{s}_k = G_*^{1/2} s_k, \quad \tilde{y}_k = G_*^{-1/2} y_k, \quad \tilde{B}_k = G_*^{-1/2} B_k G_*^{-1/2}$$

where  $G_* = \nabla^2 f(x^*)$  and  $x^*$  is a minimizer of function  $f$ . Define:

$$\cos \tilde{\theta}_k = \frac{\tilde{s}_k^T \tilde{B}_k \tilde{s}_k}{\|\tilde{s}_k\| \|\tilde{B}_k \tilde{s}_k\|}, \quad \tilde{q}_k = \frac{\tilde{s}_k^T \tilde{B}_k \tilde{s}_k}{\|\tilde{s}_k\|^2}, \quad \tilde{M}_k = \frac{\|\tilde{y}_k\|^2}{\tilde{y}_k^T \tilde{s}_k}, \quad \tilde{m}_k = \frac{\tilde{y}_k^T \tilde{s}_k}{\tilde{s}_k^T \tilde{s}_k}.$$

Now, pre- and post-multiplying (6.22) by  $\nabla^2 F(x^*)^{-1/2}$  and grouping the terms, we obtain

$$\tilde{B}_{k+1} = \tilde{B}_k - \frac{\tilde{B}_k \tilde{s}_k \tilde{s}_k^T \tilde{B}_k}{\tilde{s}_k^T \tilde{B}_k \tilde{s}_k} + \frac{\tilde{y}_k \tilde{y}_k^T}{\tilde{y}_k^T \tilde{s}_k}.$$

Therefore, as in Theorem 6.4 above, it follows that

$$\varphi(\tilde{B}_{k+1}) = \varphi(\tilde{B}_k) + (\tilde{M}_k - \ln(\tilde{m}_k) - 1) + \left[ 1 - \frac{\tilde{q}_k}{\cos^2 \tilde{\theta}_k} + \ln \frac{\tilde{q}_k}{\cos^2 \tilde{\theta}_k} \right] + \ln \cos^2 \tilde{\theta}_k. \quad (6.44)$$

However,  $y_k - G_* s_k = (\bar{G}_k - G_*) s_k$ . Therefore,  $\tilde{y}_k - \tilde{s}_k = G_*^{-1/2} (\bar{G}_k - G_*) G_*^{-1/2} \tilde{s}_k$ . Since the Hessian matrix  $G$  is Lipschitz continuous, and having in view the definition of the average Hessian, it follows that

$$\|\tilde{y}_k - \tilde{s}_k\| \leq \|G_*^{-1/2}\|^2 \|\tilde{s}_k\| \|\bar{G}_k - G_*\| \leq \|G_*^{-1/2}\|^2 \|\tilde{s}_k\| L \varepsilon_k,$$

where  $\varepsilon_k = \max \{\|x_{k+1} - x^*\|, \|x_k - x^*\|\}$ . Therefore,

$$\frac{\|\tilde{y}_k - \tilde{s}_k\|}{\|\tilde{s}_k\|} \leq \bar{c} \varepsilon_k, \quad (6.45)$$

for some positive constant  $\bar{c}$ . This inequality has a crucial role in the local superlinear convergence of the BFGS method.

**Theorem 6.7** Suppose that  $f$  is twice continuously differentiable and the sequence  $\{x_k\}$  generated by the BFGS algorithm converges to the minimum point  $x^*$  at which the Hessian is Lipschitz continuous. Also suppose that (6.43) holds. Then, the sequence  $\{x_k\}$  converges to  $x^*$  at a superlinear rate.

**Proof** From (6.45) and from the triangle inequality, it follows that  $\|\tilde{y}_k\| - \|\tilde{s}_k\| \leq \bar{c}\epsilon_k \|\tilde{s}_k\|$ ,  $\|\tilde{s}_k\| - \|\tilde{y}_k\| \leq \bar{c}\epsilon_k \|\tilde{s}_k\|$ , so that

$$(1 - \bar{c}\epsilon_k) \|\tilde{s}_k\| \leq \|\tilde{y}_k\| \leq (1 + \bar{c}\epsilon_k) \|\tilde{s}_k\|. \quad (6.46)$$

By squaring (6.45) and using (6.46), we get

$$(1 - \bar{c}\epsilon_k)^2 \|\tilde{s}_k\|^2 - 2\tilde{y}_k^T \tilde{s}_k + \|\tilde{s}_k\|^2 \leq \|\tilde{y}_k\|^2 - 2\tilde{y}_k^T \tilde{s}_k + \|\tilde{s}_k\|^2 \leq \bar{c}^2 \epsilon_k^2 \|\tilde{s}_k\|^2,$$

and therefore,

$$2\tilde{y}_k^T \tilde{s}_k \geq (1 - 2\bar{c}\epsilon_k + \bar{c}^2 \epsilon_k^2 + 1 - \bar{c}^2 \epsilon_k^2) \|\tilde{s}_k\|^2 = 2(1 - \bar{c}\epsilon_k) \|\tilde{s}_k\|^2.$$

Hence,

$$\tilde{m}_k = \frac{\tilde{y}_k^T \tilde{s}_k}{\tilde{s}_k^T \tilde{s}_k} \geq 1 - \bar{c}\epsilon_k. \quad (6.47)$$

Now, from (6.46), it follows that

$$\tilde{M}_k = \frac{\|\tilde{y}_k\|^2}{\tilde{y}_k^T \tilde{s}_k} \leq \frac{1 + \bar{c}\epsilon_k}{1 - \bar{c}\epsilon_k}. \quad (6.48)$$

Since  $x_k \rightarrow x^*$ , it follows that  $\epsilon_k \rightarrow 0$ . By (6.48), there exists a positive constant  $c > \bar{c}$  such that the following inequalities hold for all sufficiently large  $k$ :

$$\tilde{M}_k \leq 1 + \frac{2\bar{c}}{1 - \bar{c}\epsilon_k} \epsilon_k \leq 1 + c\epsilon_k. \quad (6.49)$$

Since the function  $h(t) = 1 - t + \ln(t)$  is nonpositive, it follows that

$$\frac{-x}{1-x} - \ln(1-x) = h\left(\frac{1}{1-x}\right) \leq 0.$$

Now, for  $k$  large enough, we can assume that  $\bar{c}\epsilon_k < 1/2$ , and therefore,

$$\ln(1 - \bar{c}\epsilon_k) \geq \frac{-\bar{c}\epsilon_k}{1 - \bar{c}\epsilon_k} \geq -2\bar{c}\epsilon_k.$$

This relation and (6.47) imply that for  $k$  large enough, we have

$$\ln(\tilde{m}_k) \geq \ln(1 - \bar{c}\epsilon_k) \geq -2\bar{c}\epsilon_k > -2c\epsilon_k. \quad (6.50)$$

Therefore, from (6.44), (6.49) and (6.50), we obtain

$$0 < \varphi(\tilde{B}_{k+1}) \leq \varphi(\tilde{B}_k) + 3c\epsilon_k + \ln(\cos^2 \tilde{\theta}_k) + \left[1 - \frac{\tilde{q}_k}{\cos^2 \tilde{\theta}_k} + \ln \frac{\tilde{q}_k}{\cos^2 \tilde{\theta}_k}\right]. \quad (6.51)$$

Now, summing these expressions and using (6.43), we obtain

$$\sum_{j=0}^{\infty} \left( \ln \frac{1}{\cos^2 \tilde{\theta}_j} - \left[ 1 - \frac{\tilde{q}_j}{\cos^2 \tilde{\theta}_j} + \ln \frac{\tilde{q}_j}{\cos^2 \tilde{\theta}_j} \right] \right) \leq \varphi(\tilde{B}_0) + 3c \sum_{j=0}^{\infty} \varepsilon_j < +\infty$$

Since the term in the square brackets is nonpositive and since  $\ln(1/\cos^2 \tilde{\theta}_j) \geq 0$  for all  $j$ , it follows that

$$\lim_{j \rightarrow \infty} \ln \frac{1}{\cos^2 \tilde{\theta}_j} = 0, \quad \lim_{j \rightarrow \infty} \left( 1 - \frac{\tilde{q}_j}{\cos^2 \tilde{\theta}_j} + \ln \frac{\tilde{q}_j}{\cos^2 \tilde{\theta}_j} \right) = 0,$$

which implies that

$$\lim_{j \rightarrow \infty} \cos \tilde{\theta}_j = 1, \quad \lim_{j \rightarrow \infty} \tilde{q}_j = 1. \quad (6.52)$$

From (6.37), we have

$$\begin{aligned} \frac{\|G_*^{-1/2}(B_k - G_*)s_k\|^2}{\|G_*^{1/2}s_k\|^2} &= \frac{\|\tilde{(B_k - I)\tilde{s}_k}\|^2}{\|\tilde{s}_k\|^2} \\ &= \frac{\|\tilde{B}_k \tilde{s}_k\|^2 - 2\tilde{s}_k^T \tilde{B}_k \tilde{s}_k + \tilde{s}_k^T s_k}{\tilde{s}_k^T s_k} = \frac{\tilde{q}_k^2}{\cos^2 \tilde{\theta}_k} - 2\tilde{q}_k + 1. \end{aligned}$$

Since by (6.52) the right-hand side converges to zero, it follows that

$$\lim_{k \rightarrow \infty} \frac{\|(B_k - G_*)s_k\|}{\|s_k\|} = 0.$$

Theorem 6.5 implies that the unit stepsize  $\alpha_k = 1$  will satisfy the Wolfe line-search near the solution, and therefore, the rate of convergence is superlinear.  $\blacklozenge$

### The Memoryless BFGS Method

By considering  $H_k = I$  in (6.21), the *memoryless BFGS method* is obtained as

$$H_{k+1} = I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + \left( 1 + \frac{y_k^T y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}. \quad (6.53)$$

The corresponding search direction is  $d_{k+1} = -H_{k+1}g_{k+1}$ , where  $H_{k+1}$  is given by (6.53), i.e.,

$$d_{k+1} = -g_{k+1} + \frac{(y_k^T g_{k+1})s_k + (s_k^T g_{k+1})y_k}{y_k^T s_k} - \left( 1 + \frac{y_k^T y_k}{y_k^T s_k} \right) \frac{(s_k^T g_{k+1})s_k}{y_k^T s_k}. \quad (6.54)$$

Observe that the numerical computation of  $d_{k+1}$  from (6.54) involves only four scalar products:  $y_k^T s_k$ ,  $y_k^T y_k$ ,  $y_k^T g_{k+1}$ , and  $s_k^T g_{k+1}$ . Therefore, it is very suitable to solve large-scale problems. It is worth seeing that the search direction corresponding to the memoryless BFGS updating has three terms. Besides, it is easy to prove that this search direction satisfies the Dai-Liao conjugacy condition, i.e.,  $y_k^T d_{k+1} = -s_k^T g_{k+1}$ . For the exact line-search, that is, when  $s_k^T g_{k+1} = 0$ , the memoryless BFGS search direction is exactly the search direction of Hestenes and Stiefel. Therefore, there is a close connection between the quasi-Newton and the conjugate gradient methods. Shanno (1978a, 1978b)

was the first who observed that *the conjugate gradient methods are precisely the quasi-Newton methods where the approximation to the inverse to the Hessian is restarted as the identity matrix at every iteration.*

## 6.2 Modifications of the BFGS Method

In the following, some modifications of the BFGS updating method, both subject to its updating formula and to the line-search conditions, are going to be presented. Intensive numerical experiments on minimizing functions with different dimensions and complexities showed that the BFGS method may require a large number of iterations or function and gradient evaluations on certain problems (Gill, & Leonard, 2003). The sources of the inefficiency of the BFGS method may be caused by a poor initial approximation to the Hessian or, more importantly, by the ill-conditioning of the Hessian approximations along the iterations. To improve the efficiency and the robustness of the BFGS method and to overcome the difficulties, some modified versions of it were given. All these modified BFGS methods can be classified into three large classes: the scaling of the BFGS update matrix, the BFGS update with modified secant equation, and the modified BFGS method using different line-search conditions for the stepsize computation. The scaling of the BFGS update has two developments: *sizing*, i.e., multiplying the approximate Hessian matrix by an appropriate scalar before it is updated in the BFGS method (Contreras and Tapia (1993), Oren and Luenberger (1974), Oren and Spedicato (1976), Shanno and Phua (1978), Yabe, Martínez, and Tapia (2004)), and the *proper scaling of the terms* on the right-hand side of the BFGS updating formula with positive factors (Biggs (1971, 1973), Oren (1972), Liao (1997), Nocedal and Yuan (1993), Andrei (2018a, 2018c, 2018d, 2018f)). The purpose of the BFGS update with *modified secant equation* is to approximate the curvature of the objective function along the search direction more accurately than the standard secant equation does (Yuan (1991), Yuan and Byrd (1995), Al-Baali (1998), Zhang, Deng, and Chen (1999), Zhang, and Xu (2001), Wei, Yu, Yuan, and Lian (2004), Zhu, and Wen (2006), Yabe, Ogasawara, and Yoshino (2007), Al-Baali, and Grandinetti (2009), Yuan, and Wei (2010), Wu, and Liang (2014), Arzam, Babaie-Kafaki, and Ghanbari (2017)). The BFGS methods with *new line-search conditions* for the stepsize computation try to ensure the global convergence by modifying the Wolfe line-search conditions (Wan, Huang, and Zheng (2012), Wan, Teo, Shen, and Hu (2014), Yuan, Wei, and Lu (2017), Yuan, Sheng, Wang, Hu, and Li (2018), Dehmiry, 2019).

### Scaling the Terms on the Right-Hand Side of the BFGS Update

From (6.22), we can see that the BFGS update involves two correction matrices, each of rank one. Therefore, by the interlocking eigenvalue theorem of Wilkinson (1965), the first rank-one correction matrix which is subtracted decreases the eigenvalues, i.e., it shifts the eigenvalues to the left. On the other hand, the second rank-one matrix which is added shifts the eigenvalues to the right. More exactly, two important tools in the analysis of the properties and of the convergence of the BFGS method are the trace and the determinant of the standard  $B_{k+1}$  given by (6.22). The trace of a matrix is exactly the sum of its eigenvalues. The determinant of a matrix is the product of its eigenvalues. By direct computation, from (6.22), we get (see Appendix A)

$$\text{tr}(B_{k+1}) = \text{tr}(B_k) - \frac{\|B_k s_k\|^2}{s_k^T B_k s_k} + \frac{\|y_k\|^2}{y_k^T s_k}.$$

On the other hand,

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

As it is known, the efficiency of the BFGS method is dependent on the structure of the eigenvalues of the approximation to the Hessian matrix (Nocedal, 1992). Powell (1987) and Byrd, Liu, and Nocedal (1992) emphasized that the BFGS method actually *suffers* more from the large eigenvalues than from the small ones. Observe that the second term on the right-hand side of  $\text{tr}(B_{k+1})$  is negative. Therefore, it produces a shift of the eigenvalues of  $B_{k+1}$  to the left. Thus, the BFGS method is able to correct large eigenvalues. On the other hand, the third term on the right-hand side of  $\text{tr}(B_{k+1})$  being positive produces a shift of the eigenvalues of  $B_{k+1}$  to the right. If this term is large,  $B_{k+1}$  may have large eigenvalues, too. Therefore, a correction of the eigenvalues of  $B_{k+1}$  can be achieved by scaling the corresponding terms in (6.22), and this is the main motivation for which the scaled BFGS methods is used. There must be a balance between these eigenvalue shifts; otherwise, the Hessian approximation could either approach singularity or become arbitrarily large, thus ruining the convergence of the method. The scaling procedures of the BFGS update (6.22) with one or two parameters know the following developments.

1. *One parameter scaling the third term on the right-hand side of the BFGS update.* In this case, the general scaling BFGS updating formula is

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \gamma_k \frac{y_k y_k^T}{y_k^T s_k}, \quad (6.55)$$

where  $\gamma_k$  is a positive parameter. For the selection of the scaling factor  $\gamma_k$  in (6.55), the following procedures have been considered in literature.

- 1.1 *Scaling BFGS with Hermite interpolation conditions* (Biggs, 1971, 1973). If the objective function is cubic along the line segment connecting  $x_{k-1}$  and  $x_k$  if the Hermite interpolation is used on the same line between  $x_{k-1}$  and  $x_k$ , Biggs (1971) proposed the following value for the scaling factor  $\gamma_k$ :

$$\gamma_k = \frac{6}{y_k^T s_k} (f(x_k) - f(x_{k+1}) + s_k^T g_{k+1}) - 2. \quad (6.56)$$

For one-dimensional problems, Wang and Yuan (1992) showed that the scaling BFGS (6.55) with  $\gamma_k$  given by (6.56) and without line-search is  $r$ -linearly convergent.

- 1.2 *Scaling BFGS with a simple interpolation condition* (Yuan, 1991). By using a simple interpolation condition on the quadratic approximation of the minimizing function  $f$ , the value for the scaling parameter in (6.55) suggested by Yuan (1991) is

$$\gamma_k = \frac{2}{y_k^T s_k} (f(x_k) - f(x_{k+1}) + s_k^T g_{k+1}). \quad (6.57)$$

Powell (1986a) showed that the scaling BFGS update (6.55) with  $\gamma_k$  given by (6.57) is globally convergent for convex functions with inexact line-search. However, for general nonlinear functions, the inexact line-search does not involve the positivity of  $\gamma_k$ . In these cases, Yuan

restricted  $\gamma_k$  in the interval [0.01, 100] and proved the global convergence of this variant of the scaling BFGS method.

1.3 *Spectral scaling BFGS* (Cheng, & Li, 2010). In this update, the scaling parameter  $\gamma_k$  in (6.54) is computed as

$$\gamma_k = \frac{y_k^T s_k}{\|y_k\|^2}, \quad (6.58)$$

which is obtained as solution of the problem:  $\min \|s_k - \gamma_k y_k\|^2$ . Observe that  $\gamma_k$  given by (6.58) is exactly one of the spectral stepsizes introduced by Barzilai and Borwein (1988). Therefore, the scaling BFGS method (6.55) with  $\gamma_k$  given by (6.58) is viewed as the spectral scaling BFGS method. It is proved that this spectral scaling BFGS method with the Wolfe line-search is globally convergent and  $r$ -linearly convergent for convex optimization problems. Cheng and Li (2010) presented the computational evidence that their spectral scaling BFGS algorithm is the top performer versus the standard BFGS and also versus the scaling BFGS algorithms by Al-Baali (1998), Yuan (1991), and Zhang and Xu (2001).

1.4 *Scaling BFGS with diagonal preconditioning and conjugacy condition* (Andrei, 2018a). Andrei (2018a) introduced another scaling BFGS update given by (6.54), in which the scaling parameter  $\gamma_k$  is computed in an adaptive manner as

$$\gamma_k = \min \left\{ \frac{y_k^T s_k}{\|y_k\|^2 + \beta_k}, 1 \right\}, \quad (6.59)$$

where  $\beta_k > 0$  for all  $k = 0, 1, \dots$ . Since under the Wolfe line-search conditions (6.5) and (6.6)  $y_k^T s_k > 0$  for all  $k = 0, 1, \dots$ , it follows that  $\gamma_k$  given by (6.59) is bounded away from zero, i.e.,  $0 < \gamma_k \leq 1$ . If  $\gamma_k$  is selected as in (6.59), where  $\beta_k > 0$  for all  $k = 0, 1, \dots$ , then the large eigenvalues of  $B_{k+1}$  given by (6.55) are shifted to the left (Andrei, 2018a). Intensive numerical experiments showed that this scaling BFGS algorithm with  $\beta_k = |s_k^T g_{k+1}|$  is one of the best, being more efficient and more robust versus the standard BFGS algorithm as well as versus some other scaling BFGS algorithms, including the versions of Biggs (1971, 1973), Yuan (1991), and Cheng and Li (2010).

Andrei (2018a) gives the following theoretical justification for selecting the parameter  $\gamma_k$  as in (6.59) with  $\beta_k = |s_k^T g_{k+1}|$ . To have a good algorithm, we need  $\gamma_k I$  to be a diagonal preconditioner of  $\nabla^2 f(x_{k+1})$  that reduces the condition number to the inverse of  $\nabla^2 f(x_{k+1})$ . Such matrix  $\gamma_k I$  should be a rough approximation to the inverse of  $\nabla^2 f(x_{k+1})$ . Therefore,  $\gamma_k$  can be computed to minimize  $\|s_k - \gamma_k y_k\|^2$ . On the other hand, for nonlinear functions, as known, the classical conjugacy condition used by Hestenes and Stiefel (1952) for quadratic functions which incorporate the second-order information is  $d_{k+1}^T y_k = -s_k^T g_{k+1}$ . Therefore, in this algorithm,  $\gamma_k I$  is selected to be a diagonal preconditioner of  $\nabla^2 f(x_{k+1})$  and also to minimize the conjugacy condition, i.e.,  $\gamma_k$  is selected to minimize a combination of these two conditions:

$$\min \left\{ \|s_k - \gamma_k y_k\|^2 + \gamma_k^2 |s_k^T g_{k+1}| \right\}.$$

2. *One parameter scaling the first two terms of the BFGS update* ((Oren, & Luenberger, 1974) and (Nocedal, & Yuan 1993)). This scaling BFGS update was introduced by Oren and Luenberger

(1974) in their study on self-scaling variable metric algorithms for the unconstrained optimization and was defined as

$$B_{k+1} = \delta_k \left[ B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \right] + \frac{y_k y_k^T}{y_k^T s_k}, \quad (6.60)$$

where  $\delta_k$  is a positive parameter. Oren and Luenberger (1974) suggested

$$\delta_k = \frac{y_k^T s_k}{s_k^T B_k s_k} \quad (6.61)$$

as being one of the best factors, since it simplifies the analysis of the eigenvalues structure of the inverse Hessian approximation. Furthermore, Nocedal and Yuan (1993) presented a deep analysis of this scaling quasi-Newton method and showed that even if the corresponding algorithm with inexact line-search is superlinear convergent on general functions, it is computationally expensive as regards the stepsize computation.

3. *Two parameters scaling the terms on the right-hand side of the BFGS update* ((Liao, 1997) and (Andrei, 2018c, 2018d, 2018f)). In these methods, the scaling parameters of the terms on the right-hand side of the BFGS update are selected to modify the structure of the eigenvalues of the iteration matrix  $B_{k+1}$ , mainly to cluster them and to shift the large ones to the left. The following two approaches are known.

3.1 *Scaling the first two terms on the right-hand side of the BFGS update with a positive parameter and the third one with another positive parameter* (Andrei, 2018c). Motivated by the idea of changing the structure of the eigenvalues of the BFGS approximation to the Hessian matrix, Andrei (2018c) proposed a *double parameter scaling BFGS method* in which the updating of the approximation Hessian matrix  $B_{k+1}$  is computed as

$$B_{k+1} = \delta_k \left[ B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \right] + \gamma_k \frac{y_k y_k^T}{y_k^T s_k}, \quad (6.62)$$

where  $\delta_k$  and  $\gamma_k$  are positive parameters. In this scaling BFGS method, the parameter  $\delta_k$  is selected to cluster the eigenvalues of  $B_{k+1}$ . On the other hand,  $\gamma_k$  is determined to reduce the large eigenvalues of  $B_{k+1}$ , i.e., to shift them to the left, thus obtaining a better distribution of the eigenvalues:

$$\gamma_k = \min \left\{ \frac{y_k^T s_k}{\|y_k\|^2 + |s_k^T g_{k+1}|}, 1 \right\} \quad (6.63)$$

and

$$\delta_k = \frac{n - \gamma_k \frac{\|y_k\|^2}{y_k^T s_k}}{n - \frac{\|B_k s_k\|^2}{s_k^T B_k s_k}}. \quad (6.64)$$

**Theorem 6.8** *If the stepsize  $\alpha_k$  is determined by the Wolfe line-search (6.5) and (6.6),  $B_k$  is positive definite, and  $\gamma_k > 0$ , then  $B_{k+1}$  given by (6.62) is also positive definite.*  $\blacklozenge$

For general nonlinear functions, this scaling BFGS algorithm with inexact line-search is globally convergent under the very reasonable condition that the scaling parameters are bounded. Intensive numerical experiments using over 80 unconstrained optimization test problems of different structures and complexities showed that this double parameter scaling BFGS update is more efficient than the standard BFGS algorithm and also than some other well-known scaling BFGS algorithms, including those by Biggs (1971, 1973), Cheng and Li (2010), Liao (1997), Nocedal and Yuan (1993), and Yuan (1991).

*3.2 Scaling the first two terms on the right-hand side of the BFGS update with a positive parameter and the third one with another positive parameter using the measure function of Byrd and Nocedal* (Andrei, 2018d, 2018f). In this method, the BFGS update is scaled as in (6.62), where parameters  $\delta_k$  and  $\gamma_k$  are computed to minimize the measure function  $\varphi(\cdot)$  of Byrd and Nocedal (1989). Minimizing the function  $\varphi(B_{k+1}) = \text{tr}(B_{k+1}) - \ln(\det(B_{k+1}))$  with respect to the parameters  $\delta_k$  and  $\gamma_k$ , where  $B_{k+1}$  is given in (6.62), the following values are obtained:

$$\delta_k = \frac{n-1}{\text{tr}(B_k) - \frac{\|B_k s_k\|^2}{s_k^T B_k s_k}}. \quad (6.65)$$

$$\gamma_k = \frac{y_k^T s_k}{\|y_k\|^2}. \quad (6.66)$$

**Theorem 6.9** *If the stepsize  $\alpha_k$  is determined by the Wolfe line-search (6.5) and (6.6), then the scaling parameters  $\delta_k$  and  $\gamma_k$  given by (6.65) and (6.66), respectively, are the unique global solution of the problem  $\min_{\delta_k > 0, \gamma_k > 0} \varphi(B_{k+1})$ .* ♦

Intensive numerical experiments in Andrei (2018d) proved that this scaling procedure of the BFGS with two parameters is more efficient and more robust than the other scaling procedures, including those of Biggs (1971, 1973), Cheng and Li (2010), Yuan (1991), Nocedal and Yuan (1993), Liao (1997), and Andrei (2018c).

*3.3 Scaling the last terms on the right-hand side of the BFGS update with two positive parameters* (Liao, 1997). Liao (1997) introduced the two parameter scaling BFGS method as

$$B_{k+1} = B_k - \delta_k \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \gamma_k \frac{y_k y_k^T}{y_k^T s_k} \quad (6.67)$$

and proved that this method corrects the large eigenvalues better than the standard BFGS method given by (6.22). In other words, it was proved that this scaling BFGS method has a strong self-correcting property with respect to the determinant (Liao, 1997). In Liao's method, the parameters scaling the terms in the BFGS update are computed in an adaptive way subject to the values of a positive parameter as

$$(\delta_k, \gamma_k) = \begin{cases} \left( \frac{s_k^T B_k s_k}{s_k^T B_k s_k + y_k^T s_k}, \frac{y_k^T s_k}{s_k^T B_k s_k + y_k^T s_k} \right), & \text{if } \frac{s_k^T B_k s_k}{s_k^T B_k s_k + y_k^T s_k} \geq \tau_k, \\ (\tau_k, 1), & \text{otherwise,} \end{cases} \quad (6.68)$$

where  $0 < \tau_k < 1$ . (Liao proposed  $\tau_k = \exp(-1/k^2)$ .) Liao proved that the scaling BFGS method given by (6.67) and (6.68) with the Wolfe line-search generates iterations which converge superlinearly to the optimal solution. Limited numerical experiments with Liao's scaling BFGS method proved that this is competitive with the standard BFGS method, and it corrects large eigenvalues better than the standard BFGS method does. However, subject to other scaling BFGS updates, the scaling BFGS update by Liao is less efficient and less robust (Andrei, 2018d).

### Numerical Study: Scaling the Terms on the Right-Hand Side of the BFGS Update

In the following, let us consider the set of 80 unconstrained optimization test problems from the UOP collection (Andrei, 2020a) each of them with  $n = 100$  variables. The following algorithms were compared: BFGS defined by (6.22), BFGSB defined by (6.55) with (6.56), BFGSY defined by (6.55) with (6.57), BFGSC defined by (6.55) with (6.58), BFGSD defined by (6.55) with (6.59), BFGSE defined by (6.62) with (6.63) and (6.64), and BFGSFI defined by (6.62) with (6.65) and (6.66). For the BFGSB and BFGSY, the scaling parameter  $\gamma_k$  given by (6.56) and (6.57), respectively, is restricted in the interval [0.01, 100]. Besides, at the very first iteration of these methods, the scaling is not applied. The stepsize is computed by the Wolfe line-search conditions (6.5) and (6.6), where  $\sigma = 0.8$  and  $\rho = 0.0001$ . The iterations are stopped if the inequality  $\|g_k\|_\infty \leq 10^{-5}$  is satisfied, where  $\|\cdot\|_\infty$  is the maximum absolute component of a vector, or if the number of iterations exceeds 1000. Figure 6.2 presents the Dolan and Moré performance profiles of these algorithms for this set of unconstrained optimization test problems based on the CPU time metric.

From Fig. 6.2, we can see that BFGSFI is more efficient versus BFGS, BFGSB, and BFGSY and more robust versus the same algorithms. Scaling is an important ingredient in increasing the performances of BFGS.

### BFGS with Modified Secant Equation

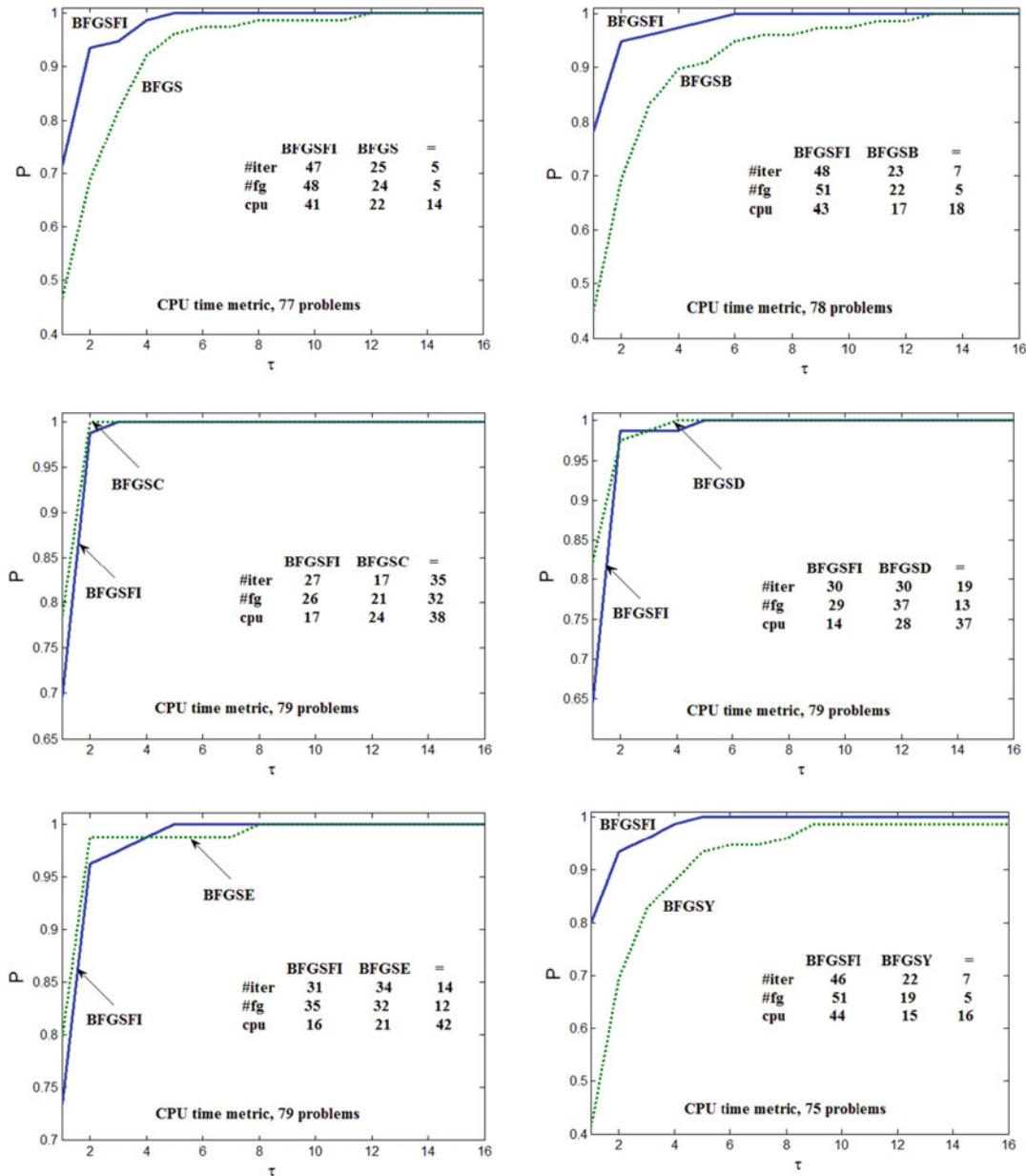
The standard secant equation (6.10) or its equivalent form (6.18) uses only the gradient information, without referring to the function values. Therefore, to obtain better approximations for the Hessian, the idea was to modify the secant equation in order to include more available information about the minimizing function  $f$ . Wei, Li, and Qi (2006a) and Wei, Yu, Yuan, and Lian (2004) proposed the following modified secant equation. Suppose that the objective function  $f$  is smooth enough. From the Taylor series, we obtain

$$f_k = f_{k+1} - s_k^T g_{k+1} + \frac{1}{2} s_k^T \nabla^2 f(x_{k+1}) s_k - \frac{1}{6} s_k^T (T_{k+1} s_k) s_k + O\left(\|s_k\|^4\right), \quad (6.69)$$

where

$$s_k^T (T_{k+1} s_k) s_k = \sum_{i,j,l}^n \frac{\partial^3 f(x_{k+1})}{\partial x^i \partial x^j \partial x^l} s_k^i s_k^j s_k^l. \quad (6.70)$$

After some simple algebraic manipulations, from (6.69), we get



**Fig. 6.2** Performance profiles of BFGSFI versus BFGS, BFGSB, BFGSC, BFGSD, BFGSE, and BFGSY. CPU time metric

$$s_k^T \nabla^2 f(x_{k+1}) s_k = s_k^T y_k + 2(f_k - f_{k+1}) + s_k^T (g_k + g_{k+1}) + \frac{1}{3} s_k^T (T_{k+1} s_k) s_k + O(\|s_k\|^4).$$

Therefore, neglecting the terms which include the tensor, the following approximation is obtained:

$$s_k^T \nabla^2 f(x_{k+1}) s_k = s_k^T y_k + \vartheta_k,$$

where

$$\vartheta_k = 2(f_k - f_{k+1}) + s_k^T(g_k + g_{k+1}). \quad (6.71)$$

Hence, the following *modified secant equation* is obtained:

$$B_{k+1}s_k = z_k, \quad z_k = y_k + \frac{\vartheta_k}{s_k^T u_k} u_k, \quad (6.72)$$

where  $u_k \in \mathbb{R}^n$  is a vector satisfying  $s_k^T u_k \neq 0$  (see: Yuan (1991), Yuan and Byrd (1995), Babaie-Kafaki (2011)).

Another modified secant equation was suggested by Zhang, Deng, and Chen (1999). From the Taylor series, we get

$$s_k^T g_k = s_k^T g_{k+1} - s_k^T \nabla^2 f(x_{k+1}) s_k + \frac{1}{2} s_k^T (T_{k+1} s_k) s_k + O(\|s_k\|^4). \quad (6.73)$$

Now, combining (6.69) and (6.73) and cancelling the terms including the tensor, it follows that

$$s_k^T \nabla^2 f(x_{k+1}) s_k = s_k^T y_k + 3\vartheta_k + O(\|s_k\|^4),$$

where  $\vartheta_k$  is defined in (6.71). Therefore, the following *modified secant equation* can be obtained:

$$B_{k+1}s_k = w_k, \quad w_k = y_k + \frac{3\vartheta_k}{s_k^T u_k} u_k, \quad (6.74)$$

where  $u_k \in \mathbb{R}^n$  is a vector satisfying  $s_k^T u_k \neq 0$ .

The theoretical advantages of the modified secant equations (6.72) and (6.74) can be seen from the following theorem, which shows their accuracy versus the standard secant equation (6.10) (see Zhang, Deng, and Chen (1999), Wei, Li, and Qi (2006a)).

**Theorem 6.10** *If the function  $f$  is sufficiently smooth and  $\|s_k\|$  is small enough, then the following estimating relations hold:*

$$\begin{aligned} s_k^T \nabla^2 f(x_{k+1}) s_k - s_k^T y_k &= \frac{1}{2} s_k^T (T_{k+1} s_k) s_k + O(\|s_k\|^3), \\ s_k^T \nabla^2 f(x_{k+1}) s_k - s_k^T z_k &= \frac{1}{3} s_k^T (T_{k+1} s_k) s_k + O(\|s_k\|^4), \\ s_k^T \nabla^2 f(x_{k+1}) s_k - s_k^T w_k &= O(\|s_k\|^4), \end{aligned}$$

where  $T_{k+1}$  is the tensor of  $f$  at  $x_{k+1}$  defined in (6.70). ◆

In this context, we can see that for quadratic objective functions,  $\vartheta_k = 0$ , and therefore, the modified secant equations (6.72) and (6.74) reduce to the standard secant equation. As regards the vector  $u_k$ , it can usually be selected as  $u_k = s_k$  or  $u_k = y_k$  provided that the line-search satisfies the Wolfe conditions (6.5) and (6.6). To get positive definite quasi-Newton approximations for the Hessian based on the modified secant equations (6.72) or (6.74), we should have  $s_k^T z_k > 0$  and  $s_k^T w_k > 0$ , respectively. To overcome this difficulty, a simple procedure is to replace  $\vartheta_k$  in (6.72) and (6.74) by  $\max\{0, \vartheta_k\}$ .

For nonconvex objective functions, Li and Fukushima (2001a, 2001b) proposed a new modified BFGS (called *cautious* BFGS), for which the local and the global superlinear convergence were proved. The method is based on the following modified secant equation:

$$B_{k+1}s_k = \bar{y}_k, \bar{y}_k = y_k + h_k \|g_k\|^r s_k, \quad (6.75)$$

where  $r$  is a positive constant and  $h_k$  is defined as

$$h_k = C + \max \left\{ 0, -\frac{y_k^T s_k}{\|s_k\|^2} \right\} \|g_k\|^{-r}$$

for a positive constant  $C$ . In the cautious BFGS method, the update of  $B_k$  is defined as

$$B_{k+1} = \begin{cases} B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, & \text{if } \frac{y_k^T s_k}{\|s_k\|^2} \geq \epsilon \|g_k\|^r, \\ B_k, & \text{otherwise,} \end{cases} \quad (6.76)$$

where  $\epsilon$  and  $r$  are positive constants. For the modified secant equation (6.75),  $\bar{y}_k^T s_k > 0$  independent of the line-search conditions and of the objective function convexity. This ensures the heredity of the positive definiteness of the corresponding BFGS update (Guo, Liu, & Wang, 2008).

Recently, Babaie-Kafaki (2012, 2013, 2014) and Babaie-Kafaki and Ghanbari (2014) proposed scaled memoryless BFGS methods with modified secant equations (6.72), (6.74), or (6.75) which satisfy the sufficient descent property  $d_k^T g_k \leq -c \|g_k\|^2$ , where  $c$  is a positive constant. A new approach using the polynomial interpolation of the data from the most recent  $m$  steps in modified secant equations was developed by Ford and Moghrabi (1994, 1996a, 1996b) and by Ford, Narushima, and Yabe (2008).

### BFGS with Modified Line-Search

Usually, the BFGS method is implemented by using the Wolfe line-search (6.5) and (6.6). This is important since it ensures that  $y_k^T s_k > 0$  for any  $k = 1, \dots$ , thus conserving the positive definiteness of the BFGS approximations along the iterations. Suppose that the gradient of the minimizing function is Lipschitz continuous with the constant  $L$ . Let  $L_k$  be an approximation of  $L$ . Wan, Huang, and Zheng (2012) proposed the cautious BFGS method (6.76) with the following modification of the Armijo line-search procedure. Set  $\beta_k = -g_k^T d_k / (L_k \|d_k\|^2)$ . “Find the stepsize  $\alpha_k$  as the largest component in the set  $\{\beta_k, \beta_k \rho, \beta_k \rho^2, \dots\}$  so that the inequality

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \sigma \alpha_k \left( g_k^T d_k - \frac{1}{2} \alpha_k \mu L_k \|d_k\|^2 \right) \quad (6.77)$$

holds, where  $\sigma \in (0, 1)$ ,  $\mu \in [0, \infty)$  and  $\rho \in (0, 1)$  are given constants.” Under classical assumptions, if  $\|B_k s_k\| \leq a_1 \|s_k\|$  and  $a_2 \|s_k\|^2 \leq s_k^T B_k s_k$ , then  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ , where  $a_1$  and  $a_2$  are positive constants.

Another interesting modification of the strong Wolfe line-search (6.12) was given by Wan, Teo, Shen, and Hu (2014). In the Wolfe line-search, the choice of  $\sigma$  can affect the cost of finding a stepsize satisfying (6.5) and (6.6) or (6.5) and (6.12). For a larger value of  $\sigma$ , the cost of searching a stepsize decreases. Therefore, it seems reasonable to select a sufficiently large value for  $\sigma$ . On the other hand, for a large value for  $\sigma$ , the obtained stepsize might be far away from the optimal one, i.e., the one obtained by the exact line-search. With the value of  $s_k^T y_k$  far away from  $-s_k^T g_k$ , it is clear that  $g(x_k + \alpha_k d_k)^T d_k$  is far away from zero. In other words, a large value for  $\sigma$  may give rise to a stepsize which is not a good approximation to the one obtained by the exact line-search. To overcome this difficulty, Wan, Teo, Shen, and Hu (2014) replaced the strong Wolfe condition (6.12) by

$$-(1 + \sigma^U)s_k^T g_k \geq s_k^T y_k \geq -(1 - \sigma^L)s_k^T g_k, \quad (6.78)$$

where  $\sigma^L$  and  $\sigma^U$  are two sufficiently small constants satisfying  $0 \leq \sigma^L \leq \sigma < 1$  and  $0 \leq \sigma^U \leq \sigma$  (see also Al-Baali and Grandinetti, (2009)). As above, it is proved that if  $\|B_k s_k\| \leq b_1 \|s_k\|$  and  $b_2 \|s_k\|^2 \leq s_k^T B_k s_k$ , where  $B_k$  is the BFGS update, then  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ , where  $b_1$  and  $b_2$  are positive constants. Numerical experiments reported by the authors show that this variant of the BFGS with modified line-search (6.78) is competitive versus the standard BFGS or the cautious BFGS (6.76).

Dai (2003) presented a numerical example showing that the standard BFGS method fails in the case of nonconvex functions under the Wolfe line-search. Therefore, the question is “*are there any other inexact line-searches that possess the global convergence of the BFGS method for general functions?*” A positive answer was given by Yuan, Wei, and Lu, (2017) and Yuan, Sheng, Wang, Hu, and Li (2018). They presented the following modified Wolfe line-search:

$$f(x_k + \alpha_k d_k) \leq f_k + \rho \alpha_k g_k^T d_k + \alpha_k \min \left\{ -\rho_1 g_k^T d_k, \rho \frac{\alpha_k}{2} \|d_k\|^2 \right\}, \quad (6.79)$$

$$g(x_k + \alpha_k d_k)^T d_k \geq \sigma g_k^T d_k + \min \left\{ -\rho_1 g_k^T d_k, \rho \alpha_k \|d_k\|^2 \right\}, \quad (6.80)$$

where  $\rho \in (0, 1/2)$ ,  $\rho_1 \in (\rho/2, \rho)$ , and  $\sigma \in (\rho, 1)$ . Under classical assumptions, if  $\|B_k s_k\| \leq b_1 \|s_k\|$  and  $b_2 \|s_k\|^2 \leq s_k^T B_k s_k$ , where  $B_k$  is the BFGS update (6.22), then, for the corresponding algorithm with modified line-search given by (6.79) and (6.80),  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ , where  $b_1$  and  $b_2$  are positive constants.

An improvement of the global convergence of the BFGS method with Yuan-Wei-Lu line-search (6.79) and (6.80) was presented by Dehmiry (2019)

$$f(x_k + \alpha_k d_k) \leq f_k + \rho \alpha_k g_k^T d_k - \rho \frac{\alpha_k^2}{2\beta_k} \|d_k\|^2, \quad (6.81)$$

$$g(x_k + \alpha_k d_k)^T d_k \geq \sigma g_k^T d_k - \frac{\rho \alpha_k}{\beta_k} \|d_k\|^2, \quad (6.82)$$

$$\alpha_k < -\frac{\beta_k(1 - \sigma)}{\varepsilon_0 \beta_k + \rho} \frac{g_k^T d_k}{\|d_k\|^2}, \quad (6.83)$$

where  $\rho \in (0, 1/2)$ ,  $\sigma \in (\rho, 1)$ ,  $\varepsilon_0$  is a small parameter ( $\varepsilon_0 = 10^{-6}$ ) and  $\{\beta_k\}$  is an arbitrary increasing sequence of positive numbers so that  $\lim_{k \rightarrow \infty} \beta_k/k = +\infty$ . Like in Byrd and Nocedal (1989), Dhemiriy (2019) proved that there exist the constants  $b_1 > b_2 > 0$  such that  $\|B_k s_k\| \leq b_1 \|s_k\|$  and  $b_2 \|s_k\|^2 \leq s_k^T B_k s_k$  for at least  $t/2$  values of  $k \in \{1, \dots, t\}$  with any positive integer  $t$ . Therefore, the sequence  $\{g_k\}$  generated by the corresponding algorithm with the modified line-search given by (6.81), (6.82), and (6.83) satisfies  $\liminf_{k \rightarrow \infty} \|g_k\| = 0$ . Numerical experiments with the BFGS algorithm, where the stepsize is determined by the modified Yuan-Wei-Lu line-search (6.81), (6.82), and (6.83), show that it is more efficient and more robust versus the algorithm with Yuan-Wei-Lu line-search (6.79) and (6.80) and versus the Li and Fukushima (2001b) algorithm.

### 6.3 Quasi-Newton Methods with Diagonal Updating of the Hessian

A relative recent idea to generate simple minimization algorithms for the unconstrained optimization in the frame of the quasi-Newton methods is to approximate the Hessian of the minimizing function by a diagonal matrix with positive diagonal elements. This approach is motivated by Theorem 4.16

which shows that, for the Newton method, in the norm of the error, the inaccuracy in the Hessian has a smaller influence than the inaccuracy of the gradient. These methods were introduced by Gill and Murray (1979) and discussed by Gilbert and Lemaréchal (1989). The search direction is computed as

$$d_{k+1} = -B_{k+1}^{-1}g_{k+1}, \quad (6.84)$$

where  $B_{k+1} = \text{diag}(b_{k+1}^1, \dots, b_{k+1}^n)$  is a positive definite diagonal matrix retaining only the diagonal elements of the BFGS update matrix (6.22)

$$b_{k+1}^i = b_k^i - \frac{(b_k^i)^2 (s_k^i)^2}{\sum_{i=1}^n b_k^i (s_k^i)^2} + \frac{(y_k^i)^2}{y_k^T s_k}, \quad i = 1, \dots, n. \quad (6.85)$$

This diagonal-updating approach uses only  $O(n)$  storage. If we assume that  $y_k^T s_k > 0$ , then  $B_{k+1}$  is positive definite.

Another quasi-Newton algorithm with diagonal approximation to the Hessian is based on the weak quasi-Newton equation  $s_k^T B_{k+1} s_k = s_k^T y_k$  and was introduced and studied by Dennis and Wolkowicz (1993). The update proposed by Dennis and Wolkowicz is as follows:

$$B_{k+1} = B_k + \frac{s_k^T y_k - s_k^T B_k s_k}{(s_k^T B_k s_k)^2} B_k s_k s_k^T B_k, \quad (6.86)$$

where  $B_k$  is positive definite. The condition  $s_k^T y_k > 0$  implies that  $B_{k+1}$  in (6.86) is also positive definite. If  $B_k$  is taken to be a positive definite diagonal matrix  $B_k = \text{diag}(b_k^1, \dots, b_k^n)$ , then (6.86) can be restricted to update only the diagonal elements of  $B_{k+1}$  as

$$b_{k+1}^i = b_k^i + \frac{s_k^T y_k - \sum_{i=1}^n b_k^i (s_k^i)^2}{\left(\sum_{i=1}^n b_k^i (s_k^i)^2\right)^2} (b_k^i)^2 (s_k^i)^2, \quad i = 1, \dots, n, \quad (6.87)$$

yielding a positive definite diagonal matrix. The search direction in this algorithm is computed as in (6.84), where the diagonal elements of  $B_{k+1}$  are computed as in (6.87).

The diagonal quasi-Newton approximation algorithm presented by Zhu, Nazareth, and Wolkowicz (1999) is as follows. Suppose that  $B_k$  is a positive definite diagonal matrix and  $B_{k+1}$ , which is also diagonal, is the updated version of  $B_k$ . The algorithm requires that the updated  $B_{k+1}$  satisfy the quasi-Newton secant equation and the deviation between  $B_k$  and  $B_{k+1}$  be minimized under a variational principle. The search direction is computed as  $d_{k+1} = -B_{k+1}^{-1}g_{k+1}$ , where the diagonal elements of  $B_{k+1} = \text{diag}(b_{k+1}^1, \dots, b_{k+1}^n)$  are computed as

$$b_{k+1}^i = b_k^i + \frac{s_k^T y_k - \sum_{i=1}^n b_k^i (s_k^i)^2}{\sum_{i=1}^n (s_k^i)^4} (s_k^i)^2, \quad i = 1, \dots, n. \quad (6.88)$$

In the same way of developments, Andrei (2019a) presented a new diagonal quasi-Newton updating method, where the elements of the diagonal matrix approximating the Hessian are determined by minimizing both the size of the change from the previous estimate and the trace of the update, with respect to the weak secant equation. The search direction in this algorithm is computed as

$$d_{k+1}^i = -\frac{g_{k+1}^i}{b_{k+1}^i}, \quad i = 1, \dots, n, \quad (6.89)$$

and

$$b_{k+1}^i = b_k^i + \frac{s_k^T y_k + s_k^T s_k - \sum_{i=1}^n b_k^i (s_k^i)^2}{\sum_{i=1}^n (s_k^i)^4} (s_k^i)^2 - 1, \quad i = 1, \dots, n. \quad (6.90)$$

Another approach that uses the same paradigm of the diagonal quasi-Newton updating method was given by Andrei (2018e), where the diagonal elements are determined by minimizing the measure function of Byrd and Nocedal. The components of the search direction in this approach are computed as

$$d_{k+1}^i = -g_{k+1}^i \left( 1 + \lambda (s_k^i)^2 \right), \quad i = 1, \dots, n, \quad (6.91)$$

where

$$\lambda = \begin{cases} r + \theta, & \text{if } \bar{\lambda} < r, \\ \bar{\lambda} & \text{if } \bar{\lambda} \geq r, \end{cases} \quad \text{and} \quad \bar{\lambda} = \frac{t(s_k^T g_{k+1}) - y_k^T g_{k+1}}{\sum_{i=1}^n (y_k^i g_{k+1}^i (s_k^i)^2)},$$

and

$$r = -1 / (s_k^i)^2 = \max_{i=1, \dots, n; s_k^i \neq 0} \left\{ -1 / (s_k^i)^2 \right\}.$$

$t$  is a positive parameter and  $\theta$  is a small perturbation (e.g.,  $\theta = 1$ ).

Andrei (2019b) has recently presented a new quasi-Newton method, in which the Hessian of the function  $f$  is approximated as a positive definite diagonal matrix. In this method, the approximation Hessian  $B_{k+1}$  is a diagonal matrix computed as

$$B_{k+1} = Y_k S_k^{-1}, \quad (6.92)$$

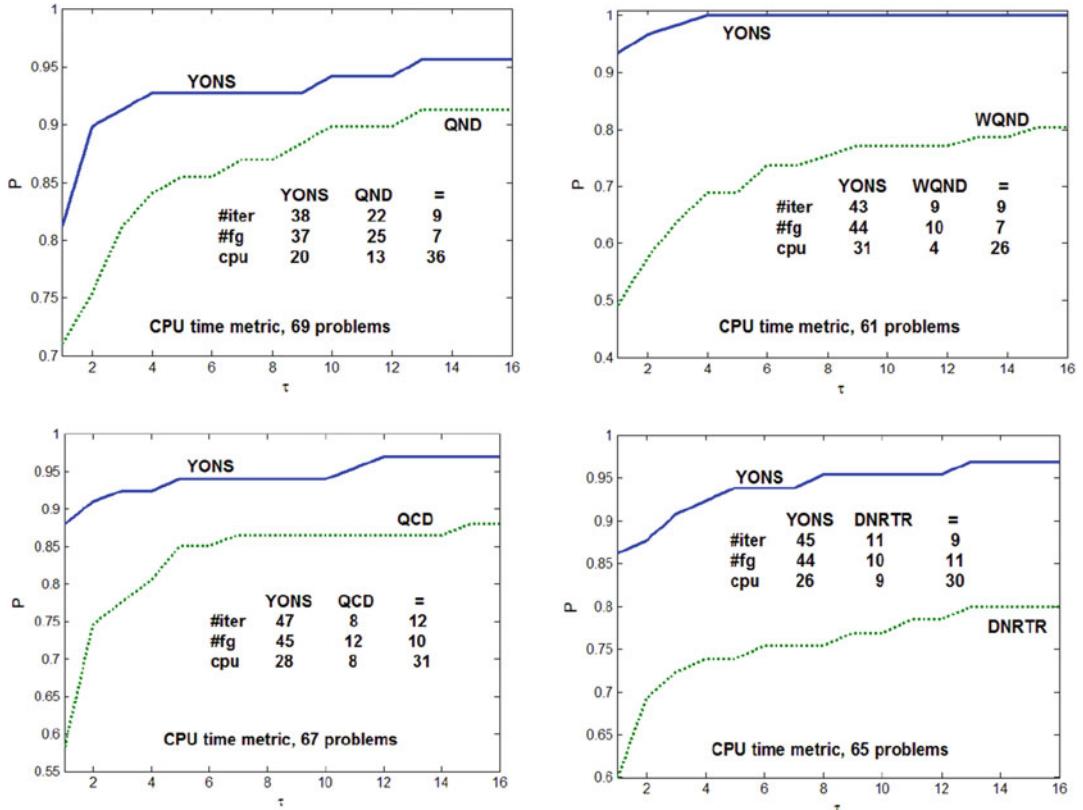
where  $Y_k = \text{diag}(y_k^1, \dots, y_k^n)$  and  $S_k = \text{diag}(s_k^1, \dots, s_k^n)$ ,  $y_k^i$ ,  $i = 1, \dots, n$  being the components of the vector  $y_k$  and  $s_k^i$ ,  $i = 1, \dots, n$  being the components of the vector  $s_k$ . In other words,

$$b_{k+1}^i = \frac{y_k^i}{s_k^i} = \frac{g_{k+1}^i - g_k^i}{x_{k+1}^i - x_k^i} = \frac{g^i(x_k + \alpha_k d_k) - g^i(x_k)}{\alpha_k d_k^i}, \quad i = 1, \dots, n, \quad (6.93)$$

where  $g_k^i$  is the  $i$ -th component of the gradient in  $x_k$  and  $d_k^i$  is the  $i$ -th component of the search direction  $d_k$ . Therefore, in this approach, the element  $b_{k+1}^i$  may be considered as an approximation of the second-order derivative of the function  $f$ , corresponding to the  $i$ -th diagonal element of the Hessian computed in  $x_{k+1}$  by a *scaled forward finite differences directional derivative scheme*.

## Numerical Study

Consider the set of 80 unconstrained optimization test functions from the UOP collection (Andrei, 2020a). Figure 6.3 presents the Dolan and Moré performance profiles of YONS defined by (6.92) with (6.93) versus QND defined by (6.84) with (6.85), WQND defined by (6.84) with (6.87), QCD defined by (6.84) with (6.88), and DNRTR defined by (6.84) with (6.90), for minimizing this set of unconstrained optimization test problems with  $n = 500$  variables, subject to the CPU time metric.



**Fig. 6.3** Performance profiles of YONS versus QND, WQND, QCD, and DNRTR. CPU time metric,  $n = 500$

The stepsize is determined by the Wolfe line-search conditions (6.5) and (6.6), implemented with  $\sigma = 0.8$  and  $\rho = 0.0001$ . The iterations are stopped if the inequality  $\|g_k\|_\infty \leq \varepsilon_g$  is satisfied, where  $\|\cdot\|_\infty$  is the maximum absolute component of a vector, or if the number of iterations exceeds a prespecified limit (50000).

Observe that YONS is significantly more efficient and more robust versus all the algorithms considered in this numerical study. More numerical results with quasi-Newton methods with diagonal updating to the Hessian, presented in Andrei (2018e, 2019a, 2019b, 2020b), proved that the algorithm in which the diagonal elements of the approximation to the Hessian are computed by a scaled forward finite differences directional derivative scheme is competitive among the algorithms in this class.

## 6.4 Limited-Memory Quasi-Newton Methods

Using (6.15) in (6.20), the inverse BFGS updating can be written as

$$H_{k+1} = \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) H_k \left( I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}.$$

Therefore, the corresponding search direction based on this inverse BFGS formula is

$$\begin{aligned} d_{k+1} &= -H_{k+1}\nabla f(x_{k+1}) \\ &= -\left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) \nabla f(x_{k+1}) - \frac{s_k s_k^T}{y_k^T s_k} \nabla f(x_{k+1}). \end{aligned}$$

Note that this formula requires no matrix storage except for storage of  $H_k$  and, therefore, uses only vector operations. This is *one version* of a limited-memory BFGS method. However, more elaborate formulae can be developed as it is going to be presented in the following.

The limited-memory quasi-Newton methods are dedicated to solving large-scale unconstrained optimization problems whose Hessian matrix cannot be computed and stored at a reasonable cost (Nocedal, 1980). Instead of storing an  $n \times n$  approximation to the Hessian, these methods save only a few vectors which can be used to represent the approximation implicitly. Among different limited-memory methods, the best known and used is L-BFGS, which is based on the BFGS update. The main idea of this method is to use the curvature information only from the most recent iterations in order to construct the Hessian approximation.

As known, each step of the BFGS method is computed as

$$x_{k+1} = x_k - \alpha_k H_k g_k, \quad k = 0, 1, \dots, \quad (6.94)$$

where  $\alpha_k$  is the stepsize,  $g_k = \nabla f(x_k)$ , and  $H_k$  is the approximation to the inverse Hessian updated by the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \quad (6.95)$$

where

$$\rho_k = \frac{1}{y_k^T s_k}, \quad V_k = I - \rho_k y_k s_k^T. \quad (6.96)$$

Since the inverse Hessian  $H_k$  is generally a dense matrix, the limited-memory BFGS method will implicitly store a modified version of  $H_k$  by storing only a certain number (say  $m$ ) of the vector pairs  $\{s_i, y_i\}$  used in the updating formulae (6.95) and (6.96). After the new iterate has been computed, the oldest vector pair in the set of pairs  $\{s_i, y_i\}$  is replaced by the new pair  $\{s_k, y_k\}$  obtained from the current iteration. Therefore, the set of vector pairs includes the curvature information from the most recent  $m$  iterations. It is worth mentioning that only a small number of vector pairs, between 3 and 11, needs to be used for solving large-scale optimization problems. The product  $H_k g_k$  is computed during the updating process by performing a sequence of inner products and vector summations involving only  $g_k$  and the pairs  $\{s_i, y_i\}$ . Nocedal and Wright (2006) presented a two-loop recursion L-BFGS algorithm to compute the product  $H_k g_k$  in (6.94). At iteration  $k$ , the current iterate is  $x_k$ , and the set of vector pairs is given by  $\{s_i, y_i\}$  for  $i = k-m, \dots, k-1$ . Choose some initial Hessian approximation  $H_k^0$ , and by repeated application of (6.95), find that the L-BFGS approximation  $H_k$  satisfies the following formula:

$$\begin{aligned} H_k &= (V_{k-1}^T \cdots V_{k-m}^T) H_0 (V_{k-m} \cdots V_{k-1}) \\ &\quad + \rho_{k-m} (V_{k-1}^T \cdots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\ &\quad + \rho_{k-m+1} (V_{k-1}^T \cdots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) \\ &\quad + \cdots \\ &\quad + \rho_{k-1} s_{k-1} s_{k-1}^T. \end{aligned}$$

This expression of  $H_k$  may be used to derive a recursive procedure for computing the product  $H_k \nabla f(x_k)$ . Given  $H_k^0$ , the two-loop recursion algorithm for the search direction computation is as follows (Liu, & Nocedal, 1989).

**Algorithm 6.2** *L-BFGS two-loop recursion*

---

Set  $q = g_k$ . The *first loop* is:

For  $i = k - 1, k - 2, \dots, k - m$  compute:

$$\alpha_i = \rho_i s_i^T q, \quad q = q - \alpha_i y_i$$

End for

$$\text{Set } r = H_k^0 q.$$

The *second loop* is:

For  $i = k - m, k - m + 1, \dots, k - 1$  compute:

$$\beta = \rho_i y_i^T r, \quad r = r + s_i(\alpha_i - \beta)$$

End for

$$\text{Set } H_k g_k = r$$

◆

Usually,  $H_k^0$  is chosen as a diagonal matrix. Often,  $H_k^0 = \gamma_k I$ , where  $\gamma_k = (s_{k-1}^T y_{k-1}) / \|y_{k-1}\|^2$ . The parameter  $\gamma_k$  is a scaling factor that attempts to estimate the size of the true Hessian along the most recent search direction. This selection of  $\gamma_k$  ensures that the search direction is well scaled, and therefore, as a by-product, the stepsize  $\alpha_k = 1$  is accepted in most iterations (Gilbert & Lemaréchal, 1989). In the implementation of Liu and Nocedal (1989), L-BFGS is one of the best algorithms (and computing programs) for minimizing large-scale unconstrained optimization functions. Now, the limited-memory BFGS algorithm can be presented as follows.

**Algorithm 6.3** *L-BFGS*

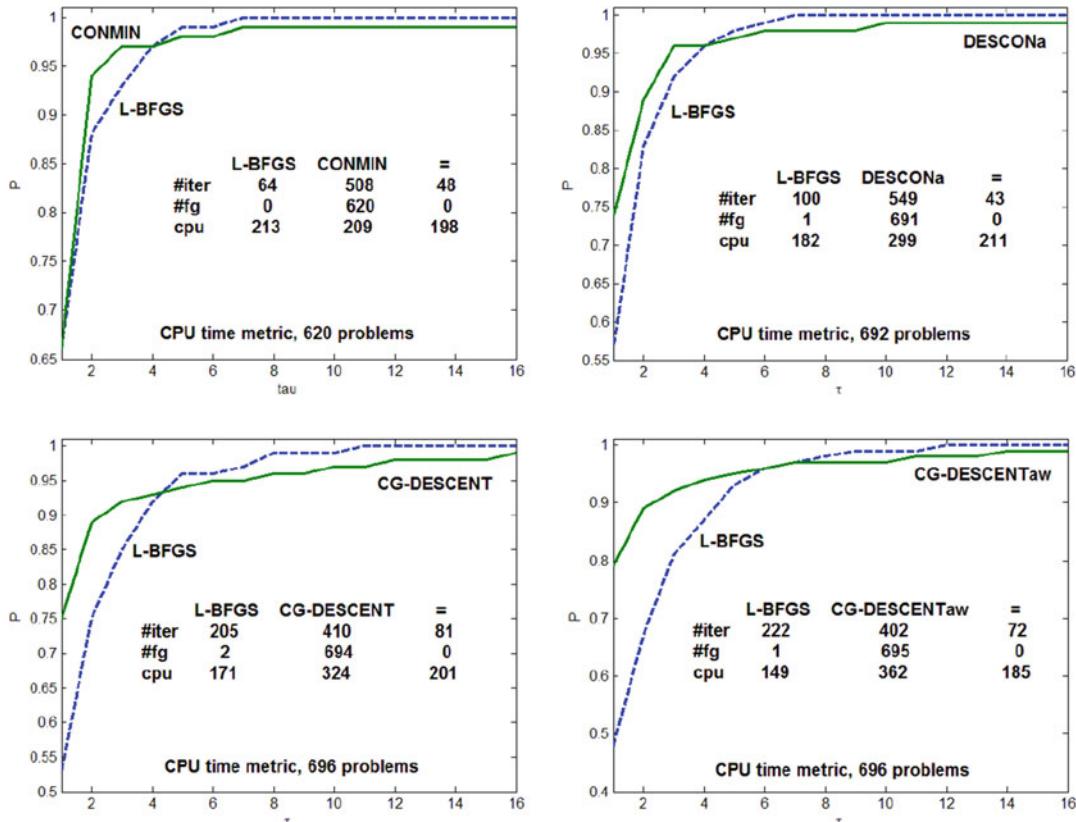
- |    |  |
|----|--|
| 1. | Initialization. Choose the initial point $x_0$ , the convergence tolerance $\varepsilon > 0$ sufficiently small, and a value for integer $m$ , and set $k = 0$ |
| 2. | Test a criterion for stopping the iterations. For example, if $\ \nabla f(x_k)\  \leq \varepsilon$ , then stop; otherwise, set $k = k + 1$ and go to step 3    |
| 3. | Choose $H_k^0$ . For example, $H_k^0 = \gamma_k I$ , where $\gamma_k = (s_{k-1}^T y_{k-1}) / \ y_{k-1}\ ^2$  |
| 4. | Use Algorithm 6.2 (two-loop recursion) to compute $d_k = -H_k \nabla f(x_k)$   |
| 5. | Compute the stepsize $\alpha_k$ by using the Wolfe line-search   |
| 6. | Compute $x_{k+1} = x_k + \alpha_k d_k$   |
| 7. | If $k > m$ , then discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from the storage  |
| 8. | Compute and save $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$   |
| 9. | Set $k = k + 1$ and go to step 2   |

◆

During the first  $m - 1$  iterations, if the initial matrix  $H_0$  is the same in both algorithms and if L-BFGS chooses  $H_k^0 = H_0$  at each iteration, then Algorithm 6.3 is equivalent to Algorithm 6.1 (BFGS method). Observe that the performance of Algorithm 6.3 is subject to the level of memory  $m$ . Table 6.1 shows the performances of L-BFGS (Liu & Nocedal, 1989) for solving five unconstrained optimization problems, where  $\varepsilon = 10^{-6}$ . The steplength is determined at each iteration by means of the line-search routine MCVSRCH, which is a slight modification of the routine CSRCH written by Moré and Thuente (1990, 1992, 1994).

**Table 6.1** Performances of L-BFGS for different values of  $m$ 

Problem	$n$	$m=3$			$m=5$			$m=17$		
		#iter	#fg	cpu	#iter	#fg	cpu	#iter	#fg	cpu
FreuRoth	10,000	16	18	0.00	17	18	0.01	19	20	0.01
Rosenbrock	10,000	68	93	0.01	67	93	0.01	69	92	0.02
BDQRTIC	10,000	191	252	0.03	168	223	0.04	61	90	0.02
NONDQUAR	10,000	1805	2001	0.29	1796	2001	0.33	1796	2001	0.89
DIXON3DQ	10,000	1892	2001	0.26	1938	2001	0.30	1948	2001	0.94

**Fig. 6.4** Performance profiles of L-BFGS versus CONMIN, DESCONa, CG-DESCENT, and CG-DESCENTaw

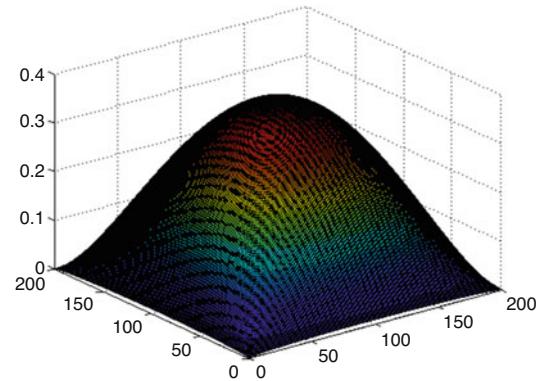
From Table 6.1, we can see that the optimal choice of  $m$  is problem dependent. For large values of  $m$ , the cost of each iteration increases, and therefore, L-BFGS requires more CPU computing time. The best CPU time is obtained for small values of  $m$ .

### Numerical Study: L-BFGS

Figure 6.4 shows the performance profiles of L-BFGS versus CONMIN, DESCONa (accelerated DESCON), CG-DESCENT (CG-DESCENT with Wolfe line-search), and CG-DESCENTaw (CG-DESCENT with approximate Wolfe line-search) for solving 800 unconstrained optimization problems from the UOP collection with the number of variables in the range [1000, 10000].

**Table 6.2** Performances of L-BFGS Elastic-plastic torsion.  $n = 40,000$ .  $\varepsilon = 10^{-6}$ 

	#iter	#fg	cpu
L-BFGS (m=3)	322	338	20.90
L-BFGS (m=5)	328	677	21.51
L-BFGS (m=7)	340	1027	22.96
L-BFGS (m=9)	318	1353	22.19

**Fig. 6.5** Solution of application A1. Elastic-plastic torsion.  $nx = 200$ ,  $ny = 200$ **Table 6.3** Performances of L-BFGS Pressure distribution in a journal bearing.  $n = 40,000$ .  $\varepsilon = 10^{-6}$ 

	#iter	#fg	cpu
L-BFGS (m=3)	898	935	58.47
L-BFGS (m=5)	788	1750	52.15
L-BFGS (m=7)	813	2584	55.33
L-BFGS (m=9)	782	3386	54.94

Observe that L-BFGS is more robust than all these conjugate gradient algorithms. However, DESCN, CG-DESCENT, and CG-DESCENTaw are more efficient than L-BFGS.

### Applications from the MINPACK-2 Collection

In the following, let us see the performances of L-BFGS in the implementation of Liu and Nocedal (1989) for solving the applications from the MINPACK-2 collection (see Appendix D) for different values of  $m$ .

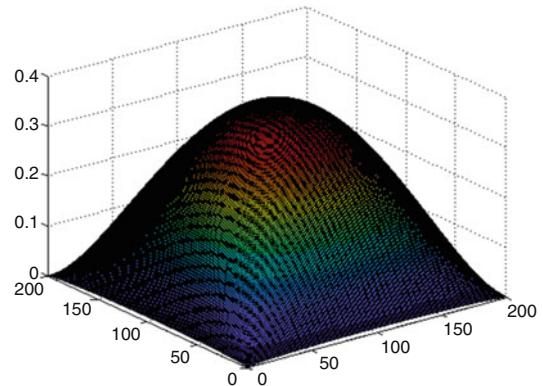
#### Application A1. Elastic-Plastic Torsion

For  $nx = 200$ ,  $ny = 200$ , and  $c = 5$ , for different values of  $m$ , the performances of L-BFGS for solving elastic-plastic torsion are as in Table 6.2, where #iter is the number of iterations, #fg is the number of function and its gradient evaluations, and cpu is the CPU computing time in seconds. Figure 6.5 shows the solution of this application for  $m = 3$ .

#### Application A2. Pressure Distribution in a Journal Bearing

For  $nx = 200$ ,  $ny = 200$ ,  $b = 10$ , and  $\varepsilon = 0.1$ , Table 6.3 shows the performances of L-BFGS for solving this application.

**Fig. 6.6** Solution of application A2. Pressure distribution in a journal bearing.  $nx = 200$ ,  $ny = 200$



**Table 6.4** Performance of L-BFGS. Optimal design with composite materials.  $n = 40,000$ ,  $\varepsilon = 10^{-6}$

	#iter	#fg	cpu
L-BFGS (m=3)	1468	1482	121.10
L-BFGS (m=5)	856	2343	71.83
L-BFGS (m=7)	864	3210	74.30
L-BFGS (m=9)	659	3871	57.93

**Fig. 6.7** Solution of application A3. Optimal design with composite materials.  $nx = 200$ ,  $ny = 200$

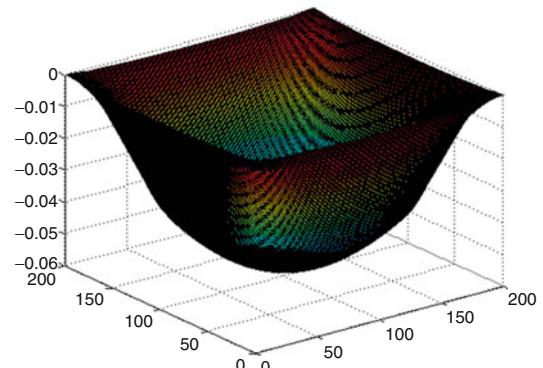


Figure 6.6 illustrates the solution of this application for  $nx = 200$ ,  $ny = 200$ , and  $m = 3$ .

#### Aplication A3. Optimal Design with Composite Materials

For  $nx = 200$  and  $ny = 200$ , the performances of L-BFGS for solving optimal design with composite materials with parameters  $\lambda = 0.008$ ,  $\mu_1 = 1$ , and  $\mu_2 = 2$  are presented in Table 6.4.

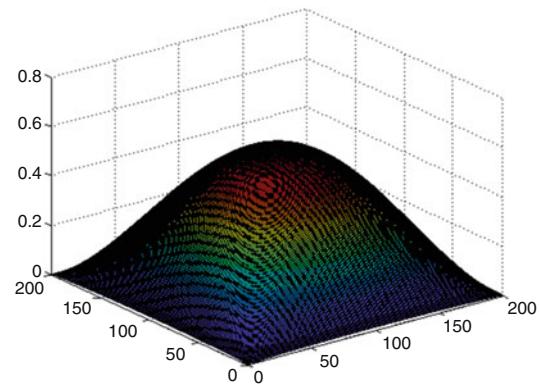
Figure 6.7 shows the solution of this application for  $nx = 200$ ,  $ny = 200$ , and  $m = 3$ .

#### Application A4. Steady-State Combustion

For  $nx = 200$  and  $ny = 200$ , the performances of L-BFGS for solving steady-state combustion with the Frank-Kamenetskii parameter  $\lambda = 5$  are as in Table 6.5.

**Table 6.5** Performances of L-BFGS. Steady-state combustion.  $n = 40,000$ .  $\varepsilon = 10^{-6}$ 

	#iter	#fg	cpu
L-BFGS (m=3)	721	763	80.08
L-BFGS (m=5)	523	1308	58.06
L-BFGS (m=7)	524	1846	58.61
L-BFGS (m=9)	412	2272	47.10

**Fig. 6.8** Solution of application A4. Steady-state combustion.  $nx = 200$ ,  $ny = 200$ **Table 6.6** Performances of L-BFGS. Minimal surfaces with Enneper boundary conditions.  $n = 40,000$ .  $\varepsilon = 10^{-6}$ 

	#iter	#fg	cpu
L-BFGS (m=3)	484	507	37.77
L-BFGS (m=5)	459	977	35.93
L-BFGS (m=7)	424	1408	33.94
L-BFGS (m=9)	427	1844	35.10

Figure 6.8 illustrates the solution of this application for  $nx = 200$ ,  $ny = 200$ , and  $m = 3$ .

#### Application A5. Minimal Surfaces with Enneper Boundary Conditions

For  $nx = 200$  and  $ny = 200$ , the performances of L-BFGS for minimal surface with Enneper boundary conditions are as in Table 6.6. The solution of this application is illustrated in Fig. 6.9.

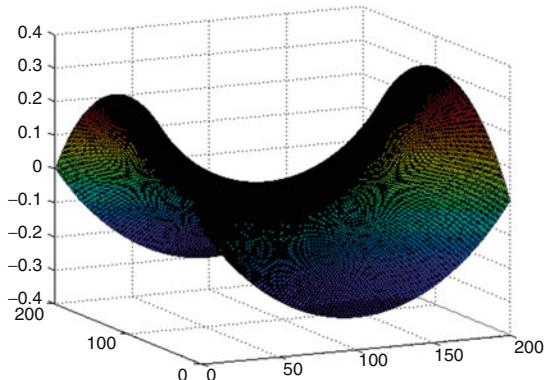
#### Application A6. Inhomogeneous Superconductors: 1-D Ginzburg-Landau

For  $d = 3.2 \text{ \AA}$  and the temperature  $T = 5$ , for  $n = 1000$ , the performances of L-BFGS for solving 1-D Ginzburg-Landau application are as in Table 6.7. The solution of this application is presented in Fig. 6.10.

#### Compact Representation of the L-BFGS Updating

Limited-memory quasi-Newton approximations are an important component of the unconstrained or constrained optimization methods. L-BFGS is a line-search algorithm that updates an approximation to the inverse Hessian  $H_k$ . On the other hand, the trust-region methods require an approximation to the

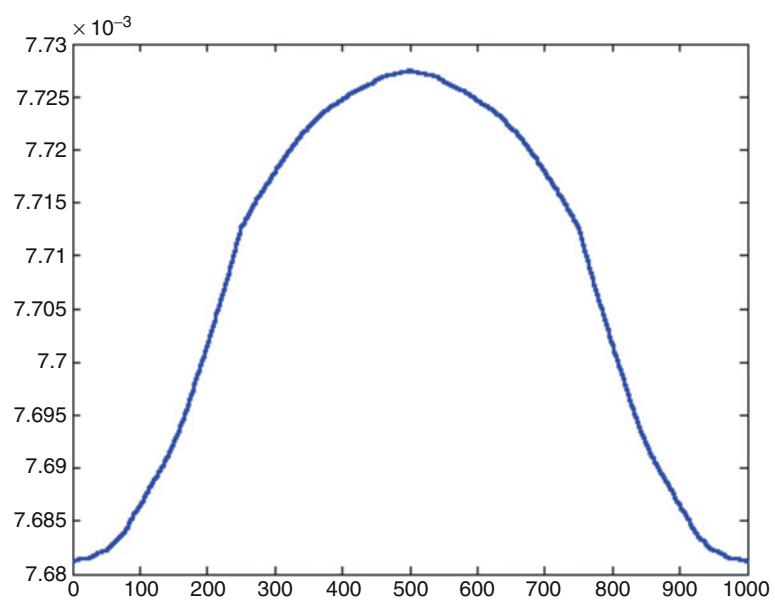
**Fig. 6.9** Solution of application A5. Minimal surfaces with Enneper boundary conditions.  
 $nx = 200, ny = 200$



**Table 6.7** Performances of L-BFGS. Inhomogeneous superconductors: 1-D Ginzburg-Landau.  $n = 1000$ .  $\varepsilon = 10^{-6}$

	#iter	#fg	cpu
L-BFGS (m=3)	1904	2001	9.45

**Fig. 6.10** Solution of application A6. Inhomogeneous superconductors: 1-D Ginzburg-Landau.  
 $n = 1000$



Hessian matrix  $B_k$ , not to its inverse. In the following, we show that, by representing quasi-Newton matrices in a compact form, an efficient implementation of all quasi-Newton formulae and their inverse can be derived. These compact representations are useful in designing limited-memory methods for the constrained optimization, where the approximations to the Hessian or to the reduced Hessian of the Lagrangian are needed.

In the following, let us describe the limited-memory updating that is based on representing quasi-Newton matrices in compact form. This is illustrated for the case of the BFGS approximation  $B_k$  to the Hessian (Byrd, Nocedal, & Schnabel, 1994b).

**Theorem 6.11** Consider  $B_0$  a symmetric and positive definite matrix, and assume that the  $k$  vector pairs  $\{s_i, y_i\}$  satisfy  $s_i^T y_i > 0$ , for all  $i = 0, \dots, k - 1$ . Using (6.22), let  $B_k$  be obtained by applying  $k$  BFGS updates with these vector pairs to  $B_0$ . Then,

$$B_k = B_0 - [B_0 S_k \quad Y_k] \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix}, \quad (6.97)$$

where

$$S_k = [s_0, \dots, s_{k-1}] \in \mathbb{R}^{n \times k}, Y_k = [y_0, \dots, y_{k-1}] \in \mathbb{R}^{n \times k}, \quad (6.98)$$

while  $L_k \in \mathbb{R}^{k \times k}$  and  $D_k \in \mathbb{R}^{k \times k}$  are defined by

$$(L_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & \text{if } i > j, \\ 0, & \text{otherwise,} \end{cases} \quad D_k = \text{diag}[s_0^T y_0, \dots, s_{k-1}^T y_{k-1}]. \quad (6.99)$$

The proof is obtained by induction.  $\blacklozenge$

Observe that the conditions  $s_i^T y_i > 0$ ,  $i = 0, \dots, k - 1$ , ensure that the middle matrix in (6.97) is nonsingular, so  $B_k$  is well-defined. Now, let us consider this result in the limited-memory updating.

As we know, the L-BFGS algorithm keeps the  $m$  most recent pairs  $\{s_i, y_i\}$ , and at every iteration, this set of vectors is refreshed by removing the oldest pair and by adding a newly generated pair. Obviously, during the first  $m$  iterations, the update procedure described in Theorem 6.11 can be used without any modification, except that  $B_0^k = \delta_k I$  for the initial matrix, where  $\delta_k = 1/\gamma_k$  and  $\gamma_k = (s_{k-1}^T y_{k-1})/(y_{k-1}^T y_{k-1})$ .

For the iterations  $k > m$ , the update procedure has to be slightly modified in order to take into consideration the vector pairs  $\{s_i, y_i\}$  for  $i = k - m, k - m + 1, \dots, k - 1$ . Define

$$S_k = [s_{k-m}, \dots, s_{k-1}] \in \mathbb{R}^{n \times m}, Y_k = [y_{k-m}, \dots, y_{k-1}] \in \mathbb{R}^{n \times m}. \quad (6.100)$$

Therefore, the matrix  $B_k$  obtained after  $m$  updates of  $B_0^k = \delta_k I$  is given by

$$B_k = \delta_k I - [\delta_k S_k \quad Y_k] \begin{bmatrix} \delta_k S_k^T S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} \delta_k S_k^T \\ Y_k^T \end{bmatrix}, \quad (6.101)$$

where  $L_k \in \mathbb{R}^{m \times m}$  and  $D_k \in \mathbb{R}^{m \times m}$  are defined by

$$(L_k)_{i,j} = \begin{cases} s_{k-m-1+i}^T y_{k-m-1+j} & \text{if } i > j, \\ 0 & \text{otherwise,} \end{cases} \quad D_k = \text{diag}[s_{k-m}^T y_{k-m}, \dots, s_{k-1}^T y_{k-1}]. \quad (6.102)$$

As soon as the new iterate  $x_{k+1}$  has been generated, the matrix  $S_{k+1}$  is obtained by deleting  $s_{k-m}$  from  $S_k$  and by adding the new  $s_k$ . The matrix  $Y_{k+1}$  is updated in a similar way. Similarly, the new matrices  $L_{k+1}$  and  $D_{k+1}$  are obtained.

Since the middle matrix in (6.101) is of small dimension  $2m$ , its factorization requires a negligible amount of computation. The idea of the compact representation (6.101) is that the corrections to the initial matrix can be expressed as the outer product of  $[\delta_k S_k \quad Y_k]$  and its transpose with an intervening multiplication by a small  $2m \times 2m$  matrix. The limited-memory updating of  $B_k$  requires approximately  $2mn + O(m^3)$  operations and the matrix-vector products of the form  $B_k v$  which can be

performed at a cost of  $(4m + 1)n + O(m^2)$  multiplications. Therefore, updating the direct limited-memory BFGS matrix  $B_k$  is economical when  $m$  is small ( $m = 3$ , or  $m = 5$ ).

This approximation  $B_k$  can be used in methods for the bound constrained and the general constrained optimization. For example, the program L-BFGS-B (Byrd, Lu, Nocedal, & Zhu, 1995b; Zhu, Byrd, Lu, & Nocedal, 1997) makes extensive use of compact limited-memory approximations to solve large-scale nonlinear optimization problems with bound constraints (see Chap. 12). Other codes for the general constrained optimization like KNITRO (Byrd, Hribar, & Nocedal, 1999) and IPOPT (Wächter, & Biegler, 2001, 2006) also make use of the compact limited-memory matrix  $B_k$  to approximate the Hessian of the Lagrangians (see Chaps. 15, 17, and 19, respectively).

A formula similar to (6.97), but this time for a compact representation of the inverse BFGS approximation  $H_k$ , was developed by Byrd, Nocedal, and Schnabel (1994b). An implementation of the unconstrained L-BFGS algorithm based on this expression requires a similar amount of computation as the algorithm described above.

## 6.5 The SR1 Method

The symmetric rank-one SR1 update formula can be derived as solution of the following simple problem. “Given a symmetric matrix  $B_k$  and the vectors  $s_k$  and  $y_k$ , find a new symmetric matrix  $B_{k+1}$  such that  $B_{k+1} - B_k$  has rank one, such that the secant equation  $B_{k+1}s_k = y_k$  is satisfied.” It is easy to see that if  $(y_k - B_k s_k)^T s_k \neq 0$ , then the unique solution of the above problem is

$$(SR1\text{direct}) \quad B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \quad (6.103)$$

If  $y_k = B_k s_k$ , then the solution is  $B_{k+1} = B_k$ . However, if  $(y_k - B_k s_k)^T s_k = 0$  and  $y_k \neq B_k s_k$ , then there is no solution to the problem.

Let  $H_k$  be the inverse approximation to the Hessian at iteration  $k$ . By using the Sherman-Morrison-Woodbury formula, from (6.103), the following update to the inverse Hessian for SR1 is obtained as

$$(SR1\text{inverse}) \quad H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}. \quad (6.104)$$

This variant of the algorithm is only applicable in cases in which the inverse  $H_k$  exists. The main drawbacks of the SR1 update are as follows:

1. The denominator  $(y_k - B_k s_k)^T s_k$  of the SR1 update term in (6.103) may vanish, i.e.,  $(y_k - B_k s_k)^T s_k \cong 0$ , cases in which  $B_{k+1}$  is not well-defined.
2. The step directions computed by using the SR1 updating formula given by (6.103) may no longer be uniform linear independent, thus leading to slowing down the convergence or even the stalling.
3. The SR1 Hessian approximation may not be positive definite along the iterations, thus resulting in a direction that does not produce descent.

To prevent the method from failing because of the first drawback, one simple remedy is to set  $B_{k+1} = B_k$ . However, this may slow down the convergence of the method. Conn, Gould, and Toint (1991b) and Khalfan, Byrd, and Schnabel (1993) showed that the denominator of (6.103) rarely

vanishes in practice, and setting  $B_{k+1} = B_k$  does not have a significant impact on the performances of the SR1 method subject to the number of the iterations or runtimes.

The second drawback is more delicate, being in close connection with the uniform linear independence of the search directions generated by the SR1 algorithm. A more precise definition of the uniform linear independence was given by Conn, Gould, and Toint (1991b). “A sequence  $\{s_k\}$  is *uniformly linearly independent* if there exist  $\xi > 0$ ,  $k_0$  and  $m \geq n$  such that, for each  $k \geq k_0$ , there are  $n$  distinct indices  $k \leq k_1 \leq k_2 \leq \dots \leq k_n \leq k + m$  for which the minimum singular value of the matrix  $S = \begin{bmatrix} \frac{s_{k_1}}{\|s_{k_1}\|}, \dots, \frac{s_{k_n}}{\|s_{k_n}\|} \end{bmatrix}$  is at least  $\xi$ .” Roughly speaking, uniformly linearly independent steps mean that the steps do not tend to fall in a subspace of dimension less than  $n$ .

Conn, Gould, and Toint (1991b) proved that the sequence of matrices generated by the SR1 formula converges to the exact Hessian when the sequence of iterates converges to a limit point and the sequence of steps is uniformly linearly independent. Kelley and Sachs (1998) provided similar convergence results by removing the first of these assumptions. Fiacco and McCormick (1968) showed that if the search directions are linearly independent and the denominator of (6.103) is always non-zero, then the SR1 method without line searches minimizes a strongly convex quadratic function in at most  $n + 1$  steps. In this case,  $B_{n+1}$  is exactly the Hessian of the quadratic function. Observe that this result is significant since it does not require the exact line-search as in the case of the BFGS update. Generally, the above condition given by the definition of the uniform linear independency is not implemented in practice, but it serves only as one of the main assumptions of a proof that the SR1 approximations to the Hessian converge to the true Hessian as the iterates converge to the solution of (6.1).

Subject to the uniform linear independency of the search directions, Khalfan, Byrd, and Schnabel (1993) showed that many problems do not satisfy this requirement. Instead, they proved the local convergence of the SR1 method by using only the positive definiteness and boundedness assumptions for the approximate Hessian. Moreover, Conn, Gould, and Toint (1991b) proved that if the minimizing function  $f$  is twice continuously differentiable and its Hessian is bounded and Lipschitz continuous, the iterates generated by the SR1 method converge to a point  $x^*$ , and in addition, if for all  $k$ ,

$$|(y_k - B_k s_k)^T s_k| \geq \xi \|y_k - B_k s_k\| \|s_k\|, \quad (6.105)$$

for some  $\xi \in (0, 1)$ , ( $\xi = 10^{-8}$ ), and the steps  $s_k$  are uniformly linearly independent, then

$$\lim_{k \rightarrow \infty} \|B_k - \nabla^2 f(x^*)\| = 0. \quad (6.106)$$

The SR1 methods have the ability to generate good Hessian approximations. Let us prove this property for the quadratic functions. Observe that for the functions of this type, the choice of the stepsize does not affect the update. Therefore, to examine the properties of this update, we can assume a uniform step of length 1, that is,  $d_k = -H_k \nabla f(x_k)$  and  $x_{k+1} = x_k + d_k$ , i.e.,  $s_k = d_k$ . In fact, the following results can be proved.

**Theorem 6.12** Suppose that  $f(x) = \frac{1}{2}x^T Ax + b^T x$  is a strongly convex quadratic function, where  $x \in \mathbb{R}^n$  and  $A$  is symmetric positive definite. Then, for any initial point  $x_0$  and any initial symmetric matrix  $H_0$ , the iterates  $\{x_k\}$  generated by the SR1 method (6.104) converge to the minimizer in at most  $n$  steps, provided that  $(s_k - H_k y_k)^T y_k \neq 0$  for all  $k$ . Assume a uniform step length of 1. Moreover, if  $n$  steps are performed and if the search directions  $d_k = -H_k \nabla f(x_k)$  are linearly independent, then  $H_n = A^{-1}$ .

**Proof** Since  $(s_k - H_k y_k)^T y_k \neq 0$ , it follows that the SR1 method is well-defined. Let us show that

$$H_k y_j = s_j, \quad j = 0, 1, \dots, k-1. \quad (6.107)$$

In other words, let us show that the secant equation is satisfied not only along the most recent search directions but also along all the previous directions.

Observe that  $H_1 y_0 = s_0$ . Assume that (6.107) holds for a certain value  $k > 1$ , and show that it also holds for  $k + 1$ . From (6.107), since  $f$  is a quadratic function, we have  $y_i = As_i$ . Therefore,

$$(s_k - H_k y_k)^T y_j = s_k^T y_j - y_k^T (H_k y_j) = s_k^T y_j - y_k^T s_j = 0, \quad \text{for all } j < k.$$

Using this in (6.104), we get  $H_{k+1} y_j = H_k y_j = s_j$ , for all  $j < k$ . Since  $H_{k+1} y_k = s_k$  by the secant equation, it follows that (6.107) holds for  $k + 1$ . By induction, this relation holds for all  $k$ .

If the algorithm performs  $n$  steps and if the steps  $\{s_j\}$  are linearly independent, we obtain  $s_j = H_n y_j = H_n A s_j$ , for  $j = 0, 1, \dots, n-1$ . Hence,  $H_n A = I$ , that is,  $H_n = A^{-1}$ . In other words, the step taken at  $x_n$  is the Newton step. Therefore, the next iterate  $x_{n+1}$  will be the solution and the algorithm terminates.

Now, consider the case in which the steps become linearly dependent. For this, suppose that  $s_k$  is a linear combination of the previous steps, that is,  $s_k = \xi_0 s_0 + \dots + \xi_{k-1} s_{k-1}$ , for some scalars  $\xi_i$ ,  $i = 0, 1, \dots, k-1$ . From (6.107), we have

$$\begin{aligned} H_k y_k &= H_k A s_k \\ &= \xi_0 H_k A s_0 + \dots + \xi_{k-1} H_k A s_{k-1} \\ &= \xi_0 H_k y_0 + \dots + \xi_{k-1} H_k y_{k-1} \\ &= \xi_0 s_0 + \dots + \xi_{k-1} s_{k-1} = s_k. \end{aligned}$$

However,  $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ . Since  $s_k = d_k = -H_k \nabla f(x_k)$ , it follows that

$$H_k (\nabla f(x_{k+1}) - \nabla f(x_k)) = -H_k \nabla f(x_k).$$

By the nonsingularity of  $H_k$ , it follows that  $\nabla f(x_{k+1}) = 0$ , that is,  $x_{k+1}$  is the minimum point. ◆

The relation (6.107) proved in Theorem 6.12 for SR1 shows that, for quadratic functions, the secant equation is satisfied along all the previous search directions, no matter how the line-search is implemented. Such a result is established for the BFGS updating only under the assumption of the exact line-search.

It is important to notice that for general nonlinear functions, under certain conditions, the SR1 update continues to generate good Hessian approximations, as shown further:

**Theorem 6.13** Suppose that  $f$  is twice continuously differentiable and its Hessian is bounded and Lipschitz continuous in a neighborhood of a point  $x^*$ . Let  $\{x_k\}$  be any sequence of iterates such that  $x_k \rightarrow x^*$  for some  $x^* \in \mathbb{R}^n$ . Additionally, suppose that (6.105) holds for all  $k$ , where  $\xi \in (0, 1)$ , and that the steps  $s_k$  are uniformly linearly independent. Then, the matrices  $B_k$  generated by the SR1 updating formula (6.103) satisfy

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

Often, the condition (6.105) is used in the implementations of SR1 to make sure this update behaves well. If this condition is not satisfied, then the update is skipped. Conn, Gould, and Toint (1991b) and Khalfan, Byrd, and Schnabel (1993) provided theoretical and computational results, respectively, that if the uniform linear independence assumption is satisfied, then the approximations to the Hessian generated by the SR1 method are more accurate than those generated by BFGS, and SR1 converges faster than BFGS to the true Hessian. Therefore, if all the above drawbacks are addressed in a reliable and efficient manner, then SR1 can be used for solving (6.1) instead of the rank-two updates. More details on the SR1 method concerning the undefined updates, the choice of the initial approximate  $B_0$ , and the uniform linear independence of the steps are found in Benson and Shanno (2018) and Chen, Lam, and Chan (2019).

The convergence properties of the SR1 method are not understood so well as those of the BFGS method. No global results like in Theorem 6.3 have been established for SR1, apart from some results for quadratic functions. We emphasize that there is no rank-one update formula that maintains both the symmetry and the positive definiteness of the Hessian approximations. However, there is an infinity of rank-two formulae that does it. The most widely used and considered to be the most effective is the BFGS update formula (6.22).

### The Memoryless SR1 Method with Generalized Secant Equation

This quasi-Newton method introduced by Andrei (2021c, 2021d, 2022) is based on the *generalized secant equation*

$$y_k = \gamma_k B_{k+1} s_k, \quad (6.108)$$

where  $\gamma_k$  is a positive parameter. The generalized secant equation is obtained from the requirement that the quadratic model of the minimizing function with scaled approximation to the Hessian should match the gradient of the minimizing function at the latest two iterations  $x_k$  and  $x_{k+1}$ .

Suppose that in the current point  $x_k$  we know the approximation  $B_k$  to the Hessian to be a symmetric matrix. To derive the SR1 method with generalized secant equation, we establish that  $B_{k+1}$ , which is the approximation to the Hessian in  $x_{k+1}$ , satisfies (6.108) and is obtained after a rank-one update of  $B_k$ , i.e., it has the form

$$B_{k+1} = B_k + \delta_k u u^T, \quad (6.109)$$

where  $\delta_k$  is a scalar and  $u \in \mathbb{R}^n$ . Substituting this form into the generalized secant equation, we get

$$y_k = \gamma_k B_k s_k + \gamma_k \delta_k u u^T s_k,$$

or, alternatively,

$$y_k - \gamma_k B_k s_k = \gamma_k \delta_k (u^T s_k) u.$$

Since  $\gamma_k \delta_k (u^T s_k)$  is a scalar, in order to satisfy this equation, we can simply set  $\delta_k = \frac{1}{\gamma_k (u^T s_k)}$  and  $u = y_k - \gamma_k B_k s_k$ . Therefore, introducing these elements in (6.109), the SR1 method with generalized secant equation is obtained as

$$B_{k+1} = B_k + \frac{(y_k - \gamma_k B_k s_k)(y_k - \gamma_k B_k s_k)^T}{\gamma_k (y_k - \gamma_k B_k s_k)^T s_k}. \quad (6.110)$$

Note that the SR1 update formula (6.110) is unique, that is, there is exactly one rank-one update satisfying the generalized secant equation. Moreover, if  $\gamma_k = 1$  in (6.110), then the SR1 update (6.103) is obtained.

If  $y_k = \gamma_k B_k s_k$ , then the solution is  $B_{k+1} = B_k$ . However, if  $(y_k - \gamma_k B_k s_k)^T s_k = 0$  and  $y_k \neq \gamma_k B_k s_k$ , then there is no solution to the problem.

Let  $H_k$  be the inverse approximation to the Hessian at iteration  $k$ . By using the Sherman-Morrison-Woodbury formula in (6.110), the following update to the *inverse Hessian for SR1 with generalized secant equation* is obtained as

$$H_{k+1} = H_k - \frac{(H_k y_k - \gamma_k s_k)(H_k y_k - \gamma_k s_k)^T}{(H_k y_k - \gamma_k s_k)^T y_k}. \quad (6.111)$$

This variant of the algorithm is only applicable in cases in which the inverse  $H_k$  exists.

From (6.111), the search direction corresponding to the inverse Hessian SR1 updating with generalized secant equation is  $d_{k+1} = -H_{k+1} g_{k+1}$ , i.e.,

$$d_{k+1} = -H_k g_{k+1} + \frac{(H_k y_k - \gamma_k s_k)^T g_{k+1}}{(H_k y_k - \gamma_k s_k)^T y_k} (H_k y_k - \gamma_k s_k). \quad (6.112)$$

Now, the *memoryless SR1 method with generalized secant equation* is obtained by considering  $B_k = I$  in (6.110), i.e.,

$$B_{k+1} = I + \frac{(y_k - \gamma_k s_k)(y_k - \gamma_k s_k)^T}{\gamma_k (y_k - \gamma_k s_k)^T s_k}. \quad (6.113)$$

Observe that this is a very simple updating formula, in which the information about the Hessian is not accumulated from iteration to iteration. Besides, we can see that the memoryless SR1 method with generalized secant equation has the same drawbacks as the SR1 method, i.e., when the denominator  $(y_k - \gamma_k s_k)^T s_k$  is zero, or is very close to zero, then the method is not defined.

Now, choosing in (6.111)  $H_k = I$ , i.e.,

$$H_{k+1} = I - \frac{(y_k - \gamma_k s_k)(y_k - \gamma_k s_k)^T}{(y_k - \gamma_k s_k)^T y_k}, \quad (6.114)$$

the *memoryless inverse of SR1 with generalized secant equation* is obtained. Therefore, from (6.114), the *memoryless SR1 search direction with generalized secant equation* is  $d_{k+1} = -H_{k+1} g_{k+1}$ , that is,

$$d_{k+1} = -g_{k+1} + \frac{(y_k - \gamma_k s_k)^T g_{k+1}}{(y_k - \gamma_k s_k)^T y_k} (y_k - \gamma_k s_k). \quad (6.115)$$

The main advantage of the memoryless SR1 update (6.115) is that, for its implementation in computer programs, only two scalar products  $(y_k - \gamma_k s_k)^T g_{k+1}$  and  $(y_k - \gamma_k s_k)^T y_k$  must be computed. This is advantageous for solving large-scale problems. Observe that in this memoryless SR1 update, the information from the previous iteration is not accumulated into the current iteration.

**Proposition 6.2** If

$$\gamma_k > \frac{y_k^T y_k}{s_k^T y_k}, \quad (6.116)$$

then the memoryless SR1 search direction with generalized secant equation (6.115) is a descent direction.

**Proof** From (6.116), it follows that  $(y_k - \gamma_k s_k)^T y_k < 0$ . Therefore, by direct computation, from (6.115), we get

$$g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2 + \frac{(y_k - \gamma_k s_k)^T g_{k+1}}{(y_k - \gamma_k s_k)^T y_k} < 0, \quad \blacklozenge$$

Note that the memoryless SR1 search direction with generalized secant equation (6.115) has three terms. The first is the negative gradient  $-g_{k+1}$ ; the last two terms involve  $s_k$  and  $y_k$ , both of them being multiplied by some scalars. It is obvious that the search direction (6.115) satisfies the conjugacy condition, i.e.,  $y_k^T d_{k+1} = -\gamma_k s_k^T g_{k+1}$ , where  $\gamma_k$  is a positive parameter. Therefore, the memoryless SR1 method with generalized secant equation is a conjugate gradient method which satisfies the Dai and Liao (2001) conjugacy condition.

The memoryless algorithms corresponding to SR1 with generalized secant equation (MM-SR1gen) and to the memoryless BFGS (MM-BFGS) are very simple. In both of them, the stepsize is computed by the Wolfe line-search. The search direction in the memoryless SR1 method with generalized secant equation is computed as in (6.115), while the search direction in the memoryless BFGS method is computed as in (6.54). In the numerical experiments with these algorithms, an acceleration scheme developed by Andrei (2006b, 2009c) was implemented.

#### Algorithm 6.4 Accelerated MM-SR1gen and MM-BFGS

	MM-SR1gen	MM-BFGS
1.	Consider an initial point $x_0$ . Set $k = 0$ . Select some values for the Wolfe line-search conditions $\sigma$ and $\rho$ with $0 < \rho < \sigma < 1$ . Compute $g_0 = \nabla f(x_0)$ and set $d_0 = -g_0$ . Select the sufficiently small parameters: $\varepsilon > 0$ used in the criterion for stopping the iterations, $\varepsilon_q > 0$ used in the search direction computation, and $\varepsilon_A > 0$ used in the acceleration scheme	
2.	Test a criterion for stopping the iterations: if $\ g_k\ _\infty \leq \varepsilon$ , then stop the iterations; otherwise, go to step 3	
3.	Compute the stepsize $\alpha_k$ using the Wolfe line-search conditions	
4.	Update the variables $x_{k+1} = x_k + \alpha_k d_k$ and compute $f_{k+1}$ and $g_{k+1}$ . Compute $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$	
5.	Acceleration scheme: <ol style="list-style-type: none"> <li>Compute: <math>z = x_k + \alpha_k d_k</math>, <math>g_z = \nabla f(z)</math>, and <math>y_k = g_k - g_z</math></li> <li>Compute: <math>\bar{a}_k = \alpha_k g_k^T d_k</math> and <math>\bar{b}_k = -\alpha_k y_k^T d_k</math></li> <li>If <math> \bar{b}_k  \geq \varepsilon_A</math>, then compute <math>\eta_k = -\bar{a}_k / \bar{b}_k</math> and update the variables as <math>x_{k+1} = x_k + \eta_k \alpha_k d_k</math>. Otherwise, update the variables as <math>x_{k+1} = x_k + \alpha_k d_k</math>. Compute <math>f_{k+1}</math> and <math>g_{k+1}</math>. Compute <math>y_k = g_{k+1} - g_k</math> and <math>s_k = x_{k+1} - x_k</math></li> </ol>	
6.	Select a value for the parameter $\gamma_k$ as in (6.116). If $ y_k^T s_k  \geq \varepsilon_q$ , then compute the search direction $d_{k+1}$ as in (6.54). Otherwise, set $d_{k+1} = -g_{k+1}$	If $ y_k^T s_k  \geq \varepsilon_q$ , compute the search direction $d_{k+1}$ as in (6.54). Otherwise, set $d_{k+1} = -g_{k+1}$
7.	Restart the iterations. If $g_{k+1}^T d_{k+1} > -10^{-3} \ g_{k+1}\  \ d_{k+1}\ $ , then set $d_{k+1} = -g_{k+1}$	
8.	Consider $k = k + 1$ and go to step 2	

Observe that the algorithm is equipped with an acceleration scheme (see step 5). This scheme modifies the stepsize determined by the Wolfe line-search conditions in such a way as to improve the reduction of the minimizing function values along the iterations. It is proved that this acceleration scheme is linear convergent, but the reduction in the function value is significantly improved (see Sect. 3.4 in Chap. 3).

If  $f$  is bounded along the direction  $d_k$ , then there exists a stepsize  $\alpha_k$  satisfying the Wolfe line-search conditions (6.5) and (6.6). The first trial of the stepsize crucially affects the practical behavior of the algorithm. At every iteration  $k \geq 1$ , the starting guess for the step  $\alpha_k$  in the line-search is computed as  $\alpha_{k-1}\|d_{k-1}\|/\|d_k\|$ . Observe that in step 6 of the algorithms, if the search direction for the memoryless SR1 method with generalized secant equation is not defined, i.e., if  $(y_k - \gamma_k s_k)^T y_k$  is close to zero or if the search direction for the memoryless BFGS is not defined, i.e., if  $y_k^T s_k$  is close to zero, then the search direction is commuted to be the steepest descent. A restarting condition has been introduced in our algorithm. If this condition is satisfied, then the algorithm is restarted with the negative gradient.

### Convergence of the MM-SR1gen Method

The global convergence of the MM-SR1gen algorithm is established under the following assumptions:

- (A1) *The level set  $S = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  is bounded, i.e., there exists a constant  $b > 0$  such that for any  $x \in S$ ,  $\|x\| \leq b$ .*
- (A2) *The function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable, and its gradient is Lipschitz continuous in a neighborhood  $N$  of  $S$ , i.e., there exists a constant  $L > 0$  such that  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ , for any  $x, y \in N$ .*

Note that under these assumptions, there exists a constant  $\Gamma > 0$  such that  $\|\nabla f(x)\| \leq \Gamma$  for any  $x \in S$ .

**Proposition 6.3** *Suppose that  $(y_k - \gamma_k s_k)^T y_k < 0$  and  $(y_k - \gamma_k s_k)(y_k - \gamma_k s_k)^T$  is a matrix bounded above in norm. Then, the memoryless SR1 with generalized secant equation update matrix  $H_{k+1}$  given by (6.114) is bounded above in norm.*

**Proof** From (6.114), since the denominator  $(y_k - \gamma_k s_k)^T y_k$  is strictly negative, the numerator  $(y_k - \gamma_k s_k)(y_k - \gamma_k s_k)^T$  is bounded above in norm, and  $\|s_k\| = \|x_{k+1} - x_k\| \leq \|x_{k+1}\| + \|x_k\| \leq 2b$ , it follows that  $H_{k+1}$  given by (6.114) is bounded above in norm. ◆

**Proposition 6.4** *Suppose that  $(y_k - \gamma_k s_k)^T y_k < 0$ . Then, the memoryless SR1 with generalized secant equation search direction (6.115) is a descent direction, that is,  $g_{k+1}^T d_{k+1} < 0$ . Besides,  $\|d_{k+1}\| \leq D$  for some  $D > 0$ .*

**Proof** Observe that the numerator of the update (6.114) is a rank-one positive semidefinite matrix and the denominator of (6.114) is strictly negative. Therefore, the update matrix  $H_{k+1}$  given by (6.114) is positive semidefinite. Without loss of generality, it follows that  $H_{k+1}$  given by (6.114) is positive definite. Thus,  $g_{k+1}^T d_{k+1} = -g_{k+1}^T H_{k+1} g_{k+1} < 0$ .

The boundedness of  $d_{k+1}$  may be established as follows. If the restart condition is satisfied, then  $d_{k+1} = -g_{k+1}$ . Therefore,  $\|d_{k+1}\| = \|g_{k+1}\| < D$ , where  $D = \Gamma$  by assumption (A2). On the other hand, for non-restart iterations, the search direction is computed as  $d_{k+1} = -H_{k+1}g_{k+1}$ , where  $H_{k+1}$  is given by (6.114). In this case, by Proposition 6.3, the sequence of the Hessian matrices given by

(6.114) remains bounded above in norm. Therefore,  $\|d_{k+1}\| = \|-H_{k+1}g_{k+1}\| \leq \|H_{k+1}\|\|g_{k+1}\|$ , which is bounded above by some constant  $D > 0$ .  $\blacklozenge$

**Theorem 6.14** Suppose that the assumptions (A1) and (A2) are satisfied, and assume that the generated matrix  $H_k$  is positive definite, and there are the positive constants  $m$  and  $M$  such that  $mI \leq H_k \leq MI$ . Furthermore, assume that the stepsize  $\alpha_k$  satisfies the weak Wolfe conditions (6.5) and (6.6) and that the function  $f$  is bounded below in  $\mathbb{R}^n$ . Then,  $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ .

**Proof** From the first Wolfe condition (6.5), it follows that  $f(x_k + \alpha_k d_k) < f(x_k) + \rho \alpha_k g_k^T d_k$ . Taking the summation of both sides from 0 to  $k$ , we have

$$f(x_{k+1}) < f(x_0) + \rho \sum_{i=0}^k \alpha_i g_i^T d_i.$$

Since from Proposition 6.4 the algorithm generates descent directions and the function  $f$  is bounded below, it follows that  $-\sum_{i=0}^k \alpha_i g_i^T d_i$  is bounded above by a positive scalar. Now, by Proposition 2.4, for any  $i$ , the stepsize  $\alpha_i$  is bounded below as

$$\alpha_i \geq \frac{1 - \sigma}{L} \frac{|g_i^T d_i|}{\|d_i\|^2}.$$

Therefore,

$$f(x_0) - f(x_{k+1}) \geq \frac{\rho(1 - \sigma)}{L} \sum_{i=0}^k \frac{(g_i^T d_i)^2}{\|d_i\|^2}.$$

In other words,

$$\sum_{i=0}^{\infty} \frac{(g_i^T d_i)^2}{\|d_i\|^2} \leq \gamma, \quad \text{where } \gamma = \frac{L(f(x_0) - f_{\min})}{\rho(1 - \sigma)} \geq 0,$$

where  $f_{\min}$  is the minimum value of the sequence  $\{f(x_k)\}_{k=1}^{\infty}$ . Since  $d_i = -H_i g_i$ , we have

$$\frac{(g_i^T d_i)^2}{\|d_i\|^2} = \frac{(g_i^T H_i g_i)^2}{g_i^T H_i H_i g_i} \geq \frac{m^2}{M^2} \|g_i\|^2.$$

Therefore,

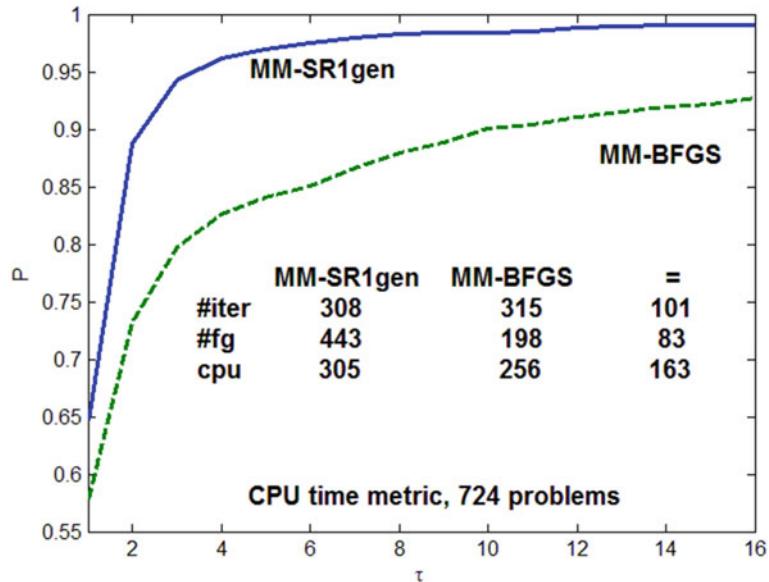
$$\frac{m^2}{M^2} \sum_{i=1}^{\infty} \|g_i\|^2 \leq \sum_{i=0}^{\infty} \frac{(g_i^T d_i)^2}{\|d_i\|^2} < \gamma,$$

that is,

$$\sum_{i=0}^{\infty} \|g_i\|^2 \leq \gamma \frac{M^2}{m^2} < \infty,$$

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

**Fig. 6.11** Accelerated MM-SR1gen versus MM-BFGS, range [1000, 10000]



Hence, the algorithm MM-SR1gen is globally convergent to a point in which the first-order optimality conditions are satisfied.

### Numerical Study: MM-SR1gen versus MM-BFGS and versus BFGS

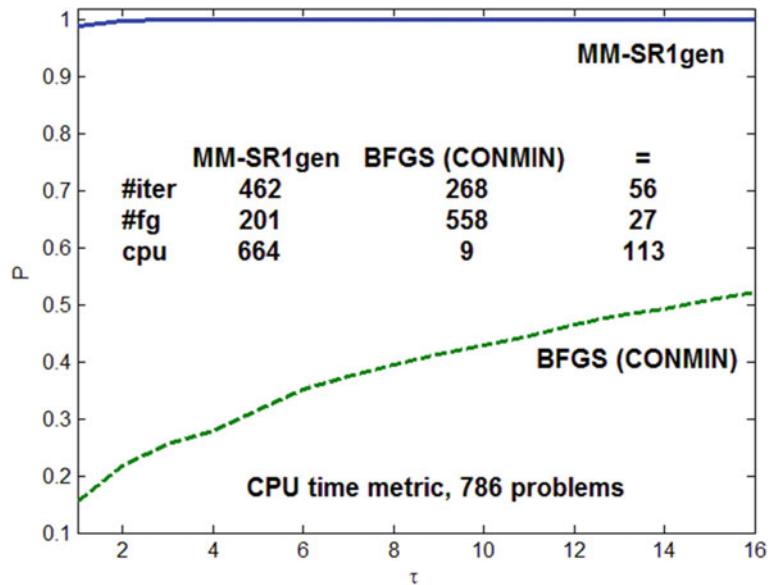
In the following, let us report some numerical results obtained with the accelerated MM-SR1gen and the MM-BFGS methods for solving 800 unconstrained optimization problems from the UOP collection. In the first set of numerical experiments, we compare accelerated MM-SR1gen versus MM-BFGS for solving the unconstrained minimization problems from UOP, with the number of variables in the range [1000, 10000]. Figure 6.11 shows the Dolan and Moré performance profiles of these algorithms; see Remark 1.1.

From Fig. 6.11, we can see that MM-SR1gen is more efficient and more robust than MM-BFGS. In the second set of numerical experiments, we compare the performances of the accelerated MM-SR1gen versus BFGS which is implemented in CONMIN (Shanno, 1983). The CONMIN package includes two optimization algorithms: a BFGS preconditioned conjugate gradient and a variant of the BFGS quasi-Newton algorithm. Figure 6.12 shows the performance profiles of these algorithms for solving 800 unconstrained optimization problems from our collection, with the number of variables in the range [100, 1000].

Obviously, MM-SR1gen is more efficient and more robust than BFGS from CONMIN, one of the best implementations of this quasi-Newton method. Observe that subject to the CPU computing time, MM-SR1 is faster in 664 problems, while BFGS from CONMIN is faster only in nine problems. The BFGS from CONMIN is a variable metric method with initial scaling which approximately needs  $n^2/2 + 11n/2$  double precision words of working storage. By comparison, MM-SR1gen requires approximately  $6n$  double precision words of working storage.

BFGS requires more memory and involves a greater computational effort. We emphasize that at every iteration, BFGS from CONMIN updates an approximation to the Hessian by accumulating the information from the previous iterations. On the other hand, at every iteration, the memoryless MM-SR1gen algorithm updates the identity matrix (see (6.113), or (6.114) for the inverse Hessian

**Fig. 6.12** Accelerated MM-SR1gen versus BFGS from CONMIN, range [100, 1000]



**Table 6.8** Performances of MM-SR1gen versus MM-BFGS (40,000 variables, cpu seconds)

		MM-SR1gen				MM-BFGS			
		#iter	#fg	cpu	#ig	#iter	#fg	cpu	#ig
A1	40,000	372	772	7.79	0	6711	20133	180.56	6611
A2	40,000	1257	2547	31.97	0	9312	27937	298.50	8883
A3	40,000	4093	10001	202.28	0	861	2584	53.28	15
A4	40,000	609	1260	65.10	0	666	1997	140.00	416
A5	40,000	308	697	9.47	0	2177	6520	148.46	199
Total		6639	15277	316.61	0	19727	59171	820.80	16124

updating). Obviously, MM-SR1gen does not accumulate the information from iteration to iteration when updating the approximation to the Hessian. However, MM-SR1gen, having a very simple updating formula, is way more efficient and more robust than BFGS from CONMIN.

Now, let us present comparisons between the MM-SR1gen and MM-BFGS algorithms for solving five applications from the MINPACK-2 collection, each of them with 40,000 variables (see Appendix D). The performances of the MM-SR1gen method versus the MM-BFGS method are given in Table 6.8 where #iter is the number of iterations, #fg is the number of function and its gradient evaluations, #ig is the number of iterations in which the search direction is the negative gradient, and cpu is the CPU computing time for solving these applications in seconds.

From Table 6.8, we can see that MM-SR1gen is more efficient for solving these applications from the MINPACK-2 collection. We can see that MM-SR1gen needs 316.61 seconds, while MM-BFGS needs 820.80 seconds, i.e., MM-SR1gen is 2.60 times faster than MM-BFGS. It is worth pointing out that, for solving all these applications, MM-SR1gen does not use the negative gradient in any iteration. This is in sharp contrast with MM-BFGS, which, out of 19727 iterations for solving all the five applications, the negative gradient is used in exactly 16124 iterations (i.e., 81.73%).

The performances of MM-SR1gen and of MM-BFGS should be compared with the performances of the conjugate gradient algorithms in Tables 5.2, 5.3, 5.4, 5.6, 5.7, 5.9, 5.10, 5.11, and 5.12. Observe that the conjugate gradient algorithms are more efficient than MM-SR1gen or MM-BFGS.

Both these algorithms, MM-SR1gen and MM-BFGS, are memoryless, i.e., they do not accumulate the information from iteration to iteration. They use the same initialization and exactly the same implementation of the Wolfe line-search conditions (6.5) and (6.6). The differences are in the formula for the search direction computation.

---

## 6.6 Sparse Quasi-Newton Updates

These updates are designed for solving large-scale optimization problems. The idea is that the quasi-Newton approximations  $B_k$  have the same sparsity pattern as the true Hessian. Obviously, this approach would reduce the storage requirement and perhaps would generate a more accurate Hessian approximations (Toint, 1977, 1981).

Consider

$$\Omega \triangleq \left\{ (i, j) : (\nabla^2 f(x))_{i,j} \neq 0 \text{ for some } x \text{ in the domain of } f \right\}.$$

Suppose that the current Hessian approximation  $B_k$  has the same zero/nonzero structure of the exact Hessian, i.e.,  $(B_k)_{i,j} = 0$  for all  $(i, j) \notin \Omega$ . Now, in updating  $B_k$  to get  $B_{k+1}$ , the following requirements are imposed:  $B_{k+1}$  satisfies the secant condition, and it has the same sparsity pattern as  $B_k$  and is as close as possible to  $B_k$ . Specifically,  $B_{k+1}$  is defined as solution of the following quadratic minimization problem

$$\begin{aligned} \min_B \|B - B_k\|_F^2 &= \sum_{(i,j) \in \Omega} (B_{i,j} - (B_k)_{i,j})^2, \\ \text{subject to} \\ B s_k &= y_k, \quad B = B^T, \quad \text{and} \quad B_{i,j} = 0 \quad \text{for } (i, j) \notin \Omega. \end{aligned} \tag{6.117}$$

The solution  $B_{k+1}$  of this problem can be obtained by solving an  $n \times n$  linear system whose sparsity pattern is  $\Omega$ , the same as the sparsity of the true Hessian. The main drawbacks of this approach are as follows: the possible  $B_{k+1}$  solution of (6.117) is not positive definite and the updating process is not scale invariant under linear transformations of the variables. However, the fundamental weakness of this approach is that (6.117) is an inadequate model and can produce poor Hessian approximations.

An alternative approach is to relax the secant equation by requiring that it should be approximately satisfied along the last few steps rather than on the latest step. For this, define  $S_k$  and  $Y_k$  as in (6.100) so that they contain the  $m$  most recent difference pairs. The new Hessian approximation  $B_{k+1}$  is obtained as solution of the following quadratic problem:

$$\begin{aligned} \min_B \|BS_k - Y_k\|_F^2 \\ \text{subject to} \\ B = B^T \text{ and } B_{i,j} = 0 \text{ for } (i, j) \notin \Omega. \end{aligned} \tag{6.118}$$

This minimizing problem has a solution which is not easy to compute. Moreover, this approach can produce singular or poorly conditioned Hessian approximations (Nocedal and Wright, 2006). Details on sparse quasi-Newton update can be found in Toint (1981); Gill, Murray, Saunders, and Wright (1982); Lucia (1983); Spedicato and Zhao (1993); Byrd, Nocedal, and Schnabel (1994b); Fletcher (1995); and Yamashita (2005).

## 6.7 Quasi-Newton Methods and Separable Functions

A *separable function*  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  is a function which can be decomposed into a sum of *element functions*, each element function depending on different components of the vector  $x$ . For example, the function

$$f(x) = (x_3 - x_1^2)^2 + (x_4^3 - x_2)^2 \quad (6.119)$$

is a separable function because it can be written as a sum of two element functions

$$f(x) = f_1(x) + f_2(x),$$

where  $f_1(x) = (x_3 - x_1^2)^2$  and  $f_2(x) = (x_4^3 - x_2)^2$ . In general, a separable function can be written as

$$f(x) = \sum_{i=1}^{ne} f_i(x), \quad (6.120)$$

where each of the element function  $f_i$  depends on only a few components of  $x$  and no component of  $x$  appears in more than one element function. *Separable unconstrained minimization* consists of the minimization of each element function in turn. It is easy to see that the gradient and the Hessian of the function (6.120) are

$$\nabla f(x) = \sum_{i=1}^{ne} \nabla f_i(x), \quad \nabla^2 f(x) = \sum_{i=1}^{ne} \nabla^2 f_i(x).$$

The quasi-Newton approximations of the Hessian  $\nabla^2 f(x)$  of the function  $f$  can be computed and handled by computing the quasi-Newton approximations of the Hessians  $\nabla^2 f_i(x)$ ,  $i = 1, \dots, ne$ , corresponding to each element function.

To clarify the concept and explain the quasi-Newton method with separable functions, let us consider the function from (6.119). Observe that  $f_1$  is a function which formally depends on  $x$ , but it actually depends only on  $x_1$  and  $x_3$ , which we call *element variables*. Assembling the element variables into a vector  $x_{[1]} = [x_1, x_3]^T$  and defining the *compactifying matrix*

$$U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

we get  $x_{[1]} = U_1 x$ . Introducing the function  $\varphi_1(z_1, z_2) = (z_2 - z_1^2)^2$ , we can write  $f_1(x) = \varphi_1(U_1 x)$ . By applying the chain rule, we get

$$\nabla f_1(x) = U_1^T \nabla \varphi_1(U_1 x), \quad \nabla^2 f_1(x) = U_1^T \nabla^2 \varphi_1(U_1 x) U_1. \quad (6.121)$$

In our case, we have

$$\nabla^2 \varphi_1(U_1 x) = \begin{bmatrix} 12x_1^2 - 4x_3 & -4x_1 \\ -4x_1 & 2 \end{bmatrix}, \quad \nabla^2 f_1(x) = \begin{bmatrix} 12x_1^2 - 4x_3 & 0 & -4x_1 & 0 \\ 0 & 0 & 0 & 0 \\ -4x_1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The idea of using the quasi-Newton method with separable functions is *instead of storing and handling a quasi-Newton approximation to  $\nabla^2 f_1$ , it is more profitable to store and handle a  $2 \times 2$ -*

dimensional quasi-Newton approximation  $B_{[1]}$  of  $\nabla^2\varphi_1$  and use (6.121) to transform it into a quasi-Newton approximation to  $\nabla^2f_1$ . Obviously, after a typical iteration from  $x$  to  $x^+$ , both  $s_{[1]} = x_{[1]}^+ - x_{[1]}$  and  $y_{[1]} = \nabla\varphi_1(x_{[1]}^+) - \nabla\varphi_1(x_{[1]})$  have to be recorded and used in the BFGS or SR1 updating formulae to obtain a new approximation  $B_{[1]}^+$  of  $\nabla^2\varphi_1(U_1x)$ . With this, the approximation of the element Hessian  $\nabla^2f_1$  is obtained from (6.121), that is,  $\nabla^2f_1(x) \approx U_1^T B_{[1]}^+ U_1$ . Obviously, this operation will map the elements of  $B_{[1]}$  to their correct positions in the full  $n \times n$  Hessian approximation to  $f_1$ .

From (6.120), the function  $f$  can be written as

$$f(x) = \sum_{i=1}^{ne} \varphi_i(U_i x).$$

Therefore, the  $n \times n$ -dimensional quasi-Newton approximation to the full Hessian  $\nabla^2f$  is obtained by summing the quasi-Newton approximations  $B_{[i]}$ ,  $i = 1, \dots, ne$  of the element function as follows:

$$B = \sum_{i=1}^{ne} U_i^T B_{[i]} U_i. \quad (6.122)$$

With this approximation to the Hessian, the search direction can be computed as solution of the system  $B_k d_k = -\nabla f(x_k)$ . Of course, for solving this system, it is not necessary to assemble  $B_k$  explicitly, but rather use the conjugate gradient algorithm which is based only on matrix-vector products of the form  $B_k v$  by performing operations with the matrices  $U_i$  and  $B_{[i]}$ ,  $i = 1, \dots, ne$ .

It is important to compare these two approaches. Suppose that we have to minimize a separable function which depends on 1000 variables and where each element function depends only on two variables. Of course, ignoring the partially separable structure of the minimizing function, a quasi-Newton approximation BFGS or SR1 to a  $1000 \times 1000$ -dimensional matrix has to be computed. In this case, many iterations are needed to get a good quality quasi-Newton approximation of  $\nabla^2f(x)$ . By contrast, in the quasi-Newton with separable functions, the functions  $\varphi_i$  still depend only on two variables, so that each quasi-Newton approximation to the Hessian  $B_{[i]}$  is a  $2 \times 2$ -dimensional matrix. After a few iterations, we have enough directions  $s_{[i]}$  and differences  $y_{[i]}$  to compute each  $B_{[i]}$  as an approximate to  $\nabla^2\varphi_i$  accurate enough. Therefore, the full quasi-Newton approximation (6.122) tends to be a very good one to  $\nabla^2f(x)$ .

Obviously, there is no guarantee that the curvature condition  $s_{[i]}^T y_{[i]} > 0$  is satisfied for all  $i = 1, \dots, ne$ . In this case, even if the full Hessian  $\nabla^2f(x)$  is at least positive semidefinite at the solution  $x^*$ , some of the element Hessians  $\nabla^2\varphi_i(\cdot)$  may be indefinite. A solution to overcome this situation is to apply the SR1 update to each of the element Hessians. This approach is used in the LANCELOT package (Conn, Gould, & Toint, 1992b), which is designed to take full advantage of the partial separability of the minimizing function.

The main limitations of this approach based on the separability of the minimizing function are the difficulty of identifying a good partially separable structure of  $f$  and the cost of the step computation management.

Another possibility to take advantage of the separability of the minimizing function  $f$  is to identify the vectors  $g_{[i]}$  with the components of the gradient  $\nabla f(x)$  corresponding to the components of each  $x_{[i]}$  and solve the systems  $\nabla^2\varphi_i(Ux)d_{[i]} = -g_{[i]}$ , for  $i = 1, \dots, ne$ , thus obtaining the components  $d_{[i]}$  of the search direction. In this context, another idea is to compute  $\nabla^2\varphi_i(\cdot)$  by finite-differences (see Chap. 4).

## 6.8 Solving Large-Scale Applications

The quasi-Newton methods are more efficient than the Newton methods. Using only the change in gradients, the quasi-Newton methods construct a model of the minimizing function that produces superlinear convergence. Optimization software libraries contain a large variety of quasi-Newton programs for solving unconstrained, constrained, and large-scale optimization problems. From the practical viewpoint, the most popular quasi-Newton method is BFGS, in which the stepsize is computed by using the Wolfe line-search. The performances of BFGS can degrade if the line-search is not based on the Wolfe conditions.

Two difficulties affect the BFGS method: the storage and the strategy for dealing with the possibility that  $y_k^T s_k \leq 0$ . The storage difficulty is solved by using an efficient storage way to compute the BFGS step by using the history of the iteration rather than the full matrix storage (Byrd, Nocedal, & Schnabel, 1994b). A strategy based on reducing the storage cost to one vector for each iteration was described by Kelley (1999). Another strategy is the limited-memory BFGS method (L-BFGS) (see Nocedal (1980)). Neither of these approaches for controlling the storage, although essential in practice for large-scale problems, has the superlinear convergence properties like the full-storage BFGS algorithm. On the other hand, when  $y_k^T s_k$  is not sufficiently positive, the BFGS update is restarted with the identity matrix. One of the best implementations of the limited-memory BFGS, with a highly practical efficiency, is L-BFGS of Liu and Nocedal (1989).

Another point which should be emphasized is the Hessian initialization of the L-BFGS method. A popular Hessian initialization is  $H_0 = (y_k^T y_k / y_k^T s_k) I$ . There are also more sophisticated Hessian initializations. Gilbert and Lemaréchal (1989) proposed a sparse Hessian initialization.

Table 6.9 shows the performances of L-BFGS ( $m = 5$ ) of Liu and Nocedal for solving five applications from the MINPACK-2 collection, each of them with 250,000 variables.

Comparisons of L-BFGS ( $m = 5$ ) (see Table 6.9) versus the modern conjugate gradient algorithms CG-DESCENT (Table 5.13), DESCON (Table 5.14), and DK (Table 5.15) show that L-BFGS ( $m = 5$ ) is faster. Subject to the number of iterations, DESCONa, with 5810 iterations for solving all the five applications, is better than L-BFGS ( $m = 5$ ).

MM-SR1gen is very easy to implement and requires only two scalar products  $(y_k - \gamma_k s_k)^T g_{k+1}$  and  $(y_k - \gamma_k s_k)^T y_k$  per iteration, where  $\gamma_k$  is given by (6.114). In our numerical experiments,  $\gamma_k$  is computed as  $\gamma_k = 100(y_k^T y_k / s_k^T y_k)$ . Table 6.10 presents the performances of MM-SR1gen for solving five applications from the MINPACK2 collection, each of them with 250,000 variables.

Observe that the limited-memory BFGS, L-BFGS ( $m = 5$ ) is more efficient than the memoryless SR1 method with the generalized secant equation MM-SR1gen. This is because MM-SR1gen does not accumulate information about the Hessian from iteration to iteration.

**Table 6.9** Performances of L-BFGS ( $m = 5$ ) for solving five applications from the MINPACK-2 collection (250,000 variables, *cpu* seconds)

	<i>n</i>	#iter	#fg	<i>cpu</i>
A1	250,000	672	695	80.00
A2	250,000	1924	1986	222.73
A3	250,000	1999	2001	366.84
A4	250,000	1295	1338	439.87
A5	250,000	921	937	137.65
Total	-	6811	6957	1247.09

**Table 6.10** Performances of MM-SR1gen for solving five applications from the MINPACK-2 collection (250,000 variables, *cpu* seconds)

	<i>n</i>	#iter	#g	<i>cpu</i>
A1	250,000	805	1638	89.96
A2	250,000	3384	6799	580.42
A3	250,000	4114	10001	1258.14
A4	250,000	1375	2775	729.36
A5	250,000	742	1561	49.08
Total	-	10420	22774	2706.96

## Notes and References

The quasi-Newton methods were introduced by Davidon (1959, 1980). A comprehensive description of quasi-Newton methods was given by Dennis and Schnabel (1983), Dennis and Moré (1977), and Fletcher (1987). The convergence of the BFGS matrices was considered by Ge and Powell (1983) and by Boggs and Tolle (1994). The global convergence of the BFGS method was established by Powell (1976). This result was extended to the restricted Broyden class, except for the DFP method, by Byrd, Nocedal, and Yuan (1987a). For the *self-correcting property* of the quasi-Newton methods, see Nocedal (1992). The early analysis of the quasi-Newton methods was based on the *bounded deterioration* principle. This is a tool for the local analysis that treats the case of the worst behavior of the quasi-Newton updating. In principle, assuming that the initial point is sufficiently close to the minimum point  $x^*$  and the initial Hessian approximation is sufficiently close to  $\nabla^2 f(x^*)$ , then the bounded deterioration proves that the iteration cannot stay away from the solution. This property can be used to show the superlinear convergence. For details, see Dennis and Schnabel (1983) and the landmark paper by Dennis and Moré (1977).

The limited-memory BFGS method was designed by Nocedal (1980). The main weakness of the L-BFGS method is that it slowly converges on ill-conditioned problems, that is, on problems for which the Hessian matrix contains a wide distribution of eigenvalues.

Plenty of papers were dedicated to the modifications of the BFGS quasi-Newton method. They are based on *sizing*, i.e., multiplying by an appropriate scalar the approximate Hessian matrix before it is updated in the BFGS method, on the *proper scaling of the terms* on the right-hand side of the BFGS updating formula with positive factors, on the *modified secant equation* in order to approximate the curvature of the objective function along the search direction more accurately than the standard secant equation does, and on the *new line-search conditions* for the stepsize computation to ensure the global convergence by modifying the Wolfe line-search conditions.

The quasi-Newton methods with diagonal updating of the Hessian are based on Theorem 4.16 (see 4.89), which shows that for the Newton method, in the norm of the error, the inaccuracy in the Hessian has a smaller influence than the inaccuracy of the gradient. In other words, the inaccuracy evaluation of the Hessian of the minimizing function is not so important. It is the accuracy of the evaluation of the gradient which is more important. A diagonal approximation of the Hessian by finite differences for the unconstrained optimization was developed by Andrei (2020c) (see also Andrei, 2019a, 2019b, 2020d).

The memoryless SR1 method with generalized secant equation was introduced by Andrei (2021c, 2021d).

Sparse quasi-Newton updates were studied by Toint (1977, 1981), Fletcher (1995), and Fletcher, Grothey, and Leyffer (1996). An efficient computation of the sparse Jacobian was presented by Andrei (1983).

The numerical experience with the limited-memory quasi-Newton (and the truncated Newton) methods is described in Zou, Navon, Berger, Phua, Schlick, and Le Dimet (1993). Limited-memory BFGS methods are implemented in L-BFGS (Liu, & Nocedal, 1989), M1QN3 (Gilbert, & Lemaréchal, 1989), E04DGF (Gill, & Murray, 1979), and BBVSCG (Buckley, & Lenir, 1985). Gill and Leonard (2003) give a variant of the limited-memory BFGS that requires less storage and appears to be quite efficient. The compact limited-memory representations are used in L-BFGS-B (Zhu, Byrd, Lu, & Nocedal, 1997), IPOPT (Wächter & Biegler, 2000) (see Chap. 19), and KNITRO (Byrd, Hribar, & Nocedal, 1999) (see Chaps. 15 and 17).

The quasi-Newton methods for partially separable functions were developed by Griewank and Toint (1982a, 1982b). The LANCELOT package by Conn, Gould, and Toint (1992b) implements the concept of partial separability of the minimizing functions.



## Inexact Newton Methods

7

The main idea behind the *inexact Newton method*, also known as the *truncated Newton method*, introduced by Dembo, Eisenstat, and Steihaug (1982) and analyzed by Dembo and Steihaug (1983) and Deuflhard (1990) is to approximate the solution of the Newton system

$$\nabla^2 f(x_k) d_k = -\nabla f(x_k) \quad (7.1)$$

for the search direction  $d_k$ . Since far away from a local minimum the objective function cannot be well approximated by a quadratic model, it follows that it is not necessary to spend too much time on computing an *accurate* Newton search vector. Therefore, in the following, we describe some procedures for obtaining approximations of the search direction  $d_k$ , which are good enough and inexpensive to calculate. Solving (7.1) in an iterative manner has the advantage of avoiding the factorization of the Hessian  $\nabla^2 f(x_k)$  and, therefore, the fill-in during the factorization. Besides, these methods can be implemented in a Hessian-free manner, so the Hessian  $\nabla^2 f(x_k)$  need not be explicitly calculated or stored. Solving (7.1) is based on using the conjugate gradient method with some modifications to handle the negative curvature in the Hessian.

The criterion for terminating the iterative method for solving (7.1) is based on the *residual*

$$r_k = \nabla^2 f(x_k) d_k + \nabla f(x_k), \quad (7.2)$$

where  $d_k$  is the inexact Newton step. Usually, the conjugate gradient iterations are terminated when

$$\|r_k\| \leq \eta_k \|\nabla f(x_k)\|, \quad (7.3)$$

where the sequence  $\{\eta_k\}$  with  $0 < \eta_k < 1$  for all  $k$  is called the *forcing sequence*. Therefore, in the inexact Newton method, the linear algebraic system (7.1) for the Newton step is solved by an iterative method and (7.3) is considered the termination criterion. It is standard to refer to the sequence of the Newton steps  $\{x_k\}$  as the *outer iteration* and to the sequence of iterates for solving (7.1) as the *inner iterations* (Nash, 1984b, 1985). In this context, the naming convention is that the Newton-CG, for example, refers to the Newton iterative method in which the conjugate gradient algorithm is used to perform the inner iterations (see Remark 5.2). The Newton-CC is particularly appropriate for the large-scale optimization since we expect positive definite Hessians near a local minimizer.

## 7.1 The Inexact Newton Method for Nonlinear Algebraic Systems

In the following, let us discuss the inexact Newton method for solving nonlinear algebraic systems

$$F(x) = 0, \quad (7.4)$$

where  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ . Suppose that

- H1. There exists a point  $x^*$  such that  $F(x^*) = 0$ .
- H2.  $F$  is continuously differentiable in a neighborhood of  $x^*$ .
- H3. The Jacobian matrix  $J(x^*) = \nabla F(x^*)$  is nonsingular.

As we know, for solving (7.4) the Newton method is defined by the Newton system

$$J(x_k)s_k = -F(x_k) \quad (7.5)$$

and

$$x_{k+1} = x_k + s_k, \quad (7.6)$$

where  $J(x_k)$  is the Jacobian matrix of  $F$  in point  $x_k$ .

The *inexact* or *truncated* Newton method is defined by

$$J(x_k)s_k = -F(x_k) + r_k, \quad (7.7)$$

where

$$\|r_k\| \leq \eta_k \|F(x_k)\| \quad (7.8)$$

and

$$x_{k+1} = x_k + s_k. \quad (7.9)$$

In this context of truncation,

$$r_k = J(x_k)s_k + F(x_k) \quad (7.10)$$

represents the *residual*, while  $\{\eta_k\}$ , with  $0 < \eta_k < 1$  is a *forcing sequence* which controls the approximation with which the Newton system is solved. We are interested in the local convergence of this method.

**Proposition 7.1** *Let  $F : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a continuously differentiable function in a neighborhood of  $x^* \in D$  and  $J(x^*) = \nabla F(x^*)$  nonsingular. Then, there exist  $\delta > 0$ ,  $\xi > 0$ , and  $\varepsilon > 0$ , such that as soon as  $\|y - x^*\| < \delta$  and  $y \in D$ ,  $J(y)$  is nonsingular and*

$$\|J(y)^{-1}\| \leq \xi. \quad (7.11)$$

Also  $J(y)^{-1}$  is continuous in  $x^*$ , that is,

$$\|J(y)^{-1} - J(x^*)^{-1}\| < \varepsilon. \quad (7.12)$$

**Proof** Denote  $\alpha = \|J(x^*)^{-1}\|$ . For a  $\beta < \alpha^{-1}$ , choose  $\delta$  so that, when  $\|y - x^*\| < \delta$  with  $y \in D$ , it follows that

$$\|J(x^*) - J(y)\| \leq \beta.$$

Using the von-Neumann lemma (see Appendix A), it results that  $J(y)$  is nonsingular, and therefore (7.11) is true with  $\xi = \alpha/(1 - \beta\alpha)$ . Hence,

$$\begin{aligned} \|J(x^*)^{-1} - J(y)^{-1}\| &= \|J(x^*)^{-1}(J(y) - J(x^*))J(y)^{-1}\| \\ &\leq \alpha\xi\|J(x^*) - J(y)\| \leq \alpha\beta\xi \triangleq \varepsilon, \end{aligned}$$

which proves that the continuity of  $J$  guarantees the continuity of  $J^{-1}$ .  $\blacklozenge$

With this, the convergence of the inexact Newton method can be established as follows:

**Theorem 7.1** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a function which satisfies the assumptions H1–H3. Suppose that the forcing sequence  $\{\eta_k\}$  satisfies  $0 \leq \eta_k \leq \eta < t < 1$ . Then, for  $\varepsilon > 0$ , if the initial point  $x_0$  is sufficiently close to  $x^*$ , the sequence  $\{x_k\}$  generated by the inexact Newton method (7.7), (7.8), and (7.9) converges to  $x^*$ , and the rate of convergence is linear, that is,

$$\|x_{k+1} - x^*\|_* \leq t\|x_k - x^*\|_*, \quad (7.13)$$

where  $\|y\|_* = \|J(x^*)y\|$ .

**Proof** Since  $J(x^*)$  is nonsingular, then for  $y \in \mathbb{R}^n$  it follows that

$$\frac{1}{\mu}\|y\| \leq \|y\|_* \leq \mu\|y\|, \quad (7.14)$$

where

$$\mu = \max \left\{ \|J(x^*)\|, \|J(x^*)^{-1}\| \right\}. \quad (7.15)$$

Since  $\eta < t$ , then there exists  $\gamma > 0$  sufficiently small, so that

$$(1 + \gamma\mu)(\eta(1 + \mu\gamma) + 2\mu\gamma) \leq t. \quad (7.16)$$

Now, choose  $\varepsilon > 0$  sufficiently small, so that if  $\|y - x^*\| \leq \mu^2\varepsilon$ , we have

$$\|J(y) - J(x^*)\| \leq \gamma, \quad (7.17)$$

$$\|J(y)^{-1} - J(x^*)^{-1}\| \leq \gamma, \quad (7.18)$$

$$\|F(y) - F(x^*) - J(x^*)(y - x^*)\| \leq \gamma\|y - x^*\|. \quad (7.19)$$

Let  $\|x_0 - x^*\| \leq \varepsilon$ . We prove (7.13) by induction. From (7.14), (7.15), and the inductive hypothesis, it follows that

$$\|x_k - x^*\| \leq \mu\|x_k - x^*\|_* \leq \mu t^k\|x_0 - x^*\|_* \leq \mu^2\|x_0 - x^*\| \leq \mu^2\varepsilon.$$

When  $y = x_k$ , it follows that (7.17), (7.18), and (7.19) are true. Since

$$\begin{aligned}
& J(x^*)(x_{k+1} - x^*) \\
&= J(x^*) \left( x_k - x^* - J(x_k)^{-1} F(x_k) + J(x_k)^{-1} r_k \right) \\
&= J(x^*) J(x_k)^{-1} (J(x_k)(x_k - x^*) - F(x_k) + r_k) \\
&= \left[ I + J(x^*) \left( J(x_k)^{-1} - J(x^*)^{-1} \right) \right] \times [r_k + (J(x_k) - J(x^*))(x_k - x^*) \\
&\quad - (F(x_k) - F(x^*) - J(x^*)(x_k - x^*))],
\end{aligned} \tag{7.20}$$

then, taking the norm, we obtain

$$\begin{aligned}
& \|x_{k+1} - x^*\|_* \\
&\leq \left[ 1 + \|J(x^*)\| \|J(x_k)^{-1} - J(x^*)^{-1}\| \right] \times [\|r_k\| + \|J(x_k) - J(x^*)\| \|x_k - x^*\| \\
&\quad + \|F(x_k) - F(x^*) - J(x^*)(x_k - x^*)\|] \\
&\leq (1 + \mu\gamma)[\eta_k \|F(x_k)\| + \gamma \|x_k - x^*\| + \gamma \|x_k - x^*\|].
\end{aligned} \tag{7.21}$$

However,

$$F(x_k) = [J(x^*)(x_k - x^*)] + [F(x_k) - F(x^*) - J(x^*)(x_k - x^*)];$$

therefore,

$$\|F(x_k)\| \leq \|x_k - x^*\|_* + \gamma \|x_k - x^*\|. \tag{7.22}$$

Using (7.22) in (7.21), we obtain

$$\begin{aligned}
\|x_{k+1} - x^*\|_* &\leq (1 + \mu\gamma)[\eta_k (\|x_k - x^*\|_* + \gamma \|x_k - x^*\|) + 2\gamma \|x_k - x^*\|] \\
&\leq (1 + \mu\gamma)[\eta(1 + \mu\gamma) + 2\mu\gamma] \|x_k - x^*\|_* \\
&\leq t \|x_k - x^*\|_*. \quad \blacklozenge
\end{aligned}$$

To prove the superlinear convergence of the inexact Newton method, some conditions on the residual have to be introduced.

**Proposition 7.2** *Let*

$$\alpha = \max \left\{ \|J(x^*)\| + \frac{1}{2\beta}, 2\beta \right\},$$

where  $\beta = \|J(x^*)^{-1}\|$ . Then, for  $\|y - x^*\|$  sufficiently small, the following inequality

$$\frac{1}{\alpha} \|y - x^*\| \leq \|F(y)\| \leq \alpha \|y - x^*\|. \tag{7.23}$$

holds.

**Proof** Since  $F$  is continuously differentiable, then there exists  $\delta > 0$  sufficiently small, so that as soon as  $\|y - x^*\| < \delta$ ,

$$\|F(y) - F(x^*) - J(x^*)(y - x^*)\| \leq \frac{1}{2\beta} \|y - x^*\|.$$

However,

$$F(y) = [J(x^*)(y - x^*)] + [F(y) - F(x^*) - J(x^*)(y - x^*)].$$

Therefore,

$$\begin{aligned} \|F(y)\| &\leq \|J(x^*)\| \|y - x^*\| + \|F(y) - F(x^*) - J(x^*)(y - x^*)\| \\ &\leq \left( \|J(x^*)\| + \frac{1}{2\beta} \right) \|y - x^*\|. \end{aligned} \quad (7.24)$$

On the other hand,

$$\begin{aligned} \|F(y)\| &\geq \|J(x^*)^{-1}\|^{-1} \|y - x^*\| - \|F(y) - F(x^*) - J(x^*)(y - x^*)\| \\ &\geq \left( \|J(x^*)^{-1}\|^{-1} - \frac{1}{2\beta} \right) \|y - x^*\| \\ &= \frac{1}{2\beta} \|y - x^*\|. \end{aligned} \quad (7.25)$$

From (7.24) and (7.25), we obtain (7.23).  $\diamond$

**Theorem 7.2** Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a function which satisfies the assumptions H1–H3. Suppose that the sequence  $\{x_k\}$  generated by the inexact Newton method (7.7), (7.8), and (7.9) converges to  $x^*$ . Then, the convergence is superlinear if and only if

$$\|r_k\| = o(\|F(x_k)\|) \quad (7.26)$$

for  $k \rightarrow \infty$ .

**Proof** Suppose that the sequence  $\{x_k\}$  is superlinear convergent to  $x^*$ . Therefore,

$$\begin{aligned} r_k &= F(x_k) + J(x_k)(x_{k+1} - x_k) \\ &= [F(x_k) - F(x^*) - J(x^*)(x_k - x^*)] - [J(x_k) - J(x^*)](x_k - x^*) \\ &\quad + [J(x^*) + (J(x_k) - J(x^*))](x_{k+1} - x^*). \end{aligned}$$

Having in view the assumptions H1–H3 and that  $\{x_k\}$  is superlinear convergent, it follows that

$$\begin{aligned} \|r_k\| &\leq \|F(x_k) - F(x^*) - J(x^*)(x_k - x^*)\| + \|J(x_k) - J(x^*)\| \|x_k - x^*\| \\ &\quad + [\|J(x^*)\| + \|J(x_k) - J(x^*)\|] \|x_{k+1} - x^*\| \\ &= o(\|x_k - x^*\|) + o(1) \|x_k - x^*\| + [\|J(x^*)\| + o(1)] o(\|x_k - x^*\|). \end{aligned}$$

Now, from Proposition 7.2, when  $k \rightarrow \infty$ , we obtain

$$\|r_k\| = o(\|x_k - x^*\|) = o(\|F(x_k)\|). \quad (7.27)$$

On the contrary, let us suppose that  $\|r_k\| = o(\|F(x_k)\|)$ . Then, from (7.20), it follows that

$$\begin{aligned} \|x_{k+1} - x^*\| &\leq \left[ \|J(x^*)^{-1}\| + \|J(x_k)^{-1} - J(x^*)^{-1}\| \right] \times \\ &\quad [\|r_k\| + \|J(x_k) - J(x^*)\| \|x_k - x^*\| + \|F(x_k) - F(x^*) - J(x^*)(x_k - x^*)\|] \\ &= \left[ \|J(x^*)^{-1}\| + o(1) \right] [o(\|F(x_k)\|) + o(1) \|x_k - x^*\| + o(\|x_k - x^*\|)]. \end{aligned}$$

Therefore, from Proposition 7.2, we get

$$\|x_{k+1} - x^*\| = o(\|F(x_k)\|) + o(\|x_k - x^*\|) = o(\|x_k - x^*\|),$$

which proves the superlinear convergence of  $\{x_k\}$  generated by the inexact Newton method (7.7), (7.8), and (7.9).  $\blacklozenge$

Observe that if  $\{\eta_k\} \rightarrow 0$ , then the sequence  $\{x_k\}$  generated by the inexact Newton method (7.7), (7.8), and (7.9) converges to  $x^*$  superlinearly. This result can be proved as the following corollary.

**Corollary 7.1** *Suppose that the sequence  $\{x_k\}$  generated by the inexact Newton method (7.7), (7.8), and (7.9) converges to  $x^*$ . If  $\{\eta_k\}$  is convergent to zero, then the sequence  $\{x_k\}$  is superlinear convergent to  $x^*$ .*

**Proof** If  $\lim_{k \rightarrow \infty} \eta_k = 0$ , then

$$\limsup_{k \rightarrow \infty} \frac{\|r_k\|}{\|F(x_k)\|} = 0,$$

that is,  $\|r_k\| = o(\|F(x_k)\|)$ .  $\blacklozenge$

**Theorem 7.3** *Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a function which satisfies the assumptions H1–H3. Suppose that the forcing sequence  $\{\eta_k\}$  satisfies  $0 \leq \eta_k \leq \eta < 1$ . For  $\epsilon > 0$ , if the initial point  $x_0$  is sufficiently close to  $x^*$ , then the sequence  $\{x_k\}$  generated by the inexact Newton method (7.7), (7.8), and (7.9) converges to  $x^*$ , and the rate of convergence is linear, that is, for  $k$  sufficiently large,*

$$\|x_{k+1} - x^*\| \leq c \|x_k - x^*\|, \quad (7.28)$$

where  $0 < c < 1$ . If  $\eta_k \rightarrow 0$ , then the sequence  $\{x_k\}$  is superlinear convergent to  $x^*$ . If  $\eta_k = O(\|F(x_k)\|)$ , then the sequence  $\{x_k\}$  is quadratic convergent to  $x^*$ .

**Proof** From (7.7), it follows that

$$s_k = J(x_k)^{-1}[-F(x_k) + r_k].$$

Then, from Proposition 7.1, we have

$$\|s_k\| \leq \xi(\|F(x_k)\| + \|r_k\|) \leq \xi(1 + \eta)\|F(x_k)\| \leq 2\xi\|F(x_k)\|. \quad (7.29)$$

Therefore, using the Taylor development, we get

$$F(x_{k+1}) = F(x_k) + J(x_k)s_k + O(\|s_k\|^2) = r_k + O(\|F(x_k)\|^2). \quad (7.30)$$

Hence,

$$\|F(x_{k+1})\| \leq \eta_k\|F(x_k)\| + O(\|F(x_k)\|^2). \quad (7.31)$$

Now, dividing both members of (7.31) by  $\|F(x_k)\|$  and having in view that  $\eta_k < \eta < 1$ , then, for  $k \rightarrow \infty$ , we obtain

$$\limsup_{k \rightarrow \infty} \frac{\|F(x_{k+1})\|}{\|F(x_k)\|} \leq \eta < 1. \quad (7.32)$$

From Proposition 7.2, it follows that

$$\limsup_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq C \limsup_{k \rightarrow \infty} \frac{\|F(x_{k+1})\|}{\|F(x_k)\|}, \quad (7.33)$$

where  $C$  is a constant. Therefore, if  $x_k$  is sufficiently close to  $x^*$  and  $C\eta < 1$ , then, locally, the sequence  $\{x_k\}$  is linear convergent to  $x^*$ .

Now, let us suppose that  $\eta_k \rightarrow 0$ . Then,

$$\limsup_{k \rightarrow \infty} \frac{\|r_k\|}{\|F(x_k)\|} = 0,$$

that is,  $\|r_k\| = o(\|F(x_k)\|)$ . Using (7.30) it follows that

$$\limsup_{k \rightarrow \infty} \frac{\|F(x_{k+1})\|}{\|F(x_k)\|} = 0, \quad (7.34)$$

thus proving the superlinear convergence of the values of the function  $F(x)$ . As above, from Proposition 7.2, it follows that

$$\limsup_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0. \quad (7.35)$$

If  $\eta_k = O(\|F(x_k)\|)$ , then there exists a constant  $c_1$  so that  $\eta_k \leq c_1 \|F(x_k)\|$ . From (7.8), we get

$$\limsup_{k \rightarrow \infty} \frac{\|r_k\|}{\|F(x_k)\|^2} \leq c_1,$$

thus proving that

$$r_k = O(\|F(x_k)\|^2).$$

Therefore, from (7.30), we obtain

$$\limsup_{k \rightarrow \infty} \frac{\|F(x_{k+1})\|}{\|F(x_k)\|^2} = c, \quad (7.36)$$

where  $c$  is an arbitrary constant, thus proving the quadratic convergence of the sequence  $\{F(x_k)\}$ , that is,

$$\limsup_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} = c.$$

◆

## 7.2 Inexact Newton Methods for Functions Minimization

As we know, for solving the minimizing problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad (7.37)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable, the Newton method in the current point  $x_k$  minimizes the quadratic model

$$m_2(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T \nabla^2 f(x_k) d \quad (7.38)$$

and considers the new approximation to the minimizer  $x^*$  of the function  $f$  as

$$x_{k+1} = x_k + \alpha_k d_k, \quad (7.39)$$

where  $d_k$  is the minimum point of  $m_2(d)$  and  $\alpha_k > 0$  is the stepsize. Obviously, the minimum of  $m_2(d)$ , i.e., the search direction  $d_k$ , is obtained as solution of the following algebraic linear system

$$\nabla^2 f(x_k) d_k = -\nabla f(x_k), \quad (7.40)$$

known as the Newton system.

On the other hand, for minimizing the function  $f$ , the inexact Newton method does not consider the exact solution of the Newton system. Instead, the inexact Newton method considers an approximate solution of (7.40) based on the *residual*

$$r_k = \nabla^2 f(x_k) d_k + \nabla f(x_k). \quad (7.41)$$

A strategy for implementing this idea is to require that

$$\|r_k\| \leq \eta_k \|\nabla f(x_k)\|, \quad (7.42)$$

where, as above, the sequence  $\{\eta_k\}$  with  $0 < \eta_k < 1$  for all  $k$  is called the *forcing sequence*. The local convergence of the truncated Newton method is simply obtained by ensuring that the sequence  $\{\eta_k\}$  is bounded away from 1.

The following results proved by Dembo, Eisenstat, and Steihaug (1982) are local and show the convergence of the inexact Newton method.

**Theorem 7.4** Suppose that  $\nabla^2 f(x)$  exists and is continuous in a neighborhood of a minimizer  $x^*$ , with  $\nabla^2 f(x^*)$  positive definite. Consider the iteration  $x_{k+1} = x_k + d_k$ , where  $d_k$  satisfies  $\|r_k\| \leq \eta_k \|g_k\|$ , and assume that  $\eta_k \leq \eta$  for some constant  $\eta \in (0, 1)$ . If the starting point  $x_0$  is sufficiently close to  $x^*$ , then the sequence  $\{x_k\}$  generated by the truncated Newton method converges to  $x^*$  and satisfies

$$\|\nabla^2 f(x^*)(x_{k+1} - x^*)\| \leq \hat{\eta} \|\nabla^2 f(x^*)(x_k - x^*)\|, \quad (7.43)$$

for some constant  $\hat{\eta}$  with  $\eta < \hat{\eta} < 1$ .

**Proof** In the following, an informal proof will be presented (Nocedal & Wright, 2006). Since the Hessian matrix  $\nabla^2 f(x^*)$  is positive definite at  $x^*$  and continuous near  $x^*$ , it follows that there exists a positive constant  $L$  such that  $\|\nabla^2 f(x_k)^{-1}\| \leq L$  for all  $x_k$  sufficiently close to  $x^*$ . Therefore, from (7.41) and since  $\|r_k\| \leq \eta_k \|\nabla f(x_k)\|$  where  $\eta_k < 1$ , it follows that the inexact Newton step satisfies

$$\|d_k\| \leq L(\|\nabla f(x_k)\| + \|r_k\|) \leq 2L\|\nabla f(x_k)\|.$$

From the continuity of  $\nabla^2 f(x)$ , we have

$$\begin{aligned} \nabla f(x_{k+1}) &= \nabla f(x_k) + \nabla^2 f(x_k)d_k + \int_0^1 (\nabla f(x_k + td_k) - \nabla f(x_k))d_k dt \\ &= \nabla f(x_k) + \nabla^2 f(x_k)d_k + o(\|d_k\|) \\ &= \nabla f(x_k) - (\nabla f(x_k) - r_k) + o(\nabla f(x_k)) \\ &= r_k + o(\|\nabla f(x_k)\|). \end{aligned} \tag{7.44}$$

Having in view (7.42), from (7.44), we get

$$\|\nabla f(x_{k+1})\| \leq \eta_k \|\nabla f(x_k)\| + o(\|\nabla f(x_k)\|) \leq (\eta_k + o(1))\|\nabla f(x_k)\|. \tag{7.45}$$

When  $x_k$  is close enough to  $x^*$ , it follows that the  $o(1)$  term in the last estimate is bounded by  $(1 - \eta)/2$ . Therefore, from (7.45), we obtain

$$\|\nabla f(x_{k+1})\| \leq (\eta_k + (1 - \eta)/2)\|\nabla f(x_k)\| \leq \frac{1 + \eta}{2}\|\nabla f(x_k)\|, \tag{7.46}$$

that is, the norm of the gradient decreases by a factor of  $(1 + \eta)/2$  at this iteration. By choosing the initial point  $x_0$  sufficiently close to  $x^*$ , it follows that this rate of decreases occurs at every iteration.

To prove (7.43), observe that under the smoothness assumptions, we have

$$\nabla f(x_k) = \nabla^2 f(x^*)(x_k - x^*) + o(\|x_k - x^*\|).$$

Therefore, for  $x_k$  close to  $x^*$ , it can be shown that the gradient  $\nabla f(x_k)$  differs from the scaled error  $\nabla^2 f(x^*)(x_k - x^*)$  by only a relatively small perturbation. A similar estimate holds at  $x_{k+1}$ , so (7.43) follows from (7.46).  $\blacklozenge$

For the superlinear convergence, from (7.45), observe that

$$\frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|} \leq \eta_k + o(1). \tag{7.47}$$

If  $\lim_{k \rightarrow \infty} \eta_k = 0$ , it follows that

$$\lim_{k \rightarrow \infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|} = 0, \tag{7.48}$$

showing the superlinear convergence of the gradient norms  $\|\nabla f(x_k)\|$  to zero. As a consequence, we have the superlinear convergence of the iterates  $\{x_k\}$  to  $x^*$ .

By making the additional assumption that the Hessian  $\nabla^2 f(x)$  is Lipschitz continuous near  $x^*$ , the quadratic convergence of  $\{x_k\}$  to  $x^*$  is obtained. In this case, the estimate (7.44) can be tightened to

$$\nabla f(x_{k+1}) = r_k + O\left(\|\nabla f(x_k)\|^2\right).$$

Therefore, by choosing the forcing sequence so that  $\eta_k = O(\|\nabla f(x_k)\|)$ , it follows that

$$\|\nabla f(x_{k+1})\| = O\left(\|\nabla f(x_k)\|^2\right),$$

thus indicating the quadratic convergence of the gradient norms to zero and also the quadratic convergence of the iterates  $\{x_k\}$  to  $x^*$ . These remarks can be assembled in the following theorem.

**Theorem 7.5** *Suppose that the conditions of Theorem 7.4 hold and assume that the iterates  $\{x_k\}$  generated by the inexact Newton method converge to  $x^*$ . Then, the rate of convergence is superlinear if  $\eta_k \rightarrow 0$ . If in addition  $\nabla^2 f(x)$  is Lipschitz continuous for  $x$  near  $x^*$  and if  $\eta_k = O(\|\nabla f(x_k)\|)$ , then the convergence is quadratic.*  $\blacklozenge$

The best selection of the sequence  $\{\eta_k\}$  is unknown. The only requirement is  $\eta_k \rightarrow 0$ . If  $\eta_k \rightarrow 0$ , then the rate of convergence of the inexact Newton method is superlinear. If in addition  $\nabla^2 f(x)$  is Lipschitz continuous for  $x$  near  $x^*$  and if  $\eta_k = O(\|\nabla f(x_k)\|)$ , then the convergence is quadratic. To obtain the superlinear convergence, set  $\eta_k = \min\{0.5, \sqrt{\|\nabla f(x_k)\|}\}$ . The quadratic convergence is obtained for  $\eta_k = \min\{0.5, \|\nabla f(x_k)\|\}$ .

### 7.3 The Line-Search Newton-CG Method

In this method, the search direction is computed by applying the conjugate gradient method to the Newton system (7.1) until the termination test (7.3) is satisfied. Recall that the conjugate gradient method is designed to solve positive definite linear systems. However, it is quite possible for the Hessian  $\nabla^2 f(x_k)$  to have negative eigenvalues when  $x_k$  is far away from  $x^*$ . Therefore, the algorithm terminates as soon as a direction of negative curvature is generated. This adaptation of Algorithm 5.2 produces a search direction  $d_k$  which is a descent direction. Moreover, the adaptation guarantees that the fast convergence rate of the pure Newton method is preserved, provided that the stepsize  $\alpha_k = 1$  is used whenever it satisfies the acceptance criteria.

The following algorithm is a modification of Algorithm 5.2, which implements the inner iterations to compute the search direction  $d_k$ . In this algorithm, the linear system (7.1) is written as  $B_k d = -g_k$ , where  $B_k$  represents  $\nabla^2 f(x_k)$  and  $g_k$  is  $\nabla f(x_k)$ . For the inner conjugate gradient iterations, the search directions are denoted by  $d_j$ , and the sequence of iterates generated is denoted by  $z_j$ . When  $B_k$  is positive definite, the inner iteration sequence  $\{z_j\}$  converges to the Newton step  $d_k^N$ , solution of  $B_k d = -g_k$ . At each outer iteration, a tolerance  $\varepsilon_k$  is computed, which specifies the accuracy of the computed solution. To obtain the superlinear convergence, the forcing sequence is selected to be  $\eta_k = \min\{0.5, \sqrt{\|\nabla f(x_k)\|}\}$ .

**Algorithm 7.1** Line-search Newton-CG (*truncated Newton*)

1.	Initialization. Select the initial point $x_0$ . Set $k = 0$
2.	Compute the tolerance $\epsilon_k = \min\{0.5, \sqrt{\ \nabla f(x_k)\ }\} \ \nabla f(x_k)\ $
3.	Set $z_0 = 0$ , $r_0 = \nabla f(x_k)$ , $d_0 = -r_0 = -\nabla f(x_k)$
4.	For $j = 0, 1, 2, \dots$ If $d_j^T B_k d_j \leq 0$ If $j = 0$ return $d_k = -\nabla f(x_k)$ Else return $d_k = z_j$ End if End if Set: $\alpha_j = r_j^T r_j / d_j^T B_k d_j$ , $z_{j+1} = z_j + \alpha_j d_j$ , $r_{j+1} = r_j + \alpha_j B_k d_j$ If $\ r_{j+1}\  < \epsilon_k$ return $d_k = z_{j+1}$ End if Set: $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$ , $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$ End for
5.	Compute the stepsize $\alpha_k$ by means of Armijo, Goldstein, or Wolfe line-search
6.	Set $x_{k+1} = x_k + \alpha_k d_k$
7.	Go to step 2

◆

The main differences between the inner loop of Algorithm 7.1 and of Algorithm 5.2 are as follows: the specific starting point  $z_0 = 0$  is used; the positive tolerance  $\epsilon_k$  allows the conjugate gradient iterations to terminate at an inexact solution; and the negative curvature test  $d_j^T B_k d_j \leq 0$  ensures that  $d_k$  is a descent direction for  $f$  at  $x_k$ . If the negative curvature is detected at the very first iteration  $j = 0$ , then the returned direction  $d = -\nabla f(x_k)$  is both a descent direction and a direction of nonpositive curvature for  $f$  at  $x_k$  (Nocedal & Wright, 2006). Algorithm 7.1 can be modified by introducing the preconditioning in conjugate gradient iterations.

It is worth saying that the line-search Newton-CG algorithm does not require explicit knowledge of the Hessian  $B_k = \nabla^2 f(x_k)$ . Instead, it requires only the Hessian-vector products of the form  $B_k d$  for a given vector  $d$ . When the second-order derivatives are difficult to compute or the Hessian requires too much storage, then the finite differencing can be used to calculate the Hessian-vector products. In this case, the product  $\nabla^2 f(x_k) d$  can be approximated as

$$\nabla^2 f(x_k) d = \frac{\nabla f(x_k + hd) - \nabla f(x_k)}{h} \quad (7.49)$$

for a certain small differencing interval  $h$ . The differencing interval  $h$  can be computed as

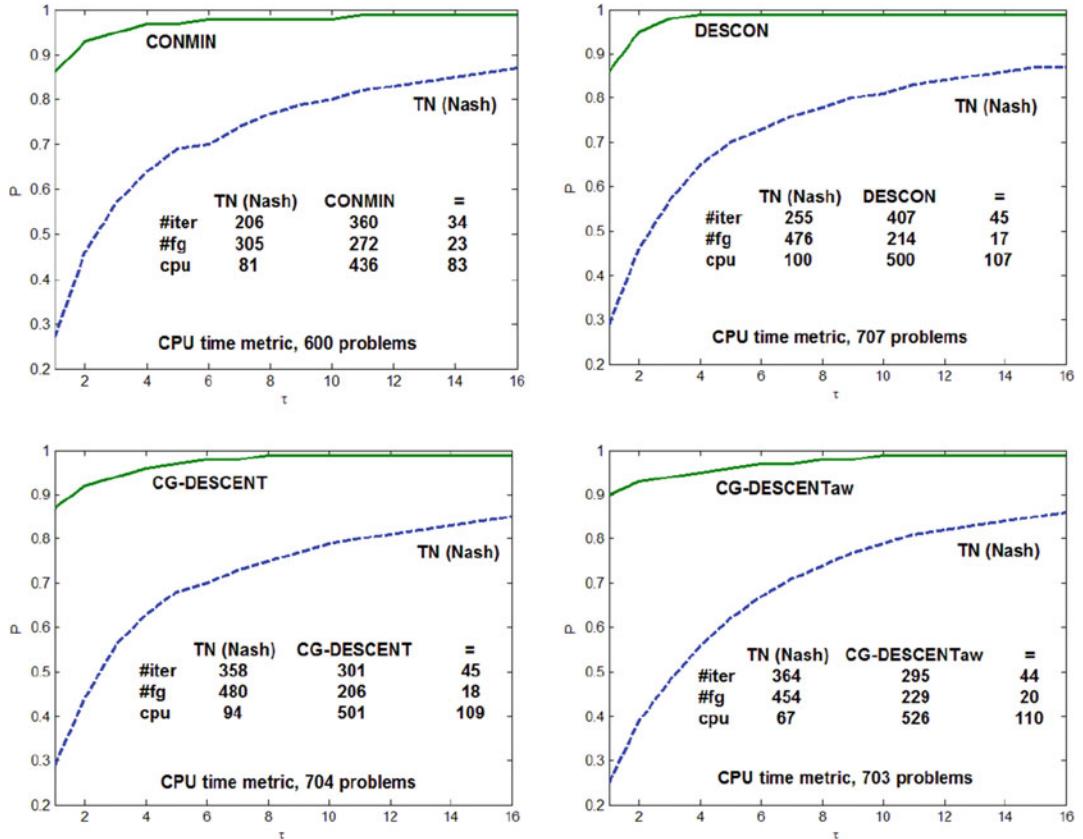
$$h = \frac{2\sqrt{\epsilon_m}(1 + \|x_k\|)}{1 + \|d_k\|}, \quad (7.50)$$

where  $\epsilon_m$  is the epsilon machine.

---

## 7.4 Comparison of TN Versus Conjugate Gradient Algorithms

Observe that the truncated Newton method has two imbedded loops. The first *outer* loop implements the Newton algorithm. The second *inner* loop implements the mechanism for an approximate solution



**Fig. 7.1** Performance of TN (Nash) versus conjugate gradient algorithms CONMIN, DESCON, CG-DESCENT, and CG-DESCENTaw

to the Newton system. Two implementations of this method are known. In the first one (TN), given by Nash (1985), a BFGS approximation of the Hessian is computed and an approximate solution to the Newton system is determined by the Newton-CG algorithm. In the second one (TNPACK), given by Schlick and Fogelson (1992a, 1992b), the Hessian matrix is computed by finite differences, and an approximate solution to the preconditioned Newton system is computed by the Newton-CG algorithm.

In the following, let us present a comparison between the truncated Newton method TN and the conjugate gradient methods CONMIN, DESCON, CG-DESCENT with Wolfe line-search (CG-DESCENT), and CG-DESCENT with the approximate Wolfe line-search (CG-DESCENTaw), for solving 800 unconstrained optimization problems from the UOP collection, with the number of variables in the range [1000, 10000]. Figure 7.1 illustrates the performance profile of these algorithms.

Figure 7.1 illustrates that the conjugate gradient algorithms CONMIN, DESCON, CG-DESCENT, and CG-DESCENTaw are way more efficient and more robust than TN. For example, subject to the CPU time metric, observe that CG-DESCENTaw is the fastest in solving 526 problems, while TN is the fastest in solving only 67 problems, etc. The modern conjugate gradient algorithms are able to better catch the curvature of the minimizing function, thus reducing the number of the function and its gradient evaluations and implicitly the CPU computing time. All these conjugate gradient algorithms implement a lot of ingredients which improve their numerical performances: automatic restarts, storing some additional information, using the approximate or the modified Wolfe line-search, etc.

On the other hand, by taking an approximate solution of the Newton system, TN spends more time in using the approximate search direction compared to the conjugate gradient algorithms.

## 7.5 Comparison of TN Versus L-BFGS

A numerical study on the performances of the limited-memory BFGS method L-BFGS (Liu & Nocedal, 1989) and on the truncated-Newton TN (Nash, 1985) was given by Nash and Nocedal (1991). Some details on the truncated Newton method in the implementation of TN, given by Nash, and on the limited-memory BFGS in the implementation of L-BFGS, given by Liu and Nocedal, are as follows.

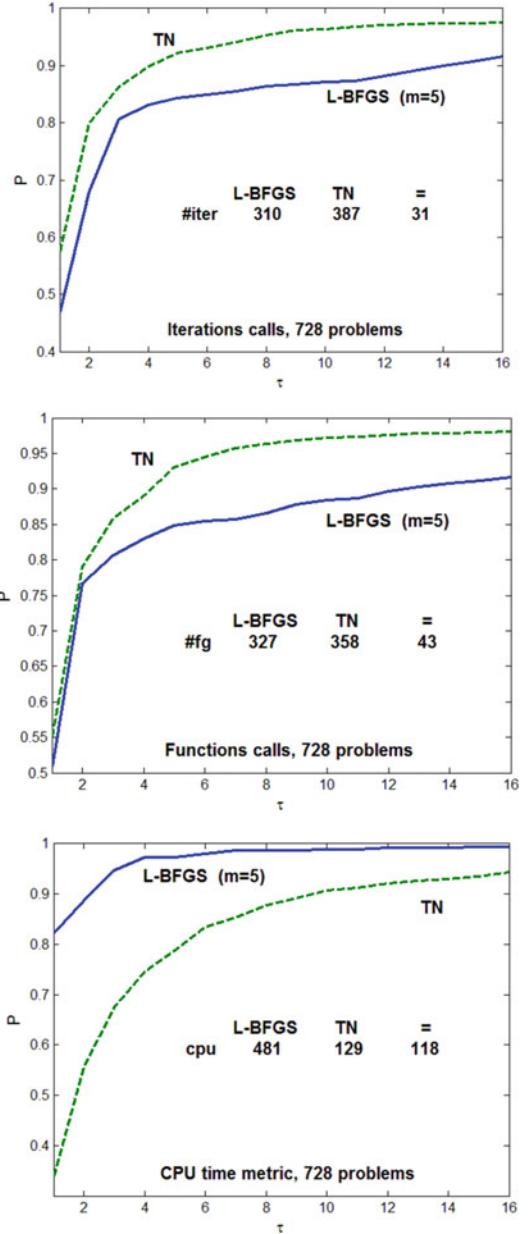
At each *outer* iteration of TN, an approximate solution to the Newton system is computed. This is done by using an *inner* iteration based on a preconditioned linear conjugate gradient algorithm, as described by Concus, Golub, and O’Leary (1976). If indefiniteness in the Hessian is detected, then the inner iteration is stopped. This approximate solution, that is, the search direction, is then used in a line-search to get a new point which approximates the minimizer of the minimizing function. TN implements the strong Wolfe line-search described by Gill and Murray (1979). In TN, the conjugate gradient used in the inner iteration is preconditioned by a scaled two-step limited memory BFGS method with Powell’s restarting strategy used to reset the preconditioner periodically. More than that, the matrix-vector products in TN required by the inner conjugate gradient algorithm are obtained by finite differencing. The truncated Newton method TN requires storage for 16 vectors of length  $n$ .

The limited memory L-BFGS in the implementation of Liu and Nocedal uses a scaling at the first iteration by a diagonal matrix suggested by Oren and Spedicato (1976). In L-BFGS, the approximation of the inverse Hessian is obtained by applying  $m$  BFGS corrections to the initial diagonal matrix, using the  $m$  previous stored vectors  $s_i$  and  $y_i$ . The stepsize is computed by using the strong Wolfe line-search implemented in the subroutine CWSRCH of Moré and Thuente (1990) and by trying the stepsize  $\alpha_k = 1$  first. The L-BFGS method requires  $2m(n + 1) + 4n$  storage locations.

The conclusions of Nash and Nocedal (1991) for comparing TN with L-BFGS are as follows. L-BFGS and TN use different principles to compute the search direction. L-BFGS uses a fixed, low-cost formula requiring no extra derivative information. TN uses an elaborate, variable-cost iteration with partial second-derivative information. Both of them use the cubic interpolation to obtain the strong Wolfe conditions. The numerical experiments showed that neither algorithm is clearly superior to the other one. In terms of the CPU computing time, neither algorithm is a clear winner: the higher iteration cost of TN is compensated by a much lower iteration count, on average. The performance of these algorithms appears to be correlated with the degree of nonlinearity: for quadratic and approximately quadratic problems, TN outperforms L-BFGS. For highly nonlinear problems, L-BFGS is the best. In terms of function evaluations, L-BFGS is preferable to TN for more highly nonlinear problems. However, TN almost always requires fewer iterations than L-BFGS, and therefore, if the number of the gradient evaluations in the inner iteration could be significantly reduced, TN would be competitive or more efficient than L-BFGS. Subject to the distribution of the eigenvalues of the Hessian, there is no clear correlation between the success of the methods and the eigenvalue structure. For problems with ill-conditioned Hessians, TN seems to be better. The clustering of the eigenvalues at the solution does not seem to be beneficial to one method more than to the other one.

In the following, we shall present the performances of the L-BFGS ( $m = 5$ ) in the implementation of Liu and Nocedal (1989) and of TN in the implementation of Nash (1985) for solving 80 unconstrained optimization test problems from our UOP collection, with  $n = 1000, \dots, 10000$ . Figure 7.2 presents the Dolan and Moré performance profiles of TN versus L-BFGS ( $m = 5$ ) for solving this set

**Fig. 7.2** Performance profiles of L-BFGS ( $m = 5$ ) versus TN (truncated Newton) based on iterations calls, function calls, and CPU time, respectively



of 800 unconstrained optimization test problems subject to the iterations calls, function calls, and CPU time metric, respectively.

By comparing L-BFGS versus TN (see Fig. 7.2) subject to the number of iterations, we can see that L-BFGS is better in 310 problems (i.e., it achieved the minimum number of iterations in 310 problems), while TN is better in 387 problems, etc. Out of 800 problems considered in this set of numerical experiments, only for 728 problems does the criterion (1.3) hold.

Note that, subject to the number of iterations and to the number of function calls, TN is a top performer. However, subject to the CPU time metric, L-BFGS is more efficient and more robust. Both these algorithms are reliable for solving a large variety of large-scale unconstrained optimization

problems. In our numerical experiments, we noticed that the performances of these methods do not depend on the structure of the Hessian matrix of the problems.

Let us now present the performances of these algorithms for solving the applications from the MINPACK-2 collection (see Appendix D). Table 7.1 presents the performances of TN in the implementation of Nash (1985) for solving these applications, where  $n$  is the number of variables ( $n = nx \times ny$ ,  $nx = 200$ ,  $ny = 200$ ),  $\#iter$  is the number of iterations to get the solution,  $\#fg$  is the number of function and its gradient calls, and  $cpu$  is the CPU computing time in seconds for solving the application.

Comparing Table 7.1 with Tables 6.2, 6.3, 6.4, 6.5, and 6.6, we can see that, subject to the number of iterations and to the function calls, TN is better than L-BFGS, but subject to the CPU computing time metric, both L-BFGS and TN have similar performances, L-BFGS being slightly faster.

## 7.6 Solving Large-Scale Applications

The inexact or truncated Newton methods are designed for solving large-scale unconstrained optimization problems. Since for large-scale problems the factorization of the Hessian is prohibitive, it follows that it is preferable to compute an approximate solution to the Newton step by using iterative linear algebra techniques. TN by Nash implements such a technique. The resulting algorithm has the global convergence property and with appropriate parameters may be superlinear convergent to the solution. Besides, TN finds an effective search direction when the Hessian  $\nabla^2 f(x_k)$  is indefinite, being implemented in a “Hessian-free” manner without explicit computation or storage of the Hessian.

Table 7.2 shows the performances of TN for solving five applications from the MINPACK-2 collection, each of them with 250,000 variables ( $n = nx \times ny$ ,  $nx = 500$ ,  $ny = 500$ ).

Comparing the performances of TN subject to the CPU computing time versus the modern conjugate gradient algorithms CG-DESCENT (Table 5.13) with 2401.01 seconds versus DESCON (Table 5.14) with 1537.55 seconds and versus DK+ (Table 5.15) with 2643.24 seconds, we can see that TN (Table 7.2) with 1223.89 seconds is a top performer. In the same realm of comparisons,

**Table 7.1** Performances of TN for solving five applications from the MINPACK-2 collection (40,000 variables)

	$n$	$\#iter$	$\#fg$	$cpu$
A1	40,000	14	320	10.35
A2	40,000	40	790	13.92
A3	40,000	53	1791	53.37
A4	40,000	26	501	21.87
A5	40,000	20	312	5.06
Total	-	153	3714	104.57

**Table 7.2** Performances of TN for solving five applications from the MINPACK-2 collection (250,000 variables)

	$n$	$\#iter$	$\#fg$	$cpu$
A1	250,000	12	649	45.61
A2	250,000	56	1933	173.11
A3	250,000	139	4205	652.80
A4	250,000	29	943	285.13
A5	250,000	17	726	67.24
Total	-	253	8456	1223.89

subject to the CPU time metric, we can see that TN (Table 7.2) with 1223.89 seconds is slightly faster than L-BFGS ( $m = 5$ ) (Table 6.9) with 1247.09 seconds.

### Notes and References

A deep analysis of the inexact Newton method is given by Conn, Gould, and Toint (2000). The choice of the forcing terms in an inexact Newton method for solving nonlinear algebraic systems of equations was presented by Eisenstat and Walker (1996) and An, Mo, and Liu (2007). A matrix-free line-search algorithm for the large-scale equality constrained optimization that allows for inexact step computations was given by Byrd, Curtis, and Nocedal (2010). Numerical experience with the truncated Newton method for the unconstrained optimization was presented by Dixon and Price (1988). Freely available software for the unconstrained optimization includes TN (Nash, 1984a, 1984b) and TNPACK (Schlick & Fogelson, 1992a, 1992b). In TNPACK, a sparse modified Cholesky factorization based on the Yale Sparse Matrix Package is used to factor the preconditioner, which need not be positive definite. Two modified Cholesky factorizations have been implemented in TNPACK (Schlick, 1993). The package TNBC (Nash, 1984a, 1984b) implements the truncated Newton method for solving simple bound optimization problems (see Chap. 12). A survey of the truncated Newton methods was presented by Nash (2000).



# The Trust-Region Method

# 8

In the unconstrained optimization, two approaches are fundamental: the line-search and the trust-region. Both of them generate steps by using a quadratic model of the minimizing function, but in different ways. The line-search methods, presented in the previous chapters, generate a descent search direction  $d$  and then determine a suitable stepsize  $\alpha$  along this direction, hoping that the function values will be reduced. On the other hand, the trust-region methods define a region around the current iterate within which we trust the quadratic model to be an adequate representation of the minimizing function and to choose the step which is the approximate minimizer of the model in this region. Therefore, the trust-region methods choose the direction and the stepsize simultaneously. Of course, if a step is not acceptable, the size of the region will be reduced and a new minimizer will be found. The size of the trust-region is important in the economy of each step. If the region is too small, then the algorithm will take small steps. If it is too large, the minimizer of the model may be far from the minimizer of the function. The size of the region is selected based on the performance of the algorithm at the previous iteration.

*The purpose of this chapter is to present the trust-region method as well as its properties for solving unconstrained optimization problems.* For this, the concept of trust-region, the convergence of the corresponding algorithm, and different techniques for solving the subproblems associated to this method are to be developed. One of the best descriptions of the trust-region method is given by Nocedal and Wright, (2006). We follow their developments.

---

## 8.1 The Trust-Region

Consider the problem  $\min\{f(x) : \mathbb{R}^n \rightarrow \mathbb{R}\}$ , where  $f$  is continuously differentiable. Assume that for the function  $f$ , around the current point  $x_k$ , a quadratic model  $m_k$  is used, which is based on the Taylor series development

$$f(x_k + d) = f(x_k) + g_k^T d + \frac{1}{2} d^T \nabla^2 f(x_k + td) d, \quad (8.1)$$

where  $g_k = \nabla f(x_k)$  and  $t$  is a scalar in the interval  $(0,1)$ . Considering an approximation  $B_k$  to the Hessian of the minimizing function  $f$ , the model  $m_k$  is defined as

$$m_k(d) = f(x_k) + g_k^T d + \frac{1}{2} d^T B_k d, \quad (8.2)$$

where  $B_k$  is a symmetric matrix. Observe that the difference between  $f(x_k + d)$  and  $m_k(d)$  is of order  $O(\|d\|^2)$ , which is small when  $\|d\|$  is small.

When  $B_k$  is equal to the true Hessian  $\nabla^2 f(x_k)$ , the above difference is of order  $O(\|d\|^3)$ , showing that the model is accurate when  $\|d\|$  is small. This choice of  $B_k = \nabla^2 f(x_k)$  defines the trust-region Newton method.

The step of the trust-region method is computed as solution of the following subproblem:

$$\begin{aligned} \min_{d \in \mathbb{R}^n} m_k(d) &= f(x_k) + g_k^T d + \frac{1}{2} d^T B_k d, \\ \text{subject to} \\ \|d\| &\leq \Delta_k, \end{aligned} \tag{8.3}$$

where  $\Delta_k > 0$  is the *trust-region radius*.

In (8.3),  $\|\cdot\|$  is the Euclidian norm, so the solution  $d_k^*$  of (8.3) is the minimizer of  $m_k$  in the ball of radius  $\Delta_k$ . Therefore, the trust-region approach involves the solving of a sequence of subproblems (8.3) in which the objective function and the constraints are both quadratic. If  $B_k$  is positive definite and  $\|B_k^{-1} g_k\| \leq \Delta_k$ , then the solution of (8.3) is easy to compute as the unconstrained minimum  $d_k^B = -B_k^{-1} g_k$  of the quadratic  $m_k(d)$ . In this case,  $d_k^B$  is called the *full step*. In other cases, the solution of (8.3) is not so obvious, and the computational effort associated to the trust-region method is to solve the subproblem (8.3). However, in the trust-region method, we need only an approximate solution of (8.3), which is not so difficult to obtain.

Considering  $B_k = \nabla^2 f(x_k)$ , the optimality conditions for (8.3) show that  $d_k$  is the solution of the linear system

$$(\nabla^2 f(x_k) + \lambda I) d_k = -\nabla f(x_k),$$

where the scalar  $\lambda \geq 0$  is the Lagrange multiplier associated to the trust-region constraints, the matrix  $(\nabla^2 f(x_k) + \lambda I)$  is positive semidefinite and

$$\lambda(\Delta_k - \|d_k\|) = 0.$$

If  $\nabla^2 f(x_k)$  is positive definite and  $\Delta_k$  is sufficiently large, then the solution of (8.3) is the solution of the linear system

$$\nabla^2 f(x_k) d = -\nabla f(x_k),$$

the Newton system. Otherwise, the trust-region method guarantees that

$$\Delta_k \geq \|d_k\| = \left\| (\nabla^2 f(x_k) + \lambda I)^{-1} \nabla f(x_k) \right\|,$$

and so, if  $\Delta_k \rightarrow 0$ , then  $\lambda \rightarrow \infty$  and  $d_k \approx -\frac{1}{\lambda} \nabla f(x_k)$ . Therefore,  $d_k$  is a function of  $\lambda$  and indirectly a function of  $\Delta_k$ . As  $\lambda$  varies between 0 and  $\infty$ , it can be shown that  $d_k = d_k(\lambda)$  continuously varies between the Newton direction (in the positive definite case) and a multiple of  $-\nabla f(x_k)$ . (More discussion on the solution of (8.3) is given in Sect. 8.5, where an iterative technique for solving the subproblem (8.3) is presented.)

The most important ingredient in any trust-region method is the strategy for selecting the trust-region radius  $\Delta_k$  at each iteration. The selectin is based on the agreement between the model  $m_k$  and the minimizing function  $f$  at the previous iteration.

Given the step  $d_k$ , let us define the *ratio*

$$r_k = \frac{f(x_k) - f(x_k + d_k)}{m_k(0) - m_k(d_k)}, \quad (8.4)$$

where  $f(x_k) - f(x_k + d_k)$  is called the *actual reduction* and  $m_k(0) - m_k(d_k)$  is called the *predicted reduction* (predicted by the model). The value of  $r_k$  is crucial in the trust-region method.

Since the step  $d_k$  is obtained by minimizing the model  $m_k$  over a domain that includes  $d = 0$ , it follows that the predicted reduction will always be nonnegative. Therefore, if  $r_k$  is negative, then the new value of the minimizing function  $f(x_k + d_k)$  is greater than the current value  $f(x_k)$ , and in this case, this step must be rejected. On the other hand, if  $r_k$  is close to 1, then there is a good agreement between the model  $m_k$  and the function  $f$  over this step, so it is safe to expand the trust-region for the next iteration. Finally, if  $r_k$  is positive but significantly smaller than 1, then the trust-region is not altered, but if it is close to zero or negative, the trust-region is shrank by reducing  $\Delta_k$  at the next iteration. This strategy is described on steps as the trust-region algorithm.

### Algorithm 8.1 Trust-region

1.	Initialization. Select an initial point $x_0$ and the initial trust-region radius $\Delta_0 > 0$ . Choose the constants $0 < \mu < \eta < 1$ (e.g., $\mu = 1/4$ and $\eta = 3/4$ ). Set $k = 0$
2.	For $k = 0, 1, \dots$ <ul style="list-style-type: none"> <li>(i) If <math>x_k</math> is optimal, stop</li> <li>(ii) Compute <math>d_k</math> as an approximate solution of (8.3)</li> <li>(iii) Evaluate the ratio <math>r_k</math> from (8.4)</li> <li>(iv) If <math>r_k \leq \mu</math>, then set <math>x_{k+1} = x_k</math> (unsuccessful step), else set <math>x_{k+1} = x_k + d_k</math> (successful step)</li> <li>(v) Update <math>\Delta_k</math>:               <ul style="list-style-type: none"> <li>If <math>r_k \leq \mu</math>, then set <math>\Delta_{k+1} = \Delta_k/2</math>,</li> <li>If <math>\mu &lt; r_k &lt; \eta</math>, then set <math>\Delta_{k+1} = \Delta_k</math>,</li> <li>If <math>r_k \geq \eta</math>, then set <math>\Delta_{k+1} = 2\Delta_k</math>.</li> </ul> </li> </ul> End for

◆

In Algorithm 8.1, observe that the radius is increased only if  $\|d_k\|$  actually reaches the boundary of the trust-region. Otherwise, if the step stays strictly inside the region, then the value of the radius is not changed for the next iteration. The value of  $r_k$  shows how well the model predicts the reduction in the function value. If  $r_k$  is small, that is,  $r_k \leq \mu$ , then the actual reduction in the function value is much smaller than the one predicted by  $m_k(d_k)$ , that is, the model cannot be trusted for a radius as large as  $\Delta_k$ . In this case, the step  $d_k$  will be rejected and  $\Delta_k$  will be reduced. On the other hand, if  $r_k$  is large, that is,  $r_k \geq \eta$ , then the model is adequately predicting the reduction in the function value, suggesting that the model can be trusted over an even wider region. In this case, the radius  $\Delta_k$  will be increased.

We now state a convergence theorem of the trust-region methods (see Griva, Nash, & Sofer, 2009). Additional theoretical results and discussions of the second-order optimality conditions can be found in the paper by Moré (1983).

**Theorem 8.1** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a real-valued function. Consider  $x_0$  as an initial point and  $\{x_k\}$  the sequence of points generated by the trust-region algorithm. Assume that the level set  $S = \{x : f(x) \leq f(x_0)\}$  is a compact set and  $f(x)$ ,  $\nabla f(x)$ , and  $\nabla^2 f(x)$  are continuous for all  $x \in S$ . Then,

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

**Proof** Before presenting the proof, some comments are needed. Observe that the theorem does not state that the sequence  $\{x_k\}$  converges to a local minimum of  $f$ . It only states that  $\nabla f(x_k) \rightarrow 0$ . The proof has two parts.

In the first part, it is proved that a subsequence of  $\{\|\nabla f(x_k)\|\}$  converges to zero. The proof is by contradiction. If no such subsequence converges to zero, then, for all sufficiently large values of  $k$ , it follows that  $\|\nabla f(x_k)\| \geq \varepsilon > 0$ , where  $\varepsilon$  is a constant. However, we are interested only in the asymptotic behavior of the trust-region algorithm, and therefore, the early iterations may be ignored, that is, we may as well assume that  $\|\nabla f(x_k)\| \geq \varepsilon$  for all  $k$ . The first part of the proof has five major steps. The first two steps establish the relationships among the quantities  $f(x_k) - f(x_{k+1})$ ,  $m_k(d_k)$ ,  $\Delta_k$ , and  $\|\nabla f(x_k)\|$ . The remaining steps use the assumption that  $\|\nabla f(x_k)\| \geq \varepsilon$  to obtain a contradiction. Step 3 shows that  $\lim_{k \rightarrow \infty} \Delta_k = 0$ . If  $\Delta_k$  is small, then  $\|d_k\|$  is too, and the quadratic model must be a good prediction of the actual reduction in the function value, that is,  $\lim_{k \rightarrow \infty} r_k = 1$ . If this is true, then the trust-region algorithm will not reduce  $\Delta_k$ , that is,  $\lim_{k \rightarrow \infty} \Delta_k \neq 0$ , thus contradicting the result of step 3 and proving the overall result. Let  $M$  be a constant satisfying  $\|\nabla^2 f(x_k)\| \leq M$  for all  $k$ . Observe that this upper bound  $M$  exists because  $\nabla^2 f(x)$  is continuous on the level set  $S$  that is closed and bounded (compact).

The first part of the proof is as follows:

1. *A bound on the predicted reduction:* Let us prove that

$$f(x_k) - m_k(d_k) \geq \frac{1}{2} \|\nabla f(x_k)\| \min \left\{ \Delta_k, \frac{\|\nabla f(x_k)\|}{M} \right\}$$

by investigation how small  $m_k$  could be if  $d_k$  is a multiple of  $-\nabla f(x_k)$ . For this, define the function

$$\begin{aligned} \Phi(\alpha) &= m_k \left( -\alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \right) - f(x_k) \\ &= -\alpha \frac{\nabla f(x_k)^T \nabla f(x_k)}{\|\nabla f(x_k)\|} + \frac{1}{2} \alpha^2 \frac{\nabla f(x_k)^T (\nabla^2 f(x_k)) \nabla f(x_k)}{\|\nabla f(x_k)\|^2} \\ &= -\alpha \|\nabla f(x_k)\| + \frac{1}{2} \alpha^2 M_k, \end{aligned}$$

where  $M_k = \nabla f(x_k)^T (\nabla^2 f(x_k)) \nabla f(x_k) / \|\nabla f(x_k)\|^2 \leq \|\nabla^2 f(x_k)\| \leq M$ . Let  $\alpha^*$  be the minimizer of  $\Phi$  on the interval  $[0, \Delta_k]$ . Observe that  $\alpha^* > 0$ . If  $0 < \alpha^* < \Delta_k$ , then  $\alpha^*$  can be determined by setting  $\Phi'(\alpha) = 0$ , showing that  $\alpha^* = \|\nabla f(x_k)\|/M_k$  and

$$\Phi(\alpha^*) = -\frac{1}{2} \|\nabla f(x_k)\|^2 / M_k \leq -\frac{1}{2} \|\nabla f(x_k)\|^2 / M.$$

On the other hand, supposing that  $\alpha^* = \Delta_k$ , it follows that  $M_k \Delta_k \leq \|\nabla f(x_k)\|$ . This follows from the fact that if  $M_k \leq 0$ , then this is trivially satisfied; otherwise, this is a consequence of setting  $\Phi'(\alpha) = 0$ , since the solution of this equation must be  $\geq \Delta_k$ . Thus,

$$\Phi(\alpha^*) = \Phi(\Delta_k) = -\Delta_k \|\nabla f(x_k)\| + \frac{1}{2} \Delta_k^2 M_k \leq -\frac{1}{2} \Delta_k \|\nabla f(x_k)\|.$$

Therefore, the desired result is obtained by observing that  $m_k(d_k) - f(x_k) \leq \Phi(\alpha^*)$ .

2. A bound on  $f(x_k) - f(x_{k+1})$ : If a successful step is taken, then

$$\mu \leq r_k = \frac{f(x_k) - f(x_{k+1})}{f(x_k) - m_k(d_k)},$$

where  $\mu$  is the constant used to test  $r_k$  in the trust-region algorithm. Hence, from step 1, it follows that

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq (f(x_k) - m_k(d_k))\mu \\ &\geq \frac{1}{2}\mu\|\nabla f(x_k)\| \min \left\{ \Delta_k, \frac{\|\nabla f(x_k)\|}{M} \right\}. \end{aligned}$$

3.  $\lim_{k \rightarrow \infty} \Delta_k = 0$ : Since  $f$  is bounded below on  $S$  and the trust-region algorithm ensures that  $f$  cannot increase at any iteration, it follows that  $\lim_{k \rightarrow \infty} f(x_k)$  exists and is finite. Assume that  $\|\nabla f(x_k)\| \geq \varepsilon > 0$ . If  $k$  is a successful step, then step 2 shows that

$$f(x_k) - f(x_{k+1}) \geq \frac{1}{2}\mu\varepsilon \min \left\{ \Delta_k, \frac{\varepsilon}{M} \right\}.$$

The limit of the left-hand side is zero, so  $\lim_{k_i \rightarrow \infty} \Delta_{k_i} = 0$ , where  $\{k_i\}$  are the indices of the iterations where successful steps are taken. At successful steps, the trust-region radius is either kept constant or doubled; at unsuccessful steps, the radius is reduced. Therefore, between successful steps,  $2\Delta_{k_i} \geq \Delta_{k_{i+1}} \geq \dots \geq \Delta_{k_{i+1}}$ . Thus,  $\lim_{k \rightarrow \infty} \Delta_k = 0$ .

4.  $\lim_{k \rightarrow \infty} r_k = 1$ : From the Taylor series of  $f(x_k + d_k)$ , it follows that

$$\begin{aligned} |f(x_k + d_k) - m_k(d_k)| &= \left| f(x_k) + \nabla f(x_k)^T d_k + \frac{1}{2} d_k^T \nabla^2 f(x_k + \xi_k d_k) d_k - m_k(d_k) \right| \\ &= \left| -\frac{1}{2} d_k^T \nabla^2 f(x_k) d_k + \frac{1}{2} d_k^T \nabla^2 f(x_k + \xi_k d_k) d_k \right| \\ &\leq \frac{1}{2} M \|d_k\|^2 + \frac{1}{2} M \|d_k\|^2 = M \|d_k\|^2 \leq M \Delta_k^2. \end{aligned}$$

Using the bound from step 1 and the result of step 3, for large values of  $k$ , we get

$$f(x_k) - m_k(d_k) \geq \frac{1}{2}\varepsilon\Delta_k.$$

Therefore,

$$\begin{aligned} |r_k - 1| &= \left| \frac{f(x_k) - f(x_k + d_k)}{f(x_k) - m_k(d_k)} - 1 \right| \\ &= \frac{|f(x_k + d_k) - m_k(d_k)|}{|f(x_k) - m_k(d_k)|} \\ &\leq \frac{M \Delta_k^2}{\varepsilon \Delta_k / 2} = \frac{2M}{\varepsilon} \Delta_k \rightarrow 0. \end{aligned}$$

5.  $\lim_{k \rightarrow \infty} \Delta_k \neq 0$ : If  $\lim_{k \rightarrow \infty} r_k = 1$ , then, for large values of  $k$ , the trust-region algorithm will not decrease  $\Delta_k$ . Therefore,  $\Delta_k$  will be bounded away from zero.

This is a contradiction establishing that a subsequence of the sequence  $\{\|\nabla f(x_k)\|\}$  converges to zero, thus completing the first part of the proof.

The second part of the proof is as follows. By contradiction, it is proved that  $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ . If this result is not true, then  $\|\nabla f(x_{k_i})\| \geq \varepsilon > 0$  for a subset  $\{k_i\}$  of the iterations of the trust-region algorithm. However, since a subsequence of  $\{\|\nabla f(x_k)\|\}$  converges to zero, there must exist a set of indices  $\{l_i\}$  such that

$$\begin{aligned}\|\nabla f(x_k)\| &\geq \frac{1}{4}\varepsilon \text{ for } k_i \leq k < l_i, \\ \|\nabla f(x_{l_i})\| &< \frac{1}{4}\varepsilon.\end{aligned}$$

If  $k_i \leq k < l_i$  and iteration  $k$  is successful, then step 2 shows that

$$f(x_k) - f(x_{k+1}) \geq \frac{1}{2}\mu\left(\frac{1}{4}\varepsilon\right) \min\left\{\Delta_k, \frac{\varepsilon/4}{M}\right\}.$$

The left-hand side of the above inequality goes to zero, so that

$$f(x_k) - f(x_{k+1}) \geq \varepsilon_1 \|x_{k+1} - x_k\|,$$

where  $\varepsilon_1 = (\mu\varepsilon)/8$ . Because  $\|x_{k+1} - x_k\| = 0$  for an unsuccessful step, it follows that this result holds for  $k_i \leq k < l_i$ . Using this result repeatedly, we obtain

$$\begin{aligned}\varepsilon_1 \|x_{k_i} - x_{l_i}\| &\leq \varepsilon_1 (\|x_{k_i} - x_{k_i+1}\| + \|x_{k_i+1} - x_{k_i+2}\| + \cdots + \|x_{l_i-1} - x_{l_i}\|) \\ &\leq f(x_{k_i}) - f(x_{k_i+1}) + f(x_{k_i+1}) - f(x_{k_i+2}) + \cdots + f(x_{l_i-1}) - f(x_{l_i}) \\ &= f(x_{k_i}) - f(x_{l_i}).\end{aligned}$$

Since the right-hand side of the above result goes to zero, it follows that the left-hand side can be made arbitrarily small. Since  $\nabla f(x)$  is continuous on  $S$  and  $S$  is a compact, by choosing  $i$  large enough it is possible to guarantee that

$$\|\nabla f(x_{k_i}) - \nabla f(x_{l_i})\| \leq \frac{\varepsilon}{4}.$$

With this, the following contradiction is obtained:

$$\begin{aligned}\varepsilon &\leq \|\nabla f(x_{k_i})\| = \|(\nabla f(x_{k_i}) - \nabla f(x_{l_i})) + \nabla f(x_{l_i})\| \\ &\leq \|\nabla f(x_{k_i}) - \nabla f(x_{l_i})\| + \|\nabla f(x_{l_i})\| \leq \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{\varepsilon}{2} < \varepsilon.\end{aligned}$$

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

The main problem with Algorithm 8.1 is how to solve the trust-region subproblem (8.3). In the following, three strategies for obtaining an approximate solution of the subproblem (8.3) are presented. They are based on the Cauchy point. This point is the minimizer of  $m_k$  along the steepest descent direction given by the negative gradient subject to the trust-region bound. The first approximate strategy is the *dogleg method*, which is appropriate when the Hessian  $B_k$  is positive definite. The

second strategy, known as the *two-dimensional subspace minimization*, can be used when  $B_k$  is indefinite. The third strategy, known as the *trust-region Newton-CG strategy*, is based on the conjugate gradient method to minimize  $m_k$  and is appropriate when  $B_k$  is large and sparse.

## 8.2 Algorithms Based on the Cauchy Point

We emphasize that we seek an optimal solution for (8.3), but for the global convergence, it is enough to find an approximate solution  $d_k$  that lies inside the trust-region and gives a sufficient reduction in the model.

### The Cauchy Point

The sufficient reduction in the model can be measured in terms of the Cauchy point, denoted by  $d_k^C$ , which is computed by the following procedure.

#### Algorithm 8.2 Cauchy point computation

- |    |  |  |
|----|--|--|
| 1. | Find the vector $d_k^s$ solution of the linear version of (8.3), that is,                                      | $d_k^s = \arg \min \{f(x_k) + g_k^T d\} \text{ subject to } \ d\  \leq \Delta_k \quad (8.5)$                     |
| 2. | Compute the scalar $\tau_k > 0$ that minimizes $m_k(\tau_k d_k^s)$ subject to the trust-region bound, that is, | $\tau_k = \arg \min_{\tau > 0} \{m_k(\tau d_k^s)\} \text{ subject to } \ \tau d_k^s\  \leq \Delta_k \quad (8.6)$ |
| 3. | Set $d_k^C = \tau_k d_k^s$   | ◆  |

Observe that in closed form the solution of (8.5) is simply obtained as

$$d_k^s = -\frac{\Delta_k}{\|g_k\|} g_k. \quad (8.7)$$

It is more elaborate to obtain  $\tau_k$  from (8.6), and for that, according to the sign of  $g_k^T B_k g_k$ , the following two cases should be considered. When  $g_k^T B_k g_k \leq 0$ , the function  $m_k(\tau d_k^s)$  monotonically decreases with  $\tau$  whenever  $g_k \neq 0$ . Therefore, in this case,  $\tau_k$  is the largest value that satisfies the trust-region bound, that is,  $\tau_k = 1$ . For the case  $g_k^T B_k g_k > 0$ ,  $m_k(\tau d_k^s)$  is a convex quadratic function in  $\tau$ . In this case,  $\tau_k$  is either the unconstrained minimizer of this quadratic, that is,  $\|g_k\|^3 / (\Delta_k g_k^T B_k g_k)$ , or the boundary value 1, whichever comes first. In conclusion, the solutions of (8.5) and (8.6) are as follows:

$$d_k^C = -\tau_k \frac{\Delta_k}{\|g_k\|} g_k, \quad (8.8)$$

where

$$\tau_k = \begin{cases} 1, & \text{if } g_k^T B_k g_k \leq 0, \\ \min \left\{ \frac{\|g_k\|^3}{\Delta_k g_k^T B_k g_k}, 1 \right\}, & \text{otherwise.} \end{cases} \quad (8.9)$$

Observe that the Cauchy point is not difficult to compute and is of a crucial importance in the economy of the trust-region algorithm. Besides, the Cauchy point does not strongly depend on the matrix  $B_k$ , which is used only in the computation of the step length. Obviously, a rapid convergence of

the algorithm is expected only if  $B_k$  is used in determining the direction and the length of the step and if  $B_k$  contains useful curvature information about the minimizing function  $f$ . Some variants of the trust-region algorithms compute the Cauchy point as above and then try to improve it. The improvement strategy is designed in such a way so that the full step  $d_k^B = -B_k^{-1}g_k$  is chosen whenever  $B_k$  is positive definite and  $\|d_k^B\| \leq \Delta_k$ . When  $B_k$  is the exact Hessian  $\nabla^2 f(x_k)$  or a quasi-Newton approximation of it, then this improvement strategy can be expected to yield superlinear convergence. Now, let us describe the methods for finding the approximate solutions to the subproblem (8.3).

### The Dogleg Method

This method can be used when  $B_k$  is positive definite. In this case, the unconstrained minimizer of  $m_k$  is exactly  $d_k^B = -B_k^{-1}g_k$ . When this point is feasible for (8.3), it follows that it is a solution, so we have  $d_k^*(\Delta_k) = d_k^B$  when  $\|d_k^B\| \leq \Delta_k$ . When  $\Delta_k$  is small relative to  $d_k^B$ , the restriction  $\|d_k\| \leq \Delta_k$  ensures that the quadratic term in  $m_k$  has little effect on the solution of (8.3). For such a small  $\Delta_k$ , an approximation to  $d_k(\Delta_k)$  can be obtained by omitting the quadratic term in (8.3) and by writing

$$d_k^*(\Delta_k) \approx -\Delta_k \frac{g_k}{\|g_k\|}, \quad (8.10)$$

For the intermediate values of  $\Delta_k$ , the solution  $d_k^*(\Delta_k)$  follows a curved trajectory. The dogleg method finds an approximate solution by replacing the curved trajectory for  $d_k^*(\Delta_k)$  with a path consisting of two line segments. The first-line segment runs from the origin to the minimizer of  $m_k$  along the steepest descent direction, which is

$$d_k^U = -\frac{g_k^T g_k}{g_k^T B_k g_k} g_k, \quad (8.11)$$

while the second-line segment runs from  $d_k^U$  to  $d_k^B$ . This trajectory is denoted by  $\tilde{d}_k(\tau)$  for  $\tau \in [0, 2]$ , where

$$\tilde{d}_k(\tau) = \begin{cases} \tau d_k^U, & 0 \leq \tau \leq 1, \\ d_k^U + (\tau - 1)(d_k^B - d_k^U), & 1 \leq \tau \leq 2. \end{cases} \quad (8.12)$$

The dogleg method chooses  $d_k$  to minimize the model  $m_k$  along this path subject to the trust-region bound. The following proposition shows that the minimum along the dogleg path can be easily found.

**Proposition 8.1** *Let  $B_k$  be positive definite. Then,*

- (i)  $\|\tilde{d}_k(\tau)\|$  is an increasing function of  $\tau$ ,
- (ii)  $m_k(\tilde{d}_k(\tau))$  is a decreasing function of  $\tau$ .

**Proof** We restrict the analysis to the case  $\tau \in [1, 2]$ . For  $\tau \in [0, 1]$ , both (i) and (ii) are easy to show. For (i), let us define the function  $h(\alpha)$  by

$$\begin{aligned}
h(\alpha) &= \frac{1}{2} \left\| \tilde{d}_k(1 + \alpha) \right\|^2 \\
&= \frac{1}{2} \| d_k^U + \alpha(d_k^B - d_k^U) \|^2 \\
&= \frac{1}{2} \| d_k^U \|^2 + \alpha(d_k^U)^T (d_k^B - d_k^U) + \frac{1}{2} \alpha^2 \| d_k^B - d_k^U \|^2.
\end{aligned}$$

To prove (i), we must show that  $h'(\alpha) \geq 0$  for  $\alpha \in (0, 1)$ . However,

$$\begin{aligned}
h'(\alpha) &= -(d_k^U)^T (d_k^U - d_k^B) + \alpha \| d_k^U - d_k^B \|^2 \\
&\geq -(d_k^U)^T (d_k^U - d_k^B) \\
&= \frac{g_k^T g_k}{g_k^T B_k g_k} g_k^T \left( -\frac{g_k^T g_k}{g_k^T B_k g_k} g_k + B_k^{-1} g_k \right) \\
&= g_k^T g_k \frac{g_k^T B_k^{-1} g_k}{g_k^T B_k g_k} \left( 1 - \frac{(g_k^T g_k)^2}{(g_k^T B_k g_k)(g_k^T B_k^{-1} g_k)} \right) \geq 0,
\end{aligned}$$

where the final inequality follows from the Cauchy-Schwarz inequality.

For proving (ii), let us define  $\hat{h}(\alpha) = m_k \left( \tilde{d}_k(1 + \alpha) \right)$  and show that  $\hat{h}(\alpha) \leq 0$  for  $\alpha \in (0, 1)$ . Substituting (8.12) in (8.3) and by differentiation, we get

$$\begin{aligned}
h'(\alpha) &= (d_k^B - d_k^U)^T (g_k + B_k d_k^U) + \alpha (d_k^B - d_k^U)^T B_k (d_k^B - d_k^U) \\
&\leq (d_k^B - d_k^U)^T (g_k + B_k d_k^U + B_k (d_k^B - d_k^U)) \\
&= (d_k^B - d_k^U)^T (g_k + B_k d_k^B) = 0,
\end{aligned}$$

proving the result. ◆

Therefore, from this proposition, if  $\| d_k^B \| \geq \Delta_k$ , it follows that the path  $\tilde{d}_k(\tau)$  intersects the trust-region boundary  $\| d_k \| = \Delta_k$  at exactly one point and nowhere else. Since  $m_k$  is decreasing along the path, the chosen value of  $d_k$  will be at  $d_k^B$  if  $\| d_k^B \| \leq \Delta_k$ , otherwise at the point of intersection of the dogleg trajectory and the trust-region boundary. In the latter case, the appropriate value of  $\tau$  is computed by solving the following scalar quadratic equation:

$$\| d_k^U + (\tau - 1)(d_k^B - d_k^U) \|^2 = \Delta_k^2.$$

Let us now discuss the case in which the exact Hessian  $\nabla^2 f(x_k)$  is available to be used in the model problem (8.3). When  $\nabla^2 f(x_k)$  is positive definite, simply set  $B_k = \nabla^2 f(x_k)$ , that is,  $d_k^B = (\nabla^2 f(x_k))^{-1} g_k$ , and apply the above procedure to find the Newton-dogleg step. Otherwise,  $d_k^B$  can be computed by choosing  $B_k$  to be one of the positive definite modified Hessian, as described in Sect. 4.5, and proceed to find the dogleg step as above.  $d_k^B$  will be set to the usual Newton step, near a solution that satisfies the second-order sufficient conditions (see Theorem 11.13), thus allowing a rapid local convergence of the Newton method. However, to use the modified Hessian in the Newton-dogleg method is not beneficial. A modified factorization perturbs the diagonal elements of  $\nabla^2 f(x_k)$  in an arbitrary way, and so

the benefits of the trust-region method may not be achieved. Anyway, the Newton-dogleg method is most appropriate when the minimizing function is convex (i.e.,  $\nabla^2 f(x_k)$  is always positive semidefinite).

### The Two-Dimensional Subspace Minimization

When  $B_k$  is positive definite, the dogleg method can be made more sophisticated by widening the search for  $d_k$  to the entire two-dimensional subspace spanned by  $d_k^U$  and  $d_k^B$  (equivalently  $g_k$  and  $-B_k^{-1}g_k$ ). The subproblem (8.3) is replaced by

$$\begin{aligned} \min_{d \in \mathbb{R}^n} m_k(d) &= f(x_k) + g_k^T d + \frac{1}{2} d^T B_k d, \\ \text{subject to} \\ \|d\| &\leq \Delta_k, \\ d &\in \text{span}[g_k, B_k^{-1}g_k]. \end{aligned} \tag{8.13}$$

The Cauchy point  $d_k^C$  is feasible for (8.13), so the optimal solution of this subproblem yields at least as much reduction in the model  $m_k$  as the Cauchy point, thus ensuring the global convergence of the algorithm. Observe that the two-dimensional subspace minimization strategy is obviously an extension of the dogleg method, since the entire dogleg path lies in  $\text{span}[g_k, B_k^{-1}g_k]$ . The case of indefinite  $B_k$  was considered by Byrd, Schnabel, and Schultz (1988) and Schultz, Schnabel, and Byrd (1985). When  $B_k$  has negative eigenvalues, the two-dimensional subspace in (8.13) is changed to

$$\text{span}\left[g_k, (B_k + \alpha I)^{-1}g_k\right] \quad \text{for some } \alpha \in (-\lambda_1, -2\lambda_1], \tag{8.14}$$

where  $\lambda_1$  is the most negative eigenvalue of  $B_k$ . Of course, this choice of  $\alpha$  ensures that  $B_k + \alpha I$  is positive definite. When  $\|(B_k + \alpha I)^{-1}g_k\| \leq \Delta_k$ , then the subspace search (8.13) and (8.14) is discarded; instead, the step is defined as

$$d_k = -(B_k + \alpha I)^{-1}g_k + v_k, \tag{8.15}$$

where  $v_k$  is a vector satisfying  $v_k^T(B_k + \alpha I)^{-1}g_k \leq 0$ . This condition ensures that  $\|d_k\| \geq \|(B_k + \alpha I)^{-1}g_k\|$ . When  $B_k$  has zero eigenvalues but no negative eigenvalues, the step is defined to be the Cauchy point, i.e.,  $d_k = d_k^C$ . Observe that the computational effort with this method lies in a single factorization of  $B_k$  or of  $B_k + \alpha I$ .

### 8.3 The Trust-Region Newton-CG Method

In the following, let us discuss the finding of an approximate solution to the trust-region subproblem (8.3) that produces improvements of the Cauchy point. This strategy is based on a modified conjugate gradient algorithm and was designed by Steihaug (1983). The following algorithm, Algorithm 8.3, generates the step  $d_k$  which is used in Algorithm 8.1 for some choice of tolerance  $\epsilon_k$  at each iteration. The Steihaug algorithm finds an approximate solution to the subproblem (8.3), where  $B_k = \nabla^2 f(x_k)$ . Denote the search directions of the modified conjugate gradient iteration by  $d_j$  and the sequence of iterates that it generates by  $z_j$ .

**Algorithm 8.3** Newton conjugate gradient (Steihaug)

1.	Initialization. Set: $z_0 = 0$ , $u_0 = \nabla f(x_k)$ , $d_0 = -u_0 = -\nabla f(x_k)$ . Set the tolerance $\varepsilon_k > 0$
2.	If $\ u_0\  \leq \varepsilon_k$ , then return with $d_k = z_0 = 0$ , otherwise go to step 3
3.	For $j = 0, 1, \dots$
	If $d_j^T B_k d_j \leq 0$ , then find $\tau$ such that $d_k = z_j + \tau d_j$ minimizes $m_k(d_k)$ in (8.3) and satisfies $\ d_k\  = \Delta_k$ , return $d_k$ ; otherwise, set $\alpha_j = u_j^T u_j / d_j^T B_k d_j$ ,
	$z_{j+1} = z_j + \alpha_j d_j$ .
	End if
	If $\ z_{j+1}\  \geq \Delta_k$ , then find $\tau \geq 0$ such that $d_k = z_j + \tau d_j$ satisfies $\ d_k\  = \Delta_k$ , return $d_k$ ;
	Otherwise, set $u_{j+1} = u_j + \alpha_j B_k d_j$
	End if
	If $\ u_{j+1}\  < \varepsilon_k$ , then return $d_k = z_{j+1}$ , set $\beta_{j+1} = u_{j+1}^T u_{j+1} / u_j^T u_k$ , $d_{j+1} = -u_{j+1} + \beta_{j+1} d_j$
	End if
	End for

◆

Some remarks are as follows. The first “if” statement inside the loop stops the algorithm if its current search direction  $d_j$  is a direction of nonpositive curvature along  $B_k$ , while the second “if” statement inside the loop determines the termination if  $z_{j+1}$  violates the trust-region bound. In both cases, the algorithm returns the step  $d_k$  obtained by intersecting the current search direction with the trust-region boundary. The choice of the tolerance  $\varepsilon_k$  at each iteration of Algorithm 8.3 is important in keeping the overall cost of the trust-region Newton-CG method at a low level. Near a well-behaved solution  $x^*$ , the trust-region bound becomes inactive, and the method reduces to the inexact (truncated) Newton method analyzed in Chap. 7 (see Theorems 7.4 and 7.5). Therefore, the rapid convergence can be obtained by choosing  $\varepsilon_k = \min \{0.5, \sqrt{\|\nabla f(x_k)\|}\} \|\nabla f(x_k)\|$  (see Algorithm 7.1).

The initialization of  $z_0$  to zero in Algorithm 8.3 is a crucial feature of the algorithm. Provided  $\|\nabla f(x_k)\|_2 \geq \varepsilon_k$ , Algorithm 8.3 terminates at a point  $d_k$  for which  $m_k(d_k) \leq m_k(d_k^C)$ , that is, when the reduction in the model equals or exceeds the reduction of the Cauchy point. To show it, consider several cases. First, if  $d_0^T B_k d_0 = (\nabla f(x_k))^T B_k \nabla f(x_k) \leq 0$ , then the condition in the first “if” statement is satisfied, and the algorithm returns the Cauchy point  $d_k = -\Delta_k (\nabla f(x_k)) / \|\nabla f(x_k)\|$ . Otherwise, Algorithm 8.3 defines  $z_1$  as follows:

$$z_1 = \alpha_0 d_0 = \frac{u_0^T u_0}{d_0^T B_k d_0} d_0 = -\frac{(\nabla f(x_k))^T \nabla f(x_k)}{(\nabla f(x_k))^T B_k \nabla f(x_k)} \nabla f(x_k).$$

If  $\|z_1\| < \Delta_k$ , then  $z_1$  is exactly the Cauchy point. Subsequent steps of Algorithm 8.3 ensure that the final  $d_k$  satisfies  $m_k(d_k) \leq m_k(z_1)$ . On the other hand, when  $\|z_1\| \geq \Delta_k$ , the second “if” statement is activated, and Algorithm 8.3 terminates at the Cauchy point. Another crucial property of Algorithm 8.3 is that the sequence of vectors  $\{z_j\}$  generated by the algorithm satisfies

$$0 = \|z_0\|_2 < \dots < \|z_j\|_2 < \|z_{j+1}\|_2 < \dots < \|d_k\|_2 \leq \Delta_k.$$

---

## 8.4 The Global Convergence

We start the global convergence analysis in this section by giving an estimate of the decrease in the  $m_k$  model achieved by the Cauchy point. Then, by using this estimate, we prove that the sequence of gradients  $\{g_k\}$  generated by Algorithm 8.1 has an accumulation point at zero, that is, this sequence

converges to zero when  $\eta$  is strictly positive. The first result is that the dogleg, the two-dimensional subspace minimization, and the Steihaug algorithm produce approximate solutions  $d_k$  to the subproblem (8.3), solutions that satisfy the following estimate of the decrease in the model function

$$m_k(0) - m_k(d_k) \geq c_1 \|g_k\| \min \left\{ \Delta_k, \frac{\|g_k\|}{\|B_k\|} \right\}, \quad (8.16)$$

where  $c_1 \in (0, 1]$  is a constant. In the next proposition, we show that the Cauchy point satisfies (8.16) with  $c_1 = 1/2$ .

**Proposition 8.2** *The Cauchy point  $d_k^C$  satisfies (8.16) with  $c_1 = 1/2$ , that is,*

$$m_k(0) - m_k(d_k^C) \geq \frac{1}{2} \|g_k\| \min \left\{ \Delta_k, \frac{\|g_k\|}{\|B_k\|} \right\}. \quad (8.17)$$

**Proof** Consider the case  $g_k^T B_k g_k \leq 0$ . We have

$$\begin{aligned} m_k(d_k^C) - m_k(0) &= m(-\Delta_k g_k / \|g_k\|) - f(x_k) \\ &= -\frac{\Delta_k}{\|g_k\|} \|g_k\|^2 + \frac{1}{2} \frac{\Delta_k^2}{\|g_k\|^2} g_k^T B_k g_k \\ &\leq -\Delta_k \|g_k\| \\ &\leq -\|g_k\| \min \left( \Delta_k, \frac{\|g_k\|}{\|B_k\|} \right), \end{aligned}$$

proving that (8.17) holds.

For the next case, consider  $g_k^T B_k g_k > 0$  and

$$\frac{\|g_k\|^3}{\Delta_k g_k^T B_k g_k} \leq 1. \quad (8.18)$$

From (8.9), it follows that  $\tau_k = \|g_k\|^3 / (\Delta_k g_k^T B_k g_k)$ . Now, from (8.8), it follows that

$$\begin{aligned} m_k(d_k^C) - m_k(0) &= -\frac{\|g_k\|^4}{g_k^T B_k g_k} + \frac{1}{2} (g_k^T B_k g_k) \frac{\|g_k\|^4}{(g_k^T B_k g_k)^2} \\ &= -\frac{1}{2} \frac{\|g_k\|^4}{g_k^T B_k g_k} \\ &\leq -\frac{1}{2} \frac{\|g_k\|^4}{\|B_k\| \|g_k\|^2} \\ &= -\frac{1}{2} \frac{\|g_k\|^2}{\|B_k\|} \\ &\leq -\frac{1}{2} \|g_k\| \min \left( \Delta_k, \frac{\|g_k\|}{\|B_k\|} \right), \end{aligned}$$

showing that (8.17) holds.

The last case is the one in which (8.18) does not hold, that is,

$$g_k^T B_k g_k < \frac{\|g_k\|^3}{\Delta_k}. \quad (8.19)$$

In this case, from (8.9),  $\tau_k = 1$ . Therefore, from (8.19), it follows that

$$\begin{aligned} m_k(d_k^C) - m_k(0) &= -\frac{\Delta_k}{\|g_k\|} \|g_k\|^2 + \frac{1}{2} \frac{\Delta_k^2}{\|g_k\|^2} g_k^T B_k g_k \\ &= -\Delta_k \|g_k\| + \frac{1}{2} \frac{\Delta_k^2}{\|g_k\|^2} \frac{\|g_k\|^3}{\Delta_k} \\ &= -\frac{1}{2} \Delta_k \|g_k\| \\ &\leq -\frac{1}{2} \|g_k\| \min\left(\Delta_k, \frac{\|g_k\|}{\|B_k\|}\right), \end{aligned}$$

showing that (8.17) holds.  $\blacklozenge$

To satisfy (8.16), the approximate solution  $d_k$  has only to achieve a reduction that is at least some fixed fraction  $c_2$  of the reduction achieved by the Cauchy point. This is shown in the following theorem.

**Theorem 8.2** *Let  $d_k$  be any vector such that  $\|d_k\| \leq \Delta_k$  and*

$$m_k(0) - m_k(d_k) \geq c_2(m_k(0) - m_k(d_k^C)).$$

*Then,  $d_k$  satisfies (8.16) with  $c_1 = c_2/2$ . If  $d_k$  is the exact solution  $d_k^*$  of (8.3), then it satisfies (8.16) with  $c_1 = 1/2$ .*

**Proof** Since  $\|d_k\| \leq \Delta_k$ , from Proposition 8.2, it follows that

$$m_k(0) - m_k(d_k) \geq c_2(m_k(0) - m_k(d_k^C)) \geq \frac{1}{2} c_2 \|g_k\| \min\left(\Delta_k, \frac{\|g_k\|}{\|B_k\|}\right),$$

proving the result.  $\blacklozenge$

Observe that both the dogleg and the two-dimensional subspace minimization algorithms satisfy (8.16) with  $c_1 = 1/2$ , because they all produce approximate solutions  $d_k$  for which  $m_k(d_k) \leq m_k(d_k^C)$  (Nocedal & Wright, 2006).

The convergence to the stationary points depends on whether the parameter  $\eta$  in Algorithm 8.1 is set to zero or to some small positive values. When  $\eta = 0$ , then we can prove that the sequence of gradients  $\{g_k\}$  has a limit point at zero. When  $\eta > 0$ , then a stronger result follows, that is,  $g_k \rightarrow 0$ . Assume that the approximate Hessians  $B_k$  are uniformly bounded in norm and the minimizing function  $f$  is bounded below on the level set

$$S = \{x : f(x) \leq f(x_0)\}. \quad (8.20)$$

Define an open neighborhood of this set by

$$S(R_0) = \{x : \|x - y\| < R_0 \text{ for some } y \in S\}, \quad (8.21)$$

where  $R_0$  is a positive constant.

In order to have a more general result, we allow the length of the approximate solution  $d_k$  of (8.3) to exceed the trust-region bound, provided that it stays within some fixed multiple of the bound, that is,

$$\|d_k\| \leq \gamma \Delta_k, \text{ for some constant } \gamma \geq 1. \quad (8.22)$$

The following two theorems, proved by Nocedal and Wright (2006), show the global convergence of Algorithm 8.1, both for the case  $\eta = 0$  and for  $\eta > 0$ .

**Theorem 8.3** *Let  $\eta = 0$  in Algorithm 8.1. Suppose that  $\|B_k\| \leq \beta$  for some constant  $\beta$ ,  $f$  is bounded below on the level set  $S$  defined by (8.20) and Lipschitz continuously differentiable in the neighborhood  $S(R_0)$  for some  $R_0 > 0$  and that all the approximate solutions of (8.3) satisfy the inequalities (8.16) and (8.22) for some positive constants  $c_1$  and  $\gamma$ . Then,*

$$\liminf_{k \rightarrow \infty} \|g_k\| = 0. \quad \blacklozenge$$

**Theorem 8.4** *Let  $\eta \in (0, 1/4)$  in Algorithm 8.1. Suppose that  $\|B_k\| \leq \beta$  for some constant  $\beta$ ,  $f$  is bounded below on the level set  $S$  defined by (8.20) and Lipschitz continuously differentiable in the neighborhood  $S(R_0)$  for some  $R_0 > 0$  and that all the approximate solutions  $d_k$  of (8.3) satisfy the inequalities (8.16) and (8.22) for some positive constants  $c_1$  and  $\gamma$ . Then,*

$$\lim_{k \rightarrow \infty} \|g_k\| = 0. \quad \blacklozenge$$

## 8.5 Iterative Solution of the Subproblem

In this section, let us describe a technique for solving the subproblem (8.3). This is based on the characterization of the exact solutions of (8.3), which is given by the following theorem due to Moré and Sorensen (1983). The theorem shows that the solution  $d_k^*$  of (8.3) satisfies

$$(B_k + \lambda I)d_k^* = -g_k, \quad (8.23)$$

for some  $\lambda \geq 0$ .

**Theorem 8.5** *The vector  $d_k^*$  is a global solution of the trust-region problem*

$$\begin{aligned} \min_{d \in \mathbb{R}^n} m_k(d) &= f(x_k) + g_k^T d + \frac{1}{2} d^T B_k d, \\ \text{subject to} \\ \|d\| &\leq \Delta_k, \end{aligned} \quad (8.24)$$

*if and only if  $d_k^*$  is feasible, and there is a scalar  $\lambda \geq 0$  such that the following conditions are satisfied:*

$$(B_k + \lambda I)d_k^* = -g_k. \quad (8.25a)$$

$$\lambda(\Delta_k - \|d_k^*\|) = 0 \quad (8.25b)$$

$$(B_k + \lambda I) \text{ is positive definite.} \quad (8.25c)$$

Before proving the theorem, let us discuss some aspects of the conditions (8.25). The condition (8.25b) is a complementarity condition that states that at least one of the nonnegative quantities  $\lambda$  and  $(\Delta_k - \|d_k^*\|)$  must be zero. Therefore, when the solution lies strictly inside the trust-region, we must have  $\lambda = 0$  and, therefore,  $B_k d_k^* = -g_k$  with  $B_k$  positive semidefinite, from (8.25a) and (8.25c), respectively.

Theorem 8.5 suggests an algorithm for finding the solution  $d_k$  of (8.24). Either  $\lambda = 0$  satisfies (8.25a) and (8.25c) with  $\|d_k\| \leq \Delta_k$ , or else define

$$d_k(\lambda) = -(B_k + \lambda I)^{-1} g_k$$

for  $\lambda$  sufficiently large such that  $(B_k + \lambda I)$  is positive definite and seeks a value  $\lambda > 0$  such that

$$\|d_k(\lambda)\| = \Delta_k. \quad (8.26)$$

Of course, this problem is a one-dimensional root finding problem in variable  $\lambda$ .

To find a value of  $\lambda$  with all the above properties, consider the eigendecomposition of  $B_k$  and use it to study the properties of  $\|d_k(\lambda)\|$ . Since  $B_k$  is symmetric, there is an orthogonal matrix  $Q$  and a diagonal matrix  $\Lambda$  such that  $B_k = Q_k \Lambda_k Q_k^T$ , where  $\Lambda_k = \text{diag}\{\lambda_1, \dots, \lambda_n\}$  and  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$  are the eigenvalues of  $B_k$ . (It should be written  $\Lambda_k = \text{diag}\{\lambda_1^k, \dots, \lambda_n^k\}$ , but we have simplified the notation.) Obviously,  $B_k + \lambda I = Q_k (\Lambda_k + \lambda I) Q_k^T$ , and for  $\lambda \neq \lambda_j$ , it follows that

$$d_k(\lambda) = -Q_k (\Lambda_k + \lambda I)^{-1} Q_k^T g_k = -\sum_{j=1}^n \frac{q_j^T g_k}{\lambda_j + \lambda} q_j, \quad (8.27)$$

where  $q_j$  is the  $j$ -th column of  $Q_k$ .

Therefore, by the orthonormality of  $q_1, \dots, q_n$ , it follows that

$$\|d_k(\lambda)\|^2 = \sum_{j=1}^n \frac{(q_j^T g_k)^2}{(\lambda_j + \lambda)^2}. \quad (8.28)$$

Observe that if  $\lambda > -\lambda_1$ , it follows that  $\lambda_j + \lambda > 0$  for all  $j = 1, \dots, n$ . Therefore,  $\|d_k(\lambda)\|$  is a continuous, nonincreasing function of  $\lambda$  in the interval  $(-\lambda_1, \infty)$ . In fact, we have

$$\lim_{\lambda \rightarrow \infty} \|d_k(\lambda)\| = 0. \quad (8.29)$$

Moreover, when  $q_1^T g_k \neq 0$ , then

$$\lim_{\lambda \rightarrow -\lambda_1} \|d_k(\lambda)\| = \infty. \quad (8.30)$$

Therefore, when  $q_1^T g_k \neq 0$ , then there is a unique value  $\lambda^* \in (-\lambda_1, \infty)$  such that  $\|d_k(\lambda^*)\| = \Delta$ . Of course, there might be other smaller values of  $\lambda$  for which  $\|d_k(\lambda)\| = \Delta$ , but they will fail to satisfy (8.25c).

Now, we are in the position to describe a procedure for computing  $\lambda^* \in (-\lambda_1, \infty)$  for which  $\|d_k(\lambda^*)\| = \Delta$ . Actually, there are two procedures: one for the case in which  $q_1^\top g_k \neq 0$  and the other one for the case in which  $q_1^\top g_k = 0$ , as follows:

*Case  $q_1^\top g_k \neq 0$ .* When  $B_k$  is positive definite and  $\|B_k^{-1}g_k\| \leq \Delta_k$ , then the value  $\lambda = 0$  satisfies (8.25), so the procedure can be terminated with  $\lambda^* = 0$ . Otherwise, by using the Newton method, we can find the value  $\lambda > -\lambda_1$  that solves

$$\varphi_1(\lambda) = \|d_k(\lambda)\| - \Delta_k = 0. \quad (8.31)$$

Nocedal and Wright (2006) show that the disadvantage of this approach can be seen by considering the form of  $\|d_k(\lambda)\|$  when  $\lambda$  is greater than, but close to  $-\lambda_1$ . For such a  $\lambda$ , the function  $\varphi_1$  can be approximated by a rational function as

$$\varphi_1(\lambda) \approx \frac{C_1}{\lambda + \lambda_1} + C_2,$$

where  $C_1 > 0$  and  $C_2$  are constants. Obviously, this approximation is highly nonlinear and the Newton method for the root-finding will be unreliable or slow. If the problem (8.31) is reformulated to be almost linear near the optimal  $\lambda$ , the results obtained are better. By defining

$$\varphi_2(\lambda) = \frac{1}{\Delta_k} - \frac{1}{\|d_k(\lambda)\|},$$

from (8.28), it can be shown that for  $\lambda$  slightly greater than  $-\lambda_1$ , we have

$$\varphi_2(\lambda) \approx \frac{1}{\Delta_k} - \frac{\lambda + \lambda_1}{C_3},$$

for some  $C_3 > 0$ . Therefore,  $\varphi_2$  is almost linear near  $-\lambda_1$  and the Newton method for root-finding will perform well, provided that it maintains  $\lambda > -\lambda_1$ . For solving the equation  $\varphi_2(\lambda) = 0$ , the Newton method generates a sequence of iterates  $\lambda^{(l)}$  by setting

$$\lambda^{(l+1)} = \lambda^{(l)} - \frac{\varphi_2(\lambda^{(l)})}{\varphi'_2(\lambda^{(l)})}, \quad l = 0, 1, \dots \quad (8.32)$$

The following algorithm is a practical implementation of (8.32).

**Algorithm 8.4** *Trust-region subproblem*

1. Initialization. Consider $\lambda^{(0)}$ and $\Delta_k > 0$ 2. For $l = 0, 1, \dots$ Factor: $B_k + \lambda^{(l)}I = R_k^\top R_k$ Solve: $R_k^\top R_k d_l = -g_k, \quad R_k^\top q_l = d_l$ Set: $\lambda^{(l+1)} = \lambda^{(l)} + \left( \frac{\ d_l\ }{\ q_l\ } \right)^2 \left( \frac{\ d_l\  - \Delta_k}{\Delta_k} \right)$ End for	◆
---	---

Algorithm 8.4 can be made practical by adding some safeguarding, for instance, when  $\lambda^{(l)} < -\lambda_1$ , the Cholesky factorization  $B_k + \lambda^{(l)}I = R_k^\top R_k$  will not exist. Anyway, an improved version of

Algorithm 8.4 always converges to a solution of (8.26). The most difficult operation in Algorithm 8.4 is the Cholesky factorization of  $B_k + \lambda^{(l)}I$ . Of course, it is not necessary to get an accurate solution for (8.26). Algorithm 8.4 is stopped after two or three iterations when an approximate solution is computed.

*Case  $q_1^T g_k = 0$ .* This case is a little more difficult because the limit (8.30) does not hold for  $\lambda_j = \lambda_1$ , and therefore, there might not be a value  $\lambda \in (-\lambda_1, \infty)$  such that  $\|d_k(\lambda)\| = \Delta_k$ . Of course, in this case, the root-finding techniques described above will not work, since there is no solution for  $\lambda$  in the open interval  $(-\lambda_1, \infty)$ . However, Theorem 8.4 assures us that the right value of  $\lambda$  lies in the interval  $[-\lambda_1, \infty)$ , so there is only one possibility:  $\lambda = -\lambda_1$ .

Observe that  $(B_k - \lambda_1 I)$  is singular. Therefore, there is a vector  $z$  such that  $\|z\| = 1$  and  $(B_k - \lambda_1 I)z = 0$ . In fact,  $z$  is an eigenvector of  $B_k$  corresponding to the eigenvalue  $\lambda_1$ , so, by the orthogonality of  $Q$ , we have  $q_j^T z = 0$  for  $\lambda_j \neq \lambda_1$ . From this property, it follows that, if we set

$$d_k(\lambda) = \sum_{j:\lambda_j \neq \lambda_1} \frac{q_j^T g_k}{\lambda_j + \lambda} q_j + \tau z \quad (8.33)$$

for any scalar  $\tau$ , we have

$$\|d_k(\lambda)\|^2 = \sum_{j:\lambda_j \neq \lambda_1} \frac{(q_j^T g_k)^2}{(\lambda_j + \lambda)^2} + \tau^2,$$

so it is always possible to choose  $\tau$  to ensure that  $\|d_k(\lambda)\| = \Delta_k$ . Therefore, the condition (8.25) holds for this choice of  $d_k$  and for  $\lambda = -\lambda_1$ .

The following proposition deals with the unconstrained minimization of quadratics with the Hessian positive semidefinite and constitutes a support for proving Theorem 8.5.

**Proposition 8.3** *Let  $m$  be the quadratic function*

$$m(d) = g^T d + \frac{1}{2} d^T B d, \quad (8.34)$$

where  $B$  is any symmetric matrix. Then, the following statements are true:

- (i)  $m$  attains a minimum if and only if  $B$  is positive semidefinite and  $g$  is in the range of  $B$ . If  $B$  is positive semidefinite, then every  $d$  satisfying  $Bd = -g$  is a global minimizer of  $m$ .
- (ii)  $m$  has a unique minimizer if and only if  $B$  is positive definite.

**Proof**

- (i) The “if” part. Since  $g$  is in the range of  $B$ , there is a  $d$  such that  $Bd = -g$ . For all  $w \in \mathbb{R}^n$ , we can write

$$\begin{aligned} m(d + w) &= g^T (d + w) + \frac{1}{2} (d + w)^T B (d + w) \\ &= \left( g^T d + \frac{1}{2} d^T B d \right) + g^T w + (Bd)^T w + \frac{1}{2} w^T B w \\ &= m(d) + \frac{1}{2} w^T B w \\ &\geq m(d), \end{aligned} \quad (8.35)$$

since  $B$  is positive semidefinite. Hence,  $d$  is a minimizer of  $m$ .

For the “only if” part, let  $d$  be a minimizer of  $m$ . Since  $\nabla m(d) = Bd + g = 0$ , it follows that  $g$  is in the range of  $B$ . Also, we have  $\nabla^2 m(d) = B$  positive semidefinite, giving the result.

- (ii) The “if” part. The same argument as in (i) suffices with the additional argument that  $w^T Bw > 0$  whenever  $w \neq 0$ . For the “only if” part, deduce as above that  $B$  is positive semidefinite. If  $B$  is not positive definite, then there is a vector  $w \neq 0$  such that  $Bw = 0$ . Therefore, from (8.35), it follows that  $m(d + w) = m(d)$ , so the minimizer is not unique, giving a contradiction.  $\blacklozenge$

### Proof of Theorem 8.5

Assume that there is  $\lambda \geq 0$  such that the conditions (8.25) are satisfied. Proposition 8.3 implies that  $d_k^*$  is a global minimum of the quadratic

$$\hat{m}_k(d) = g_k^T d + \frac{1}{2} d^T (B + \lambda I) d = m_k(d) + \frac{\lambda}{2} d^T d. \quad (8.36)$$

Since  $\hat{m}_k(d) \geq \hat{m}_k(d_k^*)$ , it follows that

$$m_k(d) \geq m_k(d_k^*) + \frac{\lambda}{2} \left( (d_k^*)^T d_k^* - d^T d \right). \quad (8.37)$$

Since  $\lambda(\Delta_k - \|d_k^*\|) = 0$  and, therefore,  $\lambda(\Delta_k^2 - (d_k^*)^T d_k^*) = 0$ , we have

$$m_k(d) \geq m_k(d_k^*) + \frac{\lambda}{2} (\Delta_k^2 - d^T d).$$

Therefore, from  $\lambda \geq 0$ , it follows that  $m_k(d) \geq m_k(d_k^*)$  for all  $d$  with  $\|d\| \leq \Delta_k$ . Therefore,  $d_k^*$  is a global minimizer of (8.24).

Conversely, we assume that  $d_k^*$  is a global solution of (8.24) and show that there is a  $\lambda \geq 0$  that satisfies (8.25). In the case  $\|d_k^*\| < \Delta_k$ ,  $d_k^*$  is an unconstrained minimizer of  $m_k$ , and so  $\nabla m_k(d_k^*) = B_k d_k^* + g_k$ ,  $\nabla^2 m_k(d_k^*) = B_k$  is positive semidefinite, and so the properties (8.25) hold for  $\lambda = 0$ .

For the remainder of the proof, assume that  $\|d_k^*\| = \Delta_k$ . Then, (8.25b) is immediately satisfied, and therefore,  $d_k^*$  also solves the constrained problem

$$\min m_k(d) \text{ subject to } \|d\| = \Delta_k.$$

By applying the optimality conditions for this problem, we find that there is a  $\lambda$  such that the Lagrangian function defined by

$$L(d, \lambda) = m_k(d) + \frac{\lambda}{2} (d^T d - \Delta_k^2)$$

has a stationary point at  $d_k^*$ . By setting  $\nabla_d L(d_k^*, \lambda)$  to zero, it follows that

$$B_k d_k^* + g_k + \lambda d_k^* = 0 \Rightarrow (B_k + \lambda I) d_k^* = -g_k, \quad (8.38)$$

so that (8.25a) holds. Since  $m_k(d) \geq m_k(d_k^*)$  for any  $d$  with  $d^T d = (d_k^*)^T d_k^* = \Delta_k^2$ , for such vectors  $d$ , we have

$$m_k(d) \geq m_k(d_k^*) + \frac{\lambda}{2} \left( (d_k^*)^\top d_k^* - d^\top d \right).$$

Now, substituting the expression of  $g_k$  from (8.38) into the above expression, we get

$$\frac{1}{2} (d - d_k^*)^\top (B_k + \lambda I) (d - d_k^*) \geq 0. \quad (8.39)$$

Since the set of directions

$$\left\{ w : w = \pm \frac{d - d_k^*}{\|d - d_k^*\|}, \text{ for some } d \text{ with } \|d\| = \Delta_k \right\}$$

is dense on the unit sphere, (8.39) suffices to prove (8.25c).

It remains to show that  $\lambda \geq 0$ . As in Nocedal and Wright (2006), since (8.25a) and (8.25c) are satisfied by  $d_k^*$ , from Proposition 8.3(i), it follows that  $d_k^*$  minimizes  $\hat{m}_k$ , so (8.37) holds. Suppose that there are only negative values of  $\lambda$  that satisfy (8.25a) and (8.25c). Then, from (8.37), we have  $m_k(d) \geq m_k(d_k^*)$  whenever  $\|d\| \geq \|d_k^*\| = \Delta_k$ . Since we already know that  $d_k^*$  minimizes  $m_k$  for  $\|d\| \leq \Delta_k$ , it follows that  $d_k^*$  is in fact the global unconstrained minimizer of  $m_k$ . From Proposition 8.3 (i), it follows that  $B_k d = -g_k$  and  $B_k$  is positive semidefinite. Therefore, conditions (8.25a) and (8.25c) are satisfied by  $\lambda = 0$ , which contradicts our assumption that only negative values of  $\lambda$  can satisfy these conditions.  $\blacklozenge$

The key to attaining the fast rate of convergence is that the solution of the trust-region subproblem is well inside the trust-region and becomes closer and closer to the true Newton step. The steps that satisfy this property are said to be *asymptotically similar to the Newton steps*. The following result applies to any algorithm of the form of Algorithm 8.1 that generates steps that are asymptotically similar to the Newton steps whenever the Newton steps satisfy the trust-region bound.

**Theorem 8.6** *Let  $f$  be Lipschitz continuously differentiable in a neighborhood of a point  $x^*$  at which the second-order sufficient conditions (Theorem 11.13) are satisfied. Suppose that the sequence  $\{x_k\}$  converges to  $x^*$  and that for all  $k$  sufficiently large the trust-region algorithm based on (8.3) with  $B_k = \nabla^2 f(x_k)$  chooses the steps  $d_k$  that satisfy the Cauchy point based on the reduction criterion (8.16) and are asymptotically similar to the Newton steps  $d_k^N$  whenever  $\|d_k^N\| \leq \Delta_k/2$ , that is,*

$$\|d_k - d_k^N\| = o(\|d_k^N\|). \quad (8.40)$$

*Then, the trust-region bound  $\Delta_k$  becomes inactive for all  $k$  sufficiently large, and the sequence  $\{x_k\}$  converges superlinearly to  $x^*$ .*

**Proof** Firstly, we seek a lower bound on the predicted reduction  $m_k(0) - m_k(d_k)$  for all the sufficiently large  $k$ . Assume that  $k$  is sufficiently large, so that  $o(\|d_k^N\|)$  in (8.40) is smaller than  $\|d_k^N\|$ . When  $\|d_k^N\| \leq \Delta_k/2$ , it follows that  $\|d_k\| \leq \|d_k^N\| + o(\|d_k^N\|) \leq 2\|d_k^N\|$ , while if  $\|d_k^N\| > \Delta_k/2$ , it follows that  $\|d_k\| \leq \Delta_k < 2\|d_k^N\|$ . Therefore, in both cases,

$$\|d_k\| \leq 2\|d_k^N\| \leq 2\|\nabla^2 f(x_k)^{-1}\| \|g_k\|,$$

and so  $\|g_k\| \geq \frac{1}{2} \|d_k\| / \|\nabla^2 f(x_k)^{-1}\|$ .

From the reduction (8.16), it follows that

$$\begin{aligned} m_k(0) - m_k(d_k) &\geq c_1 \|g_k\| \min \left( \Delta_k, \frac{\|g_k\|}{\|\nabla^2 f(x_k)\|} \right) \\ &\geq c_1 \frac{\|d_k\|}{2\|\nabla^2 f(x_k)^{-1}\|} \min \left( \|d_k\|, \frac{\|d_k\|}{2\|\nabla^2 f(x_k)\| \|\nabla^2 f(x_k)^{-1}\|} \right) \\ &= c_1 \frac{\|d_k\|^2}{4\|\nabla^2 f(x_k)\| \|\nabla^2 f(x_k)^{-1}\|^2}. \end{aligned}$$

Since  $x_k \rightarrow x^*$ , by the continuity of  $\nabla^2 f(x)$  and by the positive definiteness of  $\nabla^2 f(x^*)$ , it follows that the following bounds hold for all  $k$  sufficiently large

$$\frac{c_1}{4\|\nabla^2 f(x_k)\| \|\nabla^2 f(x_k)^{-1}\|^2} \geq \frac{c_1}{8\|\nabla^2 f(x^*)\| \|\nabla^2 f(x^*)^{-1}\|^2} \triangleq c_3,$$

where  $c_3 > 0$  is a constant. Therefore,

$$m_k(0) - m_k(d_k) \geq c_3 \|d_k\|^2,$$

for all  $k$  sufficiently large. By the Lipschitz continuity of  $\nabla^2 f(x)$  near  $x^*$  and by using Taylor's theorem, it follows that

$$\begin{aligned} |(f(x_k) - f(x_k + d_k)) - (m_k(0) - m_k(d_k))| &= \left| \frac{1}{2} d_k^T \nabla^2 f(x_k) d_k - \frac{1}{2} \int_0^1 d_k^T \nabla^2 f(x_k + td_k) d_k dt \right| \\ &\leq \frac{L}{4} \|d_k\|^3, \end{aligned}$$

where  $L > 0$  is the Lipschitz constant associated to  $\nabla^2 f(\cdot)$ . Therefore, from (8.4), it follows that

$$|r_k - 1| \leq \frac{\|d_k\|^3 (L/4)}{c_3 \|d_k\|^2} = \frac{L}{4c_3} \|d_k\| \leq \frac{L}{4c_3} \Delta_k. \quad (8.41)$$

As we know, the trust-region can be reduced only if  $r_k < 1/4$ , so, from (8.41), it follows that the sequence  $\{\Delta_k\}$  is bounded away from zero. Since  $x_k \rightarrow x^*$ ,  $\|d_k^N\| \rightarrow 0$ , and therefore, from (8.40),  $\|d_k\| \rightarrow 0$ . Therefore, for  $k$  sufficiently large, the trust-region bound is inactive, and eventually, the bound  $\|d_k^N\| \leq \Delta_k/2$  is always satisfied.

To prove the superlinear convergence, from (8.40), we have

$$\begin{aligned} \|x_k + d_k - x^*\| &\leq \|x_k + d_k^N - x^*\| + \|d_k^N - d_k\| \\ &= o(\|x_k - x^*\|^2) + o(\|d_k^N\|) = o(\|x_k - x^*\|), \end{aligned}$$

thus proving the superlinear convergence. ♦

Observe that if  $d_k = d_k^N$  for all  $k$  sufficiently large, it follows that the sequence  $\{x_k\}$  converges to  $x^*$ . Under the condition of Theorem 8.6, a reasonable implementation of the dogleg and the subspace

**Table 8.1** Performances of TRON for solving five applications from the MINPACK-2 collection

	<i>n</i>	#iter	#f	#g	#h	#Hv	#cg	cpu
A1	40,000	21	8	8	8	72	43	2.95
A2	40,000	21	8	8	8	72	73	3.50
A3	40,000	2074	1117	480	480	4320	1763	319.90
A4	40,000	15	6	6	6	54	35	4.69
A5	40,000	149	74	39	39	351	142	22.71
Total	-	2280	1213	541	541	4869	2056	353.75

In this table,  $n$  is the number of variables, #iter is the number of iterations for solving the application, #f is the number of function evaluations, #g is the number of gradient evaluations, #h is the number of the Hessian evaluations, #Hv is the number of the Hessian-vector evaluations, #cg is the number of the conjugate gradient iterations, and cpu is the CPU computing time in seconds.

minimization with  $B_k = \nabla^2 f(x_k)$  will eventually use the steps  $d_k = d_k^N$ , and therefore, the convergence is quadratic. In the case of the dogleg and of two-dimensional subspace minimization methods, the exact step  $d_k^N$  is a candidate for  $d_k$ , since  $d_k^N$  lies inside the trust-region along the dogleg path and inside the two-dimensional subspace (Nocedal & Wright, 2006).

### Numerical Study: TRON for the Unconstrained Optimization

In the following, let us present the performances of TRON (Lin & Moré, 1999) for solving five applications from the MINPACK-2 collection (see Appendix D). TRON implements a trust-region method for the solution of large bound-constrained optimization problems. The Cauchy step is generated by a gradient projection method. The search direction is computed by a preconditioned conjugate gradient method with an incomplete Cholesky factorization. The projected searches allow TRON to examine the faces of the feasible set by generating a small number of minor iterates, even for problems with a large number of variables. As a result, TRON is remarkably efficient for solving large bound-constrained optimization problems. In our numerical experiments, the large bounds on variables are fixed, thus obtaining unconstrained optimization applications. Table 8.1 shows the performances of TRON for solving these applications, each of them with 40,000 variables ( $n = nx \times ny$ ,  $nx = 200$ ,  $ny = 200$ ).

---

## 8.6 The Scaled Trust-Region

The poor scaling of the problem is when the minimizing function is highly sensitive to small changes in certain components of the variable  $x$  and relative insensitive to changes in other components of the variable  $x$ . Poor scaling of function  $f$  means that  $x^*$  lies in a narrow valley, that is, the contours of the minimizing function near  $x^*$  are eccentric ellipses. Since the trust-region around the current iterate within the model  $m_k(\cdot)$  is an adequate representation of the minimizing function  $f$ , it is easy to see that a *spherical* trust-region may not be appropriate when  $f$  is poorly scaled. Therefore, better is to use the *elliptical* trust-regions in which the axes are short in the sensitive directions and longer in the less sensitive directions. Elliptical trust-regions are defined by

$$\|Dd_k\| \leq \Delta_k,$$

where  $D$  is a diagonal matrix with positive diagonal elements. Elliptical trust-region yield to the following scaled trust-region subproblem:

$$\min_{d \in \mathbb{R}^n} m_k(d) \triangleq f(x_k) + d^T g_k + \frac{1}{2} d^T B_k d, \text{ subject to } \|Dd\| \leq \Delta_k.$$

When the minimizing function  $f$  is highly sensitive to the value of the  $i$ -th component  $x_i$  of vector  $x$ , then the corresponding diagonal element  $d_{ii}$  of  $D$  is assigned to a large value, while  $d_{ii}$  is small for less-sensitive components of  $x$ . Obviously, the diagonal elements of the scaling matrix  $D$  may be obtained from the second derivatives  $\partial^2 f / \partial x_i^2$ .

For scaled trust-region method, the Cauchy point calculation is changed according to the following algorithm.

**Algorithm 8.5** *Generalized Cauchy point*

- |    |  |
|----|--|
| 1. | Find the vector $d_k^s$ as solution of the subproblem:<br>$\min_{d \in \mathbb{R}^n} f(x_k) + g_k^T d, \quad \text{subject to } \ Dd\  \leq \Delta_k$  |
| 2. | Compute the scalar $\tau_k > 0$ that minimize $m_k(\tau d_k^s)$ subject to satisfying the trust-region bound:<br>$\tau_k = \arg \min_{\tau > 0} m_k(\tau d_k^s) \quad \text{subject to } \ \tau D d_k^s\  \leq \Delta_k$ |

◆

For this scaled trust-region version, we find that

$$d_k^s = -\frac{\Delta_k}{\|D^{-1}g_k\|} D^{-2} g_k$$

and

$$\tau_k = \begin{cases} 1, & \text{if } g_k^T D^{-2} B_k D^{-2} g_k \leq 0, \\ \min \left\{ 1, \frac{\|D^{-1}g_k\|^3}{\Delta_k g_k^T D^{-2} B_k D^{-2} g_k} \right\}, & \text{otherwise.} \end{cases}$$

The generalized Cauchy point is computed as

$$d_k^C = \tau_k d_k^s.$$

The scaling matrix  $D$  may be changed from iteration to iteration; most of the theoretical results on trust-region methods will still apply with minor modifications under condition that each diagonal element  $d_{ii}$  of  $D$  satisfies  $0 < d_{lo} \leq d_{ii} < d_{hi} < \infty$ , where  $d_{lo}$  and  $d_{hi}$  define a range  $[d_{lo}, d_{hi}]$ .

### Notes and References

The idea of trust-region method was first proposed by Levenberg (1944) and Marquardt (1963) as a technique for solving nonlinear least-squares problems. The proof of the convergence theorem of the trust-region algorithm is originally due to Powell (1975). Methods for computing the search directions within a trust-region method are described by Gay (1981) and Sorensen (1982). Convergence theory is discussed by Moré (1983). Advances in trust-region algorithms were presented by Yuan (2015). The content of this chapter is taken from Nocedal and Wright (2006), which is one of the best descriptions of the trust-region methods. The numerical experiments for solving the applications from MINPACK-2 collection were given by the package TRON (Lin & Moré, 1999), one of the most respectable software. The developments of algorithms and the corresponding

software for trust-region methods before 1982 were presented by Moré (1983). Dennis and Schnabel (1989) survey trust-region methods emphasizing important developments in literature. The general theory of the inexact trust-region methods was presented by Byrd, Schabel, and Schultz (1988). They introduced the two-dimensional subspace minimization method for trust-region and discussed the case of indefinite  $B$  to ensure stronger convergence results than that given by Theorems 8.3 and 8.4. An exhaustive presentation of the state of the art in trust-region methods both for unconstrained and constrained optimization could be found in the monograph of Conn, Gould, and Toint (2000). For the convergence of trust-region algorithms for unconstrained minimization without derivatives, see Powell (2011).

In our presentation, the trust-region method was defined in Euclidian norm, but it could be defined in other norms as  $\|d_k\|_1 \leq \Delta_k$  or  $\|d_k\|_\infty \leq \Delta_k$ , or scaled as  $\|Dd_k\|_1 \leq \Delta_k$  or  $\|Dd_k\|_\infty \leq \Delta_k$ , where  $D$  is a diagonal positive definite matrix. The most advantageous is the  $\infty$ -norm, in which case the feasible region is the rectangular box defined by  $x_k + d \geq 0$ ,  $d \geq -\Delta_k e$ , and  $d \leq \Delta_k e$ , where  $e = [1, 1, \dots, 1]^T$ . In this case, the solution of the subproblem is easily computed by using the simple bound-constrained optimization (see Chap. 12).

The trust-region algorithm is also implemented in filterSQP (Fletcher & Leyffer, 1998) which is a sequential quadratic programming solver with a filter to promote global convergence and in KNITRO/INTERIOR-CG (Byrd, Hribar, & Nocedal, 1999) with sequential quadratic programming.



# Direct Methods for Unconstrained Optimization

9

Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x), \quad (9.1)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is bounded from below on  $\mathbb{R}^n$  and  $n$  is relatively small, let us say  $n \leq 50$ . Many algorithms have been proposed to solve this problem, but in this chapter, we are interested in dealing with the case in which the derivatives of this function are unavailable, impractical to obtain, or unreliable. The optimization methods that use only the function values are called *derivative-free optimization* or *direct search* methods or *zeroth-order* methods.

Excellent reviews and perspectives with an emphasis on highlighting recent developments of the derivative-free optimization methods both for unconstrained and constrained optimization with deterministic, stochastic, or structured objectives were given by Kolda, Lewis, and Torczon (2003), Rios and Shainidis (2013), and Larson, Menickelly, and Wild (2019). A review of the derivative-free algorithms followed by a systematic comparison of 22 related implementations using a test set of 502 problems was given by Rios and Shainidis (2013). Their conclusion is not definitive. They emphasize that the ability of all these solvers to obtain good solutions diminishes with the increasing problem size. Besides, attaining the best solutions even for small problems is a challenge for most current derivative-free solvers, and there is no single solver whose performance dominates the performances of all the others. The dimensionality of the problems and the non-smoothness rapidly increase the complexity of the search and decrease the performances of all the solvers. In conclusion, there is a large diversity of derivative-free optimization methods with an impressive development. More details on derivative-free methods can be found in Andrei (2021a).

Addressing the optimization problems that arise from scientific engineering, from artificial intelligence applications, or from some other areas of activity for which the objective and the constraints are available as a mixture of a mathematical model and the output of a black-box or a simulation oracle that does not provide derivative information, this chapter has a special position in the architecture of the book. From the multitude of derivative-free optimization algorithms, we have selected only the algorithm of Nelder-Mead (NELMED) (see Nelder & Mead, 1965), Powell (NEWUOA) (see Powell, 2003, 2004, 2006), and Andrei (DEEPS) (see Andrei, 2021a).

## 9.1 The NELMED Algorithm

The Nelder-Mead algorithm has numerous interpretations. In the following, one of the simplest but sufficiently complete will be discussed. This method is based on simplexes. A simplex in  $\mathbb{R}^n$  is a  $n + 1$  element set of  $n$ -dimensional vertices with a nondegenerate set of edges. More exactly, a simplex  $S \subset \mathbb{R}^n$  is a set of  $n + 1$  vertices  $\{x_1, \dots, x_{n+1}\} \subset \mathbb{R}^n$  connected by the edges

$$E = (x_2 - x_1, x_3 - x_1, \dots, x_{n+1} - x_1)$$

which form a basis of  $\mathbb{R}^n$ , i.e.,  $\text{span}(E) = \mathbb{R}^n$ .

At iteration  $k$ , the Nelder-Mead simplex algorithm applied to the function  $f$  from (9.1) maintains an ordered set of simplex vertices  $(x_1^k, \dots, x_{n+1}^k)$ . The ordering of the simplex vertices is

$$f(x_1^k) \leq f(x_2^k) \leq \dots \leq f(x_{n+1}^k). \quad (9.2)$$

In this context, the vertex  $x_1^k$  with the lowest function value is referred to as the *lowest vertex*. The vertex with the highest value is the *highest vertex*.

At the beginning of each iteration, the *centroid* point is computed as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i^k.$$

At each step, a *trial point* is computed. Each iteration may have one of the following two outcomes: (1) one of the computed trial points with a lower function value replaces the highest vertex or (2) a shrink operation is performed.

The Nelder-Mead simplex algorithm uses four different trial points  $x_r$ ,  $x_e$ ,  $x_{oc}$ , and  $x_{ic}$  as follows:

$$1. \text{ Reflection } (\rho > 0) : \quad x_r^k = \bar{x} + \rho(\bar{x} - x_{n+1}^k), \quad (9.3)$$

$$2. \text{ Expansion } (\eta > \max\{1, \rho\}) : \quad x_e^k = \bar{x} + \eta(x_r^k - \bar{x}), \quad (9.4)$$

$$3. \text{ Outside contraction } (0 < \gamma < 1) : \quad x_{oc}^k = \bar{x} + \gamma(x_r^k - \bar{x}), \quad (9.5)$$

$$4. \text{ Inside contraction } (0 < \gamma < 1) : \quad x_{ic}^k = \bar{x} - \gamma(\bar{x} - x_{n+1}^k). \quad (9.6)$$

If none of the trial points yields improvement of the minimizing function, then a *shrink* operation is performed. Standard values of the parameters in (9.3, 9.4, 9.5, and 9.6) are  $\rho = 1$ ,  $\eta = 2$ ,  $\gamma = 1/2$ . A shrink operation transforms the simplex vertices  $\{x_2^k, \dots, x_{n+1}^k\}$  according to the following rule:

$$\tilde{x}_i^k = x_1^k + \sigma(x_i^k - x_1^k), \quad (9.7)$$

where  $0 < \sigma < 1$ . ( $\sigma = 1/2$ ).

One iteration of the Nelder-Mead algorithm is presented as Algorithm 9.1.

**Algorithm 9.1** Nelder-Mead algorithm (NELMED) (one iteration)

1. *Order.* Order the simplex vertices according to (9.2)
2. Compute the centroid  $\bar{x}$
3. *Reflect.* Compute  $x_r^k$  as in (9.3)
4. If  $f(x_1^k) \leq f(x_r^k) < f(x_n^k)$ , then set  $x_{n+1}^k = x_r^k$
5. *Expand.* If  $f(x_r^k) < f(x_1^k)$ , then compute  $x_e^k$  as in (9.4). If  $f(x_e^k) < f(x_r^k)$ , then set  $x_{n+1}^k = x_e^k$ , else set  $x_{n+1}^k = x_r^k$
6. *Contract.* If  $f(x_r^k) \geq f(x_n^k)$ , then
 

*Contract outside:*  
If  $f(x_n^k) \leq f(x_r^k) < f(x_{n+1}^k)$ , then compute  $x_{oc}^k$  as in (9.5). If  $f(x_{oc}^k) \leq f(x_r^k)$ , then set  $x_{n+1}^k = x_{oc}^k$   
else  
    Go to step 7  
End if  
*Contract inside:*  
If  $f(x_r^k) \geq f(x_{n+1}^k)$ , then compute  $x_{ic}^k$  as in (9.6). If  $f(x_{ic}^k) < f(x_{n+1}^k)$ , then set  $x_{n+1}^k = x_{ic}^k$   
else  
    Go to step 7  
End if
7. *Shrink.* Apply (9.7) to the vertices of the current simplex ◆

The hope is that the successive decreases of the values of the function  $f$  at the simplex vertices eventually lead to the convergence to  $x^*$ . The convergence of the algorithm was established only for problems with a small number of variables (Lagarias, Reeds, Wright, & Wright, 1998).

Since the algorithm is based only on comparisons among the function values, then a reasonable stopping criterion is  $|f(x_{n+1}^k) - f(x_1^k)| \leq \epsilon$ , where  $\epsilon$  is a positive constant, sufficiently small. Another stopping criterion is based on the volume of the simplex  $V(S_k) \leq \epsilon$ , where  $V(S_k)$  is computed as

$$V(S_k) = \frac{|\det(E_k)|}{n!}.$$

As regards the initial simplex, a practical choice is to compute the vertices as  $x_{i+1}^0 = x_1^0 + \lambda_i e_i$ ,  $i = 1, \dots, n$ , where  $x_1^0$  is the given starting point  $x_0$ ,  $\lambda_i$  for  $i = 1, \dots, n$ , are given scalars, and  $e_i$  are the unit vectors corresponding to the  $i$ -th coordinate axis. In this case, the volume of the initial simplex can be computed as

$$V(S_0) = \frac{1}{n!} \prod_{i=2}^{n+1} \|x_i^0 - x_1^0\|.$$

Several details on the properties and convergence of the Nelder-Mead algorithm can be found in Lagarias, Reeds, Wright, and Wright (1998). A convergent variant of the Nelder-Mead algorithm was given by Price, Coope, and Byatt (2002).

---

## 9.2 The NEWUOA Algorithm

NEWUOA is an iterative algorithm which seeks the least value of a function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ , when  $f$  can be computed for any vector of variables  $x$ . A short description of NEWUOA is as follows. At the

beginning of each iteration, the NEWUOA algorithm uses a quadratic model  $Q$  which is embedded into a trust region procedure for adjusting the variables. The model  $Q$  is revised at each iteration. When  $Q$  is revised, the new  $Q$  interpolates  $f$  at  $m$  points, where the recommended value of  $m$  is  $2n + 1$ . The remaining freedom in the new  $Q$  is taken up by minimizing the Frobenius norm of the change to  $\nabla^2 Q$ . Only one interpolating point is altered at each iteration. Therefore, the amount of work per iteration is only of order  $(m + n)^2$ .

At the beginning of each iteration, the model  $Q(x)$  has to satisfy only  $m$  interpolating conditions  $Q(x_i) = f(x_i)$ ,  $i = 1, \dots, m$ , where the position of different points  $x_i$ ,  $i = 1, \dots, m$ , is generated automatically. The algorithm implements a technique that is suggested by the symmetric Broyden method for updating  $\nabla^2 Q$  when the first derivatives of  $f$  are available. Let the old model  $Q_{old}$  be known and the new model  $Q_{new}$  be asked to satisfy some conditions that are compatible and that leave some freedom in the parameters of  $Q_{new}$ . The technique tries to minimize  $\|\nabla^2 Q_{new} - \nabla^2 Q_{old}\|_F$ , where  $\|\cdot\|_F$  is the Frobenius norm. The conditions on the new model  $Q = Q_{new}$  are the interpolating equations  $Q(x_i) = f(x_i)$ ,  $i = 1, \dots, m$ . Thus,  $\nabla^2 Q_{new}$  is uniquely defined and  $Q_{new}$  itself is also unique.

At the beginning of the current iteration, the variables are updated as  $x_{new} = x_{opt} + d$ , where  $x_{opt}$  corresponds to  $f(x_{opt})$ , the least calculated value of  $f$  so far. If the error  $|f(x_{new}) - Q_{old}(x_{new})|$  is relatively small, then the model predicted well the new value of  $f$ , even if the errors of the approximation  $\nabla^2 Q \approx \nabla^2 f$  are substantial. On the other hand, if  $|f(x_{new}) - Q_{old}(x_{new})|$  is relatively large, then by satisfying  $Q_{new}(x_{new}) = f(x_{new})$ , the updating technique should improve the accuracy of the model. The change in the variables  $d$  is an approximate solution to the trust region subproblem

$$\min Q(x_{opt} + d) \quad \text{subject to} \quad \|d\| \leq \Delta.$$

The algorithm uses a number of parameters:  $\rho_{beg}$ ,  $\rho_{end}$ ,  $\rho_{old}$ ,  $\rho_{new}$ , and  $\Delta$ . The initial interpolating points  $x_i$ ,  $i = 2, \dots, m$ , include  $x_0$ , while the other points have the property  $\|x_i - x_0\|_\infty = \rho_{beg}$ . The parameter  $\rho_{end}$ , which has to satisfy  $\rho_{end} \leq \rho_{beg}$ , should have the magnitude of the required accuracy in the final values of variables. The parameter  $\rho$  is a lower bound on the trust region radius  $\Delta$  from the interval  $[\rho_{end}, \rho_{beg}]$ . The purpose of  $\rho$  is to maintain enough distance between the interpolating points. The value of  $\Delta$  is revised on most iterations.

On steps, the algorithm NEWUOA is as follows:

#### Algorithm 9.2 Powell algorithm (NEWUOA)

1.	Select the initial interpolating points. Let $x_{opt}$ be an initial point where $f$ is the least
2.	Compute $d$ by approximately minimizing $Q(x_{opt} + d)$ subject to $\ d\  < \Delta$
3.	If $\ d\  \geq \rho/2$ , then go to step 4; otherwise, go to step 10
4.	Calculate $f(x_{opt} + d)$ and set $r = (f(x_{opt}) - f(x_{opt} + d))/(Q(x_{opt}) - Q(x_{opt} + d))$ . Thus, $\Delta$ is revised subject to $\Delta \geq \rho$ . The parameter $move$ is set to zero or to the index of the interpolating point that will be dropped next
5.	If $move > 0$ , then $Q$ is modified so that $Q$ interpolates $f$ at $x_{opt} + d$ instead of $x_{move}$ . If $f(x_{opt} + d) < f(x_{opt})$ , then $x_{opt}$ is overwritten by $x_{opt} + d$ . Go to step 6
6.	If $r \geq 0.1$ , then go to step 2; otherwise, go to step 7
7.	Let $x_{move}$ be the current interpolating point that maximizes the distance $dist = \ x_{move} - x_{opt}\ $
8.	If $dist \geq 2\Delta$ , then $x_{move}$ is replaced by $x_{opt} + d$ , set $r = 1$ and go to step 5. Otherwise, if $\max\{\ d\ , \Delta\} \leq \rho$ and $r \leq 0$ , go to step 9
9.	If $\rho = \rho_{end}$ , then compute $f(x_{opt} + d)$ and stop. Otherwise, reduce $\rho$ about a factor of 10 subject to $\rho \geq \rho_{end}$ . Reduce $\Delta = \max[\rho_{old}/2, \rho_{new}]$ and go to step 2
10.	If three recent values of $\ d\ $ and $f - Q$ are small, then go to step 9; otherwise, reduce $\Delta$ by a factor of 10 or to its lower bound $\rho$ . Set $r = -1$ . Set $x_{move}$ to be the current interpolating point that maximizes the distance $dist = \ x_{move} - x_{opt}\ $ and go to step 8

Details on NEWUOA are given in Powell (2002, 2003, 2004, 2006).

### 9.3 The DEEPS Algorithm

In the following, we present a new derivative-free random search method at two levels, which is completely different from the known randomized methods for the unconstrained optimization. Roughly speaking, our method is close to pure random search, but extended at two levels (Andrei, 2021a)

Suppose that all the variables are in a domain  $D \subset \mathbb{R}^n$  defined by some bounds on variables. Some of these bounds are imposed by the physical constraints which define the problem. Others are artificially introduced, for being used in our algorithm. Anyway, we consider that all the variables are bounded by some known bounds, the same for all of them. Suppose that the minimum value of the function  $f$  is known as  $f^{opt}$ . Assume that the function  $f$  is continuous and bounded below on  $\mathbb{R}^n$ .

The idea of the algorithm is as follows. Set the current iteration  $iter = 1$  and assign a value to the maximum number of iterations admitted by the algorithm as  $maxiter$ . Suppose that the searching of the minimum of the function  $f$  starts from the initial point  $x_0 \in \mathbb{R}^n$ . Consider an initial domain  $D \subset \mathbb{R}^n$  defined by its bounds (lower and upper) to each of the  $n$  variables. These limits may be the same for all the variables, let us say  $lobnd$  and  $upbnd$ . In other words, if on components we have  $x_i = [x_i^1, \dots, x_i^n]$ , then assume that  $lobnd \leq x_i^l \leq upbnd$  for all  $l = 1, \dots, n$ . Of course, the bounds  $lobnd$  and  $upbnd$  can be different for different components of the variables  $x_i$ , but in the following, we assume that they are the same for each component  $x_i^l, l = 1, \dots, n$ . We emphasize that these bounds  $lobnd$  and  $upbnd$  may be the bounds defining the optimization problem and imposed by the engineering constructive specifications of the minimizing problem.

Around the initial point  $x_0$ , we randomly generate over  $D$  a number  $N$  of trial points,  $x_1, \dots, x_N$ , where  $x_j \in \mathbb{R}^n$ , for  $j = 1, \dots, N$ . If  $N = n + 1$ , then these points define a simplex in  $\mathbb{R}^n$ , i.e., the convex hull of its  $n + 1$  points  $x_1, \dots, x_{n+1}$ . Otherwise, if  $N \neq n + 1$ , we say that we have a geometric structure which we call a *complex* in  $\mathbb{R}^n$ . Obviously,  $N$  may be larger or smaller than the number of variables  $n$ , or even equal to  $n$ . Evaluate the minimizing function  $f$  in these points, thus obtaining  $f(x_1), \dots, f(x_N)$ .

Now, for every trial point  $x_j, j = 1, \dots, N$ , let us define the local domains  $D_j \subset \mathbb{R}^n$  by specifying the bounds, let us say  $lobndc$  and  $upbndc$ . Of course, these bounds may be different for every trial point, but in our development, we consider them equal for every trial point. Using these bounds  $lobndc$  and  $upbndc$  around each trial point  $x_j$ , we randomly generate  $M$  local trial points  $x_j^1, \dots, x_j^M$ , where  $x_j^i \in \mathbb{R}^n$ , for  $i = 1, \dots, M$ . In other words, around the trial point  $x_j$ , the local trial points  $x_j^1, \dots, x_j^M$  are bounded by the limits  $lobndc$  and  $upbndc$ . Usually, the local domains  $D_j$  are smaller than the domain  $D$ , but this is not compulsory. Now, evaluate the function  $f$  in these local trial points, thus obtaining  $f(x_j^1), \dots, f(x_j^M)$  for any  $j = 1, \dots, N$ .

With this, determine the local point  $x_j^k$  corresponding to the minimum value from the set  $\{f(x_j^1), \dots, f(x_j^M)\}$ , i.e.,  $f(x_j^k) = \min\{f(x_j^1), \dots, f(x_j^M)\}$ . The following decision is now taken: if  $f(x_j^k) < f(x_j)$ , then replace the trial point  $x_j$  with the local trial point  $x_j^k$  and the value  $f(x_j)$  with  $f(x_j^k)$ .

The above procedure is repeated for every trial point  $x_j, j = 1, \dots, N$ . At the end of this cycle, if the case, the trial points  $x_j, j = 1, \dots, N$ , are replaced by the local trial points  $x_j^k, j = 1, \dots, N$ , for which the function value  $f(x_j^k)$  might be smaller than  $f(x_j)$ . Let us denote these new trial points by  $y_1, \dots, y_N$ ,

i.e.,  $y_j = x_j^k$ ,  $j = 1, \dots, N$ , and the function values in these points by  $f(y_1), \dots, f(y_N)$ , i.e.,  $f(y_j) = f(x_j^k)$ ,  $j = 1, \dots, N$ .

Now, determine the point  $y_k$  corresponding to the minimum values of the minimizing function, i.e.,  $f(y_k) = \min \{f(y_1), \dots, f(y_N)\}$ .

For  $j = 1, \dots, N, j \neq k$ , where  $k$  is the index corresponding to the minimum value  $f(y_k)$ , compute the middle point  $z_j = (y_k + y_j)/2$  and evaluate the minimizing function in this point, thus obtaining  $f(z_j)$ . For  $j = 1, \dots, N, j \neq k$ , if  $f(z_j) < f(y_j)$ , then replace the point  $y_j$  with  $z_j$  and the value  $f(y_j)$  with  $f(z_j)$ ; otherwise, do not replace them. Set  $z_k = y_k$  and  $f(z_k) = f(y_k)$ . At this step of the algorithm, another set of new trial points denoted as  $z_1, \dots, z_N$  is obtained, for which the function values  $f(z_1), \dots, f(z_N)$  might be smaller than the previous values of the minimizing function  $f$ .

Now, determine  $f^{\max} = \max \{f(z_1), \dots, f(z_N)\}$ , corresponding to the trial point  $z^{\max}$  from the set of points  $\{z_1, \dots, z_N\}$ . Similarly, determine  $f^{\min} = \min \{f(z_1), \dots, f(z_N)\}$ , corresponding to the trial point  $z^{\min}$  from the set of points  $\{z_1, \dots, z_N\}$ . Compute the middle point  $z^m = (z^{\max} + z^{\min})/2$  and the function value in this point  $f(z^m)$ .

In the following, for  $j = 1, \dots, N$ , where the index  $j$  is different from the indices corresponding to the minimum and maximum points  $z^{\min}$  and  $z^{\max}$ , respectively, if  $f(z^m) < f(z_j)$ , then compute the reflected point  $z' = \alpha z^m - z_j$ , and if  $f(z') < f(z_j)$ , then replace  $z_j$  by the reflected point  $z'$ , i.e., set  $z_j = z'$  and  $f(z_j) = f(z')$ . Notice that  $\alpha$  is a positive coefficient known as the *reflection coefficient*, often selected as  $\alpha \in [1, 2]$ , usually close to 2.

At the end of this cycle, some of the trial points  $z_j, j = 1, \dots, N$ , have been replaced by their reflected points. Therefore, we have a new set of trial points  $\{z_1, \dots, z_N\}$  for which the function values  $\{f(z_1), \dots, f(z_N)\}$  might be smaller than the previous values.

Intensive numerical experiments have shown that the values of the minimizing function are significantly reduced in the first few iterations, after which the algorithm is stalling, i.e., the function values are very slowly reduced along the iterations. In order to accelerate the algorithm, the following step is implemented. If the difference between the minimum values of the minimizing function at two successive iterations is smaller than a positive, small enough threshold  $\delta$ , then a simple line-search with a positive step  $\beta$  or with a negative step  $\beta$  is initiated from the current minimum point along the direction determined by the minimum point and the second minimum point. Thus, if the function value is reduced, then the minimum point is replaced by the new trial point. Again, at the end of this step, it is possible for the minimum trial point  $z^{\min}$  to be replaced by another point for which the function value is smaller. Therefore, another set of trial points  $\{z_1, \dots, z_N\}$  is obtained, for which the function values  $\{f(z_1), \dots, f(z_N)\}$  might be smaller than the previous values.

Now, determine  $f^{\max} = \max \{f(z_1), \dots, f(z_N)\}$  corresponding to the trial point  $z^{\max}$  from the set of trial points  $\{z_1, \dots, z_N\}$ . Similarly, determine  $f^{\min} = \min \{f(z_1), \dots, f(z_N)\}$  corresponding to the trial point  $z^{\min}$  from the set of trial points  $\{z_1, \dots, z_N\}$ .

From this set of trial points  $\{z_1, \dots, z_N\}$ , at the iteration  $iter$ , determine the point  $z_k$  corresponding to the minimum values of the minimizing function, i.e.,  $f(z_k) \equiv f_{iter} = \min \{f(z_1), \dots, f(z_N)\}$ .

At the iteration  $iter$ , compute the difference:  $d_{iter} = |f(z_k) - f^{opt}| \equiv |f_{iter} - f^{opt}|$ . With these, the following decisions are implemented:

- If  $|f^{\max}| > B$ , where  $B$  is a positive constant sufficiently large, then reduce the bounds  $lobnd$  and  $upbnd$ , as, for example,  $lobnd/2$  and  $upbnd/2$ , respectively. Set  $iter = iter + 1$  and go to generate around the initial point  $x_0$  a new set of trial points by using the reduced bounds  $lobnd/2$  and  $upbnd/2$ . At the end of this decision, the bounds  $lobnd$  and  $upbnd$  are likely to be reduced.

- If  $d_{iter} \neq d_{iter-1}$ , then, if  $d_{iter} > \epsilon$ , set  $iter = iter + 1$  and  $d_{iter-1} = d_{iter}$ . If  $iter > \max iter$ , then go to the final step, else go to the next step. If  $d_{iter} = d_{iter-1}$ , then reduce the bounds of the local domain, i.e., set  $lobndc = lobndc/2$  and  $upbndc = upbndc/2$ . If  $d_{iter} > \epsilon$ , then set  $iter = iter + 1$ , and if  $iter > \max iter$ , then go to the final step, else go to the next step. At the end of this decision, the lower and upper bounds of the local domains  $lobndc$  and  $upbndc$  might be reduced.
- Here we have the next step, where the following decision is taken: if  $|f_{iter} - f_{iter-1}| < t$ , where  $t$  is a prespecified small threshold, then reduce the complex, i.e., for  $j = 1, \dots, N$ , compute  $z_j = z^{\min} + (z_j - z^{\min})/2$ .

At this iteration, set  $f_{iter-1} = f_{iter}$ , and for  $j = 1, \dots, N$ , set  $x_j = z_j$  and go to the next iteration by generating  $M$  local trail points  $x_j^1, \dots, x_j^M$  around each new trial points  $x_j, j = 1, \dots, N$ .

The final step: improve the minimum point obtained as above by randomly selecting a number  $Q$  of points in a domain  $C$  defined by the bounds  $lobndcc$  and  $upbndcc$ , smaller than  $lobndc$  and  $upbndc$ , respectively. Evaluate the minimizing function in these  $Q$  points, and select the minimum value from them, which corresponds to the solution of the problem.

More exactly, on steps, the algorithm may be described as follows:

**Algorithm 9.3 Andrei algorithm (DEEPS)**

1.	<i>Initialization.</i> Select: an initial point $x_0$ , the number $N$ of a set of trial points and the number $M$ of the set of local trial points around each trial point. Select $\epsilon > 0$ small enough. Select the bounds $lobnd$ and $upbnd$ which determine the set of trial points, the bounds $lobndc$ and $upbndc$ which determine the set of local trail points, the bounds $lobndcc$ and $upbndcc$ which determine the final trial points around the minimum, as well as the threshold $t$ small enough. Select $B$ a positive constant sufficiently large. Select a value for the coefficient $\alpha$ used in the reflection of points and a value for the coefficient $\beta$ used in a line-search along the direction determined by the minimum trial point and by the second minimum trial point. Select a value for the parameter $\delta$ used in initiating the above line-search. Set $iter = 1$ and $\maxiter$ as the maximum number of iterations admitted by the algorithm
2.	Around $x_0$ , randomly generate $N$ trial points $x_1, \dots, x_N$ in a domain $D$ defined by the lower bound $lobnd$ and by the upper bound $upbnd$ on variables
3.	Evaluate the minimizing function $f$ in the trial points $f(x_1), \dots, f(x_N)$
4.	For $j = 1, \dots, N$ , around $x_j$ do: <ol style="list-style-type: none"> <li>Generate <math>M</math> points <math>x_j^1, \dots, x_j^M</math> in the domains <math>D_j</math> defined by the bounds <math>lobndc</math> and <math>upbndc</math>, smaller than the bounds of the domain <math>D</math>. Evaluate the function in these points, thus obtaining the values <math>f(x_j^1), \dots, f(x_j^M)</math></li> <li>Determine the point <math>x_j^k</math> corresponding to the minimum values <math>f(x_j^k) = \min \{f(x_j^1), \dots, f(x_j^M)\}</math></li> <li>If <math>f(x_j^k) &lt; f(x_j)</math>, then replace the trial point <math>x_j</math> by <math>x_j^k</math> and the value <math>f(x_j)</math> by <math>f(x_j^k)</math></li> </ol> End for <i>[By these replacements, at the end of this cycle, another set of new trial points denoted as <math>y_1, \dots, y_N</math> is obtained, for which the function values are <math>f(y_1), \dots, f(y_N)</math>.]</i>
5.	Determine the point $y_k$ corresponding to the minimum values of the minimizing function: $f(y_k) = \min \{f(y_1), \dots, f(y_N)\}$
6.	For $j = 1, \dots, N, j \neq k$ , do: <ol style="list-style-type: none"> <li>Compute the point <math>z_j = (y_k + y_j)/2</math></li> <li>Evaluate the minimizing function in this point, thus obtaining <math>f(z_j)</math></li> <li>If <math>f(z_j) &lt; f(y_j)</math>, then replace the point <math>y_j</math> by <math>z_j</math> and the value <math>f(y_j)</math> by <math>f(z_j)</math></li> </ol> End for

7.	<p>Set <math>z_k = y_k</math> and <math>f(z_k) = f(y_k)</math>  <math>[At the end of steps 6 and 7, another set of new trial points denoted as <math>z_1, \dots, z_N</math> is obtained, for which the function values are <math>f(z_1), \dots, f(z_N)</math>.]</math></p>
8.	<p>Determine <math>f^{\max} = \max \{f(z_1), \dots, f(z_N)\}</math> and the corresponding trial point <math>z^{\max}</math> Determine <math>f^{\min} = \min \{f(z_1), \dots, f(z_N)\}</math> and the corresponding trial point <math>z^{\min}</math></p>
9.	<p>Compute the middle point <math>z^m = (z^{\max} + z^{\min})/2</math> and the function value in this point <math>f(z^m)</math></p>
10.	<p>For <math>j = 1, \dots, N</math>, where the index <math>j</math> is different from the indices corresponding to the minimum and maximum points <math>z^{\min}</math> and <math>z^{\max}</math> respectively, do:</p> <p>If <math>f(z^m) &lt; f(z_j)</math>, then</p> <p>    Compute the reflected point <math>z^r = \alpha z^m - z_j</math> and <math>f(z^r)</math></p> <p>    If <math>f(z^r) &lt; f(z_j)</math>, then</p> <p>        Set <math>z_j = z^r</math> and <math>f(z_j) = f(z^r)</math></p> <p>    End if</p> <p>End if</p> <p>End for</p> <p><math>[At the end of this cycle another set of trial points <math>z_1, \dots, z_N</math> is obtained, for which the function values are <math>f(z_1), \dots, f(z_N)</math>.]</math></p>
11.	<p>At iteration <math>\text{iter}</math> determine:</p> <p><math>f_{\text{iter}}^{\max} = \max \{f(z_1), \dots, f(z_N)\}</math> and the corresponding trial point <math>z^{\max}</math></p> <p><math>f_{\text{iter}}^{\min} = \min \{f(z_1), \dots, f(z_N)\}</math> and the corresponding trial point <math>z^{\min}</math></p>
12.	<p>If <math> f_{\text{iter}}^{\min} - f_{\text{iter}-1}^{\min}  \leq \delta</math>, then do:</p> <p>(a) In the current set of trial points determine the second minimum trial point <math>z^{\min_s}</math></p> <p>(b) Compute the point <math>z^s = z^{\min} + \beta(z^{\min} - z^{\min_s})</math></p> <p>(c) Compute <math>f^s = f(z^s)</math></p> <p>If <math>f^s &lt; f_{\text{iter}}^{\min}</math>, then</p> <p>    Set <math>z^{\min} = z^s, f_{\text{iter}}^{\min} = f^s</math> and go to step 13</p> <p>Else</p> <p>    (d) Compute the point <math>z^s = z^{\min} - \beta(z^{\min} - z^{\min_s})</math></p> <p>    (e) Compute <math>f^s = f(z^s)</math></p> <p>If <math>f^s &lt; f_{\text{iter}}^{\min}</math>, then</p> <p>    Set <math>z^{\min} = z^s, f_{\text{iter}}^{\min} = f^s</math> and go to step 13</p> <p>Else</p> <p>    Go to step 13</p> <p>End if</p> <p>End if</p> <p>End if</p>
13.	<p>If <math>upbnd &lt; 1</math> and <math>lobnd &gt; -1</math>, then go to step 15</p>
14.	<p>If <math> f_{\text{iter}}^{\max}  &gt; B</math>, then:</p> <p>(a) Reduce the bounds as <math>lobnd/2</math> and <math>upbnd/2</math></p> <p>(b) Set <math>\text{iter} = \text{iter} + 1</math>. If <math>\text{iter} &gt; \max \text{ iter}</math>, then go to step 19, else go to step 2</p>
15.	<p>At iteration <math>\text{iter}</math> compute the difference: <math> f_{\text{iter}}^{\min} - f^{\text{opt}} _{\text{iter}}</math></p>
16.	<p>If <math> f_{\text{iter}}^{\min} - f^{\text{opt}} _{\text{iter}} \neq  f_{\text{iter}-1}^{\min} - f^{\text{opt}} _{\text{iter}-1}</math>, then</p> <p>    If <math> f_{\text{iter}}^{\min} - f^{\text{opt}} _{\text{iter}} &gt; \epsilon</math>, then</p> <p>        Set <math>\text{iter} = \text{iter} + 1</math> and <math> f_{\text{iter}-1}^{\min} - f^{\text{opt}} _{\text{iter}-1} =  f_{\text{iter}}^{\min} - f^{\text{opt}} _{\text{iter}}</math></p> <p>        If <math>\text{iter} &gt; \max \text{ iter}</math>, then go to step 19, else go to step 17</p> <p>    Else</p> <p>        Go to step 19</p> <p>    End if</p> <p>Else</p> <p>    Set <math>lobndc = lobndc/2</math> and <math>upbndc = upbndc/2</math></p> <p>    If <math> f_{\text{iter}}^{\min} - f^{\text{opt}} _{\text{iter}} &gt; \epsilon</math>, then</p> <p>        Set <math>\text{iter} = \text{iter} + 1</math></p> <p>        If <math>\text{iter} &gt; \max \text{ iter}</math>, then go to step 19, else go to step 17</p>

	Else Go to step 19 End if End if
17.	If $ f_{\text{iter}}^{\min} - f_{\text{iter}-1}^{\min}  < t$ , then reduce the complex: For $j = 1, \dots, N$ compute $z_j = z^{\min} + (z_j - z^{\min})/2$ End for End if
18.	Set $f_{\text{iter}-1}^{\min} = f_{\text{iter}}^{\min}$ For $j = 1, \dots, N$ , set $x_j = z_j$ and go to step 4
19.	<i>Improve the search.</i> Around the minimum point $z_k$ obtained so far, randomly select a number $Q$ of points in the domain $C$ defined by the bounds $\text{lobndcc}$ and $\text{upbndcc}$ , smaller than $\text{lobndc}$ and $\text{upbndc}$ , respectively. Evaluate the minimizing function in these $Q$ points, and from them, select the minimum as the solution of the problem ◆

To ensure the convergence, the algorithm implements three actions in steps 14, 16, and 17.

- At step 14, if the maximum absolute value of the minimizing function  $f$  computed in the current trial points is greater than a prespecified value  $B$  sufficiently large, let us say  $B = 10^5$ , then the bounds  $\text{lobnd}$  and  $\text{upbnd}$  are halved. Therefore, the algorithm generates a sequence of points in bounded domains.
- Based on the differences between the minimum values of the minimizing function and the known optimum value at two successive iterations, step 16 implements a procedure for halving the bounds  $\text{lobndc}$  and  $\text{upbndc}$  of the local domains, thus shrinking the local search domains.
- If the difference between the minimum values of the minimization function at two successive iterations is smaller than a threshold  $t$ , then the algorithm reduces the complex in step 17.

The algorithm keeps a balance between the threshold  $\delta$  for initiating the line-searches in step 12 and the threshold  $t$  for reducing the complex in step 17. In general, the values for  $\delta$  are selected as being smaller than those of  $t$ . Usually,  $\delta = 10^{-4}$  and  $t = 10^{-2}$ , but some other values are acceptable.

Numerical experiments have shown that the algorithm has two phases: the *reduction phase* and the *stalling* one. In the reduction phase, for some problems, the function values are significantly reduced for the first few iterations. After that, the algorithm gets into the stalling phase, where the function values are reduced very slowly. Both thresholds  $\delta$  and  $t$  are introduced to accelerate the algorithm, i.e., to diminish the stalling. Depending on the value of the threshold  $\delta$ , a line-search is initiated from the minimum point along the direction determined by the minimum point and the second minimum point, corresponding to the current set of trial points. Similarly, depending on the value of the threshold  $t$ , the algorithm reduces the complex.

In the following, let us discuss and present some key elements of the algorithm by illustrating the evolution of the *maximum distance among the trial points*, as well as the evolution of the *maximum distance among the trial points and the local trial points* along the iterations.

At iteration  $\text{iter}$ , for any two points  $x_i$  and  $x_j$ , from the set of trials points, let us define the distance between these points as  $d_{ij}^{\text{iter}} = \|x_i - x_j\|_2$ , for  $i, j = 1, \dots, N, i \neq j$ . Of course, this distance depends on the bounds  $\text{lobnd}$  and  $\text{upbnd}$ . Now, let us determine a pair of trial points  $x_u$  and  $x_v$ , from the set of trials points, so that

$$d_{uv}^{\text{iter}} = \|x_u - x_v\|_2 = \max_{i, j=1, \dots, N, j>i} d_{ij}^{\text{iter}}, \quad (9.8)$$

i.e.,  $d_{uv}^{iter}$  is the maximum distance among the trial points. With this, define the sphere  $S^{iter}$  centered in point  $c_{uv} = (x_u + x_v)/2$  of ray  $r_{uv}^{iter} = d_{uv}^{iter}/2$ . Obviously, the volume of this sphere is

$$V^{iter} = \frac{\pi}{6} (d_{uv}^{iter})^3. \quad (9.9)$$

**Proposition 9.1** At iteration  $iter$ , all the trial points belong to the sphere  $S^{iter}$ .

**Proof** Suppose by contradiction that a trial point  $x_p \notin S^{iter}$ . Then, if  $\|x_u - x_p\| \leq \|x_u - x_v\|$ , then  $\|x_v - x_p\| \geq \|x_u - x_v\|$ , i.e.,  $d_{uv}^{iter}$  is not the maximum distance among the trial points at iteration  $iter$ . Similarly, if  $\|x_v - x_p\| \leq \|x_u - x_v\|$ , then  $\|x_u - x_p\| \geq \|x_u - x_v\|$ , i.e.,  $d_{uv}^{iter}$  is not the maximum distance among the trial points at iteration  $iter$ . Therefore,  $x_p$  has to belong to  $S^{iter}$ .  $\blacklozenge$

**Proposition 9.2** At iteration  $iter$ , if  $|f_{iter}^{\min} - f_{iter-1}^{\min}| < t$ , then  $V^{iter} = V^{iter-1}/8$ .

**Proof** Let  $d_{uv}^{iter-1}$  be the maximum distance among the trial points at iteration  $iter - 1$  determined by the trial points  $x_u$  and  $x_v$ . Observe that at iteration  $iter$ , if  $|f_{iter}^{\min} - f_{iter-1}^{\min}| < t$ , then algorithm DEEPS reduces the complex, i.e., for  $j = 1, \dots, N$ , the new trial points are computed as  $x_j = x^{\min} + (x_j - x^{\min})/2$ , where  $x^{\min}$  is the trial point corresponding to the minimum value of function  $f$ . Therefore, the distance from any trial point  $x_i$  to any trial point  $x_j$  is

$$\begin{aligned} d_{ij}^{iter} &= \|x_i - x_j\|_2 = \|x^{\min} + (x_i - x^{\min})/2 - x^{\min} - (x_j - x^{\min})/2\|_2 \\ &= \frac{1}{2} \|x_i - x_j\|_2 = \frac{1}{2} d_{ij}^{iter-1}. \end{aligned} \quad (9.10)$$

In particular, for the points  $x_u$  and  $x_v$  which define the maximum distance among the trial points at iteration  $iter$ , it follows that  $d_{uv}^{iter} = d_{uv}^{iter-1}/2$ . Therefore, from (9.9),  $V^{iter} = V^{iter-1}/8$ .  $\blacklozenge$

Hence, at iteration  $iter$ , if  $|f_{iter}^{\min} - f_{iter-1}^{\min}| < t$ , then the complex is reduced, and the DEEPS algorithm generates the trial points in a sphere of smaller volume.

Now, let us consider the situation in which at a certain iteration  $iter - 1$ , around the point  $x_0$ , a number  $N$  of trial points are generated as

$$x_j^{iter-1} = x_0 + r_j(upbnd - lobnd) + e(lobnd), \quad j = 1, \dots, N,$$

where  $r_j = [r_j^1, \dots, r_j^n]^T$  is a vector with the components  $r_j^k$ ,  $k = 1, \dots, n$ , random numbers in and  $e = [1, \dots, 1] \in \mathbb{R}^n$  and  $lobnd$  and  $upbnd$  are the bounds of the domain  $D$ , (see step 2). As we know, the distance among these trial points is

$$d_{ij}^{iter-1} = \|x_i^{iter-1} - x_j^{iter-1}\|_2 = \|r_i - r_j\|_2 |upbnd - lobnd|, \quad i, j = 1, \dots, N, \quad i \neq j. \quad (9.11)$$

However,  $\|r_i - r_j\|_2 = \left( \sum_{k=1}^n (r_i^k - r_j^k)^2 \right)^{1/2} \leq \sqrt{n}$ . Therefore,  $d_{ij}^{iter-1} \leq \sqrt{n} |upbnd - lobnd|$ , for any  $i, j = 1, \dots, N, i \neq j$ .

**Proposition 9.3** If at the iteration  $iter$   $|f_{iter}^{\max}| > B$ , where  $B$  is a large enough prespecified parameter, then, at this iteration, the distance among the trial points is reduced.

**Proof** In this case, at step 14 the algorithm DEEPS reduces the bounds  $lobnd$  and  $upbnd$  as  $lobnd/2$  and  $upbnd/2$ , respectively, and go to generate a new set of trial points in step 2 using these new bounds. Hence, at this iteration, the new trial points are computed as  $x_j^{iter} = x_0 + \frac{1}{2}r_j(upbnd - lobnd) + \frac{1}{2}e(lobnd)$ , for  $j = 1, \dots, N$ , where again  $r_j$  is a vector with components random numbers in  $[0, 1]$ . Therefore, at this iteration, the distance among the new trial points is

$$d_{ij}^{iter} = \|x_i^{iter} - x_j^{iter}\|_2 = \frac{1}{2} \|r_i - r_j\|_2 |upbnd - lobnd| = \frac{1}{2} d_{ij}^{iter-1}, \quad (9.12)$$

for any  $i, j = 1, \dots, N, i \neq j$ .  $\diamond$

If  $|f_{iter}^{\min} - f^{opt}|_{iter} = |f_{iter-1}^{\min} - f^{opt}|_{iter-1}$ , then, at step 16 of the algorithm DEEPS, the local bounds  $lobndc$  and  $upbndc$  are halved. Now, at iteration  $iter$ , for every trial point  $x_j, j = 1, \dots, N$ , let us define the distance  $d_j^{iter}$ ,  $j = 1, \dots, N$ , from  $x_j, j = 1, \dots, N$ , to the local points  $x_j^k, k = 1, \dots, M$ , generated by the algorithm, i.e.,

$$d_j^{iter} = \max_{k=1, \dots, M} \|x_j - x_j^k\|_2, \quad j = 1, \dots, N. \quad (9.13)$$

With this, at iteration  $iter$ , let us determine a trial point  $x_w$  for which

$$d_w^{iter} = \max \left\{ d_j^{iter} : j = 1, \dots, N \right\}. \quad (9.14)$$

In other words,  $d_w^{iter}$  is the maximum distance among all trial points and all local trial points. As in Proposition 9.3, the following proposition can be proved.

**Proposition 9.4** *If at the iteration  $iter$   $|f_{iter}^{\min} - f^{opt}|_{iter} = |f_{iter-1}^{\min} - f^{opt}|_{iter-1}$  holds, then, at this iteration, the distance  $d_w^{iter}$  among the trial points and the local trial points is reduced.*  $\diamond$

Again, since at every iteration the algorithm DEEPS chooses the minimum point among the generated local points and at some specific iteration the bounds  $lobndc$  and  $upbndc$  are halved, it follows that the maximum distance between the trial points and the local trial points tends to zero. With these, the convergence of the algorithm DEEPS can be proved as follows.

**Theorem 9.1** *Let  $f$  be a continuous function bounded from below. Then, the algorithm DEEPS initialized in a point  $x_0$  generates a sequence  $\{x_k\}$  convergent to a point  $x^*$  where  $f(x^*) \leq f(x_0)$ .*

**Proof** Since at every iteration  $k$  DEEPS chooses the minimum point  $x_k$  from the trial points, it follows that the sequence  $\{f(x_k)\}$  is monotonously decreasing, i.e.,  $f(x_k) \leq f(x_{k-1})$ , for any  $k = 1, 2, \dots$ . Function  $f$  is bounded from below, and therefore, the sequence  $\{f(x_k)\}$  is convergent. By continuity, it follows that there is a point  $x^*$  such that  $\lim_{k \rightarrow \infty} f(x_k) = f\left(\lim_{k \rightarrow \infty} x_k\right) = f(x^*)$ .  $\diamond$

Observe that  $x^*$  is only a point where  $f(x^*) \leq f(x_0)$ . Since we do not have access to the gradient (and the Hessian) of function  $f$ , nothing can be said about its optimality. Having in view the derivative scarcity of information about the minimizing function  $f$ , the result given by DEEPS may be acceptable from a practical point of view. More details on DEEPS algorithm can be found in Andrei (2021a).

In this presentation of DEEPS, there are some (*open*) problems which need to be clarified: selection of the parameters  $N$  and  $M$ , selection of the lower and upper bounds, the use of the line-search procedure in step 12, etc. Our intensive numerical experiments with DEEPS show that all these elements have a marginal effect on the performances of DEEPS (Andrei, 2021a).

## 9.4 Numerical Study: NELMED, NEWUOA, and DEEPS

In the first set of numerical experiments, consider the minimization of the following function:

$$f(x) = \sum_{i=1}^{n-1} (|x_i| + 2|x_{i+1}| - 9)^2 + (2|x_i| + |x_{i+1}| - 13)^2,$$

where  $x_0 = [1, \dots, 1]$ . For minimizing this functions with different values for  $n$ , Tables 9.1, 9.2, and 9.3 show the performances of NELMED, NEWUOA with  $\rho_{end} = 10^{-6}$ , and DEEPS, respectively.

For minimizing this function for different values of  $n$ , from Tables 9.2 and 9.3 observe that NEWUOA needs a total of 859.23 seconds, while DEEPS needs only 14.08 seconds. Observe that both NEWUOA and DEEPS give a highly accurate solution.

In the second set of numerical experiments, let us solve the applications from the SMUNO collection presented in Appendix B. This collection includes 16 real applications from different

**Table 9.1** Performances of NELMED

$n$	#iter	#nf	cpu	vfomin	vfl
50	28658	32523	0.81	605.627290	6664.0
100	157965	170796	12.15	1047.023215	13464.0
200	551573	576315	193.49	2544.360161	27064.0

FORTRAN77 version by R. O'Neill (1971), modifications by John Burkardt

**Table 9.2** Performances of NEWUOA

$n$	#iter	#nf	cpu	vfomin	vfl
50	6	1037	1.10	389.333333	6664.0
100	6	1527	6.07	789.333333	13464.0
200	6	3665	117.55	1589.333333	27064.0
300	6	7616	734.51	2389.333333	40664.0

FORTRAN 77 version by Powell

**Table 9.3** Performances of DEEPS

$n$	#iter	#nf	cpu	vfomin	vfl
50	1164	37037	0.09	389.333333	6664.0
100	1510	47962	0.22	789.333333	13464.0
200	3521	112367	1.03	1589.333333	27064.0
300	11053	349293	12.74	2389.333333	40664.0

FORTRAN 77 version by Andrei

**Table 9.4** Performances of NELMED for solving 16 small-scale applications from the SMUNO collection

<i>n</i>	#iter	#nf	cpu	vfomin	vf0	name
2	4321	14260	1	-0.2644531398164E+03	-0.374731373714E+02	1. WEBER-1
2	5749	17111	0	0.9560739834844E+01	0.7859432489718E+02	2. WEBER-2
2	485	1515	0	0.8749843722120E+01	0.7860286479337E+02	3. WEBER-3
4	58196	298649	8	0.3075923582382E-03	0.5313172272109E-02	4. ENZIMES
6	601448	2963265	50	0.3370929785002E-03	0.1961733675060E+08	5. REACTOR
8	18824	131208	2	0.6829467973914E-08	0.5334258881257E+01	6. ROBOT
4	44064	225895	35	0.6872367777921E+01	0.9958700480657E+01	7. SPECTR
4	220110632	999990007	26078	0.3822441651590E-01	0.2905300235663E+01	8. ESTIMP
5	16640	96881	1	0.2887516043798E-03	0.3312269269234E+08	9. PROPAN
2	3584	11541	1	0.1744152005589E+01	0.2563325000000E+04	10. GEAR-1
8	1232994	12451823	231	0.9315130266680E-05	0.1905692553768E+00	11. HHD
6	539699	4132548	52	0.2463694921725E-05	0.2391760016000E+02	12. NEURO
10	19060786	155723303	4155	0.8484442736605E-10	0.1219988990749E+03	13. COMBUST
9	39546	390639	71	0.1022362061183E-01	0.2964578187893E+04	14. CIRCUIT
3	54473	136162	83	0.1750977749414E+03	0.2335910048036E+10	15. THERM
4	8026086	36117397	533	0.7515500076120E-19	0.7370818569964E-03	16. GEAR-2

FORTRAN77 version by R. O'Neill (1971), modifications by John Burkardt

**Table 9.5** Performances of NEWUOA for solving 16 small-scale applications from the SMUNO collection

<i>n</i>	#iter	#nf	cpu	vfomin	vf0	name
2	7	129	0	-0.2644531378352E+03	-0.3747313737140E+02	1. WEBER-1
2	6	72	0	0.9560739742798E+01	0.7859432489718E+02	2. WEBER-2
2	6	89	0	0.8749841736967E+01	0.7860286479337E+02	3. WEBER-3
4	6	306	1	0.3075056038514E-03	0.5313172272109E-02	4. ENZIMES
6	6	4824	15	0.5297708382383E-11	0.1961733675060E+08	5. REACTOR
8	6	262	1	0.3301153056066E-13	0.5334258881257E+01	6. ROBOT
4	6	246	1	0.8312307692327E+01	0.9958700480657E+01	7. SPECTR
4	7	1588	2	0.3185717487911E-01	0.2905300235663E+01	8. ESTIMP
5	7	4114	9	0.4282211642674E-04	0.3312269269234E+08	9. PROPAN
2	6	63	0	0.1744152005588E+01	0.2563325000000E+04	10. GEAR-1
8	5	9000	47	0.6277417900490E-04	0.1905692550902E+00	11. HHD
6	5	282	1	0.4539057615182E+01	0.2391760016000E+02	12. NEURO
10	6	2812	21	0.3419994182218E-11	0.1219988990749E+03	13. COMBUST
9	6	7370	44	0.1768702508139E-09	0.2964578187893E+04	14. CIRCUIT
3	4	118	1	0.6575396003806E+05	0.2335910048036E+10	15. THERM
4	8	82	0	0.4058913495309E-21	0.7370818569964E-03	16. GEAR-2

FORTRAN 77 version by Powell

areas of activity: *chemical reaction, solution of a chemical reactor, robot kinematics problem, solar spectroscopy, propane combustion in air-reduced variant, gear train of minimum inertia, human heart dipole, neurophysiology, combustion application, circuit design, thermistor, and optimal design of a gear train*. The largest application has 10 variables. Table 9.4 shows the performances of NELMED for solving these applications.

Observe that NELMED needs 249817527 iterations and 31.301 seconds. Table 9.5 includes the performances of NEWUOA. We can see that for solving all the 16 applications, NEWUOA needs 97 iterations and 14.3 seconds. Table 9.6 presents the performances of DEEPS. The total number of

**Table 9.6** Performances of DEEPS for solving 16 small-scale applications from the SMUNO collection

<i>n</i>	#iter	#nf	cpu	vfomin	vf0	name
2	119	1485	0	-0.2644531378352E+03	-0.3747313737140E+02	1. WEBER-1
2	67	1318	0	0.9560744054913E+01	0.7859432489718E+02	2. WEBER-2
2	113	6198	1	0.8749847970082E+01	0.7860286479337E+02	3. WEBER-3
4	28	42903	2	0.3087657632221E-03	0.5313172272109E-02	4. ENZIMES
6	1375	6989983	197	0.7472925581554E-04	0.1961733675060E+08	5. REACTOR
8	43	2459	0	0.1828017526180E-07	0.5334258881257E+01	6. ROBOT
4	4	233	0	0.8316382216967E+01	0.9958700480657E+01	7. SPECTR
4	54	1705727	99	0.3185724691657E-01	0.2905300235663E+01	8. ESTIMP
5	10	10013013	274	0.2467602087429E-04	0.3312269269234E+08	9. PROPAN
2	15	342	0	0.1744152005590E+01	0.2563325000000E+04	10. GEAR-1
8	2404	36265585	1487	0.9966084682095E-04	0.1905692550902E+00	11. HHD
6	1044	7999853	219	0.8545018926146E-04	0.2391760016000E+02	12. NEURO
10	25	856	0	0.4061987800161E-08	0.1219988990749E+03	13. COMBUST
9	3731	9654682	1244	0.1036184837525E-03	0.2964578187893E+04	14. CIRCUIT
3	15	752179	251	0.1742216236340E+03	0.2335910048036E+10	15. THERM
4	6	3090	0	0.3886716443010E-13	0.7370818569964E-03	16. GEAR-2

FORTRAN version by Andrei

**Table 9.7** Performances of NEWUOA for solving 10 large-scale applications from the SMUNO collection

<i>n</i>	#iter	#nf	cpu	vfomin	vf0	name
100	6	13048	205.96	0.1822103437144E-10	0.3976246221006E+03	DENSCHNA
200	7	25965	868.09	0.7241454319422E-10	0.6585000000000E+06	DENSCHNB
200	7	11503	381.19	0.5939581794857E-08	0.4160000000000E+05	DENSCHNF
300	5	3691	321.24	0.1315442201683E-09	0.5710855664408E+01	BROWN
200	7	17026	539.42	0.2200999786590E+03	0.1174100000000E+05	ENGVAL1
100	4	100000	888.07	0.2343611622985E-05	0.7942000000000E+04	NONDQAR
200	6	15063	523.91	0.2386448816338E-09	0.5970000000000E+03	ARWHEAD
200	3	100000	3588.24	0.3474833628182E+00	0.3256542280009E+17	VARDIM
200	7	20557	708.15	0.1000000000068E+01	0.1626000000000E+04	DIXMAANA
300	7	30590	2533.33	0.1000000000360E+01	0.4494375000000E+04	DIXMAANB

FORTRAN 77 version by Powell

iterations required by DEEPS for solving all these 16 applications is 9053, while the CPU computing time is of 37.74 seconds. Observe that NEWUOA is more efficient than NELMED and DEEPS.

In the last set of numerical experiments, Tables 9.7 and 9.8 present the numerical performances of NEWUOA with  $\rho_{end} = 10^{-6}$  and DEEPS with  $\epsilon = 10^{-6}$  for solving 10 unconstrained optimization problems from the LACOP collection, with the number of variables in the range [100,300]. To get an accurate solution in NEWUOA, the number of function's evaluations was limited to 100000. Observe that both these algorithms are able to solve problems with a relative large number of variables, but the differences are significant. For solving all these 10 problems, NEWUOA needs a total of 59 iterations and 10557.60 seconds. On the other hand, DEEPS needs a total of 3976 iterations and 239.04 seconds.

In the above tables, we have  $n$  = the number of variables, #iter = the number of iterations to get a local solution, #nf = the number of function evaluations, cpu = the CPU computing time (seconds), vfomin = the minimum value of the minimizing function at solution, vf0 = the value of the minimizing function in the initial point, and name = the name of the application.

**Table 9.8** Performances of DEEPS for solving 10 large-scale applications from the SMUNO collection

<i>n</i>	#iter	#nf	cpu	<i>vfomin</i>	<i>vf0</i>	<i>name</i>
100	54	2726250	44.64	0.1933041622350E-05	0.3976246221006E+03	DENSCHNA
200	576	5066817	70.13	0.7798508331993E-05	0.6585000000000E+06	DENSCHNB
200	1919	110060	1.72	0.6170389679663E-03	0.4160000000000E+05	DENSCHNF
300	12	30930	3.82	0.2519366320438E-04	0.5710855664408E+01	BROWN
200	56	285028	4.28	0.2460315490512E+03	0.1174100000000E+05	ENGVAL1
100	11	57306	0.27	0.6641255924711E-07	0.7942000000000E+04	NONDQAR
200	277	717905	10.82	0.3723608045636E-04	0.5970000000000E+03	ARWHEAD
200	1032	5258097	85.49	0.7836853799080E-04	0.3256542280009E+17	VARDIM
200	11	643	0.05	0.1000005648123E+01	0.1626000000000E+04	DIXMAANA
300	28	143708	17.82	0.1000012240963E+01	0.4494375000000E+04	DIXMAANB

FORTRAN 77 version by Andrei

Subject to CPU computing time the most critical resources, it seems that DEEPS is more efficient than NEWUOA. However, the solution given by NEWUOA is more accurate. The direct optimization methods are recommended for minimizing functions with a small number of variables for which the derivative information are difficult to obtain or are unreliable.

### Notes and References

The derivative-free methods can be classified as *direct* and *model-based*. Both of them may be *randomized*. Briefly, the direct algorithms determine the search directions by computing the values of the function  $f$  directly, whereas the model-based algorithms construct and utilize a surrogate model of the minimizing function to guide the search process. Furthermore, these methods are classified as *local* or *global*. Finally, these algorithms can be *stochastic* or *deterministic*, depending upon whether they require random search steps or not. The direct search methods include the methods of a *simplicial* type, where one typically moves away from the worst point (a point at which the function value is the highest), and the methods of a *directional* type, where one tries to move along a direction defined by the best point. The simplex methods construct and manipulate a collection of  $n + 1$  affinely independent points in  $\mathbb{R}^n$  called vertices, which define a simplex and make use of some rules (reflection, expansion, inner contraction, and shrink) to modify these vertices of the simplex until the best vertex has been determined. The most popular and representative simplex method is of Nelder and Mead (1965). Each iteration of the *directional direct-search* methods generates a finite set of points around the current point  $x_k$ , called poll points, by taking  $x_k$  and by adding terms of the form  $\alpha_k d$ , where  $\alpha_k$  is a positive step and  $d$  is an element from a finite set of directions  $D_k$ , the poll direction set.

Some of the very well-known direct methods for solving unconstrained optimization problems are computing a zero of a nonlinear function in a given interval (Andrei, 1975a, 1975b); the univariate search; the Fibonacci and golden section search; the coordinate search; the pattern move of Hooke-Jeeves (1961); the simplex method by Spendley, Hext, and Himsorth (1962); the conjugate directions of Powell (1964); turning the coordinates of Rosenbrock (1960); UOBYQA (Unconstrained Optimization BY Quadratic Approximation) of Powell (1994, 2002); the implicit filtering by Kelley (1999) and Choi and Kelley (2000) (see Andrei, 1999a, 2009e); etc. A comparison of the numerical performances of these direct methods is given by Andrei (2009e). It seems that the direct methods based on the interpolating quadratics are some of the best direct methods. The derivative-free methods for one-dimensional problems are described by Brent (1973). Recent surveys of the direct methods for optimization can be found in the papers of Wright (1996); Conn, Scheinberg, and

Toint (1997b); Powell (1998); Lewis, Torczon, and Trosset (2000); Kolda, Lewis, and Torczon (2003); Rios and Shainidis (2013); and Andrei (2021a, b, c, d).

The main software packages for the derivative-free optimization include COBYLA (Powell, 2003), DFO (Conn, Scheinberg, & Toint, 1997b), UOBYQA (Powell, 2002), WEDGE (Marazzi, & Nocedal, 2002), NEWUOA (Powell, 2004) implementing the model-based methods using linear or quadratic models, APPS (Hough, Kolda, & Torczon, 2001) using a pattern-search method, NELMED (O'Neill, 1971), DIRECT (Jones, Pertunen, & Stuckman, 1993; Finkel, 2003) for the global optimization, and DEEPS (Andrei, 2021a, b, c, d). E04FCC (NAG – The Numerical Algorithms Group) is a comprehensive algorithm for finding an unconstrained minimum of a sum of squares of  $m$  nonlinear functions in  $n$  variables ( $m \geq n$ ).



# Constrained Nonlinear Optimization Methods: An Overview

10

Consider a particular form of a nonlinear optimization problem

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c(x) = 0, x \geq 0, \end{aligned} \tag{10.1}$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is the objective function and  $c: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are the constraints assumed to be twice continuously differentiable. For the problem (10.1), denote the *multipliers* corresponding to the equality constraint by  $y$  and the multipliers of the inequality constraints by  $z \geq 0$ . Observe that (10.1) may have inequality constraints, simple bounds on variables, or even range constraints of the form  $l_i \leq c_i(x) \leq u_i$ ,  $i = 1, \dots, m$ , which are here omitted for the sake of simplifying the presentation (Leyffer & Mahajan, 2010). To solve (10.1), the iterative methods are used in a reductionist approach. The problem (10.1) is approximated by a sequence of subproblems. Their solving gives a sequence of solutions  $\{x_k\}$  starting from an initial point  $x_0$ . The solution of a subproblem is the initial guess for the next subproblem in this sequence, and all the process has a number of the so-called major iterations. In this frame, every subproblem is also solved by an iterative process through the so-called minor iterations. A simple prototype for solving (10.1) to generate the sequence  $\{x_k\}$  is as follows:

**Algorithm 10.1** Prototype for the nonlinear optimization algorithm

1. Choose an initial estimate  $x_0$  and set  $k = 0$
2. Evaluate a criterion for stopping the iterations
3. At the current point  $x_k$  elaborate a local model of (10.1)
4. Approximately solve the local model (sub-problem) to determine an improved estimation  $x_{k+1}$  of the solution of (10.1)
5. Set  $k = k + 1$  and continue with step 2



In the following, the main components of Algorithm 10.1 for solving the problem (10.1) are discussed in a *critical manner*. The real sense of the word “criticism” does not mean “destruction,” but “creation.” Consequently, some issues may be looked upon: how the *approximate subproblem* that determines an improved new iterate is computed, the *globalization strategy* that ensures the

convergence of the algorithm initialized in remote starting points by indicating whether a new estimate of the solution is better than the current estimate, the *refining mechanisms* that reduces the step computed by the local model to enforce the globalization strategy to work far away from the solution, and the *convergence tests* as well as the *termination conditions* that check for the optimal solution or for the failure of the algorithm. *The nonlinear optimization algorithms are classified by the choice with which they implement each of these fundamental components of Algorithm 10.1.*

## 10.1 Convergence Tests

Usually, the convergence tests are based on the Karush-Kuhn-Tucker conditions (see Theorem 11.15). For example, for the problem (10.1), suitable approximate convergence tests may be presented as

$$\|c(x_k)\| \leq \varepsilon_1, \quad (10.2a)$$

$$\|\nabla f(x_k) - \nabla c(x_k)y_k - z_k\| \leq \varepsilon_2, \quad (10.2b)$$

$$\|\min\{x_k, z_k\}\| \leq \varepsilon_3, \quad (10.2c)$$

where  $\varepsilon_i > 0$ ,  $i = 1, 2, 3$ , are tolerances and the “min” in (10.2c) corresponding to the complementarity condition is taken componentwise. (Here  $\nabla f(x_k)$  is the gradient of  $f$  and  $\nabla c(x_k)$  is the Jacobian of  $c$  computed at point  $x_k$ .) It is quite possible for the convergence of the algorithm not to be ensured to an approximate KKT point (see Theorem 11.15), for example, if the constraints do not satisfy a constraint qualification (see Remark 11.2). In these cases, the test (10.2b) is replaced by

$$\|\nabla c(x_k)y_k + z_k\| \leq \varepsilon_2, \quad (10.3)$$

corresponding to a Fritz John point. It is also possible for the termination conditions to be modified to include the “relative error,” as used in some algorithms. The specific norm used in the above criteria depends upon the number of variables. For small-scale problems, the two-norm is satisfactory, while for large-scale problems, the infinity norm is recommended.

*The limits of these convergence tests are the feasibility conditions and those of optimality are separated, so that we may obtain a feasible solution which is not optimal and vice versa. However, the feasibility of the constraints is always the most important criterion for terminating the iteration. From the practical point of view, the decision that the iterate  $x_k$  is an adequate approximation to the solution is based on two major tests: whether  $x_k$  almost satisfies the sufficient conditions for optimality and whether the sequence  $\{x_k\}$  appears to have converged.*

## 10.2 Infeasible Points

As already seen in Algorithm 10.1, one needs an initial point  $x_0$  to start the iterations. The computation of an initial point is a problem in itself. Unless the problem is convex or some restrictive assumptions are made, Algorithm 10.1 cannot guarantee the convergence even to a feasible point. Besides, there are many cases in which the points of the sequence  $\{x_k\}$  determined by Algorithm 10.1 must be feasible. In these cases, a procedure for certifying the feasibility is needed. An appropriate converge test and feasibility subproblem is based on the following feasibility problem:

$$\min_{x \geq 0} \|c(x)\|, \quad (10.4)$$

which, as can be seen, may be formulated as a smooth optimization problem by introducing the slack variables. For solving the problem (10.4), one can use Algorithm 10.1, as the feasibility problem can be reformulated as a smooth nonlinear optimization problem by introducing additional variables. The objective function of (10.4) can be replaced by a weighted norm. In this case, suitable tests are

$$\|\nabla c(x_k)y_k - z_k\| \leq \varepsilon \text{ and } \|\min(x_k, z_k)\| \leq \varepsilon, \quad (10.5)$$

where  $y_k$  are the multipliers or the weights corresponding to the norm used in the objective of (10.4).

### 10.3 Approximate Subproblem: Local Models and Their Solving

In step 3 of Algorithm 10.1, the elaboration of a local model of the problem (10.1) around the current point  $x_k$  is needed. *The key difference among the nonlinear optimization algorithms is how the local model is constructed around the current point.* The idea is to generate a local model of the problem for which a solution can be easily obtained and which can improve the current iterate. Three broad classes of local models are known: *sequential linear models*, *sequential quadratic models*, and *interior point models*.

The *sequential linear programming (SLP) methods* construct a linear approximation to (10.1). Usually, these linear programming programs are unbounded. Therefore, these methods require an additional trust-region constraint

$$\begin{aligned} \min_d m_k(d) &\triangleq \nabla f(x_k)^T d \\ \text{subject to} \\ c(x_k) + \nabla c(x_k)^T d &= 0, \\ x_k + d &\geq 0, \\ \|d\|_\infty &\leq \Delta_k, \end{aligned} \quad (10.6)$$

where  $\Delta_k > 0$  is the *radius of the trust-region*. To ensure the convergence, in general,  $\Delta_k \rightarrow 0$  must converge to zero. This method was used by Griffith and Stewart (1961) with some success, without introducing a trust-region constraint. The sequential linear programming method can be considered as a steepest descent method and typically converges only linearly. *If at the solution point there are exactly  $n$  active constraints and the normals of these constraints are linearly independent, then the sequential linear programming method reduces to the Newton method for solving a square system of nonlinear equations, which is superlinear convergent.*

The *sequential quadratic programming (SQP) methods* minimize a quadratic model  $m_k(d)$  subject to a linearization of the constraints about the current point  $x_k$

$$\begin{aligned} \min_d m_k(d) &\triangleq \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d \\ \text{subject to} \\ c(x_k) + \nabla c(x_k)^T d &= 0, \\ x_k + d &\geq 0, \end{aligned} \quad (10.7)$$

where  $B_k \triangleq \nabla_{xx}^2 L(x_k, \lambda_k)$  is the Hessian of the Lagrange function associated to the problem (10.1) and  $\lambda_k$  are the Lagrange multipliers, to obtain the search direction  $d \triangleq x - x_k$ . The new iterate is given by  $x_{k+1} = x_k + d$  together with the Lagrange multipliers  $\lambda_{k+1}$  associated to the linearized constraints of

(10.7). If the matrix  $B_k$  is not positive definite on the null space of the active constraint normals, then the quadratic subproblem (10.7) is nonconvex and the SQP methods determine a local minimum of (10.7). In case of large-scale problems, the solution of the quadratic subproblems can become computationally expensive because the null space method for solving the quadratic subproblem requires the factorization of a dense-reduced Hessian matrix.

The sequential linear-quadratic programming (SLQP) methods combine the advantages of the SLP method (fast solution of the linear programming subproblem) and of the SQP methods (fast local convergence) by adding an equality constraint to the SLP method [(Fletcher & de la Maza, 1989), (Chin & Fletcher, 2003), (Byrd, Gould, Nocedal, & Waltz, 2002, 2004a)]. In these methods, two subproblems are solved: the *first one* is a linear programming subproblem which gives a step for the next iteration and also an estimate of the active set  $A_k \triangleq \left\{ i : [x_k]_i + \hat{d}_i = 0 \right\}$  obtained from a solution  $\hat{d}$  of (10.6). (Here  $[x_k]_i$  is the component  $i$  of vector  $x_k$ .) This estimation of the active set is used to construct and solve the *second problem*, which is an equality constrained quadratic programming problem defined by the active constraints

$$\begin{aligned} \min_d q_k(d) &\triangleq \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d \\ \text{subject to} \\ c(x_k) + \nabla c(x_k)^T d &= 0, \\ [x_k]_i + d_i &= 0, \quad i \in A_k. \end{aligned} \tag{10.8}$$

If  $B_k$  is positive definite on the null space of the constraint normals, then the solution of (10.8) is equivalent to the following linear algebraic system:

$$\begin{bmatrix} B_k & -\nabla c(x_k) & -I_k \\ \nabla c(x_k)^T & 0 & 0 \\ I_k^T & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) + B_k x_k \\ -c(x_k) + \nabla c(x_k)^T x_k \\ 0 \end{bmatrix}, \tag{10.9}$$

obtained by applying the KKT conditions to the subproblem (10.8). (Here,  $I_k = [e_j]_{j \in A_k}$  are the normals of the active inequality constraints;  $y_k$  and  $z_k$  are the Lagrange multipliers.) It is always possible to choose a basis from the linear programming subproblem (by simplex algorithm) such that the augmented matrix  $[A_k \ I_k]$  has full rank. If  $B_k$  is not positive definite on the null space of the active constraint normals, then the inertia of this matrix can be corrected by adding to  $B_k$  a multiple of the identity matrix, thus ensuring the descent step of the subproblem (10.8).

The sequential quadratic-quadratic programming (SQQP) methods have been recently introduced by Gould and Robinson (2008, 2010). From the very beginning, they construct and solve a quadratic programming subproblem by using a positive definite Hessian of the Lagrangian. The solution of this convex subproblem is used into an equality constrained quadratic programming model with the exact second derivative of the Lagrangian. More precisely, the SQQP procedure is a second derivative method that is globalized via the  $l_1$  merit function. Given an estimate  $x_k$  of the solution to the problem (10.1), a search direction at this point is generated from the combination of three steps: a *predictor step* defined as a solution to a strictly convex quadratic programming subproblem, a *Cauchy step* (driving convergence of the algorithm) which is computed from a special univariate global minimization problem, and a *SQP step* computed from a local solution of a special nonconvex quadratic programming subproblem. The algorithm is embedded into the trust-region approach.

**Remark 10.1** As a general characterization of these sequential linear and quadratic programming methods, the following theoretical aspects can be mentioned. If  $B_k$  is the exact Hessian of the Lagrange function and the Jacobian of the active constraints has full rank, then the SQP methods are quadratic convergent near a minimizer that satisfies both a constraint qualification and a second-order sufficient condition (the Hessian of the Lagrangian is positive definite on the null space of the active constraint normals) (Boggs & Tolle, 1995). Moreover, under the additional assumption of strict complementarity, all the four methods based on sequential linear or quadratic programming identify the optimal active set in a finite number of iterations. The above sequential methods are also known as active-set methods because the solution of each linear or quadratic subproblem provides not only a suitable new iterate but also an estimate of the active set at the solution point (Leyffer, & Mahajan, 2010).

Additionally, two major concerns are associated to the SQP methods: incompatible linearized constraints and unbounded solutions of the quadratic subproblems. For handling the unbounded solutions of (10.7), two approaches are possible. The first one is to use a positive definite approximation to the Hessian in (10.7). Thus, a strictly convex-bounded quadratic program is obtained with a unique minimizer. The second approach allows for a nonconvex quadratic program by explicitly bounding the solution via a trust-region constraint. Both approaches are effective in practice. The issue of incompatible subproblems is more challenging. Observe that the quadratic programming subproblem (10.7) may be naturally incompatible. In this case, different techniques have been suggested for dealing with incompatible subproblems: the “constraint shifting” (Vardi, 1985), the use of the “elastic mode” (Gill, Murray, & Saunders, 2002, 2005) (see Chap. 15), and a “feasibility restoration phase” (Fletcher & Leyffer, 2002) (see Chap. 18). ◆

The *interior point methods* are an alternative approach to the active set methods. Essentially, these are *perturbed Newton methods* which postpone the determination of the active constraints set at the end of the iterative process for solving the problem. Plenty of interior point methods are known, but the most successful one is the primal-dual, which can be viewed as the Newton method applied to the perturbed first-order optimality conditions of problem (10.1). The first-order optimality conditions include the *stationary condition* of the Lagrange function, the *feasibility condition* (satisfying the constraints of the problem), and the *complementarity condition* (see Definition 11.14). In the interior point methods, only these complementarity conditions are perturbed, as in the following nonlinear algebraic system:

$$F_\mu(x, y, z) = \begin{bmatrix} \nabla f(x) - \nabla c(x)^T y - z \\ c(x) \\ Xz - \mu e \end{bmatrix} = 0, \quad (10.10)$$

where  $\mu > 0$  is the barrier parameter,  $X = \text{diag}(x)$  is a diagonal matrix with the components of the vector  $x$  on the main diagonal, and  $e$  is a vector with all the components equal to one. The interior point methods start the computations from an interior point  $x_0, z_0 > 0$  and generate a sequence of interior points  $x_k, z_k > 0$  as approximate solutions of the system (10.10), for a decreasing sequence of barrier parameters. By applying the Newton method to the nonlinear primal-dual system (10.10) around  $x_k$ , we get the *local model* (i.e., the approximate subproblem):

$$\begin{bmatrix} B_k & -\nabla c(x_k) & -I \\ \nabla c(x_k)^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = -F_\mu(x_k, y_k, z_k), \quad (10.11)$$

where  $B_k$  is an approximate to the Hessian of the Lagrange function and  $Z_k = \text{diag}(z_k)$  is a diagonal matrix with the components of the vector  $z_k$  on the main diagonal. ( $X_k = \text{diag}(x_k)$ .) The next step is computed as

$$(x_{k+1}, y_{k+1}, z_{k+1}) = (x_k, y_k, z_k) + (\alpha_x \Delta x, \alpha_y \Delta y, \alpha_z \Delta z),$$

where the stepsizes  $\alpha_x$ ,  $\alpha_y$ , and  $\alpha_z$  are computed to ensure that  $x_{k+1}, z_{k+1} > 0$  remain strictly positive. A simple variant of the interior point algorithms is as follows.

**Algorithm 10.2** *Prototype of the interior point algorithm*

- |    |  |
|----|--|
| 1. | Choose an initial point estimate $(x_0, y_0, z_0)$ such that $x_0 > 0$ and $z_0 > 0$ . Choose a value for the barrier parameter $\mu_0$ , the parameter $0 < \sigma < 1$ and a decreasing sequence $\varepsilon_k$ . Set $k = 0$ |
| 2. | Evaluate a criterion for stopping the iterations concerning the optimality of $(x_k, y_k, z_k)$ .  |
| 3. | Set $(x_{k,0}, y_{k,0}, z_{k,0}) = (x_k, y_k, z_k)$ and $j = 0$  |
| 4. | If $\ F_{\mu_k}(x_{k,j}, y_{k,j}, z_{k,j})\  \leq \varepsilon_k$ , then go to step 6. Otherwise, continue with step 5  |
| 5. | Approximately solve the Newton system (10.11) for a new iterate $(x_{k,j+1}, y_{k,j+1}, z_{k,j+1})$ . Set $j = j + 1$ and go to step 4   |
| 6. | Reduce the barrier parameter $\mu_{k+1} = \sigma \mu_k$ set $k = k + 1$ and go to step 2   |

◆

Observe that this algorithm has two loops: one responsible for solving the problem and the other one for approximately solving the Newton system. Notice that this is a general prototype interior point algorithm, and many other ingredients are needed to make it efficient and robust.

At this point, it is very instructive to see the relationship of the interior point methods with the barrier methods developed by Fiacco and McCormick (1990). The importance of the barrier methods comes from the fact that they can provide polynomial-time algorithms for the linear programming problems [(Wright, 1991), (Forsgren, Gill, & Wright, 2002), (Nemirovskii, & Todd, 2008)]. The barrier methods approximately solve a sequence of barrier subproblems of the following form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} & f(x) - \mu \sum_{i=1}^n \log(x_i) \\ \text{subject to} & \\ & c(x) = 0, \end{aligned} \tag{10.12}$$

for a decreasing sequence of the barrier parameters  $\mu > 0$ . The first-order optimality conditions for the problem (10.12) are given by

$$\nabla f(x) - \mu X^{-1} e - \nabla c(x)y = 0, \tag{10.13a}$$

$$c(x) = 0. \tag{10.13b}$$

Now, applying the Newton method to the nonlinear system (10.13), the following linear algebraic system is obtained:

$$\begin{bmatrix} B_k + \mu X_k^{-2} & -\nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) - \mu X_k^{-1} e - \nabla c(x_k)y_k \\ c(x_k) \end{bmatrix}.$$

Introducing the first-order multiplier estimates  $Z(x_k) \triangleq \mu X_k^{-1}$ , which can also be written as  $Z(x_k)X_k = \mu e$ , the following linear system is obtained:

$$\begin{bmatrix} B_k + Z(x_k)X_k^{-1} & -\nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) - \mu X_k^{-1}e - \nabla c(x_k)y_k \\ c(x_k) \end{bmatrix}.$$

This system is equivalent to the primal-dual Newton system (10.11), where

$$\Delta z = -X^{-1}Z\Delta x - Ze - \mu X^{-1}e$$

has been eliminated. Therefore, the main difference between the classical barrier methods as elaborated by Fiacco and McCormick and the primal-dual interior point methods is that the matrix  $Z_k$  is not free for the barrier methods, but it is chosen as the primal multiplier  $Z(x_k) \triangleq \mu X_k^{-1}$ . This freedom in the primal-dual interior point methods avoids some difficulties with the ill-conditioning of the Hessian to the barrier function. This was the main reason for rejecting the barrier methods in solving nonlinear optimization problems.

**Remark 10.2** As a characterization of the interior point methods, we can state: if there exists a compact set of isolated local minimizers of the problem (10.1) with at least one point in the closure of the strictly feasible set, then it follows that the barrier methods converge to a local minimum of the problem (10.1) (Wright, 1991). On the other hand, for the convex nonlinear optimization problems, the interior point methods, as in the case of linear programming, lead to polynomial-time algorithms (Nesterov, & Nemirovskii, 1994).  $\diamond$

## 10.4 Globalization Strategy: Convergence from Remote Starting Points

The local models discussed above guarantee the convergence only in a small neighborhood of a regular point (see Definitions 11.13 and 11.16). However, the initialization of the algorithms near the unknown optimal solution is a difficult task. Therefore, the globalization strategies are concerned with ensuring the convergence from the remote starting points to the stationary points (see Theorem 11.3). The globalization strategy should not be confused with the global optimization. To ensure the convergence from the remote starting points, the progress of iterates generated by the approximate subproblems must be monitored. For the unconstrained optimization, monitoring is very simple because the progress of the algorithm can be measured by comparing the values of the objective function. For the constrained optimization, it is necessary to take into account the constraint violation. Three broad classes of strategies exist: the *augmented Lagrangian methods*, the *penalty and merit function methods*, and the *filter methods*.

The *augmented Lagrangian methods* for the problem (10.1) use the augmented Lagrangian as

$$L(x, y, \rho) = f(x) - c(x)^T y + \frac{\rho}{2} \|c(x)\|_2^2, \quad (10.14)$$

where  $\rho > 0$  is a penalty parameter. In this context, there are two possibilities to develop the algorithms based on the augmented Lagrangian.

- (a) *Linearly constrained Lagrangian methods*. These methods minimize a modified Lagrangian subject to a linearization of the constraints. The modified Lagrangian is defined as

$$\bar{L}(x, y, \rho) = f(x) - p_k(x)^T y + \frac{\rho}{2} \|p_k(x)\|_2^2, \quad (10.15)$$

where  $p_k(x)$  collects the higher-order nonlinear terms from the Taylor series at the current iterate  $x_k$ , i.e.,

$$p_k(x) = c(x) - c(x_k) - \nabla c(x_k)^T(x - x_k). \quad (10.16)$$

Therefore, this approach consists in forming and solving the following local model:

$$\begin{aligned} & \min_x \bar{L}(x, y_k, \rho_k) \\ & \text{subject to} \\ & \quad c(x_k) + \nabla c(x_k)^T(x - x_k) = 0, \\ & \quad x \geq 0, \end{aligned} \quad (10.17)$$

for a fixed value of the penalty parameter  $\rho_k$ . The Lagrange multipliers are updated by using a first-order multiplier update rule:

$$y_{k+1} = y_k - \rho_k c(x_{k+1}), \quad (10.18)$$

where  $x_{k+1}$  is the solution of (10.17). Therefore, the local model (10.17) is used in an iterative scheme for a decreasing sequence of the penalty parameters. This approach is implemented in MINOS (Chap. 14) (Murtagh & Saunders, 1980, 1982, 1987, 1995). Another modification of the augmented Lagrangian as a combination of penalty and barrier terms is used in SPENBAR (Chap. 14) (Andrei, 1996a, 1996b, 1996c, 1998a).

(b) *Bound-constrained Lagrangian methods.* These methods approximately minimize the augmented Lagrangian for the problem (10.1) subject to simple bounds on the variables,

$$\begin{aligned} & \min_x L(x, y_k, \rho_k) \\ & \text{subject to} \\ & \quad x \geq 0, \end{aligned} \quad (10.19)$$

for a sequence of increasing penalty parameters. The advantage of this approach is that efficient methods for the simple bounds on the variables optimization can be applied: gradient projection conjugate gradient approach by Moré and Toraldo (1991); limited memory BFGS with bounds by Byrd, Lu, Nocedal, and Zhu (1995b); spectral projected gradient method by Birgin, Martínez, and Raydan (2000, 2001); truncated Newton with simple bounds by Nash (1984a, b, 1985); etc. (see Chap. 12). The idea of an iteration which uses the local model given by (10.19) is as follows. For a fixed value of the penalty parameter  $\rho_k$ , solve the subproblem (10.19). Thus, an approximate solution  $x_k^c$  is obtained. If  $\|c(x_k^c)\| \leq \eta_k$ , then update the Lagrange multipliers as  $y_{k+1} = y_k - \rho_k c(x_k^c)$ ; otherwise, increase the value of the penalty parameter, for example,  $\rho_{k+1} = 10\rho_k$ . Here,  $\{\eta_k\}$  is a forcing sequence which controls the progress to the feasibility of the nonlinear constraints. Each minimization of (10.19) can be started from the previous iterate. Representative for the bound-constrained Lagrangian method is LANCELOT (Conn, Gould, & Toint, 1992a, 1992b).

**Remark 10.3** As a general characterization of the augmented Lagrangian methods, we mention the linearly constrained augmented Lagrangian method can be made globally convergent by adding slack variables in order to handle the infeasible subproblems (Friedlander & Saunders, 2005). On the other hand, a bound-constrained Lagrangian method converges globally if the sequence  $\{x_k\}$  is bounded and if the Jacobian of the constraints at all the limit points of  $\{x_k\}$  has the column rank not

smaller than  $m$  (the number of constraints) (Conn, Gould, & Toint, 1991a, b). The difficulty with the augmented Lagrangian methods is the procedure for selecting the penalty parameters. Introducing the filter methods by Fletcher and Leyffer (2002) avoids this difficulty. ♦

The *penalty and merit function methods* combine the objective function and a measure of the constraint violation into a single function whose local minimizers correspond to the local minimizers of the problem (10.1). The convergence from the remote starting points can be ensured by forcing the descent of the penalty or the merit function by using one of the mechanisms discussed in the next section. The exact penalty functions are alternative to the augmented Lagrangian and are defined as

$$P_\rho(x) = f(x) + \rho \|c(x)\|, \quad (10.20)$$

where  $\rho > 0$  is the penalty parameter. In (10.20), the most used norm is  $l_1$ . If  $\rho \geq \|y^*\|_D$ , where  $y^*$  is the Lagrange multiplier corresponding to the nonlinear constraints and  $\|\cdot\|_D$  is the dual norm of  $\|\cdot\|$  (i.e., the  $l_\infty$  norm in the case of the  $l_1$  exact penalty function), then the local minimum  $x^*$  of the exact penalty function  $P_\rho(x)$  is a local minimum of the problem (10.1) (Fletcher, 1987). Therefore, the classical approach minimizes a sequence of penalty subproblems  $P_\rho(x)$  for an increasing sequence of the penalty parameters. Some other merit functions are known, for example, the quadratic penalty function  $f(x) + \rho \|c(x)\|_2^2$ , the oldest one. The corresponding algorithm based on this penalty function is convergent to a local solution only if the sequence of the penalty parameters is divergent to infinity. The global convergence is also ensured by using the augmented Lagrange and the penalty functions of the type  $f(x) + c(x)^T y + \rho \|c(x)\|$ . A major difficulty with these penalty functions is the possibility that near a strictly isolated minimum, the step given by the Newton method should not be accepted. This phenomenon is known as the Maratos effect (Maratos, 1978). The remedy for the Maratos effect is the introduction of second-order corrections or the use of the nonmonotone line-search. However, the greatest difficulty with the penalty methods is the selection and the updating of the penalty parameters.

The *filter methods* try to harmonize two objectives, always present in any optimization problem: minimization of the objective function and minimization of the violation of the constraints (see Chap. 18). The filter methods keep a record of the objective function value  $f_l = f(x_l)$  and of the constraint violation  $h_l = \|c(x_l)\|$  for a number of previous iterates  $x_l$ ,  $l \in F_k$ , where  $F_k$  is the filter at iteration  $k$  (Fletcher & Leyffer, 2002). A new iterate is acceptable in the filter if it improves either the objective function or the constraint violation, compared to all the previous iterates recorded in the filter. In other words, the point  $\hat{x}$  is acceptable in the filter if  $f(\hat{x}) \leq f(x_l) - \gamma h(x_l)$  or  $h(\hat{x}) \leq \beta h(x_l)$ , for any  $l \in F_k$ , where  $\gamma > 0$  and  $0 < \beta < 1$  are constants which ensure that the iterates cannot accumulate at infeasible limit points. To ensure the convergence to a local minimizer, the filter algorithm uses a standard sufficient reduction condition taken from the unconstrained optimization:

$$f(x_k) - f(x_k + d) \geq -\sigma m_k(d), \quad (10.21)$$

where  $\sigma > 0$  is the fraction of the predicted decrease and  $m_k(d)$  is the model reduction from the approximate subproblem. This condition is used only if the model predicts a decrease in the objective function. Therefore, the filter method uses a switching condition  $m_k(d) \geq \gamma h(x_k)^2$  to decide when (10.21) should be enforced. A new iterate that satisfies both these conditions is called an iteration of type  $f$ , while an iterate for which the switching condition fails is called an iteration of type  $h$ .

**Remark 10.4** The general characteristic of the filter method is that it avoids the complications with selecting and updating the penalty parameter from the augmented Lagrangian or the penalty function methods. However, the difficulty with this method is the selection of the parameters which determine

the convergence as well as the specification of the heuristics for ensuring the feasibility of the subproblems and the convergence to a local solution. ◆

## 10.5 The Refining the Local Model

The idea of refining the local model is to reduce the stepsize that is computed by the approximate subproblem. Two mechanisms for refining the local model are known: *line-search methods* and *trust-region methods*. Both these mechanisms can be used in the frame of any of the approximate subproblems and of any globalization strategies for the convergence from the remote starting points. Therefore, a great variety of nonlinear optimization algorithms result, with different capabilities for solving complex and large-scale optimization problems.

*The line-search methods* enforce the convergence by using a backtracking line-search along the direction  $s$ . For the sequential quadratic programming methods, the searching direction is given by the quadratic programming problem (10.7),  $s = d$ . For the interior point methods, the searching direction  $s = (\Delta x, \Delta y, \Delta z)$  is the solution of the primal-dual linear algebraic system (10.11). To ensure that the model produces a descent direction, one must have  $\nabla P(x_k)^T s < 0$ , where  $P(x)$  is a merit or a penalty function. A very popular line-search is the *Armijo rule* (Armijo, 1966) (see (Andrei, 2009e)). A prototype algorithm with line-search is as follows.

**Algorithm 10.3** Prototype of the line-search method for the nonlinear optimization

- |    |  |
|----|--|
| 1. | Choose an initial point $x_0 \in \mathbb{R}^n$ , as well as $0 < \sigma < 1$ . Set $k = 0$                                       |
| 2. | Evaluate a criterion for stopping the iterations. If $x_k$ is the optimal solution of the problem, stop; otherwise, go to step 3 |
| 3. | Approximately solve an approximate sub-problem of (10.1) around $x_k$ to find a search direction $s$                             |
| 4. | Verify whether $s$ is a descent direction if $\nabla P(x_k)^T s < 0$ , for a given merit function $P(x)$                         |
| 5. | Set $\alpha_0 = 1$ and $m = 0$   |
| 6. | If $P(x_k + \alpha_m s) \geq f(x_k) + \alpha_m \sigma s^T \nabla P(x_k)$ , then continue with step 7; otherwise, go to step 8    |
| 7. | Set $\alpha_{m+1} = \alpha_m/2$ and evaluate $P(x_k + \alpha_{m+1} s)$ . Set $m = m + 1$ . Continue with step 6                  |
| 8. | Set $k = k + 1$ and go to step 2   |

◆

Observe that the algorithm has two loops. The inner loop determines the stepsize by backtracking. The outer loop is responsible for the search direction determination, i.e., the solution of the problem.

The line-search methods with filters can be defined in a similar way, but, instead of checking the descent in the merit function, a filter method is used to check the acceptance to a filter (Leyffer, & Mahajan, 2010).

*The trust-region methods* restrict the step in an explicit way by adding a trust-region constraint of the form  $\|d\| \leq \Delta_k$ , where the used norm is  $l_1$ ,  $l_2$ , or  $l_\infty$ . Most methods use the  $l_\infty$  norm in the trust-region constraints, which can be represented by the simple bounds on the variables. The trust-region radius  $\Delta_k > 0$  is adjusted at every iteration, depending on how well the approximate subproblem agrees with the nonlinear optimization problem (10.1). A prototype of the optimization algorithm with trust-region is as follows.

**Algorithm 10.4** Prototype of the trust-region method for the nonlinear optimization

1. Choose an initial point  $x_0$  and a value for the trust-region radius  $\Delta_0 > 0$ . Set  $k = 0$
2. Evaluate a criterion for stopping the iterations. If  $x_k$  is the optimal solution of the problem, stop; otherwise, go to step 3
3. Approximately solve the local model of the problem (10.1), i.e. the sub-problem (10.7) in which a trust-region constraint  $\|d\| \leq \Delta_k$  is introduced
4. If  $x_k + d$  is sufficiently better than  $x_k$ , then increase  $\Delta_k$  and continue with step 5; otherwise, reduce  $\Delta_k$ , for example,  $\Delta_k = \Delta_k/2$  and go to step 3
5. Set  $k = k + 1$  and go to step 2

◆

The trust-region methods are related to the *regularization techniques*, which refer to the positive definiteness of the Hessian matrix  $B_k$ . These techniques are implemented by adding a multiple of the identity matrix to the Hessian. Locally, the solution of the regularized problem is equivalent to the solution of a trust-region problem with an  $l_2$  norm trust-region. Clearly, the disadvantage of the trust-region methods is that the subproblems may become inconsistent as  $\Delta_k \rightarrow 0$ . This situation can be dealt with in three different ways: a *penalty function* approach (Nocedal & Wright, 2006) (see Chap. 14), a *restoration phase* in which the algorithm minimizes the constraint violation (Fletcher & Leyffer, 2003) (see Chap. 18), or a *composite step* approach (Omojokun, 1989) (see Chap. 15). All these refining mechanisms of the local model are used in nonlinear optimization algorithms. The difficulties are related to the computation of the stepsize, to the selection of the merit function, and to the techniques for regularization.

In conclusion, a large variety of constrained nonlinear optimization algorithms are known. All of them use four major ingredients: a *local model* of the problem around the current point, a *globalization strategy* which ensures the convergence of the algorithm from remote initial points, a *refining mechanism* which reduces the stepsize computed by the local model to enforce the convergence, and *some convergence tests* which certify the type of the limit point obtained by the algorithm. The diversity of the optimization algorithms is given by the way in which all these ingredients are implemented. Besides, the optimization algorithms implement a multitude of other ingredients which are crucial in the efficiency and robustness for solving complex large-scale nonlinear optimization problems. Examples of such ingredients are sparse or dense linear algebra techniques, factorization of indefinite matrices, inertia correction of matrices, complement Schur computation, identification of the active constraints, initialization of the algorithms with respect to the primal and dual variables, updating procedures for the barrier parameter or a trust-region radius, updating the parameter of the central trajectory, solving the infeasibilities (elastic programming), etc.

*The most efficient and robust nonlinear optimization algorithms use combinations of the active-set methods (local models based on sequential linear or quadratic programming), or interior point methods with globalization strategies (augmented Lagrangian, penalty and merit function or filters methods) and refining mechanisms (line-search or trust-region methods), all of them in a frame where advanced computational linear algebra techniques are used.* The most advanced nonlinear optimization algorithms implementing the above concepts are illustrated in the chapters of this book.

**Notes and References**

This overview of the constrained nonlinear optimization methods is based on a deep analysis of the nonlinear optimization problems and algorithms for solving them. All the nonlinear optimization problems are solved in a reductionist approach. A current point is considered as an approximation to the solution of the original problem. In this current point, the problem is approximated through a

linear or quadratic model. This model is solved in a specific way. Thus, the obtained solution is considered as the next approximation to the solution of the original problem and as the starting point for solving the next approximation of the problem. For the termination of this iterative process, some criteria for stopping the iterations are used. A large variety of nonlinear optimization algorithms is known. The differences among them are the strategy for initialization, the construction of the local models, the globalization, the refining mechanism, and the stopping criteria. Extremely important is the computational linear algebra for implementing the different steps of the nonlinear optimization algorithms. These ingredients of an advanced nonlinear optimization algorithm are found in different publications, like Luenberger (1973, 1984); Gill, Murray, and Wright (1981); Dennis and Schnabel (1983); Peressini, Sullivan, and Uhl (1988); Bazaraa, Sherali, and Shetty (1993); Bertsekas (1999); Conn, Gould, and Toint (2000); Nocedal, and Wright (2006); Sun and Yuan (2006); Bartholomew-Biggs (2008); Kelley (1999); Andrei (1999a, 1999b, 2009e, 2009f, 2015a, 2017c, 2020a); Leyffer and Mahajan (2010); and Luenberger and Ye (2016).

It is noteworthy that the optimization methods and techniques have two main traditions: the *linear programming* and the *calculus of variations*. These two topics have evolved along two disjoint paths with *no connection* between them. The situation dramatically changed in 1984, when Karmarkar (1984) announced a new polynomial-time interior-point algorithm for linear programming, for which he reported performance times that were 50 times faster than those of the simplex algorithm. Immediately after, in 1985, it was shown that there was a formal equivalence between Karmarkar's method and the classical logarithmic barrier method applied to linear programming problems. Since then, interior methods have continued to transform both the theory and the practice of constrained optimization. This was the *first unification* between linear programming and nonlinear optimization. At present, both the simplex algorithm and different variants of the interior-point algorithms coexist as important methods for solving linear and nonlinear optimization problems.

On the other hand, a wide variety of problems arising in the system and control theory can be reduced to a few standard convex or quasiconvex optimization problems involving linear matrix inequalities. In other words, many constraints in the system and control theory, including convex quadratic inequalities, matrix norm inequalities, Lyapunov matrix inequalities, the positive-real lemma, the bounded-real lemma, etc., can be expressed as linear matrix inequalities. Interior-point methods are important since these optimization problems can be solved numerically very efficiently using the polynomial-time algorithms (see Boyd, El Ghaoui, Feron, & Balakrishnan, 1994), thus achieving the *second unification* of optimization and of the system and control theory.

We can see that the interior-point methods were the instrument that unified linear and nonlinear optimization with the system and control theory by emphasizing the problem of complexity of the optimization algorithms. Modern constrained optimization methods are based on a combination of the interior-point with the sequential linear-quadratic programming with line-search or filters in an advanced computational linear algebra environment, as we can see in the chapters of this book.



# Optimality Conditions for Nonlinear Optimization

11

The optimization problems considered in this book involve the minimization or maximization of a function of several real variables subject to one or more constraints. The constraints may be the non-negativity of variables, the simple bounds on variables, and the equalities or inequalities as functions of these variables. These problems are known as continuous nonlinear constrained optimization or nonlinear programming.

The purpose of this chapter is to introduce the main concepts and the fundamental results in nonlinear optimization, known as optimality conditions. Plenty of very good books dedicated to these problems are known in literature: Luenberger (1973); Gill, Murray, and Wright (1981); Peressini, Sullivan, and Uhl (1988); Bazaraa, Sherali, and Shetty (1993); Bertsekas (1999); Boyd and Vandenberghe (2004); Nocedal and Wright (2006); Sun and Yuan (2006); Chachuat (2007); Andrei (2009e, 2015a, 2017c); etc.

The general continuous nonlinear optimization problem is expressed as

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_i(x) \leq 0, i = 1, \dots, m, \\ & h_i(x) = 0, i = 1, \dots, p, \end{aligned} \tag{11.1}$$

where  $x \in \mathbb{R}^n$ ,  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , are continuously differentiable functions. Usually, the function  $f$  is called the *objective function*. Each of the constraints  $c_i(x) \leq 0$ ,  $i = 1, \dots, m$ , is called an *inequality constraint*, and each  $h_i(x) = 0$ ,  $i = 1, \dots, p$ , is called an *equality constraint*. Often, (11.1) is called a *nonlinear program*. A vector  $x$  satisfying all the equality and inequality constraints is called a *feasible solution (point)* to the problem (11.1). Define

$$X = \{x : c_i(x) \leq 0, i = 1, \dots, m, h_i(x) = 0, i = 1, \dots, p\}$$

as the *feasible region* (or *feasible domain*).

In this chapter, we are interested to specify what is meant by optimality for the general nonlinear optimization problem and to give conditions under which a solution for the problem (11.1) exists. Both necessary and sufficient conditions for optimality are presented, starting with unconstrained problems and continuing with problems with inequality constraints and equality constraints and finally for general nonlinear optimization problems with equality and inequality constraints. We emphasize that *conditions which are satisfied at a local minimizer  $x^*$  are the necessary conditions*;

conditions which guarantee that  $x^*$  is a local minimizer are the sufficient conditions. The key to understand the nonlinear optimization is the Karush-Kuhn-Tucker (KKT) optimality conditions. This is a major result which identifies an algebraic system of equalities and inequalities, which corresponds to the solution to any nonlinear optimization problem. This system can often be used to develop algorithms for computing a solution for the problem or to get some additional information about the sensitivity of the minimum value of the problem subject to changes in the constraints. In general, many optimization algorithms can be interpreted as methods for numerically solving the KKT nonlinear system of equations.

In the mathematical optimization, the KKT conditions are first-order necessary conditions for a solution in nonlinear optimization to be optimal, provided that some regularity conditions are satisfied. For problems with inequality constraints, the KKT approach generalizes the method of Lagrange multipliers, which allows only equality constraints. For the development of the KKT optimality conditions, three possible approaches can be used. One is based on the *separation and support theorems* from the convex set theory. Another one uses the *penalty functions*, and the third one comes from the *theory of Lagrange multipliers*. Each of these approaches has its own virtues and provides its own insights on the KKT Theorem. In this text, we consider the optimality conditions for the continuous nonlinear optimization (the mathematical programming), using the formalism of Lagrange.

## 11.1 General Concepts in Nonlinear Optimization

In the following, we shall present some definitions and results used in the context of nonlinear programming. At the same time, we shall define a particular class of nonlinear programs, the convex programming. In this section,  $X \subset \mathbb{R}$  denotes a nonempty set of real numbers.

**Definition 11.1** (*Upper Bound, Lower Bound*). A real number  $\alpha$  is called an upper bound for  $X$  if  $x \leq \alpha$  for all  $x \in X$ . The set  $X$  is said to be bounded above if it has an upper bound. Similarly, a real number  $\alpha$  is called a lower bound for  $X$  if  $x \geq \alpha$  for all  $x \in X$ . The set  $X$  is said to be bounded below if it has a lower bound.

**Definition 11.2** (*Least Upper Bound, Greatest Lower Bound*). A real number  $\alpha$  is called the least upper bound (or supremum, or sup) of  $X$ , if (i)  $\alpha$  is an upper bound for  $X$ ; and (ii) there does not exist an upper bound for  $X$  that is strictly smaller than  $\alpha$ . The supremum, if it exists, is unique and is denoted by  $\sup X$ . A real number  $\alpha$  is called the greatest lower bound (or infimum, or inf) of  $X$ , if (i)  $\alpha$  is a lower bound for  $X$ ; and (ii) there does not exist a lower bound for  $X$  that is strictly greater than  $\alpha$ . The infimum, if it exists, is unique and is denoted by  $\inf X$ .

It is worth saying that for supers and infs, the following equivalent definition is useful.

**Definition 11.3** (*Supremum, Infimum*). The supremum of  $X$ , provided it exists, is the least upper bound for  $X$ , i.e., a real number  $\alpha$ , satisfying (i)  $z \leq \alpha$  for any  $z \in X$  and (ii) for any  $\bar{\alpha} < \alpha$ , there exists  $z \in X$  such that  $z > \bar{\alpha}$ . Similarly, the infimum of  $X$ , provided it exists, is the greatest lower bound for  $X$ , i.e., a real number  $\alpha$  satisfying (i)  $z \geq \alpha$  for any  $z \in X$  and (ii) for any  $\bar{\alpha} > \alpha$ , there exists  $z \in X$  such that  $z < \bar{\alpha}$ .

**Definition 11.4** (*Maximum, Minimum*). The maximum of a set  $X$  is its largest element if such an element exists. The minimum of a set  $X$  is its smallest element if such an element exists.

The key differences between the *supremum* and the *maximum* concepts are as follows. If a set has a maximum, then the maximum is also a supremum for this set, but the converse is not true. A finite set has always a maximum which is also its supremum, but an infinite set does not need to have a maximum. The supremum of a set  $X$  does not need to be an element of the set  $X$  itself, but the maximum of  $X$  must always be an element of  $X$ .

Concerning the existence of infima and suprema in  $\mathbb{R}$ , fundamental is the *axiom of completeness*. “*If a nonempty subset of real numbers has an upper bound, then it has a least upper bound. If a nonempty set of real numbers has a lower bound, it has a greatest lower bound.*” In other words, the completeness axiom guarantees that, for any nonempty set of real numbers that is bounded above, a supremum exists (in contrast to the maximum, which may or may not exist).

Let us consider the minimization problem

$$\min \{f(x) : x \in X\}, \quad (11.2)$$

where  $X \subset \mathbb{R}^n$  represents the *feasible set*. Any point  $x \in X$  is a *feasible point* or an *admissible point*. Any point  $x \in \mathbb{R}^n \setminus X$  is called to be *infeasible*.

**Definition 11.5 (Global Minimum, Strict Global Minimum).** A point  $x^* \in X$  is said to be a *global minimum* off on  $X$  if  $f(x) \geq f(x^*)$  for any  $x \in X$ . A point  $x^* \in X$  is said to be a *strict global minimum* off on  $X$  if  $f(x) > f(x^*)$  for any  $x \in X$  with  $x \neq x^*$ .

**Definition 11.6 (Global Maximum, Strict Global Maximum).** A point  $x^* \in X$  is said to be a *global maximum* off on  $X$  if  $f(x) \leq f(x^*)$  for any  $x \in X$ . It is a *strict global maximum* off on  $X$  if  $f(x) < f(x^*)$  for any  $x \in X$  with  $x \neq x^*$ .

The point  $x^*$  is called an *optimal solution* of the optimization problem. The real number  $f(x^*)$  is known as the *optimal value* of the objective function subject to the constraints  $x \in X$ .

Observe the distinction between the minimum/maximum and the infimum/supremum. The value  $\min\{f(x) : x \in X\}$  must be attained at one or more points  $x \in X$ . On the other hand, the value  $\inf\{f(x) : x \in X\}$  does not necessarily have to be attained at any points  $x \in X$ . However, if a minimum (maximum) exists, then its optimal value equals the infimum (supremum).

If a minimum exists, it is not necessarily unique. That is, there may be a finite number or even an infinite number of feasible points  $x^*$  that satisfy the inequality  $f(x) \geq f(x^*)$  for any  $x \in X$ . The notation

$$\arg \min \{f(x) : x \in X\} \triangleq \{x \in X : f(x) = \inf \{f(x) : x \in X\}\}$$

is reserved for the set of the minima of function  $f$  on  $X$ , that is, a set in  $\mathbb{R}^n$ .

**Definition 11.7 (Local Minimum, Strict Local Minimum).** A point  $x^* \in X$  is said to be a *local minimum* off on  $X$  if there exists  $\epsilon > 0$  such that  $f(x) \geq f(x^*)$  for any  $x \in B(x^*, \epsilon) \cap X$ , where  $B(x^*, \epsilon)$  is the open ball centered at  $x^*$  of radius  $\epsilon$ . Similarly, a point  $x^* \in X$  is said to be a *strict local minimum* of  $f$  on  $X$  if there exists  $\epsilon > 0$  such that  $f(x) > f(x^*)$  for any  $x \in B(x^*, \epsilon) \setminus \{x^*\} \cap X$ .

**Definition 11.8 (Local Maximum, Strict Local Maximum).** A point  $x^* \in X$  is said to be a *local maximum* off on  $X$  if there exists  $\epsilon > 0$  such that  $f(x) \leq f(x^*)$  for any  $x \in B(x^*, \epsilon) \cap X$ . Similarly, it is said to be a *strict local maximum* off on  $X$  if there exists  $\epsilon > 0$  such that  $f(x) < f(x^*)$  for any  $x \in B(x^*, \epsilon) \setminus \{x^*\} \cap X$ .

A fundamental problem in optimizing a function on a given set is whether a minimum or a maximum point exists in the given set. This result is known as the theorem of Weierstrass. It shows that if  $X$  is nonempty, closed, and bounded, and  $f$  is continuous on  $X$ , then a minimum of  $f$  on  $X$  exists.

**Theorem 11.1** (Weierstrass). *Let  $X$  be a nonempty and compact set. Assume that  $f : X \rightarrow \mathbb{R}$  is continuous on  $X$ . Then, the problem  $\min\{f(x) : x \in X\}$  attains its minimum.*

**Proof** If  $f$  is continuous on  $X$  and  $X$  is both closed and bounded, it follows that  $f$  is bounded below on  $X$ . Now, since  $X$  is nonempty, from the axiom of completeness, there exists a greatest lower bound  $\alpha = \inf\{f(x) : x \in X\}$ . Let  $0 < \varepsilon < 1$  and consider the set  $X_k = \{x \in X : \alpha \leq f(x) \leq \alpha + \varepsilon^k\}$ ,  $k = 1, 2, \dots$ . By the definition of the infimum, for each  $k$  it follows that  $X_k \neq \emptyset$ . Therefore, a sequence of points  $\{x_k\} \subset X$  can be constructed by selecting a point  $x_k$  for each  $k = 1, 2, \dots$ . Since  $X$  is bounded, there exists a convergent subsequence  $\{x_k\}_K \subset X$  indexed by the set  $K \subset \mathbb{N}$  with  $x^*$  as its limit. Since  $X$  is closed, it follows that  $x^* \in X$ . By continuity of  $f$  on  $X$ , since  $\alpha \leq f(x_k) \leq \alpha + \varepsilon^k$ , we have  $\alpha = \lim_{k \rightarrow \infty, k \in K} f(x_k) = f(x^*)$ . Therefore, there exists a solution  $x^* \in X$  so that  $f(x^*) = \alpha = \inf\{f(x) : x \in X\}$ , i.e.,  $x^*$  is a minimizing solution. ◆

All the hypotheses of this theorem are important. The feasible set must be *nonempty*; otherwise, there are no feasible points at which the minimum is attained. The feasible set must be *closed*, i.e., it must contain its boundary points. The objective function must be *continuous* on the feasible set; otherwise, the limit at a point may not exist or it may be different from the value of the function at that point. Finally, the feasible set must be *bounded*; otherwise, even continuous functions can be unbounded on the feasible set.

**Definition 11.9** (Convex Program). *Let  $C$  be a convex set in  $\mathbb{R}^n$  and let  $f : C \rightarrow \mathbb{R}^n$  be a convex function on  $C$ . Then,  $\min\{f(x) : x \in C\}$  is called a convex optimization problem, or a convex program.*

The fundamental result in convex programming is the following theorem.

**Theorem 11.2** *Let  $x^*$  be a local minimum of a convex program. Then,  $x^*$  is also a global minimum.*

**Proof** If  $x^*$  is a local minimum, then there exists  $\varepsilon > 0$  such that  $f(x) \geq f(x^*)$  for any  $x \in B(x^*, \varepsilon)$ . Now, suppose that  $x^*$  is not a global minimum. Then, there exists  $y \in C$  such that  $f(y) < f(x^*)$ . Let  $\lambda \in (0, 1)$  be chosen such that the point  $z = \lambda y + (1 - \lambda)x^* \in B(x^*, \varepsilon)$ . By convexity of  $C$ ,  $z \in C$ . Therefore,

$$f(z) \leq \lambda f(y) + (1 - \lambda)f(x^*) < \lambda f(x^*) + (1 - \lambda)f(x^*) = f(x^*),$$

which is a contradiction, since  $x^*$  is a local minimum. ◆

## 11.2 Optimality Conditions for Unconstrained Optimization

Let us consider the problem of minimizing a function  $f(x)$  without constraints on the variables  $x \in \mathbb{R}^n$ :

$$\min\{f(x) : x \in \mathbb{R}^n\}.$$

For a given point  $x \in \mathbb{R}^n$ , the optimality conditions determine whether or not a point is a local or a global minimum of  $f$ . To formulate the optimality conditions, it is necessary to introduce some concepts which characterize an improving direction along which the values of the function  $f$  decrease.

**Definition 11.10** (*Descent Direction*). Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is continuous at  $x^*$ . A vector  $d \in \mathbb{R}^n$  is a descent direction for  $f$  at  $x^*$  if there exists  $\delta > 0$  such that  $f(x^* + \lambda d) < f(x^*)$  for any  $\lambda \in (0, \delta)$ . The cone of descent directions at  $x^*$ , denoted by  $C_{dd}(x^*)$ , is given by

$$C_{dd}(x^*) = \{d : \text{there exists } \delta > 0 \text{ such that } f(x^* + \lambda d) < f(x^*) \text{ for any } \lambda \in (0, \delta)\}.$$

Assume that  $f$  is a differentiable function. To get an algebraic characterization for a descent direction for  $f$  at  $x^*$  let us define the set

$$C_0(x^*) = \left\{ d : \nabla f(x^*)^T d < 0 \right\}.$$

The following result shows that every  $d \in C_0(x^*)$  is a descent direction at  $x^*$ .

**Proposition 11.1** (*Algebraic Characterization of a Descent Direction*). Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable at  $x^*$ . If there exists a vector  $d$  such that  $\nabla f(x^*)^T d < 0$ , then  $d$  is a descent direction for  $f$  at  $x^*$ , i.e.,  $C_0(x^*) \subseteq C_{dd}(x^*)$ .

**Proof** Since  $f$  is continuously differentiable at  $x^*$ , it follows that

$$f(x^* + \lambda d) = f(x^*) + \lambda \nabla f(x^*)^T d + \lambda \|d\| o(\lambda d),$$

where  $\lim_{\lambda \rightarrow 0} o(\lambda d) = 0$ . Therefore,

$$\frac{f(x^* + \lambda d) - f(x^*)}{\lambda} = \nabla f(x^*)^T d + \|d\| o(\lambda d).$$

Since  $\nabla f(x^*)^T d < 0$  and  $\lim_{\lambda \rightarrow 0} o(\lambda d) = 0$ , it follows that there exists a  $\delta > 0$  such that  $\nabla f(x^*)^T d + \|d\| o(\lambda d) < 0$  for all  $\lambda \in (0, \delta)$ .  $\blacklozenge$

**Theorem 11.3** (*First-Order Necessary Conditions for a Local Minimum*). Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable at  $x^*$ . If  $x^*$  is a local minimum, then  $\nabla f(x^*) = 0$ .

**Proof** Suppose that  $\nabla f(x^*) \neq 0$ . If we consider  $d = -\nabla f(x^*)$ , then  $\nabla f(x^*)^T d = -\|\nabla f(x^*)\|^2 < 0$ . By Proposition 11.1, there exists a  $\delta > 0$  such that for any  $\lambda \in (0, \delta)$ ,  $f(x^* + \lambda d) < f(x^*)$ . However, this is in contradiction with the assumption that  $x^*$  is a local minimum for  $f$ .  $\blacklozenge$

Observe that the above necessary condition represents a system of  $n$  algebraic nonlinear equations. All the points  $x^*$  which solve the system  $\nabla f(x) = 0$  are called *stationary points*. Clearly, the stationary points need not all be local minima. They could very well be local maxima or even saddle points. In order to characterize a local minimum, we need more restrictive necessary conditions involving the Hessian matrix of the function  $f$ .

**Theorem 11.4** (*Second-Order Necessary Conditions for a Local Minimum*). Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable at point  $x^*$ . If  $x^*$  is a local minimum, then  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive semidefinite.

**Proof** Consider an arbitrary direction  $d$ . Then, using the differentiability of  $f$  at  $x^*$ , we get

$$f(x^* + \lambda d) = f(x^*) + \lambda \nabla f(x^*)^T d + \frac{1}{2} \lambda^2 d^T \nabla^2 f(x^*) d + \lambda^2 \|d\|^2 o(\lambda d),$$

where  $\lim_{\lambda \rightarrow 0} o(\lambda d) = 0$ . Since  $x^*$  is a local minimum,  $\nabla f(x^*) = 0$ . Therefore,

$$\frac{f(x^* + \lambda d) - f(x^*)}{\lambda^2} = \frac{1}{2} d^T \nabla^2 f(x^*) d + \|d\|^2 o(\lambda d).$$

Since  $x^*$  is a local minimum, for  $\lambda$  sufficiently small,  $f(x^* + \lambda d) \geq f(x^*)$ . For  $\lambda \rightarrow 0$ , it follows from the above equality that  $d^T \nabla^2 f(x^*) d \geq 0$ . Since  $d$  is an arbitrary direction, it follows that  $\nabla^2 f(x^*)$  is positive semidefinite.  $\blacklozenge$

In the above theorems, we have presented the *necessary* conditions for a point  $x^*$  to be a local minimum, i.e., these conditions *must be satisfied* at every local minimum solution. However, a point satisfying these necessary conditions need not be a local minimum. In the following theorems, the *sufficient* conditions for a global minimum are given, provided that the objective function is convex on  $\mathbb{R}^n$  (see Appendix A).

**Theorem 11.5 (First-Order Sufficient Conditions for a Strict Local Minimum).** Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable at  $x^*$  and convex on  $\mathbb{R}^n$ . If  $\nabla f(x^*) = 0$ , then  $x^*$  is a global minimum of  $f$  on  $\mathbb{R}^n$ .

**Proof** Since  $f$  is convex on  $\mathbb{R}^n$  and continuously differentiable at  $x^*$ , then from the property of convex functions given by the Proposition A4.5, it follows that for any  $x \in \mathbb{R}^n$   $f(x) \geq f(x^*) + \nabla f(x^*)^T (x - x^*)$ . However,  $x^*$  is a stationary point, i.e.,  $f(x) \geq f(x^*)$  for any  $x \in \mathbb{R}^n$ .  $\blacklozenge$

The following theorem gives the second-order sufficient conditions characterizing a local minimum point for those functions which are strictly convex in a neighborhood of the minimum point.

**Theorem 11.6 (Second-Order Sufficient Conditions for a Strict Local Minimum).** Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable at point  $x^*$ . If  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite, then  $x^*$  is a local minimum of  $f$ .

**Proof** Since  $f$  is twice continuously differentiable, for any  $d \in \mathbb{R}^n$ , we can write

$$f(x^* + d) = f(x^*) + \nabla f(x^*)^T d + \frac{1}{2} d^T \nabla^2 f(x^*) d + \|d\|^2 o(d),$$

where  $\lim_{d \rightarrow 0} o(d) = 0$ . Let  $\lambda$  be the smallest eigenvalue of  $\nabla^2 f(x^*)$ . Since  $\nabla^2 f(x^*)$  is positive definite, it follows that  $\lambda > 0$  and  $d^T \nabla^2 f(x^*) d \geq \lambda \|d\|^2$ . Therefore, since  $\nabla f(x^*) = 0$ , we can write

$$f(x^* + d) - f(x^*) \geq \left[ \frac{\lambda}{2} + o(d) \right] \|d\|^2.$$

Since  $\lim_{d \rightarrow 0} o(d) = 0$ , then there exists a  $\eta > 0$  such that  $|o(d)| < \lambda/4$  for any  $d \in B(0, \eta)$ . Hence,

$$f(x^* + d) - f(x^*) \geq \frac{\lambda}{4} \|d\|^2 > 0$$

for any  $d \in B(0, \eta) \setminus \{0\}$ , i.e.,  $x^*$  is a strict local minimum of function  $f$ .  $\blacklozenge$

If we assume  $f$  to be twice continuously differentiable, we observe that, since  $\nabla^2 f(x^*)$  is positive definite,  $\nabla^2 f(x^*)$  is positive definite in a small neighborhood of  $x^*$  and so  $f$  is strictly convex in a small neighborhood of  $x^*$ . Therefore,  $x^*$  is a strict local minimum, that is, it is the unique global minimum over a small neighborhood of  $x^*$ .

### 11.3 Optimality Conditions for Problems with Inequality Constraints

In the following, we shall discuss the nonlinear optimization problems with inequality constraints:

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & \quad x \in X, \end{aligned} \tag{11.3}$$

where  $X$  is a general set. Further on, we will be more specific and will define the problem as to minimize  $f(x)$  subject to  $c(x) \leq 0$ , where  $c(x)$  is the vector of constraint functions.

**Definition 11.11** (*Feasible direction*). Let  $X$  be a nonempty set in  $\mathbb{R}^n$ . A nonzero vector  $d \in \mathbb{R}^n$  is a feasible direction at  $x^* \in \text{cl}(X)$  if there exists a  $\delta > 0$  such that  $x^* + \eta d \in X$  for any  $\eta \in (0, \delta)$ . Moreover, the cone of feasible directions at  $x^*$ , denoted by  $C_{fd}(x^*)$ , is given by

$$C_{fd}(x^*) \triangleq \{d \neq 0, \text{ there is } \delta > 0 \text{ such that } x^* + \eta d \in X, \text{ for any } \eta \in (0, \delta)\}.$$

Clearly, a small movement from  $x^*$  along the direction  $d \in C_{fd}(x^*)$  leads to feasible points. On the other hand, a similar movement along a direction  $d \in C_0(x^*)$  (see Definition 11.10) leads to solutions which improve the value of the objective function. The following theorem, which gives a geometrical interpretation of the local minima, shows that a necessary condition for local optimality is that *every improving direction is not a feasible direction*.

**Theorem 11.7** (*Geometric Necessary Condition for a Local Minimum*). Let  $X$  be a nonempty set in  $\mathbb{R}^n$ , and let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a continuously differentiable function. Suppose that  $x^*$  is a local minimum of the problem (11.3). Then,  $C_0(x^*) \cap C_{fd}(x^*) = \emptyset$ .

**Proof** Suppose that there exists a nonzero vector  $d \in C_0(x^*) \cap C_{fd}(x^*)$ . By the Proposition 11.1 of the algebraic characterization of a descent direction, there exists  $\delta_1 > 0$  such that  $f(x^* + \eta d) < f(x^*)$  for any  $\eta \in (0, \delta_1)$ . On the other hand, by Definition 11.11 of feasible direction, there exists  $\delta_2 > 0$  such that  $x^* + \eta d \in X$  for any  $\eta \in (0, \delta_2)$ . Therefore, there exists  $x \in B(x^*, \eta) \cap X$  such that  $f(x^* + \eta d) < f(x^*)$ , for every  $\eta \in (0, \min\{\delta_1, \delta_2\})$ , which contradicts the assumption that  $x^*$  is a local minimum of  $f$  on  $X$  (see Definition 11.7).  $\blacklozenge$

So far, we have obtained a geometric characterization of the optimality condition for the problem (11.3) given by Theorem 11.7 where  $C_{fd}(x^*)$  is the cone of feasible directions. To get a practical optimality condition, implementable in computer programs, we need to convert this geometric condition into an algebraic one. For this, we introduce the concept of *active constraints* at  $x^*$  and define a cone  $C_{ac}(x^*) \subseteq C_{fd}(x^*)$  in terms of the gradients of these active constraints. Now, we specify the feasible set  $X$  as

$$X \triangleq \{x : c_i(x) \leq 0, i = 1, \dots, m\}, \quad (11.4)$$

where  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , are continuous functions. Define the vector  $c(x) = [c_1(x), \dots, c_m(x)]$ .

**Definition 11.12** (*Active Constraint, Active Set*). Let  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , which define the feasible set  $X = \{x : c_i(x) \leq 0, i = 1, \dots, m\}$ , and consider  $x^* \in X$  a feasible point. For each  $i = 1, \dots, m$ , the constraint  $c_i$  is said to be active or binding at  $x^*$  if  $c_i(x^*) = 0$ . It is said to be inactive at  $x^*$  if  $c_i(x^*) < 0$ . The set

$$A(x^*) \triangleq \{i : c_i(x^*) = 0\}$$

denotes the set of active constraints at  $x^*$ .

The following proposition gives an algebraic characterization of a feasible direction showing the relation between a cone  $C_{ac}(x^*)$  expressed in terms of the gradients of the active constraints and the cone of the feasible directions.

**Proposition 11.2** (*Algebraic Characterization of a Feasible Direction*). Let  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be continuously differentiable functions, and consider the feasible set  $X = \{x : c_i(x) \leq 0, i = 1, \dots, m\}$ . For any feasible point  $x^* \in X$ , we have

$$C_{ac}(x^*) \triangleq \left\{ d : \nabla c_i(x^*)^T d < 0, i \in A(x^*) \right\} \subseteq C_{fd}(x^*).$$

**Proof** Suppose that  $C_{ac}(x^*)$  is a nonempty set. Let  $d \in C_{ac}(x^*)$ . Observe that  $\nabla c_i(x^*)^T d < 0$  for each  $i \in A(x^*)$ . Therefore, by Proposition 11.1, the algebraic characterization of a descent direction, it follows that  $d$  is a descent direction for  $c_i$  at  $x^*$ , i.e., there exists  $\delta_2 > 0$  such that  $c_i(x^* + \eta d) < c_i(x^*) = 0$  for any  $\eta \in (0, \delta_2)$  and for any  $i \in A(x^*)$ . On the other hand, since  $c_i$  is differentiable at  $x^*$ , it follows that it is continuous at  $x^*$ . Therefore, since  $c_i(x^*) < 0$  and  $c_i$  is continuous at  $x^*$  for each  $i \notin A(x^*)$ , there exists  $\delta_1 > 0$  such that  $c_i(x^* + \eta d) < 0$  for any  $\eta \in (0, \delta_1)$  and for any  $i \notin A(x^*)$ . Besides, for all  $\eta \in (0, \min\{\delta_1, \delta_2\})$ , the points  $x^* + \eta d \in X$ . Therefore, by Definition 11.11 of feasible direction,  $d \in C_{fd}(x^*)$ .  $\blacklozenge$

**Remark 11.1** From the Theorem 11.7, we know that  $C_0(x^*) \cap C_{fd}(x^*) = \emptyset$ . However, from Proposition 11.2, we have that  $C_{ac}(x^*) \subseteq C_{fd}(x^*)$ . Therefore,  $C_0(x^*) \cap C_{ac}(x^*) = \emptyset$ , for any local optimal solution  $x^*$ .  $\blacklozenge$

The above geometric characterization of the local optimal solution (see Theorem 11.7) holds either at the interior points  $\text{int}X \triangleq \{x \in \mathbb{R}^n : c_i(x) < 0, i = 1, \dots, m\}$ , or at the boundary points. For the interior points, any direction is feasible, and the necessary condition  $C_0(x^*) \cap C_{ac}(x^*) = \emptyset$  reduces to the very well-known condition  $\nabla f(x^*) = 0$ , which is identical to the necessary optimal condition for the unconstrained optimization (see Theorem 11.3).

It is important to notice that the condition  $C_0(x^*) \cap C_{ac}(x^*) = \emptyset$  can be satisfied by non-optimal points, i.e., this condition is *necessary but not sufficient* for a point  $x^*$  to be a local minimum of the function  $f$  on  $X$ . For example, at any point  $x^*$  for which  $\nabla c_i(x^*) = 0$ , for an arbitrary index  $i \in A(x^*)$ , the condition  $C_0(x^*) \cap C_{ac}(x^*) = \emptyset$  is trivially satisfied.

In the following, in order to get an algebraic necessary optimality condition to be used in numerical computation, we want to transform the geometric necessary optimality condition  $C_0(x^*) \cap C_{ac}(x^*) = \emptyset$  to a statement in terms of the gradient of the objective function and the

gradient of the constraints. Thus, the first-order optimality conditions, known as the *Karush-Kuhn-Tucker (KKT) necessary conditions*, are obtained. In order to formulate the KKT conditions, we need to introduce the concept of *regular point* and of *KKT point*.

**Definition 11.13 (Regular Point–Inequality Constraints).** Let  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be continuously differentiable functions, and consider the feasible set  $X = \{x \in \mathbb{R}^n : c_i(x) \leq 0, i = 1, \dots, m\}$ . A point  $x^* \in X$  is a regular point if the gradient vectors  $\nabla c_i(x^*)$ ,  $i \in A(x^*)$ , are linear independent, i.e.,

$$\text{rank}[\nabla c_i(x^*), i \in A(x^*)] = \text{card}(A(x^*)).$$

**Definition 11.14 (KKT Point).** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be continuously differentiable functions. Consider the problem  $\min\{f(x) : c(x) \leq 0\}$ . If a point  $(x^*, \mu^*) \in \mathbb{R}^n \times \mathbb{R}^m$  satisfies the algebraic conditions:

$$\nabla f(x^*) + (\mu^*)^T \nabla c(x^*) = 0, \quad (11.5)$$

$$\mu^* \geq 0, \quad (11.6)$$

$$c(x^*) \leq 0, \quad (11.7)$$

$$(\mu^*)^T c(x^*) = 0. \quad (11.8)$$

then  $(x^*, \mu^*)$  is called a KKT point.

In Definition 11.14, the scalars  $\mu_i$ ,  $i = 1, \dots, m$ , are called the *Lagrange multipliers*. The first condition (11.5) is known as the *primal feasibility* condition. The conditions (11.6) and (11.7) are known as *dual feasibility* conditions. The last condition (11.8), expressed as  $\mu_i^* c_i(x^*) = 0$ ,  $i = 1, \dots, m$ , is the *complementarity slackness* (or *transversality*) condition.

We are now in the position to present the KKT necessary conditions for the optimality of the nonlinear optimization problem with inequality constraints. For this, a very useful result is given by the Theorem of Gordan (see Theorem A4.3). This is extensively used in the derivation of the optimality conditions of linear and nonlinear programming problems.

**Theorem 11.8 (KKT Necessary Conditions).** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be continuously differentiable functions. Consider the problem  $\min\{f(x) : c(x) \leq 0\}$ . If  $x^*$  is a local minimum and a regular point of the constraints, then there exists a unique vector  $\mu^*$  such that  $(x^*, \mu^*)$  is a KKT point.

**Proof** We know that  $x^*$  is an optimal solution for the problem  $\min\{f(x) : c(x) \leq 0\}$ . Therefore, using the Remark 11.1, no direction  $d \in \mathbb{R}^n$  exists such that  $\nabla f(x^*)^T d < 0$  and  $\nabla c_i(x^*)^T d < 0$ , for any  $i \in A(x^*)$ , are simultaneously satisfied. Now, let  $A \in \mathbb{R}^{(\text{card}(A(x^*))+1) \times n}$  be the matrix whose rows are  $\nabla f(x^*)^T$  and  $\nabla c_i(x^*)^T$ ,  $i \in A(x^*)$ . By the Gordan Theorem (see Theorem A4.3), there exists a nonzero vector  $p = [u_0, u_1, \dots, u_{\text{card}(A(x^*))}] \geq 0$  in  $\mathbb{R}^{\text{card}(A(x^*))+1}$  such that  $A^T p = 0$ . Therefore,

$$u_0 \nabla f(x^*) + \sum_{i \in A(x^*)} u_i \nabla c_i(x^*) = 0,$$

where  $u_0 \geq 0$  and  $u_i \geq 0$  for  $i \in A(x^*)$  and  $[u_0, u_1, \dots, u_{\text{card}(A(x^*))}]$  is not the vector zero. Considering  $u_i = 0$  for all  $i \notin A(x^*)$ , the following conditions are obtained:

$$\begin{aligned} u_0 \nabla f(x^*) + u^T \nabla c(x^*) &= 0, \\ u^T c(x^*) &= 0, \\ u_0 \geq 0, u \geq 0, \\ (u_0, u) &\neq (0, 0), \end{aligned}$$

where  $u$  is the vector with components  $u_i$  for  $i = 1, \dots, m$ , some of them being  $u_0, u_1, \dots, u_{\text{card}(A(x^*))}$  and the others being zero. Observe that  $u_0 \neq 0$ , because otherwise the assumption that the gradient of the active constraints are linear independent at  $x^*$  is not satisfied. Now, considering the vector  $\mu^*$  as the vector  $u$  whose components are divided by  $u_0$ , we get that  $(x^*, \mu^*)$  is a KKT point.  $\blacklozenge$

The above theorem shows the importance of the active constraints. A major difficulty in applying this result to get optimization algorithms for solving (11.3) with (11.4) is that we do not know in advance which constraints are active and which are inactive at the solution of the problem. In other words, we do not know the active set. The majority of algorithms for solving this optimization problem with inequalities face this difficulty of identifying the active set. Of course, the idea of investigating all possible active sets of a problem in order to get the points satisfying the KKT conditions is usually impractical. More exactly, this difficulty may be detailed as follows:

### Combinatorial Difficulty for Solving the Inequality Constrained Optimization

In direct parlance, the difficulty of inequality constrained optimization problems lies in deciding which of constraints are active at the solution and which are not. One approach used in the active-set methods starts by making a guess of the optimal active set  $A(x^*)$ , that is, the set of constraints that are satisfied as equalities at the solution  $x^*$ . This set is called the *working set* and is denoted by  $W$ . With this, solve the problem in which the constraints from the working set are imposed as equalities and the constraints which are not in  $W$  are ignored. Then, check to see if there is a choice of the Lagrange multipliers such that the solution  $x^*$  obtained for this  $W$  satisfies the KKT conditions. If so, then  $x^*$  is accepted as a local solution of the problem. Otherwise, a different choice of the working set  $W$  is considered, and then, the above procedure is repeated. This approach is based on the observation that, in general, it is easier to solve equality constrained problems than nonlinear programs with inequalities. Observe that the number of choices for the working set  $W$  may be very large, up to  $2^m$ , where  $m$  is the number of inequality constraints. Since the number of the possible working sets grows exponentially with the number of inequalities, this phenomenon is called the *combinatorial difficulty of the inequality constraints optimization*. Obviously, to design a practical algorithm by considering all the possible choices for  $W$  is impossible.

A different approach for solving inequality constrained problems which avoid the combinatorial difficulty of nonlinear optimization with inequalities is given by the interior point (or barrier) methods (see Chap. 17). These methods generate iterates that stay away from the boundary of the feasible set defined by the inequality constraints. While the solution of the nonlinear optimization problem is being approached, the barrier effects are weakened, thus allowing an increasingly accurate estimation of the solution to be obtained. Thus, interior point methods avoid the combinatorial difficulty of the inequality constrained optimization.

**Remark 11.2** (*Constrained Qualification*). Observe that *not every local minimum is a KKT point*. For a local minimum  $x^*$  to be a KKT point, an additional condition must be introduced on the behavior of the constraints. Such a condition is known as the *constraint qualification*. Observe that the first-order Taylor series expansion of the functions defining the problem about  $x$  is used to form an

approximate problem in which both the objective and the constraints are linear. If, near the current point  $x$ , the linearization is fundamentally different from the feasible set, then the linear approximation of the problem does not yield useful information about the original problem. Therefore, certain assumptions about the nature of the constraints  $c_i$  that are active at  $x$  must be introduced to ensure that near  $x$ , the linearized approximation is similar to the feasible set. Given the point  $x$  and the active set  $A(x)$ , the *linear independence constraint qualification* (LICQ) holds if the *set of active constraint gradients*  $\{\nabla c_i(x), i \in A(x)\}$  is *linear independent*. In Theorem 11.8, such a constraint qualification is that  $x^*$  is a regular point, which is also known as the *linear independence constraint qualification*. The Lagrange multipliers are guaranteed to be unique in Theorem 11.8 if LICQ holds. In general, if LICQ holds, none of the active constraint gradients can be zero. Another (weaker) constraint qualification is the *Mangasarian-Fromovitz constraint qualification* (MFCQ). It requires that there exists (at least) one direction  $d \in C_{ac}(x^*)$ , i.e., such that  $\nabla c_i(x^*)^T d < 0$ , for each  $i \in A(x^*)$ . The MFCQ is weaker than LICQ, i.e., the Lagrange multipliers are guaranteed to be unique if LICQ holds, while this uniqueness property may be lost under MFCQ. Finally, observe that the constraint qualifications are *sufficient* conditions for the linear approximation of the problem to be adequate, they are not necessary conditions.  $\blacklozenge$

In the following theorem, we present a sufficient condition which guarantees that any KKT point of an inequality constrained nonlinear optimization problem is a global minimum of the problem. Of course, this result is obtained under the convexity hypothesis.

**Theorem 11.9 (KKT Sufficient Conditions).** *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be convex and continuously differentiable functions. Consider the problem  $\min\{f(x) : c(x) \leq 0\}$ . If  $(x^*, \mu^*)$  is a KKT point, then  $x^*$  is a global minimum of the problem.*

**Proof** Let us define the function  $L(x) \triangleq f(x) + \sum_{i=1}^m \mu_i^* c_i(x)$ . Since  $f$  and  $c_i$ ,  $i = 1, \dots, m$ , are convex functions and  $\mu_i^* \geq 0$ ,  $i = 1, \dots, m$ , it follows that  $L$  is also convex. Now, the dual feasibility conditions determine that  $\nabla L(x^*) = 0$ . Therefore, by Theorem 11.5,  $x^*$  is a global minimum for  $L$  on  $\mathbb{R}^n$ , i.e.,  $L(x) \geq L(x^*)$  for any  $x \in \mathbb{R}^n$ . Therefore, for any  $x$  such that  $c_i(x) \leq c_i(x^*) = 0$ ,  $i \in A(x^*)$ , it follows that

$$f(x) - f(x^*) \geq - \sum_{i \in A(x^*)} \mu_i^* [c_i(x) - c_i(x^*)] \geq 0.$$

On the other hand, the set  $\{x \in \mathbb{R}^n : c_i(x) \leq 0, i \in A(x^*)\}$  contains the feasible set  $\{x \in \mathbb{R}^n : c_i(x) \leq 0, i = 1, \dots, m\}$ . Therefore,  $x^*$  is a global minimum for the problem with inequality constraints.  $\blacklozenge$

## 11.4 Optimality Conditions for Problems with Equality Constraints

In this section, the nonlinear optimization problem with equality constraints is considered:

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & h_i(x) = 0, i = 1, \dots, p, \end{aligned} \tag{11.9}$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , are continuously differentiable functions. The functions  $h_i(x) = 0$ ,  $i = 1, \dots, p$ , of the above problem define the vector  $h(x) = [h_1(x), \dots, h_p(x)]$ . If  $x^*$

satisfies the constraints from (11.9), i.e.,  $h_i(x^*) = 0, i = 1, \dots, p$ , it is said to be *feasible*. Otherwise, it is called *infeasible*.

The optimality of  $x^*$  can be seen as a balance between the function minimization and the constraint satisfaction. A move away from  $x^*$  cannot be made without either violating a constraint or increasing the value of the objective function. Formally, this can be stated as the following proposition.

**Proposition 11.3 (Balance Between Function and Constraints).** *If  $x^*$  is a solution of (11.9) and  $x^* + \delta x$  is a nearby point, then:*

1. *If  $f(x^* + \delta x) < f(x^*)$ , then  $h_i(x^* + \delta x) \neq 0$  for some  $i$ .*
2. *If  $h_1(x^* + \delta x) = \dots = h_p(x^* + \delta x) = 0$ , then  $f(x^* + \delta x) \geq f(x^*)$ .*



In order to establish the optimality conditions for the nonlinear optimization problems with equality constraints, we need to introduce some relevant concepts. An equality constraint  $h(x) = 0$  defines in  $\mathbb{R}^n$  a set which can be viewed as a *hypersurface*. When there are  $p$  equality constraints  $h_i(x) = 0, i = 1, \dots, p$ , then their intersection defines a (possible empty) set:

$$X \triangleq \{x \in \mathbb{R}^n : h_i(x) = 0, i = 1, \dots, p\}.$$

If the functions defining the equality constraints are continuously differentiable, then the set  $X$  is said to be a *differentiable manifold*, or a *smooth manifold*.

Now, in any point on a differentiable manifold, the *tangent set* can be defined as follows. A *curve*  $\eta$  on a manifold  $X$  is a continuous application  $\eta : I \subset \mathbb{R} \rightarrow X$ , i.e., a family of points  $\eta(t) \in X$  continuously parameterized by  $t$  in the interval  $I \subset \mathbb{R}$ . Clearly, a curve passes through the point  $x^*$  if  $x^* = \eta(t^*)$  for some  $t^* \in I$ . The *derivative* of a curve at  $t^*$ , if it exists, is defined in a classical manner as

$$\dot{\eta}(t^*) \triangleq \lim_{\xi \rightarrow 0} \frac{\eta(t^* + \xi) - \eta(t^*)}{\xi}.$$

A curve is *differentiable* or *smooth* if a derivative exists for each  $t \in I$ .

**Definition 11.15 (Tangent Set).** *Let  $X$  be a differentiable manifold in  $\mathbb{R}^n$  and a point  $x^* \in X$ . Consider the collection of all the continuously differentiable curves on  $X$  passing through  $x^*$ . Then, the collection of all the vectors tangent to these curves at  $x^*$  is the tangent set to  $X$  at  $x^*$ , denoted by  $T_X(x^*)$ .*

**Definition 11.16 (Regular Point–Equality Constraints).** *Let  $h_i : \mathbb{R}^n \rightarrow \mathbb{R}, i = 1, \dots, p$ , be continuously differentiable functions on  $\mathbb{R}^n$ , and consider the set  $X \triangleq \{x \in \mathbb{R}^n : h_i(x) = 0, i = 1, \dots, p\}$ . A point  $x^* \in X$  is a regular point if the gradient vectors  $\nabla h_i(x^*), i = 1, \dots, p$ , are linearly independent, i.e.,*

$$\text{rank}[\nabla h_1(x^*), \dots, \nabla h_p(x^*)] = p. \quad (11.10)$$

If the constraints are regular in the sense of the above definition, then  $X$  is a subspace of dimension  $n - p$ . In this case,  $T_X(x^*)$  is a subspace of dimension  $n - p$ , called *tangent space*. At regular points, the tangent space can be characterized in terms of the gradients of the constraints (Luenberger, 1973).

**Proposition 11.4** (*Algebraic Characterization of a Tangent Space*). Let  $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be continuously differentiable functions on  $\mathbb{R}^n$ , and consider the set  $X \triangleq \{x \in \mathbb{R}^n : h_i(x) = 0, i = 1, \dots, p\}$ . At a regular point  $x^* \in X$ , the tangent space is such that

$$T_X(x^*) = \left\{ d : \nabla h(x^*)^T d = 0 \right\}. \quad (11.11)$$

**Proof** Let  $T_X(x^*)$  be the tangent space at  $x^*$  and  $M(x^*) = \{d : \nabla h(x^*)^T d = 0\}$ . Consider any curve  $\eta(t)$  passing through  $x^*$  at  $t = t^*$ , having derivative  $\dot{\eta}(t^*)$  such that  $\nabla h(x^*)^T \dot{\eta}(t^*) \neq 0$ . Since such a curve would not lie on  $X$ , it follows that  $T_X(x^*) \subset M(x^*)$ . Now to prove that  $T_X(x^*) \supset M(x^*)$ , we must show that if  $d \in M(x^*)$ , then there is a curve on  $X$  passing through  $x^*$  with derivative  $d$ . In order to construct such a curve, we consider the equations

$$h\left(x^* + td + \nabla h(x^*)^T u(t)\right) = 0,$$

where for fixed  $t$  the vector  $u(t) \in \mathbb{R}^p$  is unknown. Observe that we have a nonlinear system of  $p$  equations with  $p$  unknowns, continuously parameterized by  $t$ . At  $t = 0$ , there is a solution  $u(0) = 0$ . The Jacobian matrix of the above system with respect to  $u$  at  $t = 0$  is the matrix  $\nabla h(x^*) \nabla h(x^*)^T$  which is nonsingular, since  $\nabla h(x^*)$  is of full rank if  $x^*$  is a regular point. Thus, by the implicit function theorem (see Theorem A2.6), there is a continuous solution  $u(t)$  for  $-a \leq t \leq a$ . The curve  $\eta(t) = x^* + td + \nabla h(x^*)^T u(t)$  by construction is a curve on  $X$ . By differentiating the above nonlinear system with respect to  $t$  at  $t = 0$ , we get

$$0 = \frac{d}{dt} h(\eta(t)) \Big|_{t=0} = \nabla h(x^*)^T d + \nabla h(x^*) \nabla h(x^*)^T \dot{u}(0).$$

By definition of  $d$ , we have  $\nabla h(x^*)^T d = 0$ . Therefore, since  $\nabla h(x^*) \nabla h(x^*)^T$  is nonsingular, it follows that  $\dot{u}(0) = 0$ . Therefore,  $\dot{\eta}(0) = d + \nabla h(x^*)^T \dot{u}(0) = d$  and the constructed curve has derivative  $d$  at  $x^*$ .  $\blacklozenge$

### The Method of Lagrange Multipliers

Let us now present the optimality conditions for the nonlinear optimization problems with equality constraints by using the method of Lagrange multipliers. The idea is to restrict the search of a minimum of (11.9) to the manifold  $X \triangleq \{x \in \mathbb{R}^n : h_i(x) = 0, i = 1, \dots, p\}$ . The following theorem gives the geometric necessary condition for a local minimum of a nonlinear optimization problem with equality constraints. It is shown that the tangent space  $T_X(x^*)$  at a regular local minimum point  $x^*$  is *orthogonal* to the gradient of the objective function at  $x^*$ .

**Theorem 11.10** (*Geometric Necessary Condition for a Local Minimum*). Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be continuously differentiable functions. Suppose the  $x^*$  is a local minimum point of the problem  $\min\{f(x) : h(x) = 0\}$ . Then,  $\nabla f(x^*)$  is orthogonal to the tangent space  $T_X(x^*)$ , i.e.,

$$C_0(x^*) \cap T_X(x^*) = \emptyset.$$

**Proof** Assume that there exists a  $d \in T_X(x^*)$  such that  $\nabla f(x^*)^T d \neq 0$ . Let  $\eta : I = [-a, a] \rightarrow X$ ,  $a > 0$ , be any smooth curve passing through  $x^*$ , with  $\eta(0) = x^*$  and  $\dot{\eta}(0) = d$ . Also let  $\varphi$  be the function defined as  $\varphi(t) \triangleq f(\eta(t))$  for any  $t \in I$ . Since  $x^*$  is a local minimum of  $f$  on  $X \triangleq \{x \in \mathbb{R}^n : h(x) = 0\}$ , by

Definition 11.7, it follows that there exists  $\delta > 0$  such that  $\varphi(t) = f(\eta(t)) \geq f(x^*) = \varphi(0)$  for any  $t \in B(0, \delta) \cap I$ . Therefore,  $t^* = 0$  is an unconstrained local minimum point for  $\varphi$ , and

$$0 = \nabla \varphi(0) = \nabla f(x^*)^T \dot{\eta}(0) = \nabla f(x^*)^T d.$$

However, this is in contradiction with the assumption that  $\nabla f(x^*)^T d \neq 0$ .  $\blacklozenge$

The conclusion of this theorem is that if  $x^*$  is a regular point of the constraints  $h(x) = 0$  and a local minimum point of  $f$  subject to these constraints, then all  $d \in \mathbb{R}^n$  satisfying  $\nabla h(x^*)^T d = 0$  must also satisfy  $\nabla f(x^*)^T d = 0$ .

The following theorem shows that this property that  $\nabla f(x^*)$  is orthogonal to the tangent space implies that  $\nabla f(x^*)$  is a linear combination of the gradients of  $h_i(x^*)$ ,  $i = 1, \dots, p$ , at  $x^*$ . This relation leads to the introduction of Lagrange multipliers and the Lagrange function.

**Theorem 11.11** (*First-Order Necessary Optimality Conditions*). *Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be continuously differentiable functions. Consider the problem  $\min\{f(x) : h(x) = 0\}$ . If  $x^*$  is a local minimum and it is a regular point of the constraints, then there exists a unique vector  $\lambda^* \in \mathbb{R}^p$  such that*

$$\nabla f(x^*) + \nabla h(x^*)^T \lambda^* = 0. \quad (11.12)$$

**Proof** Since  $x^*$  is a local minimum of  $f$  on  $X = \{x \in \mathbb{R}^n : h(x) = 0\}$ , by Theorem 11.10, it follows that  $C_0(x^*) \cap T_X(x^*) = \emptyset$ , i.e., the system

$$\nabla f(x^*)^T d < 0, \quad \nabla h(x^*)^T d = 0,$$

is inconsistent. Now, consider the following two sets:

$$\begin{aligned} C_1 &\triangleq \left\{ (z_1, z_2) \in \mathbb{R}^{p+1} : z_1 = \nabla f(x^*)^T d, \quad z_2 = \nabla h(x^*)^T d \right\}, \\ C_2 &\triangleq \left\{ (z_1, z_2) \in \mathbb{R}^{p+1} : z_1 < 0, \quad z_2 = 0 \right\}. \end{aligned}$$

Observe that  $C_1$  and  $C_2$  are convex sets and  $C_1 \cap C_2 = \emptyset$ . Therefore, by the separation of two convex sets, given by Proposition A4.2, there exists a nonzero vector  $(\mu, \lambda) \in \mathbb{R}^{p+1}$  ( $\mu \in \mathbb{R}$ ,  $\lambda \in \mathbb{R}^p$ ) such that for any  $d \in \mathbb{R}^n$  and for any  $(z_1, z_2) \in C_2$

$$\mu \nabla f(x^*)^T d + \lambda^T [\nabla h(x^*)^T d] \geq \mu z_1 + \lambda^T z_2.$$

Now, considering  $z_2 = 0$  and having in view that  $z_1$  can be made an arbitrary large negative number, it follows that  $\mu \geq 0$ . Additionally, considering  $(z_1, z_2) = (0, 0)$ , we must have  $[\mu \nabla f(x^*) + \lambda^T \nabla h(x^*)]^T d \geq 0$ , for any  $d \in \mathbb{R}^n$ . In particular, letting  $d = -[\mu \nabla f(x^*) + \lambda^T \nabla h(x^*)]$ , it follows that  $-\|\mu \nabla f(x^*) + \lambda^T \nabla h(x^*)\|^2 \geq 0$ , and thus,

$$\mu \nabla f(x^*) + \lambda^T \nabla h(x^*) = 0, \text{ with } (\mu, \lambda) \neq (0, 0).$$

Observe that  $\mu > 0$ , for otherwise the above relation would contradict the assumption that  $\nabla h_i(x^*)$ ,  $i = 1, \dots, p$ , are linear independent. The conclusion of the theorem follows letting  $\lambda^* = \lambda/\mu$  and noting that the linear independence assumption implies the uniqueness of the  $\lambda^*$ .  $\blacklozenge$

**Remark 11.3** The first-order necessary optimality conditions given by the Theorem 11.11 together with the constraints of the problem (11.9):

$$\nabla f(x^*) + \nabla h(x^*)^T \lambda^* = 0, \quad (11.13a)$$

$$h(x^*) = 0, \quad (11.13b)$$

represent a total of  $n + p$  nonlinear equations in the variables  $(x^*, \lambda^*)$ . These conditions determine, at least locally, a unique solution  $(x^*, \lambda^*)$ . However, as in the unconstrained case, a solution to the first-order necessary optimality conditions does not have to be a local minimum of the problem (11.9).  $\blacklozenge$

**Definition 11.17** (*Lagrange multipliers*). The scalars  $\lambda_1^*, \dots, \lambda_p^*$  in (11.12) are called the Lagrange multipliers.

**Definition 11.18** (*Constraint normals*). The vectors  $\nabla h_1(x), \dots, \nabla h_p(x)$  are called the constraint normals.

The condition (11.12) shows that  $\nabla f(x^*)$  is linearly dependent on the constraint normals. Therefore, a constrained minimum occurs when the gradients of the objective function and the gradients of the constraints interact in such a way that any reduction in  $f$  can only be obtained by violating the constraints (see Proposition 11.3).

**Definition 11.19** (*Lagrange function–Lagrangian*). The function  $L : \mathbb{R}^n \times \mathbb{R}^p \rightarrow \mathbb{R}$  associated to the nonlinear optimization problem (11.9) is defined as

$$L(x, \lambda) \triangleq f(x) + \lambda^T h(x). \quad (11.14)$$

**Remark 11.4** (*Regularity Assumption*). For a local minimum to satisfy the above first-order necessary conditions (11.13) and, in particular, for unique Lagrange multipliers to exist, it is necessary that the equality constraints  $h_i(x) = 0$ ,  $i = 1, \dots, p$ , satisfy a regularity condition. As already seen, for a local minimum of an inequality constrained nonlinear optimization problem to be a KKT point, a constrained qualification is needed. For the equality constrained nonlinear optimization problems, the condition that *the minimum point is a regular point* corresponds to *linear independence constrained qualification*.  $\blacklozenge$

If  $x^*$  is a regular local minimum of (11.9), then the first-order necessary optimality conditions (11.13) can be rewritten as

$$\nabla_x L(x^*, \lambda^*) = 0, \quad (11.15a)$$

$$\nabla_\lambda L(x^*, \lambda^*) = 0. \quad (11.15b)$$

Observe that the second condition (11.15b) is a restatement of the constraints. The solution of the optimization problem (11.9) corresponds to a saddle point of the Lagrangian.

**Theorem 11.12** (Second-Order Necessary Optimality Conditions). Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be continuously differentiable functions. Consider the problem  $\min\{f(x) : h(x) = 0\}$ . If  $x^*$  is a local minimum and it is a regular point of the constraints, then there exists a unique vector  $\lambda^* \in \mathbb{R}^p$  such that:

$$\nabla f(x^*) + \nabla h(x^*)^\top \lambda^* = 0, \quad (11.16)$$

and

$$d^\top \left( \nabla^2 f(x^*) + \nabla^2 h(x^*)^\top \lambda^* \right) d \geq 0 \quad (11.17)$$

for any  $d \in \mathbb{R}^n$  such that  $\nabla h(x^*)^\top d = 0$ .

**Proof** The first condition  $\nabla f(x^*) + \nabla h(x^*)^\top \lambda^* = 0$  follows from the Theorem 11.11. Now we concentrate to the second condition. Let  $x^*$  be a regular point and consider  $d$  an arbitrary direction from  $T_{X(x^*)}$ , i.e.,  $\nabla h(x^*)^\top d = 0$ . Let  $\eta: I = [-a, a] \rightarrow X$ ,  $a > 0$ , be an arbitrary twice-differentiable curve passing through  $x^*$  with  $\eta(0) = x^*$  and  $\dot{\eta}(0) = d$ . Consider  $\varphi$  a function defined as  $\varphi(t) \triangleq f(\eta(t))$ , for any  $t \in I$ . Since  $x^*$  is a local minimum of  $f$  on  $X \triangleq \{x \in \mathbb{R}^n : h(x) = 0\}$ , it follows that  $t^* = 0$  is an unconstrained local minimum point for  $\varphi$ . Therefore, by Theorem 11.4, it follows that

$$\nabla^2 \varphi(0) = \dot{\eta}(0)^\top \nabla^2 f(x^*) \dot{\eta}(0) + \nabla f(x^*)^\top \ddot{\eta}(0) \geq 0.$$

On the other hand, differentiating the relation  $h(\eta(t))^\top \lambda = 0$  twice, we get

$$\dot{\eta}(0)^\top \left( \nabla^2 h(x^*)^\top \lambda \right) \dot{\eta}(0) + \left( \nabla h(x^*)^\top \lambda \right)^\top \ddot{\eta}(0) = 0.$$

Now, adding the last two relations we obtain

$$d^\top \left( \nabla^2 f(x^*) + \nabla^2 h(x^*)^\top \lambda^* \right) d \geq 0,$$

which must hold for every  $d$  such that  $\nabla h(x^*)^\top d = 0$ . ◆

The above theorem says that if  $T_{X(x^*)}$  is the tangent space to  $X$  at  $x^*$ , then the matrix  $\nabla_{xx}^2 L(x^*, \lambda^*) = \nabla^2 f(x^*) + \nabla^2 h(x^*)^\top \lambda^*$  is positive semidefinite on  $T_{X(x^*)}$ .

**Remark 11.5** (Feasible Directions and Second-Order Conditions) An  $n$ -vector  $d$  is said to be a *feasible direction* at  $x^*$  if  $\nabla h(x^*)^\top d = 0$ , where  $\nabla h(x^*)$  is the Jacobian of the constraints at  $x^*$ . Let us assume that  $d$  is a feasible direction normalized so that  $\|d\| = 1$ . Considering the Taylor's expansion:

$$h(x^* + \varepsilon d) = h(x^*) + \varepsilon \nabla h(x^*)^\top d + O(\|\varepsilon d\|^2),$$

then  $h(x^* + \varepsilon d) = O(\varepsilon^2)$ . Therefore, a move away from  $x^*$  along  $d$  keeps the constraints satisfied to first-order accuracy. In particular, if all the constraints in (11.9) are linear, then  $x^* + \varepsilon d$  is a feasible point for all  $\varepsilon > 0$ . On the other hand, if any of the  $h_i(x)$  in (11.9) are nonlinear, then  $d$  is a direction *tangential* to the constraints at  $x^*$ . It is easy to see that the condition (11.12) implies that, for any feasible direction  $d$ ,

$$d^T \nabla f(x^*) = 0.$$

To distinguish a *minimum* from a *maximum* or a *saddle-point*, the second-order optimality condition must be used. These conditions can be stated as follows:

1. If the constraint functions  $h_i$  are all linear, the second-order condition that guarantees  $x^*$  is a minimum of problem (11.9) is

$$d^T \nabla^2 f(x^*) d > 0$$

for any feasible direction  $d$ .

2. If the constraint functions  $h_i$  are nonlinear, the second-order condition that guarantees  $x^*$  is a minimum of problem (11.9) is

$$d^T \nabla^2 L(x^*, \lambda^*) d > 0$$

for any feasible direction  $d$ . ◆

**Remark 11.6** (*Eigenvalues in Tangent Space*). Geometrically, the restriction of the matrix  $\nabla_{xx}^2 L(x^*, \lambda^*)$  to  $T_X(x^*)$  corresponds to the projection  $P_{T_X(x^*)} [\nabla_{xx}^2 L(x^*, \lambda^*)]$ . A vector  $y \in T_X(x^*)$  is an *eigenvector* of the projection  $P_{T_X(x^*)} [\nabla_{xx}^2 L(x^*, \lambda^*)]$  if there is a real number  $\nu$  such that

$$P_{T_X(x^*)} [\nabla_{xx}^2 L(x^*, \lambda^*)] y = \nu y.$$

The real number  $\nu$  is called the *eigenvalue* of  $P_{T_X(x^*)} [\nabla_{xx}^2 L(x^*, \lambda^*)]$ . To obtain a matrix representation for  $P_{T_X(x^*)} [\nabla_{xx}^2 L(x^*, \lambda^*)]$ , it is necessary to introduce a basis of the tangent subspace  $T_X(x^*)$ . It is best to introduce an orthonormal basis, say  $E = [e_1, \dots, e_{n-p}]$ . Any vector  $y \in T_X(x^*)$  can be written as  $y = Ez$ , where  $z \in \mathbb{R}^{n-p}$ . Now,  $\nabla_{xx}^2 L(x^*, \lambda^*) Ez$  represents the action of  $\nabla_{xx}^2 L(x^*, \lambda^*)$  on such a vector. To project this result back into  $T_X(x^*)$  and to express the result in terms of the basis  $E = [e_1, \dots, e_{n-p}]$ , it is necessary to multiply by  $E^T$ . Therefore,  $E^T \nabla_{xx}^2 L(x^*, \lambda^*) Ez$  is the vector whose components give the representation in terms of the basis  $E$ . The  $(n-p) \times (n-p)$  matrix  $E^T \nabla_{xx}^2 L(x^*, \lambda^*) E$  is the matrix representation of  $\nabla_{xx}^2 L(x^*, \lambda^*)$  restricted to  $T_X(x^*)$ . The eigenvalues of  $\nabla_{xx}^2 L(x^*, \lambda^*)$  restricted to  $T_X(x^*)$  can be determined by computing the eigenvalues of  $E^T \nabla_{xx}^2 L(x^*, \lambda^*) E$ . These eigenvalues are independent of the particular choice of the basis  $E$ . ◆

Recall that the conditions given in Theorems 11.11 and 11.12 are necessary conditions. These must hold at each local minimum point. However, a point satisfying these conditions may not be a local minimum. As in the unconstrained case, it is possible to derive second-order conditions for constrained optimization problems. The following theorem provides sufficient conditions for a stationary point of the Lagrange function to be a local minimum.

**Theorem 11.13** (*Second-Order Sufficient Conditions*). Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be twice continuously differentiable functions. Consider the problem  $\min\{f(x) : h(x) = 0\}$ . If  $x^*$  and  $\lambda^*$  satisfy

$$\nabla_x L(x^*, \lambda^*) = 0, \quad (11.18a)$$

$$\nabla_\lambda L(x^*, \lambda^*) = 0, \quad (11.18b)$$

and

$$y^T \nabla_{xx}^2 L(x^*, \lambda^*) y > 0 \quad (11.19)$$

for any  $y \neq 0$  such that  $\nabla h(x^*)^T y = 0$ , then  $x^*$  is a strict local minimum.

**Proof** Consider the augmented Lagrange function:

$$\bar{L}(x, \lambda) = f(x) + \lambda^T h(x) + \frac{c}{2} \|h(x)\|^2,$$

where  $c$  is a scalar. Clearly,

$$\nabla_x \bar{L}(x, \lambda) = \nabla_x L(x, \bar{\lambda}),$$

$$\nabla_{xx}^2 \bar{L}(x, \lambda) = \nabla_{xx}^2 L(x, \bar{\lambda}) + c \nabla h(x)^T \nabla h(x),$$

where  $\bar{\lambda} = \lambda + ch(x)$ . Since  $(x^*, \lambda^*)$  satisfy the sufficient conditions, by the Theorem A4.4, we obtain that  $\nabla_x \bar{L}(x^*, \lambda^*) = 0$  and  $\nabla_{xx}^2 \bar{L}(x^*, \lambda^*) > 0$ , for sufficiently large  $c$ .  $\bar{L}$  being positive definite at  $(x^*, \lambda^*)$ , it follows that there exist  $\rho > 0$  and  $\delta > 0$  such that

$$\bar{L}(x, \lambda^*) \geq \bar{L}(x^*, \lambda^*) + \frac{\rho}{2} \|x - x^*\|^2$$

for  $\|x - x^*\| < \delta$ . Besides, since  $\bar{L}(x, \lambda^*) = f(x)$  when  $h(x) = 0$ , we get

$$f(x) \geq f(x^*) + \frac{\rho}{2} \|x - x^*\|^2$$

if  $h(x) = 0$ ,  $\|x - x^*\| < \delta$ , i.e.,  $x^*$  is a strict local minimum.  $\blacklozenge$

### Sensitivity: Interpretation of the Lagrange Multipliers

The  $i$ -th Lagrange multiplier can be viewed as the measuring of the sensitivity of the objective function with respect to the  $i$ -th constraint, i.e., how much the optimal value of the objective function would change if that constraint was perturbed.

At the very beginning, let us consider  $p = 1$ , i.e., the problem (11.9) has one constraint  $h_1(x) = 0$ . Now, suppose that  $x^*$  is a local solution of the problem

$$\min \{f(x) : h_1(x) = 0\},$$

and consider the *perturbed* problem

$$\min \{f(x) : h_1(x) = \delta\},$$

where  $\delta$  is a known scalar. If the solution of the perturbed problem is  $x^* + u$ , then by using the Taylor's expansion, a first-order estimate of the optimum function value is  $f(x^* + u) \approx f(x^*) + u^T \nabla f(x^*)$ . However, the optimality condition for the original problem given by (11.12) states that  $\nabla f(x^*) = -\lambda_1^* \nabla h_1(x^*)$ , where  $\lambda_1^*$  is the Lagrange multiplier. Hence,

$$f(x^* + u) \approx f(x^*) - \lambda_1^* u^T \nabla h_1(x^*).$$

Since  $x^* + u$  solves the perturbed problem, it follows that  $h_1(x^* + u) = \delta$ , hence  $h_1(x^*) + u^T \nabla h_1(x^*) \approx \delta$ . However,  $h_1(x^*) = 0$ . Therefore,  $u^T \nabla h_1(x^*) \approx \delta$ , that is,

$$f(x^* + u) - f(x^*) \approx -\delta \lambda_1^*.$$

In other words, the Lagrange multiplier is an approximate measure of the change in the objective function that will occur if a unit amount is added to the right-hand side of the constraint. In general, we have the following theorem.

**Theorem 11.14** (*Interpretation of the Lagrange Multipliers*). Consider the family of problems  $\min \{f(x) : h(x) = w\}$ , where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h: \mathbb{R}^n \rightarrow \mathbb{R}^p$  are twice continuously differentiable. Suppose for  $w = 0$  there is a local solution  $x^*$  that is a regular point and that, together with its associated Lagrange multiplier vector  $\lambda$ , satisfies the second-order sufficient conditions for a strict local minimum. Then, for every  $w \in \mathbb{R}^p$  in a region containing 0, there is a  $x(w)$ , depending continuously on  $w$ , such that  $x(0) = x^*$  and such that  $x(w)$  is a local minimum of the problem. Furthermore,

$$\nabla_w f(x(w))|_{w=0} = -\lambda.$$

**Proof** Consider the system of equations:

$$\nabla f(x) + \nabla h(x)^T \lambda = 0,$$

$$h(x) = w.$$

By hypothesis, when  $w = 0$ , there is a solution  $x^*, \lambda^*$  to this system. The Jacobian matrix of this system, at this solution, is

$$\begin{bmatrix} L(x^*) & \nabla h(x^*)^T \\ \nabla h(x^*) & 0 \end{bmatrix},$$

where  $L(x^*) = \nabla^2 f(x^*) + \nabla^2 h(x^*)^T \lambda^*$ .

Since  $x^*$  is a regular point and  $L(x^*)$  is positive definite on  $\{y : \nabla h(x^*)^T y = 0\}$ , it follows that this matrix is nonsingular. Thus, by the implicit function theorem (see Theorem A2.6), there is a solution  $x(w), \lambda(w)$  to the system which is twice continuously differentiable. Therefore,

$$\nabla_w f(x(w))|_{w=0} = \nabla f(x^*)^T \nabla_w x(0),$$

$$\nabla_w h(x(w))|_{w=0} = \nabla h(x^*) \nabla_w x(0).$$

However, since  $h(x^*) = w$ , it follows that  $\nabla h(x^*) \nabla_w x(0) = I$ . On the other hand, from  $\nabla f(x) + \nabla h(x)^T \lambda = 0$ , it follows that  $\nabla_w f(x(w))|_{w=0} = -\lambda$ . ◆

## 11.5 Optimality Conditions for General Nonlinear Optimization Problems

We now present a generalization of Theorems 11.8, 11.11, 11.12, and 11.13 to nonlinear optimization problems with equality and inequality constraints:

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_i(x) \leq 0, i = 1, \dots, m, \\ & h_j(x) = 0, j = 1, \dots, p, \end{aligned} \tag{11.20}$$

where  $x \in \mathbb{R}^n$ ,  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , and  $h_j: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $j = 1, \dots, p$ , are continuously differentiable functions. Define the vectors  $c(x) = [c_1(x), \dots, c_m(x)]$  and  $h(x) = [h_1(x), \dots, h_p(x)]$ .

**Remark 11.7** (*Discarding the Inactive Constraints*). Let us consider the nonlinear optimization problem  $\min\{f(x) : c_i(x) \leq 0, i = 1, \dots, m\}$ . Suppose that  $x^*$  is a local minimum point for this problem. Clearly,  $x^*$  is also a local minimum of the above problem, where the inactive constraints  $c_i(x) \leq 0, i \notin A(x^*)$  have been *discarded*. Therefore, *the inactive constraints at  $x^*$  can be ignored in the statement of the optimality conditions*. On the other hand, the active constraints can be treated as equality constraints at a local minimum point. Hence,  $x^*$  is also a local minimum point to the equality constrained problem:

$$\min \{f(x) : c_i(x) = 0, i \in A(x^*)\}$$

The difficulty is that we do not know the set of the active constraints at  $x^*$ .

From Theorem 11.11, it follows that if  $x^*$  is a regular point, there exists a unique Lagrange multiplier vector  $\mu^* \in \mathbb{R}^m$  such that

$$\nabla f(x^*) + \sum_{i \in A(x^*)} \mu_i^* \nabla c_i(x^*) = 0.$$

Now, assigning zero Lagrange multipliers to the inactive constraints, we get

$$\begin{aligned} & \nabla f(x^*) + \nabla c(x^*)^T \mu^* = 0, \\ & \mu_i = 0, i \notin A(x^*). \end{aligned}$$

Clearly, the last condition can be rewritten as  $\mu_i^* c_i(x^*) = 0, i = 1, \dots, m$ .

It remains to show that  $\mu > 0$ . For this, assume that  $\mu_q < 0$  for some  $q \in A(x^*)$ . Now, let  $A \in \mathbb{R}^{(m+1) \times n}$  be the matrix whose rows are  $\nabla f(x^*)$  and  $\nabla c_i(x^*)$ ,  $i = 1, \dots, m$ . Since  $x^*$  is a regular point, it follows that the Lagrange multiplier vector  $\mu^*$  is unique. Therefore, the condition  $A^T y = 0$  can only be satisfied by  $y^* = \gamma(1 \ \mu^*)^T$  with  $\gamma \in \mathbb{R}$ . However,  $\mu_q < 0$ . Therefore, by Gordan's Theorem A4.3, there exists a direction  $\bar{d} \in \mathbb{R}^n$  such that  $A\bar{d} < 0$ . In other words,  $\bar{d} \in C_0(x^*) \cap C_{ac}(x^*) \neq \emptyset$ , which contradicts the hypothesis that  $x^*$  is a local minimum of the problem. All these results represent the KKT optimality conditions as stated by Theorem 11.8. Although this development is straightforward, it is somewhat limited by the regularity-type assumption at the optimal solution.  $\blacklozenge$

**Definition 11.20** (*Regular Point-General Case*). Let  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , and  $h_j: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $j = 1, \dots, p$ , be continuously differentiable functions. Consider the set  $X = \{x \in \mathbb{R}^n : c_i(x) \leq 0, i = 1, \dots, m,$

$h_j(x) = 0, j = 1, \dots, p\}$ . A point  $x^* \in X$  is a regular point of the constraints from (11.20) if the gradients  $\nabla c_i(x^*)$ ,  $i \in A(x^*)$ , and  $\nabla h_j(x^*)$ ,  $j = 1, \dots, p$ , are linearly independent.  $\blacklozenge$

Definition 11.20 introduces the linear independence constraint qualification (LICQ) for general nonlinear optimization problems, i.e., the gradients of the active inequality constraints and the gradients of the equality constraints are all linearly independent at  $x^*$ . Another constraint qualification is the linear constraint qualification (LCQ), i.e.,  $c_i(x)$ ,  $i = 1, \dots, m$ , and  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $j = 1, \dots, p$ , are affine functions. Another one is the Slater condition for a convex problem, i.e., there exists a point  $\bar{x}$  such that  $c_i(\bar{x}) < 0$ ,  $i = 1, \dots, m$ , and  $h(\bar{x}) = 0$ .

We emphasize that the constraint qualification ensures that the linearized approximation to the feasible set  $X$  captures the essential shape of  $X$  in a neighborhood of  $x^*$ .

**Theorem 11.15** (First- and Second-Order Necessary Conditions). Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , and  $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be twice continuously differentiable functions. Consider the problem  $\min\{f(x) : c(x) \leq 0, h(x) = 0\}$ . If  $x^*$  is a local minimum for this problem and it is a regular point of the constraints, then there exist unique vectors  $\mu^* \in \mathbb{R}^m$  and  $\lambda^* \in \mathbb{R}^p$  such that

$$\nabla f(x^*) + \nabla c(x^*)^T \mu^* + \nabla h(x^*)^T \lambda^* = 0, \quad (11.21a)$$

$$\mu^* \geq 0, \quad (11.21b)$$

$$c(x^*) \leq 0, \quad (11.21c)$$

$$h(x^*) = 0, \quad (11.21d)$$

$$(\mu^*)^T c(x^*) = 0, \quad (11.21e)$$

and

$$y^T \left( \nabla^2 f(x^*) + \nabla^2 c(x^*)^T \mu^* + \nabla^2 h(x^*)^T \lambda^* \right) y \geq 0, \quad (11.22)$$

for all  $y \in \mathbb{R}^n$  such that  $\nabla c_i(x^*)^T y = 0$ ,  $i \in A(x^*)$  and  $\nabla h(x^*)^T y = 0$ .  $\blacklozenge$

**Proof** Observe that since  $\mu^* \geq 0$  and  $c(x^*) \leq 0$ , (11.21e) is equivalent to the statement that a component of  $\mu^*$  is nonzero only if the corresponding constraint is active. Since  $x^*$  is a minimum point over the constraint set, it is also a minimum over the subset of that set defined by setting the active constraints to zero. Therefore, for the resulting equality constrained problem defined in a neighborhood of  $x^*$ , there are Lagrange multipliers. Hence, (11.21a) holds with  $\mu_i^* = 0$  if  $c_i(x^*) \neq 0$ .

It remains to show that  $\mu^* \geq 0$ . This is a little more elaborate. Suppose that for some  $k \in A(x^*)$ ,  $\mu_k^* < 0$ . Let  $\bar{X}$  and  $\bar{T}$  be the surface and the tangent space, respectively, defined by all the other active constraints at  $x^*$ . By the regularity assumptions, there is a  $d$  such that  $d \in \bar{T}$  and  $\nabla c_k(x^*)^T d < 0$ . Let  $\eta(t)$  be a curve on  $\bar{X}$  passing through  $x^*$  at  $t = 0$  with  $\dot{\eta}(0) = d$ . Then, for small  $t \geq 0$ , it follows that  $\eta(t)$  is feasible and

$$\frac{df}{dt}(\eta(t)) \Big|_{t=0} = \nabla f(x^*)^T d < 0$$

by (11.21a), which contradicts the fact that  $x^*$  is a minimum point.  $\blacklozenge$

The conditions (11.21) are known as the *Karush-Kuhn-Tucker conditions*, or *KKT conditions*. The conditions (11.21e) written as  $\mu_i^* c_i(x^*) = 0$ ,  $i = 1, \dots, m$ , are the *complementary conditions*. They show that either constraint  $i$  is active or the corresponding Lagrange multiplier  $\mu_i^* = 0$ , or possibly both. For a given nonlinear optimization problem (11.20) and a solution point  $x^*$ , there may be many Lagrange multipliers  $(\mu^*, \lambda^*)$  for which the conditions (11.21) and (11.22) are satisfied. However, when  $x^*$  is a regular point (the LICQ is satisfied), the optimal  $(\mu^*, \lambda^*)$  is unique.

The KKT conditions motivate the following definition which *classifies* constraints according to whether or not their corresponding Lagrange multiplier is zero.

**Definition 11.21** (*Strongly Active (Binding)–Weakly Active Constraints*). Let  $x^*$  be a local solution to the problem (11.20) and the Lagrange multipliers  $(\mu^*, \lambda^*)$  which satisfy the KKT conditions (11.21). We say that an inequality constraint  $c_i(x)$  is *strongly active* or *binding* if  $i \in A(x^*)$  and the corresponding Lagrange multiplier  $\mu_i^* > 0$ . We say that  $c_i(x)$  is *weakly active* if  $i \in A(x^*)$  and the corresponding Lagrange multiplier  $\mu_i^* = 0$ .  $\blacklozenge$

**Definition 11.22** (*Strongly Active Simple Bound Constraints–Binding Simple Bound Constraints*). Let  $x^*$  be a local solution to the problem  $\min\{f(x) : x \in \mathbb{R}^n, l_i \leq x_i \leq u_i, i = 1, \dots, n\}$ . Let  $g = \nabla f(x^*)$  be the gradient of the minimizing function in  $x^*$ . Then, the inequality constraint  $l_i \leq x_i \leq u_i$  is a *strongly active simple bound constraint* in  $x^*$  if  $x_i^* = l_i$  and  $g_i \geq 0$ , or  $x_i^* = u_i$  and  $g_i \leq 0$ , where  $g_i$  is the  $i$ -th component of  $g$ .  $\blacklozenge$

The non-negativity condition (11.21b) on the Lagrange multiplier for the inequality constraints ensures that the function  $f(x)$  will not be reduced by a move off any of the binding constraints at  $x^*$  to the interior of the feasible region.

A special case of complementarity is important because it introduces the concept of *degeneracy* in optimization.

**Definition 11.23** (*Strict Complementarity*). Let  $x^*$  be a local solution to the problem (11.20) and the Lagrange multipliers  $(\mu^*, \lambda^*)$  which satisfy the KKT conditions (11.21). We say that the *strict complementarity* holds if exactly one of  $\mu_i^*$  and  $c_i(x^*)$  is zero for each index  $i = 1, \dots, m$ . In other words, we have  $\mu_i^* > 0$  for each  $i \in A(x^*)$ .  $\blacklozenge$

Usually, satisfaction of strict complementarity is beneficial for algorithms and makes it easier to determine the active set  $A(x^*)$  so that convergence is more rapid.

**Remark 11.8** (*Degeneracy*). A property that causes difficulties for some optimization algorithms is degeneracy. This concept refers to the following two situations:

- The gradients of the active constraints  $\nabla c_i(x^*)$ ,  $i \in A(x^*)$ , are linearly dependent at the solution point  $x^*$ . Linear dependence of the gradients of the active constraints can cause difficulties during the computation of the step direction because certain matrices that must be factorized become rank deficient.
- Strict complementarity fails to hold, that is, there is some index  $i \in A(x^*)$  such that all the Lagrange multipliers satisfying the KKT conditions (11.21) have  $\mu_i^* = 0$ . In the case when the problem contains weakly active constraints, it is difficult for an algorithm to determine whether these constraints are active at the solution. For some optimization algorithms (active-set algorithms and gradient projection algorithms), the presence of weakly active constraints can cause the algorithm to zigzag as the iterates move on and off the weakly constraints along the successive iterations.  $\blacklozenge$

**Theorem 11.16 (Second-Order Sufficient Conditions).** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , and  $h_j: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $j = 1, \dots, p$ , be twice continuously differentiable functions. Consider the problem  $\min\{f(x) : c(x) \leq 0, h(x) = 0\}$ . If there exist  $x^*$ ,  $\mu^*$ , and  $\lambda^*$  satisfying the KKT conditions (11.21a), (11.21b), (11.21c), (11.21d), and (11.21e) and

$$y^T \nabla_{xx}^2 L(x^*, \mu^*, \lambda^*) y > 0,$$

for all  $y \neq 0$  such that

$$\nabla c_i(x^*)^T y = 0, i \in A(x^*) \text{ with } \mu_i^* > 0, \quad (11.23a)$$

$$\nabla c_i(x^*)^T y \leq 0, i \in A(x^*) \text{ with } \mu_i^* = 0, \quad (11.23b)$$

$$\nabla h(x^*)^T y = 0, \quad (11.23c)$$

where  $L(x, \mu, \lambda) = f(x) + \mu^T c(x) + \lambda^T h(x)$ , then  $x^*$  is a strict local minimum of the problem.

**Proof** The theorem says that the Hessian of the Lagrangian is positive definite on the critical cone  $C(x^*, \mu^*, \lambda^*)$  defined by (11.23) for  $x^*$ ,  $\mu^*$  and  $\lambda^*$  satisfying the KKT conditions (11.21a), (11.21b), (11.21c), (11.21d), and (11.21e).

Assume that  $x^*$  is not a strict local minimum, and let  $\{y_k\}$  be a sequence of feasible points converging to  $x^*$  such that  $f(y_k) \leq f(x^*)$ . Consider  $y_k$  of the form  $y_k = x^* + \delta_k s_k$  with  $\delta_k > 0$  and  $\|s_k\| = 1$ . Assume that  $\delta_k \rightarrow 0$  and  $s_k \rightarrow s^*$ . Clearly,  $\nabla f(x^*)^T s^* \leq 0$  and  $\nabla h_j(x^*)^T s^* = 0$  for  $j = 1, \dots, p$ .

On the other hand, for each active constraint  $c_i$ , we have  $c_i(y_k) - c_i(x^*) \leq 0$ . Therefore,  $\nabla c_i(x^*)^T s^* \leq 0$ .

If  $\nabla c_i(x^*)^T s^* = 0$ , for all  $i \in \{l : c_l(x^*) = 0, \mu_l^* > 0\}$ , then the proof is similar to that in Theorem 11.13. If  $\nabla c_i(x^*)^T s^* < 0$  for at least one  $i \in \{l : c_l(x^*) = 0, \mu_l^* > 0\}$ , then

$$0 \geq \nabla f(x^*)^T s^* = -\lambda^T \nabla h(x^*)^T s^* - \mu^T \nabla c(x^*)^T s^* > 0,$$

which represents a contradiction.  $\blacklozenge$

The KKT sufficient conditions for convex programming with inequality constraints given in Theorem 11.9 can immediately be generalized to nonlinear optimization problems with convex inequalities and affine equalities.

**Theorem 11.17 (KKT Sufficient Conditions for General Problems).** Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be convex and continuously differentiable functions. Also, let  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, p$ , be affine functions. Consider the problem  $\min f(x)$  subject to  $x \in X \triangleq \{x \in \mathbb{R}^n : c(x) \leq 0, h(x) = 0\}$ . If  $(x^*, \mu^*, \lambda^*)$  satisfies the KKT conditions (11.21a)–(11.21e), then  $x^*$  is a global minimum for  $f$  on  $X$ .  $\blacklozenge$

### Sensitivity: Interpretation of the Lagrange Multipliers for General Problems

As we have already seen, Theorem 11.14 presents an interpretation of the Lagrange multipliers for nonlinear optimization problems with equality constraints. Each Lagrange multiplier tells us something about the *sensitivity* of the optimal objective function value  $f(x^*)$  with respect to the corresponding constraint. Clearly, for an inactive constraint  $i \notin A(x^*)$ , the solution  $x^*$  and the function value  $f(x^*)$  are independent of whether this constraint is present or not. If we slightly perturb  $c_i$  by a tiny amount, it will still be inactive, and therefore,  $x^*$  will still be a local solution of the optimization problem. Since  $\mu_i^* = 0$  from (11.21e), the Lagrange multiplier shows that the constraint  $i$  has no

importance in the system of the constraints. Otherwise, as in Theorem 11.14, the following theorem can be presented.

**Theorem 11.18** (*Interpretation of the Lagrange Multipliers for General Problems*). Consider the family of problems  $\min\{f(x) : c(x) \leq v, h(x) = w\}$ , where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c: \mathbb{R}^n \rightarrow \mathbb{R}^m$  and  $h: \mathbb{R}^n \rightarrow \mathbb{R}^p$  are twice continuously differentiable. Suppose for  $v = 0, w = 0$  there is a local solution  $x^*$  that is a regular point and that, together with its associated Lagrange multiplier  $\mu^* \geq 0, \lambda^*$ , satisfies the second-order sufficient conditions for a strict local minimum. Then, for every  $(v, w) \in \mathbb{R}^{m+p}$ , in a region containing  $(0, 0) \in \mathbb{R}^{m+p}$ , there is a solution  $x(v, w)$  continuously depending on  $(v, w)$ , such that  $x(0, 0) = x^*$  and such that  $x(v, w)$  is a local minimum of the problem. Furthermore,

$$\nabla_v f(x(v, w))|_{0,0} = -\mu^*,$$

$$\nabla_w f(x(v, w))|_{0,0} = -\lambda^*. \quad \blacklozenge$$

## 11.6 Duality

In optimization, the duality theory shows how to construct an alternative problem from functions and data that define the original optimization problem. In this context, the original problem is called the *primal* problem. In some cases, the dual problem is easier to solve than the original problem. Besides, the dual problem can be used to obtain a lower bound on the optimal value of the objective for the primal problem.

Let us consider the nonlinear optimization problem (primal problem)

$$\min_{x \in \mathbb{R}^n} f(x) \text{ subject to } c(x) \geq 0, \quad (11.24)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a continuously differentiable function and  $c(x) \triangleq [c_1(x), \dots, c_m(x)]^T$ , with  $-c_i(x)$ ,  $i = 1, \dots, m$ , are all convex functions. The Lagrangian function for (11.24) is

$$L(x, \lambda) = f(x) - \lambda^T c(x), \quad (11.25)$$

where  $\lambda \in \mathbb{R}^m$  is the Lagrange multiplier vector. With this, the dual objective function  $q: \mathbb{R}^m \rightarrow \mathbb{R}$  is defined as

$$q(\lambda) \triangleq \inf_x L(x, \lambda). \quad (11.26)$$

Obviously, for some problems, the infimum in (11.26) is  $-\infty$  for some values of  $\lambda$ . The domain of  $q$  is defined as the set of  $\lambda$  for which  $q$  is finite, that is,  $D \triangleq \{\lambda : q(\lambda) > -\infty\}$ . Observe that the infimum in (11.26) involves finding the *global* minimizer of the function  $L(., \lambda)$  for a given  $\lambda$ , which is a very difficult problem. However, when  $f$  and  $-c_i$ ,  $i = 1, \dots, m$ , are all convex functions and  $\lambda \geq 0$ , it follows that the function  $L(., \lambda)$  is also convex. In this case, all the local minimizers are also global minimizers. *The dual problem of (11.24) is defined as*

$$\max_{\lambda \in \mathbb{R}^m} q(\lambda) \text{ subject to } \lambda \geq 0. \quad (11.27)$$

In the following, we show how these problems are related.

**Theorem 11.19** *The function  $q$  defined by (11.26) is concave and its domain  $D$  is convex.*

**Proof** For any  $\lambda_0 \in \mathbb{R}^m$  and  $\lambda_1 \in \mathbb{R}^m$ , any  $x \in \mathbb{R}^n$ , and any  $\alpha \in [0, 1]$ , it follows that

$$L(x, (1 - \alpha)\lambda_0 + \alpha\lambda_1) = (1 - \alpha)L(x, \lambda_0) + \alpha L(x, \lambda_1).$$

As we know, the infimum of a sum is greater than or equal to the sum of the infimums. Therefore, taking the infimum of both sides in the above expression and using (11.26), we obtain

$$q((1 - \alpha)\lambda_0 + \alpha\lambda_1) \geq (1 - \alpha)q(\lambda_0) + \alpha q(\lambda_1),$$

showing the concavity of  $q$ . Now, if both  $\lambda_0$  and  $\lambda_1$  belong to  $D$ , it follows that  $q((1 - \alpha)\lambda_0 + \alpha\lambda_1) \geq -\infty$ . That is,  $(1 - \alpha)\lambda_0 + \alpha\lambda_1 \in D$ , i.e.,  $D$  is convex.  $\blacklozenge$

**Theorem 11.20** *For any  $\hat{x}$  feasible for (11.24) and any  $\hat{\lambda} \geq 0$ , it follows that  $q(\hat{\lambda}) \leq f(\hat{x})$ .*

**Proof** We have  $\hat{\lambda} \geq 0$  and  $c(\hat{x}) \geq 0$ . Therefore,

$$q(\hat{\lambda}) = \inf_x f(x) - \hat{\lambda}^T c(x) \leq f(\hat{x}) - \hat{\lambda}^T c(\hat{x}) \leq f(\hat{x}).$$

$\blacklozenge$

Theorem 11.20, called the *weak duality theorem*, tells us that value of the dual problem (11.27) gives a lower bound on the optimal objective value for the primal problem (11.24).

The KKT optimality conditions for the primal problem (11.24) are as follows:

$$\nabla f(\hat{x}) - \nabla c(\hat{x})\hat{\lambda} = 0, \quad (11.28a)$$

$$c(\hat{x}) \geq 0, \quad (11.28b)$$

$$\hat{\lambda} \geq 0, \quad (11.28c)$$

$$\hat{\lambda}_i c_i(\hat{x}) = 0, \quad i = 1, \dots, m, \quad (11.28d)$$

where  $\nabla c(x) \in \mathbb{R}^{n \times m}$  is the Jacobian matrix defined by  $\nabla c(x) = [\nabla c_1(x), \dots, \nabla c_m(x)]$ . The next theorem, due to Wolfe (1961), shows that the optimal Lagrange multipliers for (11.24) are solutions of the dual problem (11.27).

**Theorem 11.21** *Suppose that  $\hat{x}$  is a solution of (11.24) and that  $f$  and  $-c_i$ ,  $i = 1, \dots, m$ , are convex functions on  $\mathbb{R}^n$  that are continuously differentiable. Then, any  $\hat{\lambda}$  for which  $(\hat{x}, \hat{\lambda})$  satisfies the KKT conditions (11.28) is a solution of the dual problem (11.27).*

**Proof** Suppose that  $(\hat{x}, \hat{\lambda})$  satisfies (11.28). Since  $\hat{\lambda} \geq 0$ , it follows that  $L(., \hat{\lambda})$  is a convex and differentiable function. Therefore, having in view (11.28a), for any  $x$ , it follows that

$$L(x, \hat{\lambda}) \geq L(\hat{x}, \hat{\lambda}) + \nabla_x L(\hat{x}, \hat{\lambda})^T (x - \hat{x}) = L(\hat{x}, \hat{\lambda}).$$

Now, from (11.28d), we have

$$q(\hat{\lambda}) = \inf_x L(x, \hat{\lambda}) = L(\hat{x}, \hat{\lambda}) = f(\hat{x}) - \hat{\lambda}^T c(\hat{x}) = f(\hat{x}).$$

From Theorem 11.20, it follows that  $q(\lambda) \leq f(\hat{x})$  for all  $\lambda \geq 0$ . Therefore, from  $q(\hat{\lambda}) = f(\hat{x})$ , it follows that  $\hat{\lambda}$  is a solution of the dual problem (11.27).  $\blacklozenge$

We have dealt so far with the duality in the *Lagrange sense*. In the following, we present a slightly different form of the duality, the so-called the *Wolfe duality* (1961), which is very convenient for computations. The Wolfe dual of (11.24) is

$$\max_{x, \lambda} L(x, \lambda) \quad (11.29a)$$

$$\text{subject to} \quad \nabla_x L(x, \lambda) = 0, \quad \lambda \geq 0. \quad (11.29b)$$

The following theorem shows the relationship between these dual problems.

**Theorem 11.22** Suppose that  $f$  and  $-c_i, i = 1, \dots, m$ , are convex and continuously differentiable on  $\mathbb{R}^n$ . Suppose that  $(\hat{x}, \hat{\lambda})$  is a solution of (11.24) at which the linear independence constrained qualification holds. Then,  $(\hat{x}, \hat{\lambda})$  solves the dual problem (11.29).

**Proof** From the KKT optimality conditions (11.28), it follows that  $(\hat{x}, \hat{\lambda})$  satisfies (11.29b) and  $L(\hat{x}, \hat{\lambda}) = f(\hat{x})$ . Therefore, having in view the convexity of  $L(., \lambda)$ , for any pair  $(x, \lambda)$  that satisfies (11.29b), we have

$$\begin{aligned} L(\hat{x}, \hat{\lambda}) &= f(\hat{x}) \geq f(\hat{x}) - \lambda^T c(\hat{x}) = L(\hat{x}, \lambda) \\ &\geq L(x, \lambda) + \nabla_x L(x, \lambda)^T (\hat{x} - x) = L(x, \lambda), \end{aligned}$$

showing that  $(\hat{x}, \hat{\lambda})$  maximizes  $L$  over the constraints (11.29b), i.e., solvers (11.29).  $\blacklozenge$

**Example 11.1** For the linear programming problem

$$\min c^T x \text{ subject to } Ax \geq b, \quad (11.30)$$

the Lagrange dual is

$$\max_{\lambda} b^T \lambda \text{ subject to } A^T \lambda = c, \lambda \geq 0. \quad (11.31)$$

The Wolfe dual is

$$\max_{\lambda} c^T x - \lambda^T (Ax - b) \text{ subject to } A^T \lambda = c, \lambda \geq 0. \quad (11.32)$$

Obviously, substituting  $A^T \lambda - c = 0$  into the objective of (11.32), we obtain (11.31).  $\blacklozenge$

**Example 11.2** For the quadratic problem

$$\min \frac{1}{2}x^T Gx + c^T s \text{ subject to } Ax - b \geq 0, \quad (11.33)$$

where  $G$  is symmetric and positive definite, the dual objective is

$$q(\lambda) = \inf_x L(x, \lambda) = \inf_x \frac{1}{2}x^T Gx + c^T s - \lambda^T (Ax - b).$$

Since  $G$  is positive definite and  $L(., \lambda)$  is a strictly convex quadratic function, it follows that the infimum is achieved when  $\nabla_x L(x, \lambda) = 0$ , that is, when  $Gx + c - A^T \lambda = 0$ . Therefore,  $x = G^{-1}(A^T \lambda - c)$ . Introducing it in the infimum, we get the dual objective as

$$q(\lambda) = -\frac{1}{2}(A^T \lambda - c)^T G^{-1}(A^T \lambda - c) + b^T \lambda.$$

The dual Wolfe of (11.33) is

$$\max_{\lambda, x} \frac{1}{2}x^T Gx + c^T x - \lambda^T (Ax - b) \text{ subject to } Gx + c - A^T \lambda = 0, \lambda \geq 0. \quad (11.34)$$

However,  $(c - A^T \lambda)^T x = -x^T Gx$ . Therefore, (11.34) can be rewritten as

$$\max_{\lambda, x} -\frac{1}{2}x^T Gx + b^T \lambda \text{ subject to } Gx + c - A^T \lambda = 0, \lambda \geq 0, \quad (11.35)$$

which is the Wolfe dual of (11.33). ◆

## Notes and References

Plenty of books and papers are dedicated to the theoretical developments of the optimality conditions for continuous nonlinear optimization. Many details and properties of the theoretical aspects of the optimality conditions and duality can be found in Bertsekas (1999), Nocedal and Wright (2006), Sun and Yuan (2006), etc. The content of this chapter is based on the books by Chachuat (2007); Bazaraa, Sherali, and Shetty (1993); Luenberger and Ye (2016); and Nocedal and Wright (2006). A thorough description of the constraint qualification and duality is given in Mangasarian (1995). As for the optimality conditions for the problems with inequality constraints, the material is inspired by Bazaraa, Sherali, and Shetty (1993). The derivation of the necessary and sufficient optimality conditions for problems with equality constraints follows the developments presented by Luenberger (1973). The sensitivity analysis and the interpretation of the Lagrange multipliers for nonlinear optimization are derived from Luenberger (1973). The duality for nonlinear programming is described by Bertsekas (1999). Our presentation of the duality follows the ideas of Nocedal and Wright (2006) and Griva, Nash, and Sofer (2009).

The KKT conditions were originally named after Harold W. Kuhn (1925–2014) and Albert W. Tucker (1905–1995), who first published them in 1951 (Kuhn & Tucker, 1951). Later on, the scholars discovered that the necessary conditions for this problem had been stated by William Karush (1917–1997) in his master's thesis in 1939 (Karush, 1939). Another approach of the optimality conditions for the nonlinear optimization problem was given in 1948 by Fritz John (1910–1994) (John 1948). Also see Cottle (2012).



## Simple Bound Constrained Optimization

12

The simple bound constrained optimization is a class of nonlinear optimization problems with a special structure, found in many real practical applications. The mathematical model of these problems is as follows:

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & x \in X = \{x \in \mathbb{R}^n, l_i \leq x_i \leq u_i, i = 1, \dots, n\}. \end{aligned} \tag{12.1}$$

The function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is supposed to be at least twice continuously differentiable. The set  $X$  defined by the real numbers  $-\infty < l_i \leq u_i < +\infty$ ,  $i = 1, \dots, n$ , which represents the *bounds on the variables*, is the *feasibility domain* of the problem (12.1). Suppose that for any  $i = 1, \dots, n$ ,  $l_i \leq u_i$ , that is, the feasibility domain  $X$  is nonempty. Since  $X$  is a compact set, it follows that (12.1) always has a local optimum solution (see [Theorem 11.1](#)). The inequalities  $-\infty < l_i \leq x_i \leq u_i < +\infty$ ,  $i = 1, \dots, n$ , are called *simple bounds*. The  $i$ -th simple bound constraint is *active* at  $x \in X$  if  $x_i = l_i$ , or  $x_i = u_i$ . In the first case, if  $x_i = l_i$ , we say that the variable  $x_i$  is assigned to its *lower bound*. In the second case, if  $x_i = u_i$ , the variable  $x_i$  is assigned to its *upper bound*. The *set of active constraints* at the point  $x \in X$  is denoted by  $A(x)$ . The *set of inactive constraints* at  $x \in X$  is denoted by  $I(x)$ . If  $l_i = u_i$ , then the  $i$ -th component of  $x$  is *fixed* to the value  $l_i = u_i$ .

This problem is often a sub-problem of the augmented Lagrangian or of the penalty computational schemes for solving the general constrained optimization (Facchinei & Lucidi, 1992a, b; Conn, Gould, & Toint, 1997a; Nocedal & Wright, 2006; Sun & Yuan, 2006). Therefore, the development of numerical algorithms to efficiently solve (12.1), especially for large-scale problems, is important in both theory and practice.

*The purpose of this chapter is to present this class of problems, the main aspects of the optimality conditions, as well as the main computational methods for solving these problems.* For this, we follow the developments given by Kelley (1999), together with some details from Bertsekas (1976, 1982a, b, 1999); Lin and Moré (1999); Birgin and Martínez (2001); Birgin, Martínez, and Raydan (2000, 2001); Byrd, Lu, and Nocedal (1995a); Byrd, Lu, Nocedal, and Zhu (1994a, 1995b); and Hager and Zhang (2006a, b). At the same time, we present some computational results, comparisons among the algorithms, as well as some nonlinear optimization applications with simple bounds. From the multitude of algorithms dedicated to this problem, we insist on the spectral projected gradient

method, on the limited memory BFGS with simple bounds, and on the truncated Newton with simple bounds.

## 12.1 Necessary Conditions for Optimality

As we know, for a continuously differentiable function of one variable, the necessary conditions for the unconstrained optimality at  $x^*$  are simply  $f'(x^*) = 0$ , and if  $f$  is twice continuously differentiable,  $f''(x^*) \geq 0$ . For a simple bound constrained problem in which the variables restrict the domain of  $f$  to an interval  $[a, b]$ , the necessary condition must be changed in order to admit the possibility for the minimizer to be one of the endpoints of the interval  $[a, b]$ . If  $x^* = a$  is a local minimizer, then  $f(x) \geq f(a)$  for all  $a \leq x$  sufficiently near  $a$ . Therefore,  $f'(a) \geq 0$ . Nothing can be said about  $f''$ . Similarly, if  $x^* = b$  is a local minimizer, then  $f'(b) \leq 0$ . Hence, all three possibilities  $x^* = a$ ,  $x^* = b$ , and  $a < x^* < b$  can be expressed by the following theorem:

**Theorem 12.1** *Let  $f$  be a continuously differentiable function of one variable on the interval  $[a, b]$ . Let  $x^*$  be a local minimum of  $f$  on  $[a, b]$ . Then,*

$$f'(x^*)(x - x^*) \geq 0 \text{ for all } x \in [a, b], \quad (12.2)$$

and if  $f$  is twice continuously differentiable on  $[a, b]$ ,

$$f''(x^*)(x^* - a)(b - x^*) \geq 0. \quad (12.3)$$

A point  $x^* \in X$  is *stationary* for the problem (12.1) if

$$\nabla f(x^*)^T(x - x^*) \geq 0 \text{ for all } x \in X. \quad (12.4)$$

As in the unconstrained case, the *stationary points* are said to satisfy the *first-order necessary conditions*.

In order to present the second-order necessary conditions for the problem (12.1), the *reduced Hessian* is introduced. Let  $f$  be twice continuously differentiable at  $x \in X$ . The reduced Hessian  $\nabla_R^2 f(x)$  is the matrix

$$(\nabla_R^2 f(x))_{ij} = \begin{cases} \delta_{ij}, & i \in A(x), j \in A(x), \\ (\nabla^2 f(x))_{ij}, & \text{otherwise.} \end{cases} \quad (12.5)$$

**Theorem 12.2** *Let  $f$  be twice Lipschitz continuously differentiable and let  $x^*$  be the solution of the problem (12.1). Then, the reduced Hessian  $\nabla_R^2 f(x^*)$  is positive semidefinite.*

**Proof** Assume that at point  $x$  there are  $t$  inactive indices and  $n - t$  active indices. With this, the vector  $x \in X$  can be partitioned by reordering the variables as  $x = [z, y]$ , where  $z$  corresponds to the inactive indices and  $y$  to the active ones. Then, the map  $\psi(z) = f(z, y^*)$  has an unconstrained local minimizer at  $z^* \in \mathbb{R}^t$ , and hence,  $\nabla^2 \psi$  is positive semidefinite. However, the reduced Hessian can be written as

$$\nabla_R^2 f(x^*) = \begin{bmatrix} \nabla^2 \psi(x^*) & \\ & I \end{bmatrix}$$

if the variables are partitioned as above.  $\diamond$

Let  $P$  be the *projection* onto  $X$ , i.e., the map that takes  $x$  to the nearest point (in the  $l_2$  norm) in  $X$  to  $x$ . Then,

$$P(x)_i = \begin{cases} l_i, & \text{if } (x)_i \leq l_i, \\ (x)_i, & \text{if } l_i < (x)_i < u_i, \\ u_i, & \text{if } (x)_i \geq u_i. \end{cases} \quad (12.6)$$

The following theorem proved in Kelley (1999) states the necessary condition for optimality.

**Theorem 12.3** *Let  $f$  be continuously differentiable. A point  $x^* \in X$  is stationary for the problem (12.1) if and only if*

$$x^* = P(x^* - \alpha \nabla f(x^*)) \quad (12.7)$$

for all  $\alpha \geq 0$ .  $\diamond$

In a Lagrangian formalism, supposing that no  $l_i$  is  $-\infty$  and no  $u_i$  is  $+\infty$ ,  $i = 1, \dots, n$ , then the KKT conditions for  $x^*$  to solve the problem (12.1) are

$$\begin{aligned} \nabla f(x^*) - \lambda^* + \mu^* &= 0, \\ \lambda^* \geq 0, \quad (l - x^*)^T \lambda^* &= 0, \\ \mu^* \geq 0, \quad (x^* - u)^T \mu^* &= 0, \\ l \leq x^* \leq u, \end{aligned}$$

where  $\lambda^*, \mu^* \in \mathbb{R}^n$  are the KKT multipliers. In this context, the strict complementarity is said to hold at the KKT point  $(x^*, \lambda^*, \mu^*)$  as follows: if  $x_i^* = l_i$  implies  $\lambda_i^* > 0$  and  $x_i^* = u_i$  implies  $\mu_i^* > 0$ . Another equivalent way to present the KKT conditions is as follows:

$$\begin{aligned} l \leq x^* \leq u, \\ l_i < x_i^* < u_i \Rightarrow \nabla f(x^*)_i &= 0, \\ x_i^* = l_i \Rightarrow \nabla f(x^*)_i &\geq 0, \\ x_i^* = u_i \Rightarrow \nabla f(x^*)_i &\leq 0. \end{aligned}$$

If the projected gradient  $\nabla_P f(x)$  is defined by

$$\nabla_P f(x^*)_i \triangleq \begin{cases} \min \{0, \nabla f(x^*)_i\}, & \text{if } x_i^* = l_i, \\ \nabla f(x^*)_i, & \text{if } l_i < x_i^* < u_i, \\ \max \{0, \nabla f(x^*)_i\}, & \text{if } x_i^* = u_i, \end{cases}$$

then the first-order necessary conditions for optimality can be written as

$$l \leq x^* \leq u, \quad \nabla_P f(x^*) = 0.$$

## 12.2 Sufficient Conditions for Optimality

The sufficient conditions are formulated by using the definition of the reduced Hessian. Observe that if  $x^*$  is stationary,  $i \in I(x^*)$ , and  $e_i$  is a unit vector in the  $i$ -th coordinate direction, then  $x^* \pm te_i \in X$  for all  $t$  sufficiently small. Since

$$\frac{df(x^* \pm te_i)}{dt} = \pm \nabla f(x^*)^T e_i \geq 0,$$

it follows that

$$(\nabla f(x^*))_i = 0 \text{ for all } i \in I(x^*). \quad (12.8)$$

In order to formulate the sufficient conditions for optimality, the concept of nondegenerate stationary point is introduced. A point  $x^* \in X$  is a nondegenerate stationary point for the problem (12.1) if  $x^*$  is a stationary point and

$$(\nabla f(x^*))_i \neq 0 \text{ for all } i \in A(x^*). \quad (12.9)$$

If  $x^*$  is a solution of the problem (12.1), then  $x^*$  is a nondegenerate local minimizer.

Nondegeneracy is important in the formulation of sufficient conditions and in the design of the termination criteria. Let  $S$  be an arbitrary set of indices. Then, define

$$P_S(x)_i = \begin{cases} x_i, & i \in S, \\ 0, & i \notin S. \end{cases} \quad (12.10)$$

The following theorem proved in Kelley (1999) gives the sufficiency conditions associated to problem (12.1).

**Theorem 12.4** Let  $x^* \in X$  be a nondegenerate stationary point for the problem (12.1). Let  $f$  be twice continuously differentiable in a neighborhood of  $x^*$ , and assume that the reduced Hessian at  $x^*$  is positive definite. Then,  $x^*$  is a solution of the problem (12.1) and, hence, a nondegenerate local minimizer.

**Proof** Let  $x \in X$  and define  $\varphi(t) = f(x^* + t(x - x^*))$ . We prove that either (i)  $\varphi'(0) > 0$  or (ii)  $\varphi'(0) = 0$ ,  $\varphi''(0) > 0$ . Consider  $e = x - x^*$ . Observe that

$$\varphi'(0) = \nabla f(x^*)^T e = \nabla f(x^*)^T (P_A(e) + P_I(e)).$$

However, from the stationarity, we have  $\nabla f(x^*)^T P_I(e) = 0$ . Now, if  $P_A(e) \neq 0$ , then, from nondegeneracy, it follows that  $\nabla f(x^*)^T P_A(e) > 0$ , i.e., (i) holds. On the other hand, if  $P_A(e) = 0$ , then

$$\varphi''(0) = (x - x^*)^T P_I(\nabla^2 f(x^*)) P_I(x - x^*) = (x - x^*)^T \nabla_R^2 f(x^*) (x - x^*) > 0,$$

which proves (ii). ◆

---

## 12.3 Methods for Solving Simple Bound Optimization Problems

One of the methods for solving simple bound optimization problems is the *active-set method*. Polyak (1969) proposed an extension of the conjugate gradient method to this class of problems with

quadratic objective. In this method, the conjugate gradient method is used to explore a face of the feasible set, and the negative gradient is used to leave a face. At every iteration, the algorithm of Polyak only added or dropped one constraint. Dembo and Tulowitzki (1983) proposed an algorithm which could add and drop many constraints at an iteration. Later, Yang, and Tolle (1991) further developed Polyak's algorithm to obtain a finite termination even when the problem was degenerate at the local minimizer. Another variant of this algorithm, which includes a rigorous convergence analysis, was given by Wright (1990). Moré and Toraldo (1991) improved this algorithm by proposing a gradient projection method to identify a working face of the feasible set, followed by a conjugate gradient method to explore the face. For a simple bound optimization problem, at point  $x \in X$ , the following two sets  $L(x) = \{i : x_i = l_i\}$  and  $U(x) = \{i : x_i = u_i\}$  can be defined. Clearly,  $A(x) = L(x) \cup U(x)$ . If  $x^*$  is a local minimizer of  $f$  in  $X$ , then  $x^*$  is a local minimizer of  $f(x)$  subject to  $x_i = l_i, i \in L(x^*)$  and  $x_i = u_i, i \in U(x^*)$ . The active-set methods aim at predicting  $L(x^*)$  and  $U(x^*)$  by using in an iterative way the disjoint sets  $L, U \subseteq \{1, \dots, n\}$ . The idea of the active-set method is as follows: given the sets  $L$  and  $U$ , the following problem

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & x_i = l_i, i \in L \text{ and } x_i = u_i, i \in U \end{aligned}$$

is approximately solved as an unconstrained optimization problem over the variables  $x_i, i \notin A = L \cup U$ . For the quadratic programming case, for which  $f$  is quadratic, the active-set methods were developed by Coleman and Hulbert (1989). The main difficulty with the active-set method is the selection of the sets  $L$  and  $U$  in an iterative process either by adding variables which violate one of their bounds or by removing those for which further progress is predicted.

An *active-set algorithm* was developed by Hager and Zhang (2006a), which consists of a nonmonotone gradient projection step, an unconstrained optimization step, and a set of rules for branching between steps. The implementation of this algorithm uses the cyclic Barzilai-Borwein algorithm for the gradient projection step and the conjugate gradient algorithm CG-DESCENT by Hager and Zhang (2005) for the unconstrained optimization step. An attractive feature of this algorithm is that the search directions are always sufficient descent directions, and when the objective function is strongly convex quadratic, the convergence of the algorithm is achieved in a finite number of iterations, even when the strict complementarity slackness does not hold (see [Definition 11.14](#)) (Hager & Zhang, 2006a).

Other methods for solving these problems is the *gradient projected* (Rosen, 1960; Levitin & Polyak, 1966; Bertsekas, 1976). These methods are extensions of the steepest-descent method to deal with convex constraints. The iteration is computed as  $x_{k+1} = P_X(x_k - \alpha_k \nabla f(x_k))$ , where  $P_X(v)$  projects  $v$  onto  $X$  and  $\alpha_k$  is the stepsize. In the case of simple bounds,  $P_X(v)$  is trivial to compute. Bertsekas (1976) showed that the method has the following important feature. For the nondegenerate problems (i.e., those problems for which the removing of one or more active constraints necessarily changes the solution), the optimal “face” of the active constraints will be determined in a finite number of iterations. If the active faces visited by consecutive iterates of the gradient projected algorithm are identical, then the Newton-like method should be used to investigate this face. Therefore, the *projected Newton method* is generated (Bertsekas, 1982a). For the quadratic case, this idea was considered by Moré and Toraldo (1991). In this case, if the initial point is sufficiently near a nondegenerate local minimizer  $x^*$ , then the search direction is computed by using the reduced Hessian. The resulting projected Newton method will take full steps, i.e.,  $\alpha_k = 1$  along the iterations. On the other hand, if the initial point is far from  $x^*$  and the reduced Hessian is not symmetric and positive definite, then the line-search may fail. Clearly, this possibility of indefiniteness is the main

weakness in any line-search method which uses  $\nabla^2 f$  when far from the minimizer. To accelerate the convergence of these methods, more research has been developed on the Newton and trust-region methods. For the nondegenerate bound optimization problems, the superlinear and quadratic convergence was established by Conn, Gould, and Toint (1988a, b, 1991a) and Facchinei, Júdice, and Soares (1998). For the degenerate problems, these convergence results were established by Facchinei, Lucidi, and Palagi (2002); Friedlander, Martínez, and Santos (1994); Lescrenier (1991); and Lin and Moré (1999). Computing the Newton step can be expensive. Therefore, approximation techniques, such as the sparse or incomplete Cholesky factorization, have been suggested in order to reduce the computational expense.

A *gradient projection method with the limited memory BFGS* matrix to approximate the Hessian of the objective function was developed by Byrd, Lu, Nocedal, and Zhu (1995b). In this algorithm, the gradient projection method is used to determine a set of active constraints at each iteration. This is a line-search algorithm (as opposed to trust-region) that uses the limited memory BFGS matrices. Another algorithm that uses the limited-memory quasi-Newton methods to update the inactive variables and a projected gradient method to update the active variables was given by Ni and Yuan (1997).

A *trust-region Newton method* is the TRON method (Lin & Moré, 1999). It uses the gradient projection to generate a Cauchy step, a preconditioned conjugate gradient method with an incomplete Cholesky factorization to generate a direction, and a projected search to compute the step. Using the projected searches allows TRON to examine the faces of the feasible domain by generating a small number of minor iterations.

The *affine-scaling interior point method* by Coleman and Li (1994, 1996, 1997) represents a different approach, related to the trust-region method. Some developments of this strategy are presented in Dennis, Heinkenschlos, and Vicente (1998); Heinkenschlos, Ulbrich, and Ulbrich (1999); and Ulbrich, Ulbrich, and Heinkenschlos (1999). These methods are characterized by a reformulation of the necessary optimality conditions obtained by multiplication with a scaling matrix. The resulting system is solved by using the Newton method. In another way of research, Zhang (2004) proposes an interior point approach for solving this linear system.

As we have already said, a projected Newton-like method for the solution of (12.1) is the *truncated Newton method* for the large-scale box constrained optimization given by Facchinei, Lucidi, and Palagi (2002). This method is based on the works of Facchinei and Lucidi (1992a, b). At each iteration  $k$ , the estimates  $L(x_k)$  and  $U(x_k)$  of the variables are defined that will supposedly be at their lower and upper bounds at the solution, respectively. At the same time, the estimate  $F(x_k)$  of the variables which we believe to be free is computed. This partition of variables suggests performing an unconstrained minimization in the space of the free variables, a typical approach in the active-set methods. For a locally fast convergent method, an obvious choice for the unconstrained minimization in the subspace of the free variables is the Newton method. In order to compensate the loss of the curvature information that we have in the subspace of those variables which are active but with zero multiplier, Facchinei, Lucidi, and Palagi (2002) introduced a correction term in the right-hand term of the Newton system. Other approaches also based on the truncated Newton method are those given by Schlick and Fogelson (1992a, b) and Nash (1984a, b, 1985).

A *nonmonotone projected gradient* (SPG) for solving simple bound optimization problems is the one given by Birgin, Martínez, and Raydan (2000, 2001). This algorithm combines the projected gradient method of Bertsekas (1976) with two new features in optimization. The first one is based on the nonmonotone line-search developed by Grippo, Lampariello, and Lucidi (1986). The second one uses the spectral step length introduced by Barzilai and Borwein (1988) and further analyzed by Raydan (1993, 1997).

Another approach that we mention and that was given by Sainvitu and Toint (2006) is the *filter-trust-region method* for the simple bound constrained optimization. The algorithm combines the filter-trust-region algorithm of Gould, Sainvitu, and Toint (2005b) with a gradient projection method.

Finally, a *direct method* for solving (12.1) is BOBYQA, by Powell (2009), where a quadratic approximation of the objective function is minimized at each iteration. Recent advances in the simple bound constrained optimization have been presented by Hager and Zhang (2006b).

In the following, we present the spectral projected gradient method (SPG) by Birgin, Martínez, and Raydan (2000, 2001), the limited-memory BFGS algorithm with gradient projection (L-BFGS-B) by Byrd, Lu, Nocedal, and Zhu (1995b), and the truncated Newton with simple bounds (TNBC) by Nash (1984a, b, 1985) for solving simple bound constrained optimization problems.

## 12.4 The Spectral Projected Gradient Method (SPG)

The gradient projected algorithm and its spectral variant are natural extensions of the steepest descent algorithm to simple bound constrained problems (Bertsekas, 1976). Let us consider a current iteration  $x$ . The new iteration is computed as

$$x_+ = P(x - \alpha \nabla f(x)),$$

where  $\alpha$  is the stepsize computed by the Armijo rule or by some other line-search procedures. The gradient projected algorithm determines a sequence of iterations which satisfies the simple bounds of the problem (12.1) and achieves a sufficient decrease of the function  $f$ . For  $\alpha > 0$ , define

$$x(\alpha) = P(x - \alpha \nabla f(x)). \quad (12.11)$$

The sufficient decrease condition for the line-search in simple bound constrained problems is expressed as

$$f(x(\alpha)) - f(x) \leq \frac{-\rho}{\alpha} \|x - x(\alpha)\|^2, \quad (12.12)$$

where  $\rho$  is a positive parameter ( $\rho = 10^{-4}$  (Dennis & Schnabel, 1983)). Therefore, the general gradient projected algorithm can be presented as the following algorithm.

### Algorithm 12.1 Gradient projected for simple bounds

- |    |  |
|----|--|
| 1. | Select an initial point $x_0$ and a value for the parameter $\beta \in (0, 1)$ . Set $k = 1$ |
| 2. | Test a criterion for stopping the iterations   |
| 3. | Compute $f(x_k)$ and $\nabla f(x_k)$   |
| 4. | (Armijo rule) Find the least integer $m$ such that (12.12) holds for $\alpha = \beta^m$      |
| 5. | Set $x_{k+1} = x(\alpha)$ , $k = k + 1$ and go to step 2                                     |

◆

The spectral projected gradient method is a method of projected gradient which includes two ingredients. The first one is an extension to the simple bound optimization problem and an extension of the globalization techniques used in the unconstrained optimization and based on the non-monotone line-search by Grippo, Lampariello and Lucidi (1986). The second one consists in using the spectral stepsize introduced by Barzilai and Borwein (1988) and analyzed by Raydan (1993). Mainly, the scaled gradient projected algorithm is a version of the gradient projected algorithm by Bertsekas, which uses the Armijo rule along a nonlinear trajectory of projections.

Consider the problem (12.1) where the function  $f$  is defined and has continuous partial derivatives on an open set that contains  $X$ . Assume that  $f$  is bounded from below on  $X$ . Consider a point  $x \in \mathbb{R}^n$ . Then, the *orthogonal projection* of  $x$  onto  $X$  is  $P_X(x)$  with

$$(P_X(x))_i = \max \{l_i, \min \{x_i, u_i\}\}, \quad i = 1, \dots, n. \quad (12.13)$$

For any  $x \in X$ , the algorithm uses the *spectral projected gradient* defined as

$$g_t(x) = [P_X(x - t\nabla f(x)) - x], \quad (12.14)$$

where  $t > 0$  is a spectral selection of the stepsize. Observe that zeroing the spectral projected gradient is equivalent to the optimality conditions of the first order. Therefore, the norm of the spectral projected gradient is a criterion for stopping the iterations.

The algorithm starts with an initial point  $x_0 \in \mathbb{R}^n$  and uses an integer  $m \geq 1$  used in the nonmonotone line-search, a small parameter  $\alpha_{\min} > 0$ , a large parameter  $\alpha_{\max} > \alpha_{\min}$ , a small parameter  $\varepsilon_1 > 0$ , a sufficient decrease parameter  $\gamma \in (0, 1)$ , and the safeguarding parameters  $0 < \sigma_1 < \sigma_2 < 1$  used as protection in the quadratic interpolation. Initially,  $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$  is arbitrary. The step length is computed by the quadratic or cubic interpolation.

### Algorithm 12.2 Spectral projected gradient—SPG

1.	<i>Initialization.</i> Consider some numerical values for the above described parameters, as well as the initial point $x_0 \in \mathbb{R}^n$ . If $x_0 \notin X$ , then set $x_0 = P_X(x_0)$ . Set $k = 0$
2.	Compute $f(x_k)$ and $\nabla f(x_k)$
3.	Compute the <i>gradient projected</i> $P_X(x_k - \nabla f(x_k))$
4.	Compute the <i>initial spectral step length</i> as $\alpha = \min \left\{ \alpha_{\max}, \max \left\{ \alpha_{\min}, \frac{1}{\ P_X(x_k - \nabla f(x_k)) - x_k\ _{\infty}} \right\} \right\}$
5.	Test for stopping the iterations. If $\ P_X(x_k - \nabla f(x_k)) - x_k\ _{\infty} \leq \varepsilon_1,$ stop; otherwise, continue with step 6
6.	Compute the <i>spectral direction of the projected gradient</i> as $d = P_X(x_k - \alpha \nabla f(x_k)) - x_k$
7.	<i>Initialization of the nonmonotone line-search.</i> Set $\alpha = 1$ and compute $x_{k+1} = x_k + ad$ . Compute $f(x_{k+1})$
8.	If $f(x_{k+1}) \leq \max_{0 \leq j \leq \min\{k, m\}} \{f(x_{k-j})\} + \gamma \alpha (d^T \nabla f(x_k)),$ then go to step 11; otherwise, continue with step 9, where the quadratic or cubic interpolation is implemented
9.	<i>Quadratic interpolation.</i> If $\alpha \leq 0.1$ , then set $\alpha = \alpha/2$ ; otherwise, compute $a_t = -\frac{\alpha^2 (d^T \nabla f(x_k))}{2(f(x_{k+1}) - f(x_k) - \alpha (d^T \nabla f(x_k)))}$ If $a_t < \sigma_1$ or $a_t > \sigma_2 \alpha$ , then set $a_t = a_t/2$ . Set $\alpha = a_t$ . Compute $x_{k+1} = x_k + ad$ and go to step 8. <i>Cubic interpolation.</i> If $\alpha = 1$ , then compute $\alpha_t = -\frac{\alpha^2 (d^T \nabla f(x_k))}{2(f(x_{k+1}) - f(x_k) - \alpha (d^T \nabla f(x_k)))}$ and go to step 10. Otherwise, compute $p = f(x_{k+1}) - f(x_k) - \alpha (d^T \nabla f(x_k)), \quad q = f_p - f(x_k) - \alpha_p (d^T \nabla f(x_k)),$ $a = \left( \frac{p}{\alpha^2} - \frac{q}{\alpha_p^2} \right) \frac{1}{\alpha - \alpha_p}, \quad b = \left( \frac{-p\alpha_p}{\alpha^2} + \frac{qa}{\alpha_p^2} \right) \frac{1}{\alpha - \alpha_p},$ $r = b^2 - 3a(d^T \nabla f(x_k)).$

If  $\alpha = 0$ , then set

$$\alpha_t = \frac{-(d^T \nabla f(x_k))}{2b},$$

otherwise,

$$\alpha_t = \frac{-b + \sqrt{r}}{3a}.$$

If  $\alpha_t > \alpha/2$ , then set  $\alpha_t = \alpha/2$

10. Set  $\alpha_p = \alpha, f_p = f(x_{k+1})$ . If  $\alpha_t \leq \alpha/10$ , then set  $\alpha = \alpha/10$ ; otherwise,  $\alpha = \alpha_t$ . Compute  $x_{k+1} = x_k + \alpha d$ ,  $f(x_{k+1})$  and continue with step 8

11. Compute  $s_k = x_{k+1} - x_k$ ,  $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$  and determine the spectral step length. If  $y_k^T s_k \leq 0$ , then set  $\alpha = \alpha_{\max}$ ; otherwise,

$$\alpha = \min \left\{ \alpha_{\max}, \max \left\{ \alpha_{\min}, \frac{s_k^T s_k}{y_k^T s_k} \right\} \right\},$$

$k = k + 1$  and continue with step 5

◆

The algorithm is based on the spectral projected gradient direction  $P_X(x_k - \alpha_k \nabla f(x_k)) - x_k$ , where  $\alpha_k$  is the safeguarded inverse Rayleigh quotient  $(s_k^T s_k) / (y_k^T s_k)$ . Observe that  $(s_k^T s_k) / (y_k^T s_k)$  is a Rayleigh quotient corresponding to the average Hessian matrix  $\int_0^1 \nabla^2 f(x_k + ts_k) dt$ .

The line-search uses the protected quadratic interpolation or the cubic interpolation implemented in steps 9 and 10, according to a value of a parameter, not explained here. The protection of the quadratic interpolation is given by the parameters  $\sigma_1$  and  $\sigma_2$ , which are initialized in step 1 of the algorithm. The quadratic interpolation acts when the minimum of the one-dimensional quadratic function  $q(\cdot)$  defined in such a way that  $q(0) = f(x_k)$ ,  $q(\alpha) = f(x_k + \alpha d_k)$ , and  $\nabla q(0) = d_k^T \nabla f(x_k)$  lies outside the interval  $[\sigma_1, \sigma_2 \alpha]$  and not when this minimum lies outside the interval  $[\sigma_1 \alpha, \sigma_2 \alpha]$ , as usually implemented. This means that when the interpolation tends to reject 90% (for  $\sigma_1 = 0.1$ ) of the original search interval (let us say  $[0,1]$ ), then we say that its prediction is not reliable and a more conservative bisection is preferred. On the other hand, the cubic interpolation considers an approximation of the minimizing function  $f$  by a cubic polynomial and uses four interpolation conditions based on the function values and its derivatives, each of them computed in two different points.

The convergence of this algorithm is proved by Birgin, Martínez, and Raydan (1999).

**Theorem 12.5** *The algorithm SPG is well defined, and any accumulation point of the sequence it generates is a constrained stationary point for the problem (12.1).* ◆

The proof is based on the results of Bertsekas (1999), which refer to the Armijo rule in the context of projections. Mainly, from the convexity of the domain  $X$ , it follows that for any  $x \in X$  and  $t \in (0, \alpha_{\max}]$ , the following results are true:

- (i)  $\nabla f(x)^T g_t(x) \leq -\frac{1}{t} \|g_t(x)\|_2^2 \leq -\frac{1}{\alpha_{\max}} \|g_t(x)\|_2^2$ ,
- (ii)  $g_t(\bar{x})$  is zero if and only if  $\bar{x}$  is a stationary point.

These results are used in Theorem 12.5 to prove that any accumulation point  $\bar{x}$  of the sequence  $\{x_k\}$  generated by the algorithm SPG is a stationary point, i.e., for any  $x \in X$ , it follows that  $\nabla f(\bar{x})^T (x - \bar{x}) \geq 0$ .

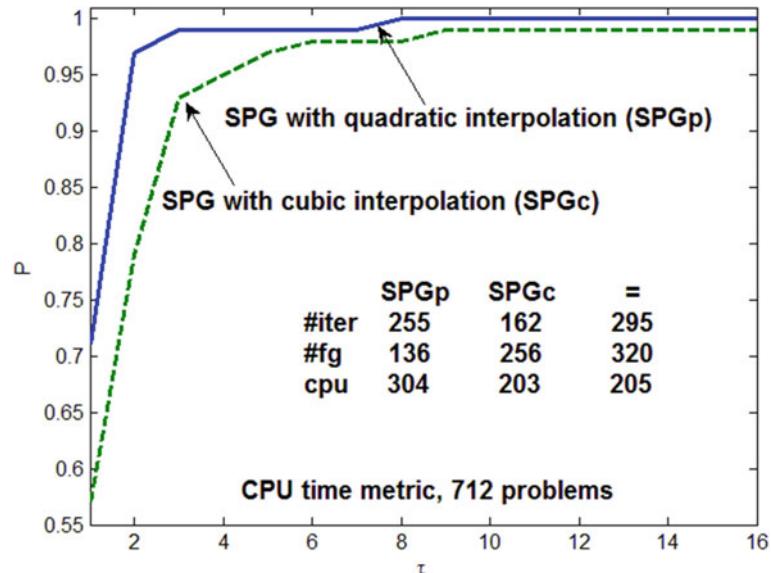
## Numerical Study—SPG: Quadratic Interpolation versus Cubic Interpolation

The SPG algorithm with both quadratic and cubic interpolation was implemented in double precision Fortran, compiled with f77 (default compiler settings), and run on a Workstation Intel Pentium 4 with 1.8 GHz. We selected a number of 80 large-scale simple bound optimization test functions in generalized or extended form presented in the UOP collection (Andrei, 2020a). For each test function, we considered 10 numerical experiments with the number of variables increasing as  $n = 1000, 2000, \dots, 10000$ . Therefore, a set of 800 simple bound optimization problems is obtained. Each problem is solved with SPG in two variants: SPG with quadratic interpolation (SPGp) and SPG with cubic interpolation (SPGc). The parameters used in SPG were initialized as  $m = 10$ ,  $\alpha_{\min} = 10^{-3}$ ,  $\alpha_{\max} = 10^3$ ,  $\gamma = 10^{-4}$ ,  $\sigma_1 = 0.1$ ,  $\sigma_2 = 0.9$ ,  $\varepsilon_1 = 0$ .

Figure 12.1 shows the Dolan and Moré (2002) performance profiles of SPGp versus SPGc, subject to the CPU time metric. From Fig. 12.1, we can see that, subject to the CPU time metric, SPGp is more efficient than SPGc, but they have the same robustness for solving this set of simple bound constrained optimization problems, SPGp being slightly more robust. Comparing SPGp versus SPGc subject to the number of iterations, we can see that SPGp was better in 255 problems (i.e., it achieved the minimum number of iterations in solving 255 problems). SPGc was better in 162 problems, and they achieved the same number of iterations in solving 295 problems. Therefore, subject to the number of iterations, SPGp is more efficient than SPGc. Similarly, SPGp is faster than SPGc.

Both these algorithms find local optimal solutions. Out of 800 problems considered in this numerical study, only for 712 problems does the criterion (1.3) hold. The conclusion of this numerical study is that both these scaled gradient projected algorithms with quadratic or cubic interpolation in the line-search represent efficient and robust algorithms for solving a large variety of simple bound optimization problems.

**Fig. 12.1** SPG: Quadratic interpolation versus cubic interpolation



## 12.5 L-BFGS with Simple Bounds (L-BFGS-B)

This section presents a very efficient and reliable algorithm for solving the simple bound optimization problem (12.1) based on the limited memory BFGS update which approximates the Hessian of the function  $f$ . The algorithm is described by Byrd, Lu, Nocedal, and Zhu (1995b) and is based on the developments given by Conn, Gould, and Toint (1988a) and Moré and Toraldo (1989). It uses the gradient projection method to determine a set of active constraints at each iteration. The main ingredient is the compact representation of the limited memory BFGS matrices described by Byrd, Nocedal, and Schnabel (1994b).

**Description of the algorithm** Suppose that at the current iterate  $x_k$ , the following elements are known: the value of the objective function  $f_k$ , the gradient  $g_k = \nabla f(x_k)$ , and a positive definite limited memory approximation  $B_k$  of the Hessian  $\nabla^2 f(x_k)$ . Therefore, the following quadratic model of  $f$  can be formed:

$$m_k(x) = f(x_k) + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k(x - x_k). \quad (12.15)$$

The algorithm approximately minimizes  $m_k(x)$  subject to the simple bounds  $l_i \leq x_i \leq u_i$ ,  $i = 1, \dots, n$ . It is done by using the gradient projection method to find a set of active constraints, followed by a minimization of  $m_k(x)$ , where the bounds are equality constraints.

For this, let us consider the piecewise linear path

$$x(t) = P_X(x_k - tg_k, l, u)$$

obtained by projecting the steepest descent direction onto the feasible set  $X$ , where

$$P_X(x, l, u)_i = \begin{cases} l_i, & x_i < l_i, \\ x_i, & l_i \leq x_i \leq u_i, \\ u_i, & x_i > u_i. \end{cases} \quad (12.16)$$

In the following, the generalized Cauchy point  $x^c$  is computed, which is defined as the first local minimizer of the univariate piecewise quadratic

$$q_k(t) = m_k(x(t)).$$

All the variables whose value at point  $x^c$  are at the lower or at the upper bound and which form the active set  $A(x^c)$  are held fixed. With this, the following quadratic programming problem over the subspace of free variables

$$\begin{aligned} & \min \{m_k(x) : x_i = x_i^c, \forall i \in A(x^c)\} \\ & \text{subject to} \\ & l_i \leq x_i \leq u_i, \quad \forall i \notin A(x^c), \end{aligned} \quad (12.17)$$

is considered. Firstly, an approximate solution of (12.17) is computed by ignoring the bounds on the free variables and by using  $x^c$  as the starting point. After an approximate solution  $\bar{x}_{k+1}$  of the problem (12.17) has been obtained, the new iteration  $x_{k+1}$  is computed by line-search along the direction  $d_k = \bar{x}_{k+1} - x_k$  using the strong Wolfe line-search conditions

$$f(x_{k+1}) \leq f(x_k) + \rho \alpha_k g_k^T d_k, \quad (12.18)$$

$$|g_{k+1}^T d_k| \leq \sigma |g_k^T d_k|, \quad (12.19)$$

where  $\alpha_k$  is the step length and  $\rho$  and  $\sigma$  are positive parameters used in the Wolfe line-search. With all these elements, the gradient at  $x_{k+1}$  is evaluated, a new limited-memory Hessian approximation  $B_{k+1}$  is computed, and the process is repeated.

Observe that the generalized Cauchy point  $x^c$ , which is a minimizer of  $m_k(x)$  on the projected steepest descent direction, satisfies  $m_k(x_k) > m_k(x^c)$  if the projected gradient is nonzero. Since  $\bar{x}_{k+1}$  is on the path from  $x^c$  to the minimizer of (12.17), along which  $m_k$  decreases, it follows that the value of  $m_k$  at  $\bar{x}_{k+1}$  is not larger than its value at  $x^c$ , i.e.,

$$f(x_k) = m_k(x_k) > m_k(x^c) \geq m_k(\bar{x}_{k+1}) = f(x_k) + g_k^T d_k + \frac{1}{2} d_k^T B_k d_k. \quad (12.20)$$

Therefore, if  $B_k$  is positive definite and  $d_k$  is not zero, the inequality (12.20) implies that  $g_k^T d_k < 0$ . In conclusion, since in our algorithm every Hessian approximation  $B_k$  is positive definite, it follows that the approximate solution  $\bar{x}_{k+1}$  of the quadratic problem (12.17) defines a descent direction  $d_k = \bar{x}_{k+1} - x_k$  for the objective function  $f$ .

**Limited-memory BFGS updates** The limited memory BFGS matrices used in the algorithm are represented in compact form. At every iteration  $x_k$ , the algorithm stores a small number, let us say  $m$ , of correction pairs  $\{s_i, y_i\}$ ,  $i = k-1, \dots, k-m$ , where  $s_k = x_{k+1} - x_k$  and  $y_k = g_{k+1} - g_k$ . These correction pairs contain information about the curvature of the function  $f$ , and in the frame of the BFGS formula, they define the limited-memory iteration matrix  $B_k$ . In the following, as described in Byrd, Nocedal, and Schnabel (1994b), we shall represent these matrices without explicitly forming them.

Firstly, the following  $n \times m$  correction matrices are formed:

$$Y_k = [y_{k-m}, \dots, y_{k-1}], \quad S_k = [s_{k-m}, \dots, s_{k-1}]. \quad (12.21)$$

Now, if  $\theta$  is a positive scaling parameter and if the  $m$  correction pairs  $\{s_i, y_i\}$ ,  $i = k-1, \dots, k-m$ , satisfy the condition  $y_i^T s_i > 0$ ,  $i = k-1, \dots, k-m$ , then the matrix obtained by updating  $\theta I$   $m$ -times using the BFGS updating formula and the pairs  $\{s_i, y_i\}$ ,  $i = k-1, \dots, k-m$ , can be written as

$$B_k = \theta I - W_k M_k W_k^T, \quad (12.22)$$

where

$$W_k = [Y_k \ \theta S_k], \quad (12.23)$$

$$M_k = \begin{bmatrix} -D_k & L_k^T \\ L_k & \theta S_k^T S_k \end{bmatrix}^{-1}, \quad (12.24)$$

and where  $L_k$  and  $D_k$  are the  $m \times m$  matrices

$$(L_k)_{ij} = \begin{cases} y_{k-m-1+j}^T s_{k-m-1+i}, & i > j, \\ 0, & i \leq j, \end{cases} \quad i, j = 1, \dots, m, \quad (12.25)$$

$$D_k = \text{diag}[y_{k-m}^T s_{k-m}, \dots, y_{k-1}^T s_{k-1}]. \quad (12.26)$$

Since  $M_k$  is a  $2m \times 2m$  matrix and since  $m$  is chosen to be a small integer, the cost of computing the inverse in (12.24) is negligible. The compact representation (12.22) is very efficient in numerical computations and involves the product of  $B_k$  with a vector, which often occurs in the algorithm.

In this context, as it is explained in Byrd, Nocedal, and Schnabel (1994b), there is a similar representation of the inverse limited-memory BFGS matrix  $H_k$  that approximates the inverse Hessian matrix

$$H_k = \frac{1}{\theta} I + \overline{W}_k \overline{M}_k \overline{W}_k^T, \quad (12.27)$$

where

$$\overline{W}_k = \left[ \begin{array}{cc} \frac{1}{\theta} Y_k & S_k \end{array} \right], \quad (12.28)$$

$$\overline{M}_k = \begin{bmatrix} 0 & -R_k^{-1} \\ -R_k^{-T} & R_k^{-T} \left( D_k + \frac{1}{\theta} Y_k^T Y_k \right) R_k^{-1} \end{bmatrix}, \quad (12.29)$$

$$(R_k)_{ij} = \begin{cases} y_{k-m-1+j}^T s_{k-m-1+i}, & i \leq j, \\ 0, & i > j. \end{cases} \quad (12.30)$$

Since the bounds on variables possibly prevent the line-search from satisfying the second Wolfe condition (12.19), there is no guarantee that the curvature condition  $y_k^T s_k > 0$  always holds. Therefore, in order to maintain the positive definiteness of the limited-memory BFGS matrix, if the curvature condition  $y_k^T s_k > \epsilon \|y_k\|^2$  is not satisfied for a small positive constant  $\epsilon$ , then the correction pairs  $(s_k, y_k)$  are discarded.

**The generalized Cauchy point** A very important element in the frame of the L-BFGS-B algorithm is the computation of the generalized Cauchy point as the first local minimizer of the quadratic model along the piecewise linear path obtained by projecting the points along the steepest descent direction  $x_k - tg_k$  onto the feasible domain. For this, define  $x^0 = x_k$  and drop the index  $k$ , so that  $x$ ,  $g$ , and  $B$  stand for  $x_k$ ,  $g_k$ , and  $B_k$ , respectively. However, subscripts are used to denote the component of a vector; for example,  $g_i$  is the  $i$ -th component of  $g$ . Superscripts will be used to represent the iterates during the piecewise search for the Cauchy point.

To define the breakpoints in each coordinate direction, the following elements are computed

$$t^i = \begin{cases} (x_i^0 - u_i)/g_i, & g_i < 0, \\ (x_i^0 - l_i)/g_i, & g_i > 0, \\ \infty, & g_i = 0, \end{cases} \quad i = 1, \dots, n, \quad (12.31)$$

and sorted in an increasing order to get the ordered set  $\{t^j : t^j \leq t^{j+1}, j = 1, \dots, n\}$ . In the following, a search along  $P_X(x^0 - tg, l, u)$  is done; thus a piecewise linear path expressed as

$$x_i(t) = \begin{cases} x_i^0 - t g_i, & t \leq t^i, \\ x_i^0 - t^i g_i, & t > t^i, \end{cases} \quad i = 1, \dots, n, \quad (12.32)$$

is obtained. Now, suppose that the interval  $[t^{j-1}, t^j]$  is examined. Let us define the  $(j-1)$ -th breakpoint as  $x^{j-1} = x(t^{j-1})$ , such that on  $[t^{j-1}, t^j]$  we have  $x(t) = x^{j-1} + \Delta t d^{j-1}$ , where  $\Delta t = t^j - t^{j-1}$  and

$$d_i^{j-1} = \begin{cases} -g_i, & t^{j-1} < t^i, \\ 0, & t^{j-1} \geq t^i, \end{cases} \quad i = 1, \dots, n. \quad (12.33)$$

With this notation, on the line segment  $[x(t^{j-1}), x(t^j)]$ , the quadratic (12.15) can be written as

$$\begin{aligned} m(x) &= f + g^T(x - x^0) + \frac{1}{2}(x - x^0)^T B(x - x^0) \\ &= f + g^T(z^{j-1} + \Delta t d^{j-1}) + \frac{1}{2}(z^{j-1} + \Delta t d^{j-1})^T B(z^{j-1} + \Delta t d^{j-1}), \end{aligned} \quad (12.34)$$

where

$$z^{j-1} = x^{j-1} - x^0. \quad (12.35)$$

Therefore, on the line segment,  $[x(t^{j-1}), x(t^j)]$   $m(x)$  can be written as a quadratic in  $\Delta t$

$$\begin{aligned} \hat{m}(\Delta t) &= \left( f + g^T z^{j-1} + \frac{1}{2}(z^{j-1})^T B z^{j-1} \right) + \left( g^T d^{j-1} + (d^{j-1})^T B z^{j-1} \right) \Delta t \\ &\quad + \frac{1}{2} \left( (d^{j-1})^T B d^{j-1} \right) \Delta t^2 \\ &= f_{j-1} + f'_{j-1} \Delta t + \frac{1}{2} f''_{j-1} \Delta t^2, \end{aligned} \quad (12.36)$$

where

$$f_{j-1} = f + g^T z^{j-1} + \frac{1}{2}(z^{j-1})^T B z^{j-1}, \quad (12.37)$$

$$f'_{j-1} = g^T d^{j-1} + (d^{j-1})^T B z^{j-1}, \quad (12.38)$$

$$f''_{j-1} = (d^{j-1})^T B d^{j-1}. \quad (12.39)$$

From the equation  $d\hat{m}(\Delta t)/d\Delta t = 0$ , we get  $\Delta t^* = -f'_{j-1}/f''_{j-1}$ . Since  $B$  is positive definite, it follows that this represents a minimizer, provided that  $t^{j-1} + \Delta t^*$  lies on  $[t^{j-1}, t^j]$ . Otherwise, the generalized Cauchy point lies at  $x(t^{j-1})$  if  $f'_{j-1} \geq 0$ , or beyond it, or even at  $x(t^j)$  if  $f'_{j-1} < 0$ .

If, after exploring the interval  $[t^{j-1}, t^j]$ , the generalized Cauchy point has not been found, we set

$$x^j = x^{j-1} + \Delta t^{j-1} d^{j-1}, \quad \Delta t^{j-1} = t^j - t^{j-1} \quad (12.40)$$

and update the directional derivatives  $f'_j$  and  $f''_j$  as the search moves to the next interval. As in Byrd, Lu, Nocedal, and Zhu (1995b), let us assume that only one variable becomes active at  $t^j$ , and let us denote its index by  $b$ . Then,  $t_b = t^j$  and we zero out the corresponding component of the search direction as

$$d^j = d^{j-1} + g_b e_b, \quad (12.41)$$

where  $e_b$  is the  $b$ -th unit vector. From (12.34) and (12.40), it follows that

$$z^j = z^{j-1} + \Delta t^{j-1} d^{j-1}. \quad (12.42)$$

Therefore, using (12.38), (12.39), (12.41), and (12.42), we get

$$\begin{aligned} f'_j &= g^T d^j + (d^j)^T B z^j \\ &= g^T d^{j-1} + g_b^2 + (d^{j-1})^T B z^{j-1} + \Delta t^{j-1} (d^{j-1})^T B d^{j-1} + g_b e_b^T B z^j \\ &= f'_{j-1} + \Delta t^{j-1} f''_{j-1} + g_b^2 + g_b e_b^T B z^j \end{aligned} \quad (12.43)$$

and

$$\begin{aligned} f''_j &= (d^j)^T B d^j \\ &= (d^{j-1})^T B d^{j-1} + 2g_b e_b^T B d^{j-1} + g_b^2 e_b^T B e_b \\ &= f''_{j-1} + 2g_b e_b^T B d^{j-1} + g_b^2 e_b^T B e_b. \end{aligned} \quad (12.44)$$

The only expensive computations in (12.43) and (12.44) are  $e_b^T B z^j$ ,  $e_b^T B d^{j-1}$ , and  $e_b^T B e_b$ , which may require  $O(n)$  operations since  $B$  is a dense limited-memory matrix. Therefore, it seems that the computation of the generalized Cauchy point could require  $O(n^2)$  operations, since, in the worst case,  $n$  segments of the piecewise linear path can be examined. For large-scale problems, this cost would be prohibitive. However, by using the limited-memory BFGS formulae (12.22) and (12.33), the updating formulae (12.43) and (12.44) become

$$f'_j = f'_{j-1} + \Delta t^{j-1} f''_{j-1} + g_b^2 + \theta g_b z_b^j - g_b w_b^T M W^T z^j, \quad (12.45)$$

$$f''_j = f''_{j-1} - \theta g_b^2 - 2g_b w_b^T M W^T d^{j-1} - g_b^2 w_b^T M w_b, \quad (12.46)$$

where  $w_b^T$  stands for the  $b$ -th row of the matrix  $W$ . The only  $O(n)$  operations remaining in (12.45) and (12.46) are  $W^T z^j$  and  $W^T d^{j-1}$ . However, from (12.41) and (12.42), it follows that  $z^j$  and  $d^j$  are updated at every iteration by a simple computation. Therefore, if we store the two  $2m$ -dimensional vectors

$$p^j \triangleq W^T d^j = W^T (d^{j-1} + g_b e_b) = p^{j-1} + g_b w_b, \quad (12.47)$$

$$c^j \triangleq W^T z^j = W^T (z^{j-1} + \Delta t^{j-1} d^{j-1}) = c^{j-1} + \Delta t^{j-1} p^{j-1}, \quad (12.48)$$

then, by means of the expressions

$$f'_j = f'_{j-1} + \Delta t^{j-1} f''_{j-1} + g_b^2 + \theta g_b z_b^j - g_b w_b^T M c^j, \quad (12.49)$$

$$f''_j = f''_{j-1} - \theta g_b^2 - 2g_b w_b^T M p^{j-1} - g_b^2 w_b^T M w_b, \quad (12.50)$$

the updating of  $f'_j$  and  $f''_j$  will require only  $O(m^2)$  operations. If more than one variable becomes active at  $t^j$ , then the above updating process is repeated before examining the new interval  $[t^j, t^{j+1}]$ .

Observe that, during the computation of the generalized Cauchy point, the examination of the first segment of the projected steepest descent path requires  $O(n)$  operations. All the subsequent segments require only  $O(m^2)$  operations, where  $m$  is the number of correction pairs stored in the limited

memory matrix. Since  $m$  is usually small, let us say less than 10, the cost of examining all the segments after the first segment is negligible. Besides, observe that it is not necessary to keep track of  $z^j \in \mathbb{R}^n$ , since only the component  $z_b^j$  corresponding to the bound that has become active is needed to update  $f'_j$  and  $f''_j$ .

**Algorithm 12.3** *Computation of the generalized Cauchy point*

1.	<i>Initialization.</i> Consider: $x, l, u, g$ and $B = \theta I - WMW^T$
2.	For $i = 1, \dots, n$ compute: $t_i = \begin{cases} (x_i - u_i)/g_i, & g_i < 0, \\ (x_i - l_i)/g_i, & g_i > 0, \\ \infty, & g_i = 0, \end{cases} \quad d_i = \begin{cases} -g_i, & t_i \neq 0, \\ 0, & t_i = 0. \end{cases}$
3.	Compute: $\begin{aligned} F &= \{i : t_i > 0\}, \quad p = W^T d, \quad c = 0, \\ f' &= g^T d = -d^T d, \quad f'' = \theta d^T d - d^T WMW^T d = -\theta f' - p^T M p, \\ \Delta t_{\min} &= -f'/f'', \quad t_{old} = 0, \quad t = \min \{t_i : i \in F\}, \\ b &= i \text{ such that } t_i = t \text{ (remove } b \text{ from } F\text{), } \Delta t = t \end{aligned}$
4.	Examination of the subsequent segments: While $\Delta t_{\min} \geq \Delta t$ do: $\begin{aligned} x_b^{cp} &= \begin{cases} u_b, & d_b > 0, \\ l_b, & d_b < 0, \end{cases} \quad z_b = x_b^{cp} - x_b, \quad c = c + \Delta t p, \\ f' &= f' + \Delta t f'' + g_b^2 + \theta g_b z_b - g_b w_b^T M c, \\ f'' &= f'' - \theta g_b^2 - 2g_b w_b^T M p - g_b^2 w_b^T M w_b, \\ p &= p + g_b w_b, \quad d_b = 0, \quad \Delta t_{\min} = -f'/f'', \\ t_{old} &= t, \quad t = \min \{t_i : i \in F\}, \\ b &= i \text{ such that } t_i = t \text{ (remove } b \text{ from } F\text{), } \Delta t = t - t_{old} \end{aligned}$ end while
5.	Compute: $\begin{aligned} \Delta t_{\min} &= \max \{\Delta t_{\min}, 0\}, \quad t_{old} = t_{old} + \Delta t_{\min}, \\ x_i^{cp} &= x_i + t_{old} d_i, \text{ for all } i \text{ such that } t_i \geq t, \\ \text{for all } i \in F \text{ with } t_i = t, \text{ remove } i \text{ from } F, \\ c &= c + \Delta t_{\min} p \end{aligned} \quad \blacklozenge$

In the last step of Algorithm 12.3, the  $2m$ -vector  $c$  is updated so that in the end,  $c = W^T(x^c - x_k)$ . This vector will be used to initialize the subspace minimization when the primal direct method or the conjugate gradient method is used.

Once the generalized Cauchy point  $x^c$  has been computed, an approximate minimum of the quadratic model  $m_k$  over the space of the free variable is determined. In Byrd, Lu, Nocedal, and Zhu (1994a, 1995b), three approaches to minimize the model are developed: a *direct primal method based on the Sherman-Morrison-Woodbury formula*, a *primal iterative method using the conjugate gradient method*, and a *direct dual method using the Lagrange multipliers*. In all these approaches, the minimization of  $m_k$  is done by ignoring the bounds. After that, an appropriate point truncates the move so as to satisfy the bound constraints. In the direct primal approach,  $n - t$  variables are fixed at their bounds at the generalized Cauchy point  $x^c$ , and the quadratic model (12.17) is solved over the subspace of the remaining  $t$  free variables, starting from  $x^c$  and imposing the free variables at the corresponding bounds from (12.17).

In the algorithm L-BFGS-B, the iterations are terminated when the projected gradient is small enough, i.e.,

$$\|P_X(x_k - g_k, l, u) - x_k\|_\infty < \epsilon_g, \quad (12.51)$$

where  $\epsilon_g > 0$  is a parameter. Observe that  $P_X(x_k - g_k, l, u) - x_k$  is the projected gradient. With this, a formal description of the L-BFGS-B algorithm can be presented as follows:

**Algorithm 12.4 L-BFGS-B**

- |    |   |
|----|---|
| 1. | Consider an initial point $x_0$ , as well as an integer $m$ that determines the number of limited memory corrections stored $(s_i, y_i)$ , $i = 1, \dots, m$ . Select the values of the parameters $\epsilon > 0$ and $\epsilon_g > 0$  |
| 2. | If the convergence test (12.51) is satisfied, stop  |
| 3. | Using Algorithm 12.3, compute the generalized Cauchy point  |
| 4. | Compute the search direction $d_k$ by the direct primal method, the conjugate gradient method or by the dual method   |
| 5. | Using the strong Wolfe line-search (12.18) and (12.19), perform a line-search along $d_k$ subject to the bounds on the problem in order to compute a stepsize $\alpha_k$  |
| 6. | Set $x_{k+1} = x_k + \alpha_k d_k$ . Compute $f(x_{k+1})$ and $\nabla f(x_{k+1})$   |
| 7. | If $y_k$ satisfies the curvature condition $y_k^T s_k > \epsilon \ y_k\ ^2$ , then the correction pair $(s_k, y_k)$ is added to the matrices $S_k$ and $Y_k$ . If more than $m$ updates are stored, delete the oldest columns from $S_k$ and $Y_k$ and place the new ones instead |
| 8. | Update: $S_k^T S_k$ , $Y_k^T Y_k$ , $L_k$ , $R_k$ and set $\theta = y_k^T y_k / y_k^T s_k$  |
| 9. | Set $k = k + 1$ and continue with step 2  |

◆

More details on the L-BFGS-B algorithm are given in Byrd, Lu, Nocedal, and Zhu (1994a, 1995b). In the following, we present a numerical study and comparisons versus the SPG algorithm.

### Numerical Study: L-BFGS-B Versus SPG

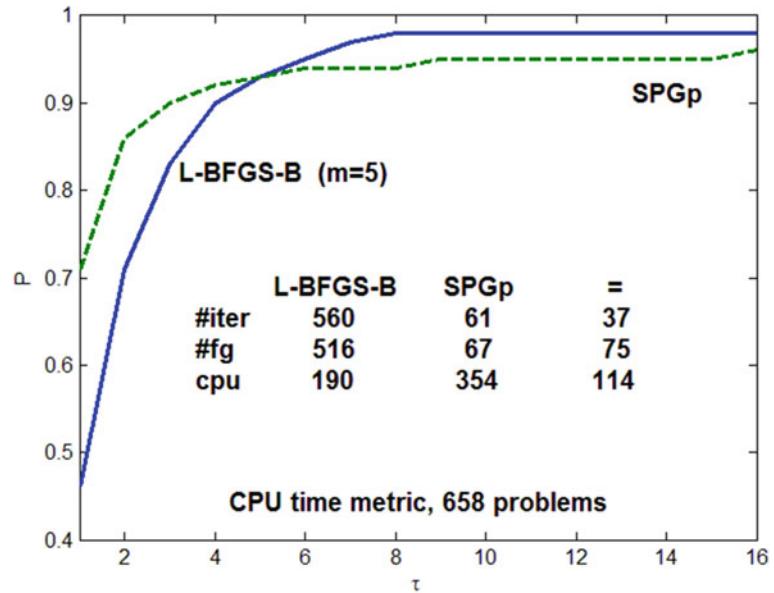
Consider the same set of 800 simple bound optimization problems from the UOP collection used in the previous numerical experiment on the SPG algorithm with quadratic or cubic interpolation. L-BFGS-B implements two criteria for stopping the iterations

$$\frac{f_k - f_{k+1}}{\max \{f_k, f_{k+1}, 1\}} \leq \tau \epsilon_m \text{ or } \|P_X(x_k - g_k, l, u) - x_k\|_\infty < \epsilon_g,$$

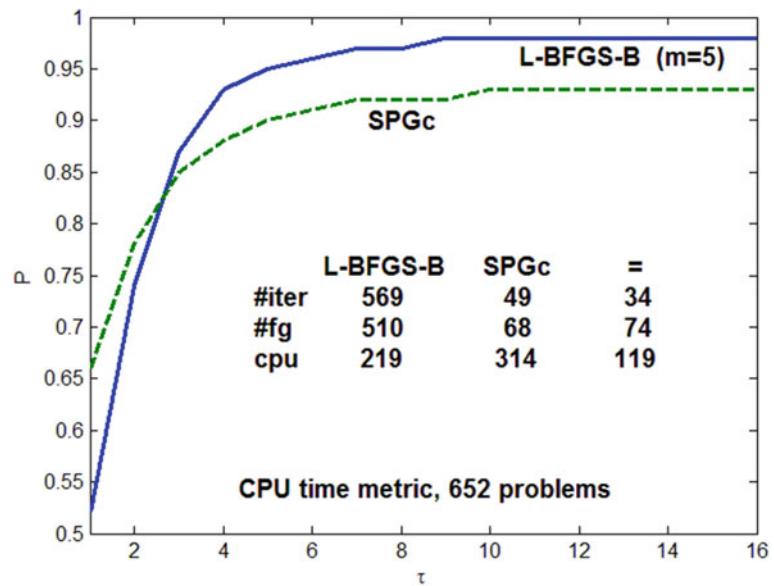
where  $\tau$  is a factor controlling the accuracy of the solution,  $\epsilon_m$  is the precision of the machine, and  $\epsilon_g$  is the tolerance on the projected gradient. If one of these two criteria is satisfied, then the iterations are stopped. The numerical values of the parameter  $\tau$  are as follows:  $\tau = 1.d + 12$  for small accuracy,  $\tau = 1.d + 7$  for medium accuracy, and  $\tau = 1.d + 1$  for high accuracy. In this numerical study, we have considered  $\tau = 1.d + 7$  and  $\epsilon_g = 1.d - 5$ . The number of correction pairs stored in computing BFGS with limited memory is  $m = 5$ .

Figures 12.2 and 12.3 show the performance profiles of L-BFGS-B ( $m = 5$ ) versus SPG with quadratic interpolation (SPGp) and versus SPG with cubic interpolation (SPGc), respectively. In Figs. 12.2 and 12.3, we have the computational evidence that both SPGp and SPGc are more efficient than L-BFGS-B, but subject to robustness, we can see that L-BFGS-B is more robust than both SPGp and SPGc. The algorithm SPG is more simple than L-BFGS-B, at each iteration requiring a projection, the computation of the spectral gradient, as well as a nonmonotone linear search. On the other hand, L-BFGS-B is more complicated, involving the computation of the generalized Cauchy

**Fig. 12.2** L-BFGS-B versus SPG with quadratic interpolation (SPGp)



**Fig. 12.3** L-BFGS-B versus SPG with cubic interpolation (SPGc)



point, the determination of the search direction, a linear search by strong Wolfe conditions (12.18) and (12.19), as well as the limited-memory BFGS update.

## 12.6 Truncated Newton with Simple Bounds (TNBC)

This method, called TNBC, elaborated by Nash (1984a, b, 2000) is especially developed to solve simple bound constrained optimization problems. The method is based on a line-search with a classical active-set strategy for treating the bounds. TNBC is a truncated Newton method (Dembo

& Steihaug, 1983) that uses the conjugate gradient projection method to obtain a new search direction. However, it computes only an approximation to the Newton direction because it is truncated before the solution to the subspace minimization problem is obtained. In other words, in the truncated Newton method, the current estimate of the solution is updated (i.e., a step is computed) by the approximate solving of the Newton system by making use of an iterative algorithm. Therefore, the algorithm implements a double iterative method: an *outer iteration* for the nonlinear optimization problem and an *inner iteration* for the Newton system. The inner iteration is stopped or “*truncated*” before the solution to the Newton system is obtained. The motivation of this approach is that far away from the solution; the Newton system does not need to be solved very accurately. Therefore, the Newton iterations can be truncated. The search direction  $d_k$  is determined such that

$$\|\nabla^2 f(x_k)d_k + \nabla f(x_k)\| \leq \eta_k \|\nabla f(x_k)\|,$$

where  $\{\eta_k\}$  is known as the “forcing sequence.” The Hessian matrix with the second-order information is not given. Therefore, the Hessian vector product  $B_k v$  for a given vector  $v$  required by the inner conjugate gradient algorithm is obtained by finite differencing as

$$\nabla^2 f(x_k)v \cong \frac{\nabla f(x_k + hv) - \nabla f(x_k)}{h},$$

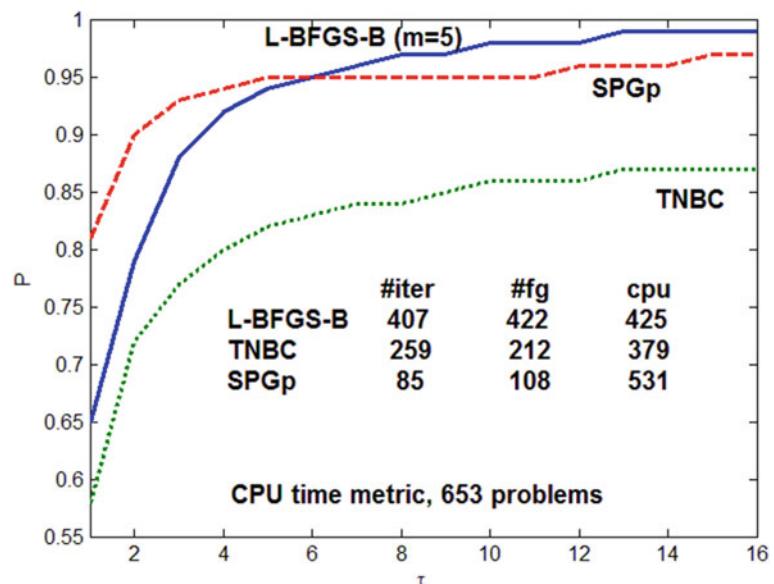
where  $h = (1 + \|x_k\|_2)\sqrt{\epsilon}$  and  $\epsilon$  is the relative machine precision. Each matrix vector product requires one gradient evaluation. The truncated Newton algorithm differs from the standard Newton approach, mainly in its use of the parameter  $\eta_k$  which determines the accuracy with which the Newton system  $\nabla^2 f(x_k)d_k = -\nabla f(x_k)$  is solved in order to obtain a search direction.

The conjugate gradient inner algorithm is preconditioned by a scaled two-step limited-memory BFGS method with Powell’s restarting strategy used to reset the preconditioner periodically (Nash, 1985).

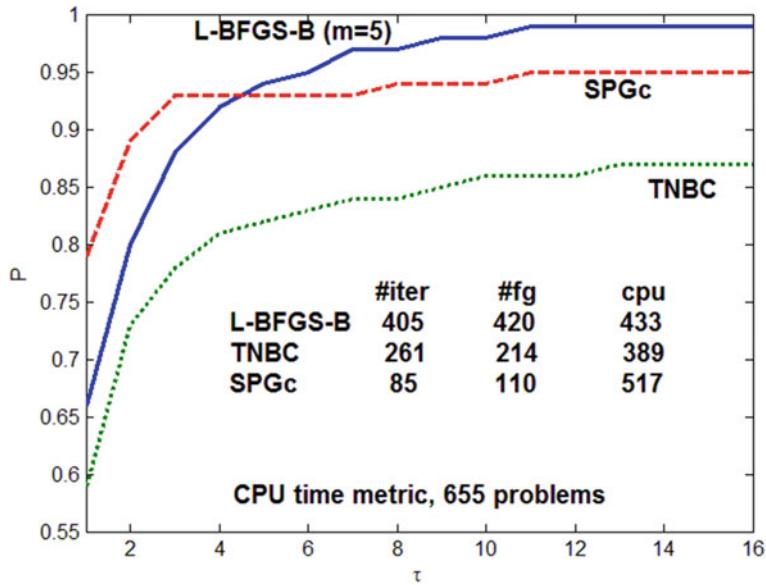
The line-search is based on the cubic interpolation and is terminated when the strong Wolfe conditions are satisfied, as described in Gill and Murray (1979).

Consider the same set of 800 simple bound optimization problems used in the previous numerical experiments, where, this time,  $n = 100, 200, \dots, 1000$ . Figures 12.4 and 12.5 present the

**Fig. 12.4** TNBC versus L-BFGS-B ( $m = 5$ ) and versus SPGp



**Fig. 12.5** TNBC versus L-BFGS-B ( $m = 5$ ) and versus SPGc



performances of the truncated Newton method with simple bounds in the implementation of Nash (1984b) versus L-BFGS-B ( $m = 5$ ) and SPGp and SPGc, respectively.

From Figs. 12.4 and 12.5, we can see that L-BFGS-B is more robust than TNBC, SPGp, and SPGc. Both SPGp and SPGc are more efficient than L-BFGS-B and TNBC. From Fig. 12.4, we can see that SPGp is faster in solving 531 problems. In contrast, L-BFGS-B is faster in solving 425 problems, while TNBC is faster in solving 379, etc.

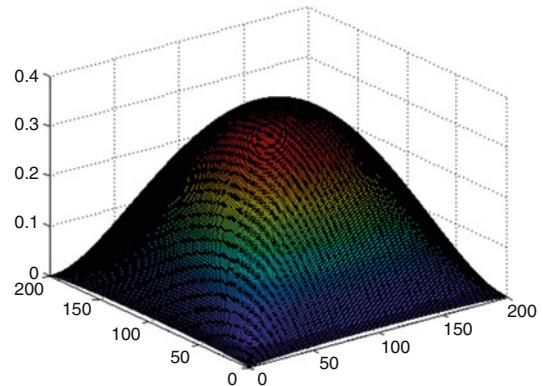
## 12.7 Applications

This section presents the numerical performances of the simple bounds optimization algorithms SPG and L-BFGS-B for solving five real industrial applications taken from the MINPACK-2 test problem collection (Averick, Carter, & Moré, 1991; Averick, Carter, Moré, & Xue, 1992). The mathematical models of these applications are expressed as unconstrained optimization problems and are described in Appendix D. All these applications *without simple bounds* have been solved with the SCALCG algorithm, which implements a double BFGS preconditioned conjugate gradient (Andrei 2008a, 2010b). In these numerical experiments, for all the applications, some simple bounds on the variables were introduced and solved with SPG and L-BFGS-B, respectively.

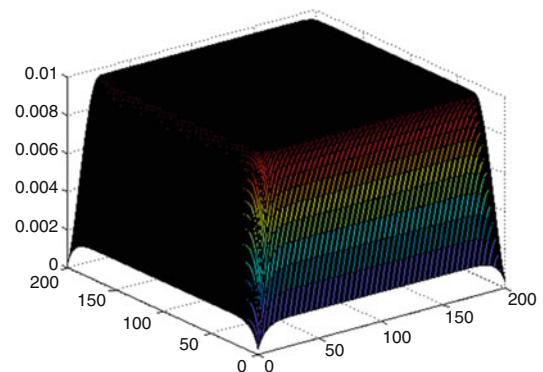
### Application A1 (Elastic-Plastic Torsion)

Considering  $D = (0, 1) \times (0, 1)$ ,  $c = 5$ , and  $nx = 200$ ,  $ny = 200$ , a minimization problem with 40,000 variables is obtained. The solution of this application without simple bounds is illustrated in Fig. 12.6. By introducing the simple bounds  $0 \leq v \leq 0.01$ , the algorithm SPG gives the solution presented in Fig. 12.7.

**Fig. 12.6** Solution of the application A1 without simple bounds.  $nx = 200$ ,  $ny = 200$



**Fig. 12.7** Solution of the application A1 with simple bounds  $0 \leq v \leq 0.01$ .  $nx = 200$ ,  $ny = 200$



**Table 12.1** Elastic-plastic torsion. SPG. 40,000 variables.  $M = 10$

	Quadratic interpolation				Cubic interpolation			
	#iter	#f	#g	cpu	#iter	#f	#g	cpu
$0 \leq v \leq 1$	3311	5433	3312	113.25	2663	3600	2664	83.21
$0 \leq v \leq 0.1$	981	1504	982	59.49	719	940	720	40.59
$0 \leq v \leq 0.01$	152	190	153	7.63	118	132	119	5.66

Table 12.1 contains the performances of the SPG algorithm with quadratic or cubic interpolation in the line-search for different values of the simple bounds:  $0 \leq v \leq 1$ ,  $0 \leq v \leq 0.1$ , and  $0 \leq v \leq 0.01$ . In this table, #iter represents the number of iterations to get the solution; #f and #g are the number of evaluations of the function and the number of evaluations of its gradient, respectively. cpu is the CPU running time in seconds. The number of values of the objective functions retained in the nonmonotone linear search is  $M = 10$ . Observe that the cubic interpolation is slightly more benefic versus the quadratic interpolation.

Table 12.2 presents the performances of the algorithm L-BFGS-B for solving this application. In this table, #sg represents the total number of segments explored during the search of the generalized Cauchy point, while #act is the number of the active constraints in the final generalized Cauchy point. Here,  $m = 5$  is the number of the pairs  $\{s_i, y_i\}$ ,  $i = 1, \dots, m$ , stored for the memoryless BFGS updating.

**Table 12.2** Elastic-plastic torsion. L-BFGS-B. 40,000 variables.  $m = 5$ 

	#iter	#f	#sg	#act	cpu
$0 \leq v \leq 1$	173	184	177	0	1.89
$0 \leq v \leq 0.1$	117	122	1774	13036	1.082
$0 \leq v \leq 0.01$	41	45	62	30920	2.74

**Table 12.3** Elastic-plastic torsion. TNBC. 40,000 variables

	#iter	#fg	#fgt	cpu
$0 \leq v \leq 1$	34	77	733	21.68
$0 \leq v \leq 0.4$	2194	2207	13176	201.24
$0 \leq v \leq 0.3$	6459	6477	30858	550.85

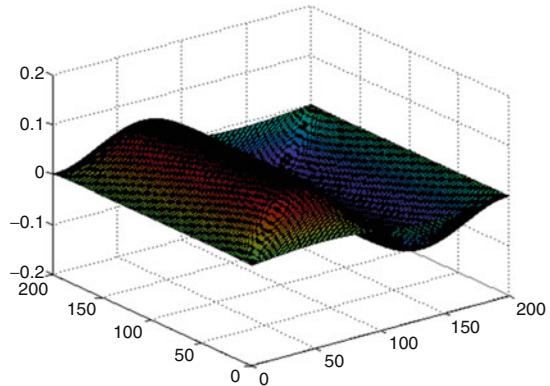
**Fig. 12.8** Solution of application A2 without simple bounds.  $nx = 200$ ,  $ny = 200$ 

Table 12.3 shows the performances of TNBC for solving this application, where  $\#fg$  is the number of callings of the subroutine for the evaluation of the minimizing function and its gradient and  $\#fgt$  is the number of function and its gradient evaluations, including the evaluations in the conjugate gradient for solving the Newton system for the line-search.

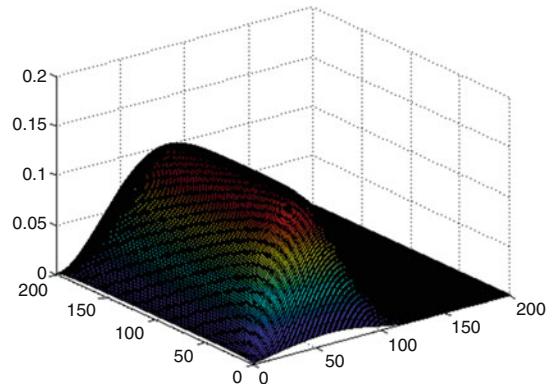
Observe that TNBC is very sensitive to the value of the bounds on variables. The closer the bounds are, the more degraded the performances are.

### Application A2 (Pressure Distribution in a Journal Bearing)

Considering  $b = 10$  and  $\epsilon = 0.1$ , as well as a discretization  $n_x \times n_y$  of the domain  $D = (0, 2\pi) \times (0, 2b)$ , where  $nx = 200$  and  $ny = 200$ , then an optimization problem with 40,000 variables is obtained. The solution of this application without simple bounds is represented in Fig. 12.8. When introducing the simple bounds as  $0 \leq v \leq 1$ , the SPG algorithm gives the solution from Fig. 12.9.

Tables 12.4 and 12.5 present the performances of SPG and L-BFGS-B for different simple bounds, respectively.

**Fig. 12.9** Solution of application A2 with simple bound  
 $0 \leq v \leq 1$ .  $nx = 200$ ,  $ny = 200$



**Table 12.4** Pressure distribution in a journal bearing. SPG. 40,000 variables.  $M = 10$

	Quadratic interpolation				Cubic interpolation			
	#iter	#f	#g	cpu	#iter	#f	#g	cpu
$-1 \leq v \leq 1$	4179	7071	4180	140.73	2663	3553	2664	83.96
$0 \leq v \leq 1$	4964	7796	4965	152.02	4808	6301	4809	133.78
$-1 \leq v \leq 0$	3881	6179	3882	116.35	3154	4170	3155	86.99
$-0.5 \leq v \leq 0.5$	4054	6820	4055	143.09	2969	4105	2970	94.84

**Table 12.5** Pressure distribution in a journal bearing. L-BFGS-B. 40,000 variables.  $m = 5$

	#iter	#f	#sg	#act	cpu
$-1 \leq v \leq 1$	623	648	623	0	7.05
$0 \leq v \leq 1$	2419	2429	25995	12862	2.62
$-1 \leq v \leq 0$	377	391	796	12904	3.84

**Table 12.6** Optimal design with composite materials. SPG. 40,000 variables.  $M = 10$

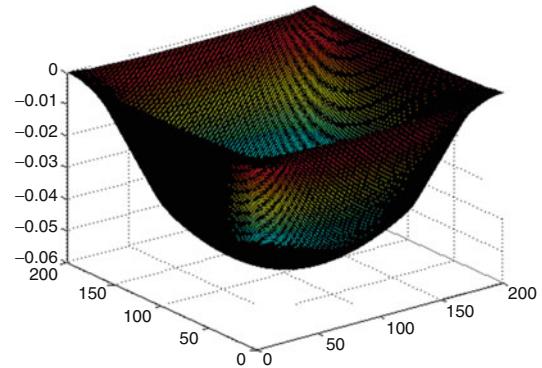
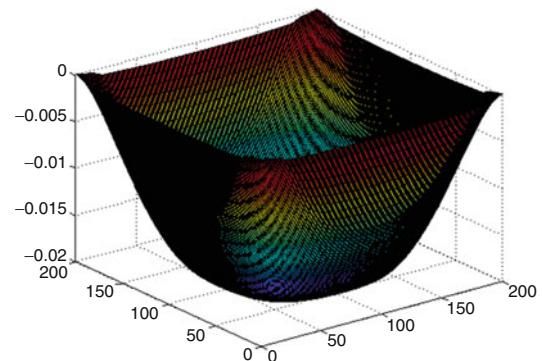
	Quadratic interpolation				Cubic interpolation			
	#iter	#f	#g	cpu	#iter	#f	#g	cpu
$-1 \leq v \leq 0$	22515	34686	22516	1259.82	20969	27964	20970	1083.20
$-0.02 \leq v \leq 0$	1855	2769	1856	86.21	2228	2890	2229	95.74

### Application A3 (Optimal Design with Composite Materials)

Considering  $\mu_1 = 1$  and  $\mu_2 = 2$ , then Tables 12.6 and 12.7 show the performances of SPG and L-BFGS-B, respectively. Figure 12.10 presents the solution of this application obtained with SPG for  $nx = 200$  and  $ny = 200$ . Figure 12.11 shows the solution of this application given by L-BFGS-B with the simple bounds  $-0.02 \leq v \leq 0$ .

**Table 12.7** Optimal design with composite materials. L-BFGS-B. 40,000 variables.  $m = 5$ 

	#iter	#f	#sg	#act	cpu
$-1 \leq v \leq 0$	646	649	744	0	7.94
$-0.02 \leq v \leq 0$	124	127	734	7420	1.38

**Fig. 12.10** Solution of application A3 without simple bounds.  $nx = 200$ ,  $ny = 200$ **Fig. 12.11** Solution of application A3 with simple bounds  $-0.02 \leq v \leq 0$ .  $nx = 200$ ,  $ny = 200$ 

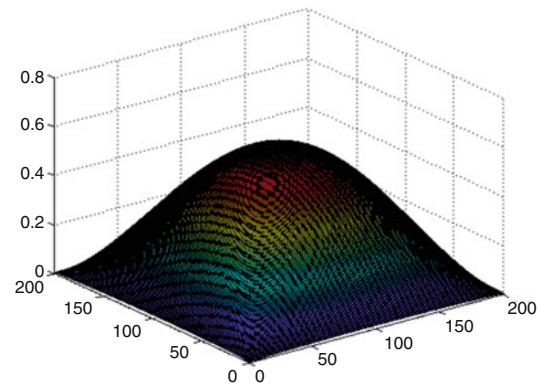
### Application A4 (Steady-State Combustion)

Considering  $\lambda = 5$ , Fig. 12.12 shows the solution of this application where  $nx = 200$  and  $ny = 200$ . Figure 12.13 presents the solution of the application with the simple bounds  $0 \leq v \leq 0.2$ , where  $\lambda = 5$ ,  $nx = 200$ , and  $ny = 200$ .

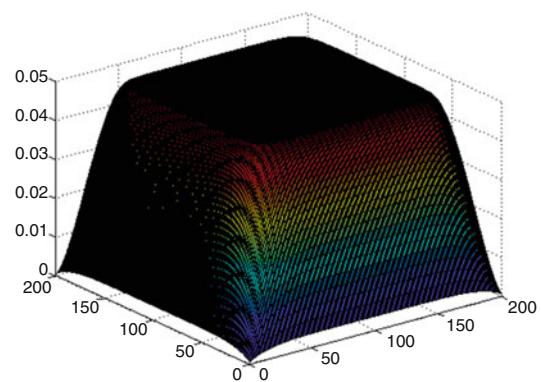
Tables 12.8 and 12.9 give the performances of SPG and of L-BFGS-B for solving this application with different simple bounds on variables.

From these tables, we note that, subject to the CPU computing time, L-BFGS-B is the best algorithm, followed by SPG with cubic interpolation and then by SPG with quadratic interpolation.

**Fig. 12.12** Solution of application A4 without simple bounds.  $nx = 200$ ,  $ny = 200$



**Fig. 12.13** Solution of application A4 with simple bounds  $0 \leq v \leq 0.2$ .  $nx = 200$ ,  $ny = 200$



**Table 12.8** Steady-state combustion. SPG. 40,000 variables.  $M = 10$

	Quadratic interpolation				Cubic interpolation			
	#iter	#f	#g	cpu	#iter	#f	#g	cpu
$0 \leq v \leq 1$	5134	8418	5135	421.92	5352	7236	5353	395.13
$0 \leq v \leq 0.2$	1843	2795	1844	137.32	1764	2286	1765	121.01
$0 \leq v \leq 0.1$	1140	1724	1141	81.52	665	854	666	43.45

**Table 12.9** Steady-state combustion. L-BFGS-B. 40,000 variables.  $m = 5$

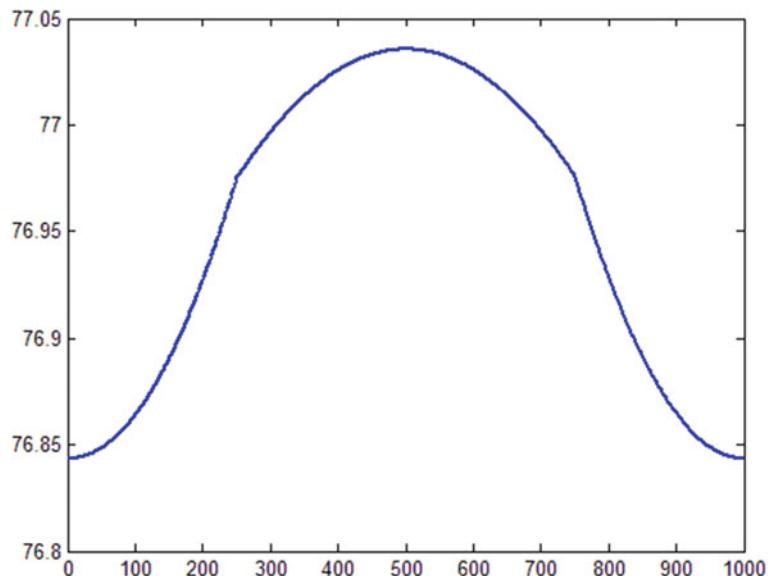
	#iter	#f	#sg	#act	cpu
$0 \leq v \leq 1$	360	372	424	0	5.48
$0 \leq v \leq 0.2$	185	196	1012	7960	2.76
$0 \leq v \leq 0.1$	128	130	909	14796	1.83

### Application A6 (Inhomogeneous Superconductors: 1-D Ginzburg-Landau)

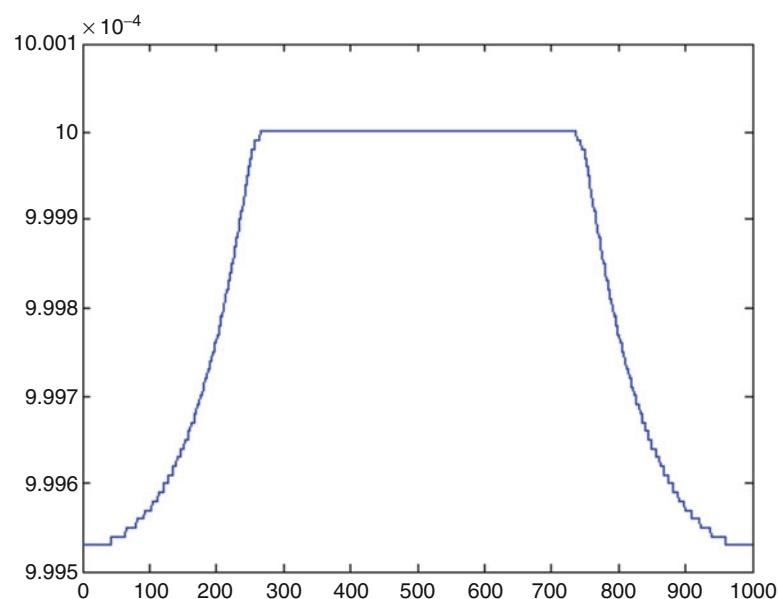
Considering  $d = 3.2 \text{ \AA}$  and the temperature  $T = 5$ , for  $n = 1000$ , the solution of this application without simple bounds is presented in Fig. 12.14. When introducing the simple bounds on the variables  $0 \leq v \leq 0.001$ , the corresponding solution given by SPG is given in Fig. 12.15.

Tables 12.10 and 12.11 present the performances of SPG and L-BFGS-B for solving this application with different simple bounds.

**Fig. 12.14** Solution of application A6 without simple bounds.  $n = 1000$



**Fig. 12.15** Solution of application A6 with simple bounds  $0 \leq v \leq 0.001$ .  $n = 1000$



**Table 12.10** 1-D Ginzburg-Landau problem. SPG. 1000 variables.  $M = 10$ 

	Quadratic interpolation				Cubic interpolation			
	#iter	#f	#g	cpu	#iter	#f	#g	cpu
$0 \leq v \leq 10^8$	3465	6001	3466	0.81	4373	6001	4373	0.87
$0 \leq v \leq 10$	3466	6001	3468	1.92	4374	6001	4375	2.20

**Table 12.11** 1-D Ginzburg-Landau problem. L-BFGS-B. 1000 variables.  $m = 5$ 

	#iter	#f	#sg	#act	cpu
$0 \leq v \leq 10^8$	34895	36076	34895	0	6.30
$0 \leq v \leq 10$	10278	10643	11438	391	1.74

Real applications of nonlinear optimization always have simple bounds on variables. In other words, for a problem with  $n$  variable, every variable  $x_j$  has a finite lower bound  $l_j$  and a finite upper bound  $u_j$ , where  $l_j \leq u_j$ ,  $j = 1, \dots, n$ . These bounds are imposed by the engineering constructive relationships of the process modeled as an optimization problem. Therefore, the simple bound optimization is an important problem with numerous practical applications.

Both SPG and L-BFGS-B are able to solve simple bound optimization problems, L-BFGS-B being slightly faster. TNBC is very sensitive to the interval in which the bounds on variables are defined.

## Notes and References

Recent advances in the simple bound constrained optimization are presented in Hager and Zhang (2006b) and Tröltzsch (2007). To guarantee the global convergence to a local solution, the algorithms for solving this class of problems use either a line-search or a trust-region-oriented framework. The algorithmic components of the line-search methods are the subspace minimization, the inexact line-search, the active-set method, the gradient projection method, the projected Newton method, the gradient projection-reduced Newton method, and the gradient projection-conjugate gradient method. On the other hand, the algorithmic components of the trust-region methods are the generalized Cauchy point and the trust-region method. For the simple bound constrained optimization, there are some specialized solvers (software). LANCELOT/SBMIN (Conn, Gould, & Toint, 1992b) and TRON (Lin & Moré, 1999) use the trust-region method. Both of them use a conjugate gradient iteration to perform the subspace minimization and apply an incomplete Cholesky preconditioner.

Other solvers like DOT (Vanderplaats, 1995), L-BFGS-B, TNBC, and IPOPT (Wächter & Biegler, 2006) use line-search methods. IPOPT uses a primal-dual interior point filter line-search method (see Chap. 19). Another approach completely different from the above categories of solvers is based on the penalty functions method of Facchinei, Lucidi, and Palagi (2002).

Our presentation follows the papers of Kelley (1999); Byrd, Lu, Nocedal, and Zhu (1995b) (L-BFGS-B); Birgin, Martínez, and Raydan (2000, 2001) (SPG); and Nash (1984a, b, 1985) (TNBC).



# Quadratic Programming

13

One of the most important nonlinear optimization problems is the quadratic programming, in which a quadratic objective function is minimized with respect to linear equality and inequality constraints. These problems are present in many methods as subproblems and in real applications from different areas of activity as mathematical models of these applications. In the beginning, we consider the equality constrained quadratic programming, after which the inequality constrained quadratic programming is presented. Finally, methods based on the elimination of variables are discussed.

## 13.1 Equality Constrained Quadratic Programming

The expression of an *equality constrained quadratic program* is as follows:

$$\begin{aligned} \min q(x) &\equiv \frac{1}{2}x^T G x + c^T x \\ \text{subject to} \\ a_i^T x + b_i &= 0, \quad i \in E \triangleq \{1, \dots, m\}, \end{aligned} \tag{13.1}$$

where  $G$  is an  $n \times n$  symmetric matrix and  $a_i, i \in E$ , are vectors in  $\mathbb{R}^n$ . All the vectors  $a_i, i \in E$ , can be assembled into the matrix  $A$  such that the constraints from (13.1) can be compactly written as  $Ax + b = 0$ , where  $A$  is an  $m \times n$  matrix. Suppose that  $m \leq n$ . If the Hessian matrix  $G$  is positive semidefinite, we say that (13.1) is a *convex quadratic program*. In this case, the problem is similar in difficulty to a linear program. *Strictly convex quadratic programs* are those in which  $G$  is positive definite. *Nonconvex quadratic programs* have  $G$  an indefinite matrix.

The first-order optimality conditions for (13.1) are

$$Gx^* + c - A^T \lambda^* = 0, \tag{13.2a}$$

$$Ax^* + b = 0, \tag{13.2b}$$

where  $x^*$  is the solution and  $\lambda^*$  is the vector of the Lagrange multipliers. Rearranging (13.2), these equations become a system of  $n + m$  linear equations known as the *KKT system*

$$\begin{bmatrix} G & -A^T \\ -A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}. \quad (13.3)$$

In the following, we consider  $Z$  to denote the  $n \times (n - m)$  matrix whose columns is a basis for the null space of  $A$ . That is,  $Z$  has full rank and satisfies  $AZ = 0$ .

### Factorization of the Full KKT System

For solving (13.1), the direct way, known as the factorization of the full KKT system, is to form the linear system (13.3) and to find  $x^*$  and  $\lambda^*$  by solving it, let us say by the Gaussian elimination. Observe that although the matrix from (13.3) is symmetric, the zeros on the main diagonal of this matrix imply that it is not positive definite, that is, the Cholesky factorization is not suitable. If  $G$  is positive definite, then the feasible stationary point obtained from (13.3) is the minimum of (13.1). Otherwise, to confirm that  $x^*$  is the minimum point, the second-order condition

$$z^T G z > 0 \text{ for all the vectors } z, \text{ such that } Az = 0$$

must be checked.

The system (13.3) can be rearranged in a very suitable form for computation by expressing  $x^*$  as  $x^* = x + d$ , where  $x$  is some estimation of the solution and  $d$  is the desired step. Using this notation, from the system (13.3), we obtain

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} -d \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ h \end{bmatrix}, \quad (13.4)$$

where  $g = c + Gx$  and  $h = Ax - b$ . The matrix in (13.4) is called the *KKT matrix*.

**Theorem 13.1** *Let  $A$  be of full rank and assume that  $Z^T G Z$  is positive definite. Then, the KKT matrix*

$$K = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \quad (13.5)$$

*is nonsingular, and therefore, there is a unique vector pair  $(x^*, \lambda^*)$  satisfying (13.3).*

**Proof** Suppose that there are the vectors  $w$  and  $v$ , such that

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = 0. \quad (13.6)$$

Since  $Aw = 0$ , from (13.6), it follows that

$$0 = [w^T \ v^T] \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = w^T G w.$$

However,  $w$  lies in the null space of  $A$ . Therefore, there is a vector  $u \in \mathbb{R}^{n-m}$ , such that  $w = Zu$ . With this, we have

$$0 = w^T Gw = u^T Z^T G Z u,$$

which by the positive definiteness of  $Z^T G Z$  implies  $u = 0$ . Therefore,  $w = 0$ , and by (13.6),  $A^T v = 0$ . Now, suppose that  $A$  is of full rank on rows. Therefore,  $v = 0$ . The conclusion is that the equation (13.6) is satisfied only if  $w = 0$  and  $v = 0$ , i.e., the matrix  $K$  is nonsingular.  $\blacklozenge$

**Theorem 13.2** *Let  $A$  be of full rank and assume that  $Z^T G Z$  is positive definite. Then,  $x^*$  satisfying (13.3) is the unique global solution of (13.1).*

**Proof** Let  $x$  be any other feasible point satisfying  $Ax = -b$ . Let  $d = x^* - x$ . Since  $Ax^* = -b$ , we have  $Ad = 0$ . Substituting  $x = x^* - d$  in  $q(x)$  from (13.1), we get

$$\begin{aligned} q(x) &= \frac{1}{2}(x^* - d)^T G(x^* - d) + c^T(x^* - d) \\ &= \frac{1}{2}d^T Gd - d^T Gx^* - c^T d + q(x^*). \end{aligned}$$

However, from (13.3), we have  $Gx^* = -c + A^T \lambda^*$ . Therefore, from  $Ad = 0$ , we have

$$d^T Gx^* = d^T(-c + A^T \lambda^*) = -d^T c.$$

With this,

$$q(x) = \frac{1}{2}d^T Gd + q(x^*).$$

Since  $d$  lies in the null space of  $A$ , there is a vector  $u \in \mathbb{R}^{n-m}$ , such that  $d = Zu$ . Hence,

$$q(x) = \frac{1}{2}u^T Z^T G Z u + q(x^*).$$

However,  $Z^T G Z$  is positive definite, hence  $q(x) > q(x^*)$ , except for the situation in which  $u = 0$ , that is, when  $x = x^*$ . Therefore,  $x^*$  is the unique global solution of (13.1).  $\blacklozenge$

## The Schur-Complement Method

The use of (13.3) to get a solution for (13.1) may be efficient only when the matrices  $G$  and  $A$  are sparse. Otherwise, another method, known as the Schur-complement, may be used, in which  $x^*$  and  $\lambda^*$  are separately determined. For example, if we multiply (13.2a) by  $AG^{-1}$  and then use (13.2b) to eliminate  $x^*$ , we can obtain  $\lambda^*$  as solution of the following linear system:

$$(AG^{-1}A^T)\lambda^* = AG^{-1}c - b. \quad (13.7)$$

Clearly,  $x^*$  is computed from

$$Gx^* = A^T \lambda^* - c. \quad (13.8)$$

A rough analysis of the computational effort shows that solving (13.7) and (13.8) requires  $O(m^3) + O(n^3)$  multiplications. This is less than  $O((n+m)^3)$  multiplications needed to solve (13.3). However, this approach requires to perform operations with  $G^{-1}$  and to compute a factorization of the  $m \times m$  matrix  $AG^{-1}A^T$ .

The name *Schur complement* derives from the fact that, by applying the block Gaussian elimination to (13.3) using  $G$  as pivot, the following block upper triangular system

$$\begin{bmatrix} G & -A^T \\ 0 & -AG^{-1}A^T \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b - AG^{-1}c \end{bmatrix} \quad (13.9)$$

is obtained. In linear algebra terminology, the matrix  $AG^{-1}A^T$  is called the Schur complement of  $G$ .

## The Null-Space Method

This method does not require that the matrix  $G$  should be nonsingular. It assumes only that  $A$  has full row rank and that the matrix  $Z^T G Z$  is positive definite, where  $Z$  is an  $n \times (n - m)$  matrix whose columns span the null space of  $A$ . The matrix  $Z^T G Z$  is called the *reduced Hessian*. Clearly, this method requires knowledge of the null space basis matrix  $Z$ .

The method considers a solution in terms of its components in two subspaces which are *normal* and *tangential* to the constraints. As we know (see Chap. 11), the optimality conditions involve feasible directions which lie in the tangent space of the constraints. If an  $n$ -variable optimization problem involves  $m$  linear constraints,  $Ax = b$ , then a feasible direction can be any member of the  $(n - m)$ -dimensional subspace of the vectors  $z$  which satisfy  $Az = 0$ . As we said before, let  $Z$  be an  $n \times (n - m)$  matrix whose columns span the null space of  $A$ . This means that  $Zw$  is a feasible direction for any  $(n - m)$  vector  $w$ . Also, let  $Y$  be an  $n \times m$ -dimensional arbitrary matrix, such that the matrix  $[Y \ Z]$  is nonsingular. Suppose that  $d$  in (13.4) is partitioned, so that

$$d = Yd_Y + Zd_Z, \quad (13.10)$$

where  $d_Y \in \mathbb{R}^m$  and  $d_Z \in \mathbb{R}^{n-m}$ . In this context, the components  $Yd_Y$  and  $Zd_Z$  are called the *vertical step* and the *horizontal step*, respectively.

Substituting  $d$  given by (13.10) into the second equation from (13.4), since  $AZ = 0$ , we get

$$(AY)d_Y = -h. \quad (13.11)$$

However,  $A$  is a matrix of full rank  $m$  and  $[Y \ Z]$  is an  $n \times n$  nonsingular matrix. Therefore, the matrix

$$A[Y \ Z] = [AY \ 0] \quad (13.12)$$

is of rank  $m$ . Hence,  $AY$  is an  $m \times m$  nonsingular matrix.  $d_Y$  is well defined as solution of the system (13.11).

Now, substituting  $d$  given by (13.10) into the first equation from (13.4), we get

$$-GYd_Y - GZd_Z + A^T\lambda^* = g. \quad (13.13)$$

Premultiplying (13.13) by  $Z^T$ , we obtain the *reduced system*

$$(Z^T G Z)d_Z = -Z^T GYd_Y - Z^T g. \quad (13.14)$$

Since  $Z^T G Z$  is a symmetric positive definite matrix, it follows that  $d_Z$  is well defined as solution of the system (13.14). Since the reduced Hessian  $Z^T G Z$  is positive definite, then the Cholesky factorization can be used to solve (13.14). In conclusion, the total step  $d$  can be computed as  $d = Yd_Y + Zd_Z$ , where  $d_Y$  is the solution of (13.11) and  $d_Z$  is the solution of (13.14), computed in this order.

To obtain the Lagrange multipliers, we multiply the first line of (13.4) by  $Y^T$  to get the linear system

$$(AY)^T \lambda^* = Y^T g + Y^T Gd. \quad (13.15)$$

In this method, solving the linear algebraic systems (13.11) and (13.14) requires  $O(m^2) + O((n - m)^3)$  multiplications, which can be significantly less than  $O((n + m)^3)$  multiplications needed to solve the KKT system (13.3), especially when  $m \approx n$ .

**Example 13.1** Let us consider the null-space method for solving the following quadratic programming problem:

$$\min Q(x) = \frac{1}{2} \underbrace{(x_1^2 + x_2^2 + 2x_3^2 + x_4^2 + x_5^2)}_{x^T Gx} + \underbrace{(x_1 + x_3 + x_5)}_{c^T x}$$

subject to the linear constraints  $Ax = b$ , where

$$\underbrace{\begin{bmatrix} 1 & 0 & 2 & 3 & 1 \\ 2 & 1 & 2 & 1 & 1 \\ 4 & 3 & 1 & 2 & 2 \end{bmatrix}}_A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \underbrace{\begin{bmatrix} 7 \\ 7 \\ 12 \end{bmatrix}}_b.$$

In this problem,  $n = 5$  and  $m = 3$ . A basis  $Z$  for the null space of  $A$  is obtained by using the reduced echelon technique. The matrix  $Y$  is selected, such that  $[Y \ Z] \in \mathbb{R}^{n \times n}$  is nonsingular. Then,

$$Z = \begin{bmatrix} -13 & -3 \\ 15 & 3 \\ 5 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{n \times (n-m)}, \quad Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{n \times m}.$$

Consider  $x = [0 \ 1 \ 0 \ 1 \ 1]^T$  an initial estimation of the solution, with which we compute

$$h = Ax - b = [-3 \ -4 \ -5]^T, \quad g = Gx + c = [1 \ 1 \ 1 \ 1 \ 2]^T.$$

From (13.11), we get

$$d_Y = -(AY)^{-1}h = -\begin{bmatrix} 5 & -6 & 2 \\ -6 & 7 & -2 \\ -2 & 3 & -1 \end{bmatrix} \begin{bmatrix} -3 \\ -4 \\ -5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Similarly, from (13.14), we obtain

$$\begin{aligned} d_Z &= -(Z^T G Z)^{-1} \{Z^T G Y d_Y + Z^T g\} \\ &= -\frac{1}{509} \begin{bmatrix} 21 & -94 \\ -94 & 445 \end{bmatrix} \left\{ \begin{bmatrix} -3 \\ -1 \end{bmatrix} + \begin{bmatrix} 8 \\ 3 \end{bmatrix} \right\} = \begin{bmatrix} 0.163064 \\ -0.825147 \end{bmatrix}. \end{aligned}$$

Therefore, from (13.10), we get

$$d = Y d_Y + Z d_Z = \begin{bmatrix} 1.355609 \\ -0.029481 \\ 0.990173 \\ 0.163064 \\ -0.825147 \end{bmatrix}.$$

Using the known initial estimation  $x$ , the local optimal solution of the problem is

$$x^* = x + d = \begin{bmatrix} 1.355609 \\ 0.970519 \\ 0.990173 \\ 1.163064 \\ 0.174853 \end{bmatrix}.$$

The most complicated steps of the algorithm are the determination of a basis of the null space of  $A$  and the solving of the linear algebraic systems (13.11) and (13.14). ♦

**Remark 13.1** (Bartholomew-Biggs, 2008) The null-space method for quadratic programming with equality constraints can be very easily extended to the nonlinear optimization with linear equality constraints:

$$\min \{f(x) : Ax + b = 0\}, \quad (13.16)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable and  $A \in \mathbb{R}^{m \times n}$  is a constant matrix. As in the unconstrained optimization, the idea is to use an iterative scheme based on the local quadratic approximation of  $f$ . That is, in a neighborhood of a solution estimate  $x$ , we suppose

$$f(x + d) \approx f(x) + \nabla f(x)^T d + \frac{1}{2} d^T B d, \quad (13.17)$$

where either  $B = \nabla^2 f(x)$  or  $B \approx \nabla^2 f(x)$  is a quasi-Newton approximation of the Hessian. The following algorithm solves the problem (13.16) by using the quadratic approximation (13.17) at each iteration. The algorithm makes use of an update approximation to the Hessian matrix more than the exact Hessian  $\nabla^2 f(x)$  and of a line-search to improve the values of the nonquadratic function  $f$ .

Therefore, the problem is now to minimize (13.17) subject to  $Ad + (Ax + b) = 0$ . The optimality conditions for this problem are

$$\begin{aligned} Bd + \nabla f(x) - A^T \lambda &= 0, \\ Ad + (Ax + b) &= 0. \end{aligned}$$

Next, we apply the technique of the null-space method. Let  $Z$  be an  $n \times (n - m)$  matrix whose columns span the null space of  $A$ . The choice of  $Z$  is not unique, but one way of getting it is by the

*orthogonal factorization* of the constraint Jacobian  $A$ . This factorization yields an orthogonal  $n \times n$  matrix  $Q$  and an  $m \times m$  lower triangular matrix  $L$ , such that  $AQ = R = [L \ 0]$  and  $Q^T Q = I$ . Now, let us define  $Y$  as the matrix composed of the first  $m$  columns of  $Q$  and  $Z$  the matrix consisting of the remaining  $(n - m)$  columns. It can be shown that  $AZ = 0$ ,  $AY = L$ , and  $Y^T Z = 0$ .

If  $d$  solves the quadratic approximation of (13.16), then its components in the  $Y$ -subspace and  $Z$ -subspace can be determined as follows. Suppose that  $\bar{y} \in \mathbb{R}^m$  and  $\bar{z} \in \mathbb{R}^{n-m}$  are vectors, such that

$$d = Y\bar{y} + Z\bar{z}.$$

Since  $AZ = 0$ , the optimality condition  $Ad + (Ax + b) = 0$  implies

$$AY\bar{y} + (Ax + b) = 0.$$

However,  $AY = L$ . Therefore,  $\bar{y}$  can be found by solving a *lower triangular system* of equations

$$AY\bar{y} = -(Ax + b).$$

Now, premultiplying the optimality condition  $Bd + \nabla f(x) - A^T \lambda = 0$  by  $Z^T$  and having in view that  $AZ = 0$ , we get  $Z^T BZ\bar{z} + Z^T \nabla f(x) + Z^T BY\bar{y} = 0$ . Therefore,  $\bar{z}$  can be found by solving the system

$$Z^T BZ\bar{z} = -Z^T \nabla f(x) - Z^T BY\bar{y},$$

where  $Z^T BZ$  is a symmetric positive definite matrix. Knowing  $\bar{y}$  and  $\bar{z}$ , we can compute  $d = Y\bar{y} + Z\bar{z}$  as a correction of the estimate  $x$ .

As in the quadratic case, in this context  $Z^T \nabla f(x)$  is called the *reduced gradient* of  $f$  and  $Z^T BZ$  is the *reduced Hessian* of  $f$ . Observe that both  $Y$  and  $Z$ , basis matrices for the range- and null-spaces of the constraints matrix  $A$ , are kept constant along the iterations of the algorithm. The corresponding algorithm is as follows:

**Algorithm 13.1** *Reduced gradient for linear equality constraints*

- |    |   |
|----|---|
| 1. | Choose an initial feasible point $x_0$ . Choose $B_0$ as a positive definite approximation of $\nabla^2 f(x_0)$ . Compute $Y$ and $Z$ as basis matrices for the range- and the null-spaces of the constraints matrix $A$ . Choose $\varepsilon > 0$ sufficiently small. Set $k = 0$ and compute $\nabla f(x_0)$ |
| 2. | Test of convergence. If $\ Z^T \nabla f(x_k)\  \leq \varepsilon$ , then stop  |
| 3. | Determine $\bar{y}$ as solution of the system $AY\bar{y} = -(Ax_k + b)$   |
| 4. | Determine $\bar{z}$ as solution of the system $(Z^T B_k Z)\bar{z} = -Z^T \nabla f(x_k) - Z^T B_k Y\bar{y}$  |
| 5. | Set $d_k = Y\bar{y} + Z\bar{z}$   |
| 6. | Compute the estimate of the Lagrange multipliers $\lambda_{k+1}$ as solution of the linear system $(AY)^T \lambda = Y^T \nabla f(x_k) + Y^T B_k d_k$  |
| 7. | Perform a line-search to determine $s$ such that $f(x_k + sd_k) < f(x_k)$ . Set $x_{k+1} = x_k + sd_k$ as a new estimate of the solution  |
| 8. | Using a quasi-Newton update, from $B_k$ compute a new estimate $B_{k+1}$ . Set $k = k + 1$ and continue with step 2   |

◆

Since the objective is not quadratic, it follows that  $x_k + d_k$  is not guaranteed to be a better point than  $x_k$ . Therefore, Algorithm 13.1 is equipped with a line-search (in step 7) which may be exact or based on the Wolfe line-search conditions.

The algorithm uses a quasi-Newton update of  $B_k$  to get  $B_{k+1}$  as a new estimate of the Hessian. The BFGS update has the advantage of generating positive definite matrices  $B_{k+1}$ . This will ensure that the approximation of the reduced Hessian  $Z^T B_k Z$  is also positive definite, and hence,  $d_k$  is a descent direction. ♦

It is noteworthy that in the nonlinear constrained optimization, the true Hessian  $\nabla^2 f(x^*)$  is often indefinite. This suggests that a positive definite updating scheme, based, for example, on BFGS formula, may be inconsistent with making  $B_k$  a good approximation. However, this is not a problem, because the second-order optimality conditions only relate to the null space of the binding constraint normals. In this subspace, the optimality conditions require that the Hessian should be positive definite.

Under reasonable assumptions about function and constraints and about the Hessian, it can be shown that the search directions and the step lengths in the feasible subspace will satisfy the Wolfe conditions. Therefore, the iterations will converge to a point where the reduced gradient is close to zero.

## Large-Scale Problems

Consider the quadratic programming problem with equality constraints

$$\begin{aligned} \min q(x) &\triangleq \frac{1}{2} x^T G x + c^T x \\ \text{subject to} \\ Ax &= b, \end{aligned} \tag{13.18}$$

where  $A \in \mathbb{R}^{m \times n}$  of full rank, with  $m < n$ , so that the constraints  $Ax = b$  are consistent. For solving large-scale quadratic programming problems, the direct factorization methods of the KKT system (13.3) are not very effective. The iterative methods are more suitable. As we know, the conjugate gradient methods are dedicated to solving very large systems of linear equations with positive definite matrices. However, the conjugate gradient methods directly applied to the KKT system (13.3) are not recommended because they can be unstable on systems with matrices that are not positive definite. It is better to apply the conjugate gradient methods to the reduced system (13.14).

## The Conjugate Gradient Applied to the Reduced System

For this, let us express the solution of (13.18) as

$$x^* = Yx_Y + Zx_Z, \tag{13.19}$$

where  $x_Y \in \mathbb{R}^m$  and  $x_Z \in \mathbb{R}^{n-m}$ .  $Z \in \mathbb{R}^{n \times (n-m)}$  is a basis for the null space of  $A$ , i.e.,  $AZ = 0$ .  $Y \in \mathbb{R}^{n \times m}$  is a matrix, such that  $[Y \ Z] \in \mathbb{R}^{n \times n}$  is nonsingular. Observe that, since  $A$  is of full row rank, it follows that  $A[Y \ Z] = [AY \ 0]$  is also of full row rank, so that the matrix  $AY$  is nonsingular. We would like to choose  $Y$  so that the matrix  $AY$  is conditioned as much as possible. This can be achieved by the  $QR$  factorization of  $A^T$ , which has the form

$$A^T P = [Q_1 \quad Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad (13.20)$$

where  $[Q_1 \quad Q_2]$  is orthogonal. The submatrices  $Q_1 \in \mathbb{R}^{n \times m}$  and  $Q_2 \in \mathbb{R}^{n \times (n-m)}$  have orthonormal columns, while  $R \in \mathbb{R}^{m \times m}$  is an upper triangular and nonsingular matrix.  $P \in \mathbb{R}^{m \times m}$  is a permutation matrix. Now, define  $Y = Q_1$  and  $Z = Q_2$ , so that the columns of  $Y$  and  $Z$  form an orthonormal basis of  $\mathbb{R}^n$ . Now, from (13.20), it is obtained that  $AY = PR^T$  and  $AZ = 0$ . Therefore,  $Y$  and  $Z$  have the desired properties, and the condition number of  $AY$  is the same as that of  $R$ , which in its turn is the same as that of  $A$ .

With this, the constraints  $Ax = b$  can be written as

$$AYx_Y = b, \quad (13.21)$$

which determines the component  $x_Y$  of  $x^*$ . Now, introducing (13.19) in (13.18), it follows that  $x_Z$  is a solution of the unconstrained reduced problem

$$\min_{x_Z} \frac{1}{2} x_Z^T Z^T G Z x_Z + c_Z^T x_Z, \quad (13.22)$$

where

$$c_Z = Z^T G Y x_Y + Z^T c. \quad (13.23)$$

Obviously, from (13.22), we can see that  $x_Z$  satisfies the following linear system:

$$Z^T G Z x_Z = -c_Z. \quad (13.24)$$

Since  $Z^T G Z$  is positive definite, it follows that the system (13.24) can be solved by applying the conjugate gradient algorithm. In conclusion, with these algebraic constructions, we can see that since  $AY$  is nonsingular, the component  $x_Y$  can be obtained from (13.21); since  $Z^T G Z$  is positive definite, the component  $x_Z$  can be obtained from (13.24), and thus, the solution  $x^*$  of (13.18) can be computed from (13.19).

In the following, let us present the preconditioned conjugate gradient algorithm for solving the reduced system (13.24). Suppose that a preconditioner  $W_{ZZ}$  is given. Then, the preconditioned conjugate gradient algorithm applied to the  $(n - m)$ -dimensional reduced system (13.24) is as follows:

### Algorithm 13.2 Preconditioned conjugate gradient for reduced systems

1.	Initialization. Choose an initial point $x_Z$
2.	Compute: $r_Z = Z^T G Z x_Z + c_Z$ , $g_Z = W_{ZZ}^{-1} r_Z$ and $d_Z = -g_Z$
3.	Test a criterion for stopping the iterations. For example, if $r_Z^T W_{ZZ}^{-1} r_Z$ is sufficiently small, then stop the iterations.
4.	Compute: $\alpha = r_Z^T g_Z / d_Z^T Z^T G Z d_Z$ , $x_Z = x_Z + \alpha d_Z$ , $r_Z^+ = r_Z + \alpha Z^T G Z d_Z$ , $g_Z^+ = W_{ZZ}^{-1} r_Z^+$ , $\beta = (r_Z^+)^T g_Z^+ / r_Z^T g_Z$ , $d_Z = -g_Z^+ + \beta d_Z$ , $g_Z = g_Z^+$ , $r_Z = r_Z^+$ .
5.	Go to step 3

◆

In Algorithm 13.2, the steps produced by the conjugate gradient are denoted by  $d_Z$ . It is worth saying that it is not necessary to explicitly form the reduced Hessian  $Z^T G Z$ . This is because the conjugate gradient algorithm requires only the computation of a matrix-vector product involving this matrix. The preconditioner  $W_{ZZ} \in \mathbb{R}^{(n-m) \times (n-m)}$  is a symmetric, positive definite matrix, which may be chosen to cluster the eigenvalues of  $W_{ZZ}^{-1/2} (Z^T G Z) W_{ZZ}^{-1/2}$ , that is to reduce the span between the smallest and the largest eigenvalues. One of the best selections of the preconditioner is one for which  $W_{ZZ}^{-1/2} (Z^T G Z) W_{ZZ}^{-1/2} = I$ , that is,  $W_{ZZ} = Z^T G Z$ . This is the main motivation for which we take the preconditioners of the following form:

$$W_{ZZ} = Z^T H Z, \quad (13.25)$$

where  $H$  is a symmetric matrix, such that  $Z^T H Z$  is positive definite.

### The Projected Conjugate Gradient Method

Another algorithm for solving (13.1) can be obtained by rewriting Algorithm 13.2 to work in the  $n$ -dimensional space. For this, the following  $n$ -vectors are introduced:  $x$ ,  $r$ ,  $g$ , and  $d$ , which satisfy  $x = Zx_Z + Yx_Y$ ,  $Z^T r = r_Z$ ,  $g = Zg_Z$ , and  $d = Zd_Z$ , respectively. The scaled projection matrix  $P \in \mathbb{R}^{n \times n}$  is also defined as  $P = Z(Z^T H Z)^{-1} Z^T$ , where  $H$  is the preconditioning matrix from (13.24). Hence, the projected conjugate gradient algorithm is as follows:

#### Algorithm 13.3 Projected conjugate gradient

- |    |  |
|----|--|
| 1. | Initialization. Choose an initial point $x$ satisfying $Ax = b$  |
| 2. | Compute: $r = Gx + c$ , $g = Pr$ and $d = -g$  |
| 3. | Test a criterion for stopping the iterations. For example, if $r^T g$ is smaller than a prespecified tolerance, then stop the iterations.  |
| 4. | Compute:<br>$\alpha = r^T g / d^T G d$ , $x = x + \alpha d$ , $r^+ = r + \alpha G d$ , $g^+ = P r^+$ ,<br>$\beta = (r^+)^T g^+ / r^T g$ , $d = -g^+ + \beta d$ , $g = g^+$ , $r = r^+$ |
| 5. | Go to step 3   |

◆

Observe that  $g^+$ , which is called *preconditioned residual*, is a vector defined in the null space of  $A$ . Therefore, in exact arithmetic, all the search directions  $d$  generated by Algorithm 13.3 also lie in the null space of  $A$ , that is, all the iterates satisfy  $Ax = b$ . The iterations of Algorithm 13.3 are well defined if  $Z^T G Z$  and  $Z^T H Z$  are positive definite. There are two simple choices of the preconditioning matrix  $H$ :  $H = \text{diag}(|G_{ii}|)$  and  $H = I$ . Nocedal and Wright (2006) give some details how to compute  $Pr$  without knowing a representation of the null space basis  $Z$ . Consider the case in which  $H = I$ . In this case,  $P$  is the orthogonal projection operator onto the null space of  $A$ . This special case of  $P$  is denoted as  $\bar{P}$ , that is,

$$\bar{P} = Z(Z^T Z)^{-1} Z^T. \quad (13.26)$$

The computation of the preconditioned residual  $g^+ = \bar{P}r^+$  can be performed in two different ways. The first one, called the *normal equations approach*, expresses  $\bar{P}$  by  $\bar{P} = I - A^T(AA^T)^{-1}A$  and thus computes  $g^+ = \bar{P}r^+$ . Therefore, we can write  $g^+ = r^+ - A^T v^+$ , where  $v^+$  is the solution of the system

$$AA^T v^+ = Ar^+. \quad (13.27)$$

The system (13.27) can be solved by using the Cholesky factorization of  $AA^T$ .

The second one, called the *augmented system approach*, is to express the projection  $Pr^+$  as the solution of the following augmented system:

$$\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} g^+ \\ v^+ \end{bmatrix} = \begin{bmatrix} r^+ \\ 0 \end{bmatrix} \quad (13.28)$$

Of course, we have

$$g^+ = Pr^+ = r^+ - A^T v^+ = r^+ - A^T (AA^T)^{-1} Ar^+ = (I - A^T (AA^T)^{-1} A) r^+ = \bar{P}r^+.$$

All these developments allow us to compute an initial point that satisfies  $Ax = b$ . The computation of  $g^+ = Pr^+$  relies on the factorization of  $AA^T$  from which we can compute  $x = A^T (AA^T)^{-1} b$ . Note that these approaches for computing  $g^+$  can give rise to significant round-off errors, so the use of the iterative refinement is recommended in order to improve the accuracy (Nocedal & Wright, 2006).

## 13.2 Inequality Constrained Quadratic Programming

An inequality constrained quadratic program has the following form:

$$\min q(x) \equiv \frac{1}{2} x^T G x + c^T x \quad (13.29a)$$

subject to

$$a_i^T x = b_i, \quad i \in E, \quad (13.29b)$$

$$a_i^T x \geq b_i, \quad i \in I, \quad (13.29c)$$

where  $G$  is an  $n \times n$  symmetric matrix,  $c, x$ , and  $\{a_i\}, i \in E \cup I$ , are vectors in  $\mathbb{R}^n$ , and  $E$  and  $I$  are finite sets of indices. The Lagrangian for this problem is

$$L(x, \lambda) = \frac{1}{2} x^T G x + c^T x - \sum_{i \in E \cup I} \lambda_i (a_i^T x - b_i).$$

As in [Definition 11.12](#), the active set  $A(x^*)$  consists of the indices of the constraints from (13.29), for which the equality holds at  $x^*$ :

$$A(x^*) = \{i \in E \cup I : a_i^T x = b_i\}. \quad (13.30)$$

Supposing that the constraint qualification LICQ is satisfied and specializing the general KKT conditions (11.21) and (11.22) to this problem, we find that any solution  $x^*$  of (13.29) satisfies the following first-order optimality conditions for some Lagrange multipliers  $\lambda_i^*, i \in A(x^*)$ :

$$Gx^* + c - \sum_{i \in A(x^*)} \lambda_i^* a_i = 0, \quad (13.31a)$$

$$a_i^T x^* = b_i, \quad \text{for all } i \in A(x^*), \quad (13.31b)$$

$$a_i^T x^* \geq b_i, \quad \text{for all } i \in I \setminus A(x^*), \quad (13.31c)$$

$$\lambda_i^* \geq 0, \quad \text{for all } i \in I \cap A(x^*). \quad (13.31d)$$

If  $G$  is positive definite, then (13.29) is called *convex quadratic programming*. For the convex quadratic programming, the conditions (13.31) are sufficient for  $x^*$  to be a global solution, as proved in the next theorem.

**Theorem 13.3** *If  $x^*$  satisfies the conditions (13.31) for some  $\lambda_i^*$ ,  $i \in A(x^*)$ , and  $G$  is positive semidefinite, then  $x^*$  is a global solution of (13.29).*

**Proof** If  $x$  is any other feasible solution for (13.29), then we have  $a_i^T x = b_i$ , for all  $i \in E$ , and  $a_i^T x \geq b_i$ , for all  $i \in A(x^*) \cap I$ . Hence,  $a_i^T(x - x^*) = 0$  for all  $i \in E$  and  $a_i^T(x - x^*) \geq 0$  for all  $i \in A(x^*) \cap I$ . Now, using (13.31a) and (13.31d), it follows that

$$(x - x^*)^T (Gx^* + c) = \sum_{i \in E} \lambda_i^* a_i^T (x - x^*) + \sum_{i \in A(x^*) \cap I} \lambda_i^* a_i^T (x - x^*) \geq 0. \quad (13.32)$$

Having in view (13.32), after some algebra, we get

$$\begin{aligned} q(x) &= q(x^*) + (x - x^*)^T (Gx^* + c) + \frac{1}{2} (x - x^*)^T G (x - x^*) \\ &\geq q(x^*) + \frac{1}{2} (x - x^*)^T G (x - x^*) \geq q(x^*), \end{aligned}$$

since  $G$  is positive semidefinite. Therefore, we have shown that  $q(x) \geq q(x^*)$  for any feasible  $x$ , i.e.,  $x^*$  is a global solution.  $\blacklozenge$

If the reduced Hessian  $Z^T G Z$  is positive definite, where  $Z$  is a null-space basis matrix for the active constraint Jacobian matrix, which is the matrix whose rows are  $a_i^T$  for all  $i \in A(x^*)$ , then the second-order sufficient conditions for  $x^*$  to be a local minimum hold. If  $G$  is not positive definite, then the general problem (13.29) may have more than one strict local solution. Such problems are called *nonconvex quadratic programming* or *indefinite quadratic programming*.

For solving the inequality constrained quadratic programming problems (13.29), we can use the *active-set methods* or the *interior-point methods*. In case of the convex quadratic programming, there are three varieties of active-set methods: *primal* (Beale, 1959; Fletcher, 1971; Bunch & Kaufman, 1977), *dual* (Lemke, 1962), and *primal-dual* (Goncalves, 1972; Goldfarb & Idnani, 1983). The active-set methods find a step from one iteration to the next one by solving a quadratic subproblem in which some of the inequality constraints (13.29c) and all the equality constraints (13.29b) are imposed as equalities. This subset of constraints is referred to as the *working set*, which at iteration  $k$  is denoted as  $W_k$ . The case in which  $G$  is an indefinite matrix raises some complications in the algorithms which are beyond the purpose of this book. For the nonconvex quadratic programming, see Gould, Orban, and Toint (2005a). In the following, we shall discuss the primal active-set method and the primal-dual active-set method by Goldfarb and Idnani (1983).

## The Primal Active-Set Method

The idea of the primal active-set method is as follows. Given the iterate  $x_k$  and the working set  $W_k$ , test whether  $x_k$  minimizes the quadratic  $q$  in the subspace defined by the working set. If not, a step  $d$  is computed by solving an equality constrained quadratic programming subproblem in which the constraints corresponding to the working set  $W_k$  are regarded as equalities and the rest of constraints are temporarily ignored. Let us define

$$d = x - x_k, \quad g_k = Gx_k + c.$$

With this, the objective function (13.29a) becomes

$$q(x) = q(x_k + d) = \frac{1}{2} d^T Gd + g_k^T d + \delta_k,$$

where  $\delta_k = \frac{1}{2} x_k^T Gx_k + c^T x_k$  is independent of  $d$ .

Since  $\delta_k$  can be dropped from the expression of  $q(x)$ , the quadratic subproblem which must be solved at the  $k$ -th iteration is

$$\min \frac{1}{2} d^T Gd + g_k^T d \quad (13.33a)$$

subject to

$$a_i^T d = 0, \quad i \in W_k. \quad (13.33b)$$

Let  $d_k$  be the solution of (13.33). Since  $a_i^T(x_k + \alpha d_k) = a_i^T x_k = b_i$  for all  $\alpha$ , observe that for each  $i \in W_k$ , it follows that the value of  $a_i^T x$  does not change as we move along  $d_k$ . Therefore, since the constraints in  $W_k$  are satisfied at  $x_k$ , they are also satisfied at  $x_k + \alpha d_k$  for any value of  $\alpha$ . Since  $G$  is positive definite, it follows that the solution of (13.33) can be computed by any method for solving equality constrained quadratic programming.

Supposing that the optimal  $d_k$  is nonzero, the question is how far we can move along this direction to conserve the feasibility. If  $x_k + d_k$  is feasible with respect to all the constraints, we set  $x_{k+1} = x_k + d_k$ . Otherwise, set

$$x_{k+1} = x_k + \alpha_k d_k, \quad (13.34)$$

where the stepsize  $\alpha_k$  is computed to be the largest value in the range  $[0, 1]$  for which all the constraints are satisfied. We must consider the definition of  $\alpha_k$  for the constraints  $i \notin W_k$ , since, as we said before, the constraints in  $W_k$  are satisfied at  $x_k + \alpha d_k$  for any value of  $\alpha$ . If  $a_i^T d_k \geq 0$  for some  $i \notin W_k$ , then for all  $\alpha_k \geq 0$  we have  $a_i^T(x_k + \alpha_k d_k) \geq a_i^T x_k \geq b_i$ . Therefore, the constraint  $i$  will be satisfied for all nonnegative  $\alpha_k$ . On the other hand, if  $a_i^T d_k < 0$  for some  $i \notin W_k$ , then  $a_i^T(x_k + \alpha_k d_k) \geq b_i$  only if

$$\alpha_k \leq \frac{b_i - a_i^T x_k}{a_i^T d_k}. \quad (13.35)$$

To maximize the decrease in  $q$ , we must take  $\alpha_k$  to be as large as possible in  $[0, 1]$  subject to the satisfying feasibility. Hence,  $\alpha_k$  is determined as

$$\alpha_k \triangleq \min \left\{ 1, \min \left\{ \frac{b_i - a_i^T x_k}{a_i^T d_k}, i \notin W_k, a_i^T d_k < 0 \right\} \right\}. \quad (13.36)$$

The constraint  $i$  for which the minimum in (13.36) is achieved is called *blocking constraint*, like in linear programming (Andrei, 2011d).

If  $\alpha_k < 1$ , that is, the step along  $d_k$  was blocked by some constraint not in  $W_k$ , then a new working set  $W_{k+1}$  is generated by adding one of the blocking constraints to  $W_k$ .

The iterations are continued in this way, by adding constraints to the working set until a point  $\hat{x}$  that minimizes the quadratic objective function over its current working set  $\hat{W}$  has been attained. Such a point is easy to be recognized because the subproblem (13.33) has as solution  $d = 0$ . Since  $d = 0$  satisfies the optimality conditions (13.3) corresponding to (13.33), we have

$$\sum_{i \in \hat{W}} a_i \hat{\lambda}_i = g = G\hat{x} + c \quad (13.37)$$

for some Lagrange multipliers  $\hat{\lambda}_i$ ,  $i \in \hat{W}$ . It follows that  $\hat{x}$  and  $\hat{\lambda}$  satisfy the first KKT optimality condition (13.31a) if we define the multipliers corresponding to the inequality constraints that are not in the working set to be zero. On the other hand, because  $\hat{x}$  is also feasible with respect to all the constraints, it follows that (13.31b) and (13.31c) are satisfied at this point.

#### Algorithm 13.4 Active-set method for convex quadratic programming

1.	Compute a feasible initial point $x_0$ . Set $W_0$ to be a subset of the active constraints at $x_0$ . Set $k = 0$
2.	Test a criterion for stopping the iterations
3.	Solve the equality quadratic programming sub-problem (13.33) to find $d_k$
4.	<p>If <math>d_k = 0</math>, then compute the Lagrange multipliers <math>\hat{\lambda}_i</math> that satisfy (13.37) with <math>\hat{W} = W_k</math>. If <math>\hat{\lambda}_i \geq 0</math>, for all <math>i \in W_k \cap I</math>, then stop with the solution <math>x^* = x_k</math>; otherwise, determine <math>j = \arg \min \{\hat{\lambda}_j : j \in W_k \cap I\}</math> and set <math>x_{k+1} = x_k</math> and <math>W_{k+1} = W_k \setminus \{j\}</math>.</p> <p>If <math>d_k \neq 0</math>, then from (13.36) compute <math>\alpha_k</math> and set <math>x_{k+1} = x_k + \alpha_k d_k</math>. If there are blocking constraints, then obtain <math>W_{k+1}</math> by adding one of the blocking constraints to <math>W_k</math>; otherwise, set <math>W_{k+1} = W_k</math> and go to step 2</p>

◆

Some properties of the algorithm are given by Nocedal and Wright (2006), where plenty and important details on the primal active-set method are presented. These properties refer to the signs of the multipliers corresponding to the inequality constraints in the working set (the indices  $i \in \hat{W} \cap I$ ), various techniques that are used to determine an initial feasible point, the strategy of removing the constraint corresponding to the most negative Lagrange multiplier, the strategy of adding or deleting one constraint at each iteration at most, the finite termination of the active-set algorithm on strictly convex quadratic programming, updating the factorizations, etc.

Algorithm 13.4 can be adapted to the situations in which the Hessian is indefinite by modifying the computation of the search direction and of the stepsize. To explain the need for modification, let us consider the computation of the step by the null-space method, that is,  $d = Zd_z$ , where  $d_z$  is given by  $(Z^T G Z)d_z = -Z^T g$ . If the reduced Hessian  $Z^T G Z$  is positive definite, then the step  $d$  is directed to the minimizer of the subproblem (13.33), and the iteration need not be changed. On the other hand, if  $Z^T G Z$  has negative eigenvalues, then  $d$  is directed to a saddle point of (13.33), and therefore, it is not a suitable step any more. In this case, an alternative direction  $s_z$  is sought, which is a direction of negative curvature for the reduced Hessian  $Z^T G Z$ . In this case,  $q(x_k + \alpha Z s_z) \rightarrow -\infty$ , as  $\alpha \rightarrow \infty$ .

Additionally, if necessary, the sign of  $s_z$  is changed to ensure that  $Zs_z$  is a non-ascent direction for  $q$  at the current point. By moving along the direction  $Zs_z$ , we encounter a constraint that can be added to the working set for the next iteration. Now, if the reduced Hessian for this new working set is not positive definite, the above procedure is repeated until enough constraints have been added to make the reduced Hessian positive definite.

Another practical class of algorithms for indefinite quadratic programming is that of inertia controlling methods. These methods never allow the reduced Hessian to have more than one negative eigenvalue. At each iteration, the algorithm will either add or remove a constraint from the working set. If a constraint is added, then the reduced Hessian is of smaller dimension and must remain positive definite or be the null matrix. Therefore, an indefinite reduced Hessian can arise only when one of the constraints is removed from the working set, which happens only when the current point is a minimizer with respect to the current working set. In this case, we will choose the new search direction to be a direction of negative curvature for the reduced Hessian. The algorithms for indefinite quadratic programming differ in the way they detect the indefiniteness and they compute the negative curvature direction and in the handling of the working set (Fletcher, 1971; Gill & Murray, 1978).

## An Algorithm for Positive Definite Hessian

This active-set algorithm is based on a repeated use of the optimality conditions for an equality quadratic programming problem. In the following, we consider the quadratic programming which has only inequality constraints. The extension to mixed equality and inequality quadratic problems is straightforward. Therefore, let us consider the problem

$$\min \frac{1}{2}x^T Gx + c^T x \quad (13.38a)$$

subject to

$$\hat{A}x + \hat{b} \geq 0, \quad (13.38b)$$

where we assume that  $G$  is positive definite. The method begins by identifying an active set of constraints. This is an estimate of those which are binding at the solution. Now, let  $A$  be the matrix and  $b$  the vector formed from the rows of  $\hat{A}$  and  $\hat{b}$  corresponding to the active constraints. With this, let  $(\tilde{x}, \tilde{\lambda})$  be a trial solution by minimizing (13.38a) subject to

$$Ax + b = 0,$$

obtained by using an algorithm for the equality quadratic programming problem. If

$$\hat{A}\tilde{x} + \hat{b} \geq 0, \quad (13.39a)$$

$$\tilde{\lambda}^T (\hat{A}\tilde{x} + \hat{b}) = 0 \text{ and } \tilde{\lambda} \geq 0, \quad (13.39b)$$

then the optimality conditions (13.31) are all satisfied and the problem (13.38) is solved by  $x^* = \tilde{x}$  and  $\lambda^* = \tilde{\lambda}$ . On the other hand, if  $(\tilde{x}, \tilde{\lambda})$  is not optimal, we must change the active set and solve another equality constrained quadratic programming subproblem. This process is repeated until the active set has become the binding set for the problem (13.38). The choice of a new active set is based on (13.39). Two ideas are followed here. The first one is that any new constraint which is violated at  $\tilde{x}$

can be regarded as a candidate for being added to the current active set. Secondly, any active constraint which corresponds to a negative component of  $\hat{\lambda}$  is a candidate for deletion from the current active set (Bartholomew-Biggs, 2008). The following algorithm formalizes the ideas just outlined, where  $\hat{a}_i$  denotes the  $i$ -th row of  $\hat{A}$ .

**Algorithm 13.5** *Active-set method with positive definite Hessian*

- |    |   |
|----|---|
| 1. | Choose an initial point $x$ and set $\lambda_i = 0$ , $i = 1, \dots, m$   |
| 2. | Test the optimality conditions. If $\hat{A}x + \hat{b} \geq 0$ , $Gx + c - \hat{A}^\top \lambda = 0$ , and $\lambda \geq 0$ , then stop   |
| 3. | Identify the active constraints as those for which<br>$\hat{a}_i^\top x + \hat{b}_i < 0 \quad \text{or} \quad (\hat{a}_i^\top x + \hat{b}_i = 0 \text{ and } \lambda_i \geq 0).$ Renumber these constraints so that the active set is $i = 1, \dots, t$ |
| 4. | Set $g = Gx + c$ and $b_i = \hat{a}_i^\top x + \hat{b}_i$ , for $i = 1, \dots, t$   |
| 5. | Find $d$ and $\mu$ as solution for the equality quadratic programming problem<br>$\min \frac{1}{2} d^\top Gd + g^\top d \text{ subject to } \hat{a}_i^\top d + b_i = 0, \quad i = 1, \dots, t$  |
| 6. | Set $(s = 1, \lambda_i = \mu_i, i = 1, \dots, t)$ and $(\lambda_i = 0, i = t + 1, \dots, m)$  |
| 7. | For $i = t + 1, \dots, m$ , i.e., for all <i>inactive</i> constraints if $\hat{a}_i^\top p < 0$ set<br>$s = \min \left\{ s, -\frac{\hat{a}_i^\top \hat{b}_i}{\hat{a}_i^\top d} \right\}$  |
| 8. | Set $x = x + sd$ and go to step 2   |

◆

The algorithm may have different variants according to the methods for solving the problems in step 5 and to the rules with regard to how many constraints may be added to or dropped from the active set. The stepsize computed in step 7 checks all the inactive constraints that might be violated by a step along the search direction  $p$  and ensures that no more than one constraint can be added to the active set on the current iteration. Details can be found in Fletcher (1971).

## Reduced Gradient for Inequality Constraints

The active-set strategy combined with the null-space approach can be very easily extended to solving problems with the nonquadratic objective function  $f(x)$  and with linear inequality constraints  $\hat{a}_i^\top x + \hat{b}_i \geq 0$ ,  $i = 1, \dots, m$ . In this case, the  $Y$  and  $Z$  matrices used in the null-space method must be computed whenever the active set is changed (Bartholomew-Biggs, 2008).

**Algorithm 13.6** *Reduced gradient for linear inequality constraints*

- |    |   |
|----|---|
| 1. | Choose an initial point $x_0$ and set $\lambda^0 = 0$ . Choose $B_0$ as a positive definite estimate of $\nabla^2 f(x_0)$ . Select a tolerance $\epsilon > 0$ . Set $k = 0$ |
| 2. | Compute $g_k = \nabla f(x_k)$   |
| 3. | Select active constraints as those with $\hat{a}_i^\top x_k + \hat{b}_i = 0$ and $\lambda_i^k \geq 0$   |
| 4. | Consider $A_k$ as the matrix of active constraint normals at $x_k$  |
| 5. | Compute $Y_k$ and $Z_k$ as basis matrices for the range and the null spaces of matrix $A_k$   |
| 6. | Determine $z$ as solution of the linear system $Z_k^\top B_k Z_k z = -Z_k^\top g_k$ . Set $d_k = Z_k z$   |
| 7. | Find $\lambda^{k+1}$ as solution of the linear system $Y_k^\top A_k^\top \lambda = Y_k^\top g_k + Y_k^\top B_k d_k$   |

8.	Step length computation. Perform a line-search to get $x_{k+1} = x_k + sd_k$ so that $f(x_{k+1}) < f(x_k)$
9.	Compute a quasi-Newton update of $B_k$ to get $B_{k+1}$
10.	Test of convergence. If $\ Z_k^T g_k\  \leq \epsilon$ , then stop; otherwise, set $k = k + 1$ go to step 2

◆

Observe that in step 8 the computations of the step length must ensure that no new constraints are violated. Therefore, the stepsize  $s$  is subject to an upper limit which allows at most one new constraint to become binding at  $x_{k+1}$ . This can be computed as in step 7 of Algorithm 13.5. The difficulty with this algorithm is that the matrices  $Y$  and  $Z$  used in the null-space method must be computed whenever the active set is changed in step 4 of the algorithm.

## The Reduced Gradient for Simple Bounds

An important situation in which  $Z$  can be very easily computed and, therefore, makes the reduced-gradient approach very attractive is when the problem has only simple bounds on variables:  $l_i \leq x_i \leq u_i$ ,  $i = 1, \dots, n$ . Suppose that  $l_i < u_i$ ,  $i = 1, \dots, n$ . In this case, at the start of each iteration the variables are classified in two classes: those which are *fixed* (i.e., on their bounds) and those which are *free*. If  $x_k$  is the current estimate at the start of iteration  $k$ , then the bound on the  $i$ -th variable is active if  $x_k$  is fixed, which means that

$$(x_{k_i} = l_i \text{ and } g_{k_i} > 0) \quad \text{or} \quad (x_{k_i} = u_i \text{ and } g_{k_i} < 0).$$

In this case, the matrix  $Z$  whose columns span the space of the free variables can be simply taken as a partition of the identity matrix. When the iterate  $x_{k+1}$  is computed from  $x_k$  along a search direction  $d_k$ , the stepsize must ensure that no new bounds are violated. A maximum stepsize to force each variable  $x_k$  to stay within its bounds can be computed as

$$s_i^{\max} = \begin{cases} (u_i - x_{k_i})/d_{k_i} & \text{if } d_{k_i} > 0, \\ (l_i - x_{k_i})/d_{k_i} & \text{if } d_{k_i} < 0. \end{cases}$$

### Algorithm 13.7 Reduced gradient for simple bounds constraints

1.	Choose an initial point $x_0$ and $B_0$ as a positive definite estimate of $\nabla^2 f(x_0)$ . Select a tolerance $\epsilon > 0$ . Set $k = 0$
2.	Compute $g_k = \nabla f(x_k)$
3.	Set $Z_k$ as the $n \times n$ identity matrix
4.	For $i = 1, \dots, n$ repeat: if $x_{k_i}$ satisfies $(x_{k_i} = l_i \text{ and } g_{k_i} > 0)$ or $(x_{k_i} = u_i \text{ and } g_{k_i} < 0)$ , then delete the $i$ -th column of $Z_k$
5.	Determine $z$ as solution of the linear system $Z_k^T B_k Z_k z = -Z_k^T g_k$ . Set $d_k = Z_k z$
6.	Use a line-search along $d_k$ to find $s$ so that $f(x_k + sd_k) < f(x_k)$
7.	For each free variable $x_{k_i}$ compute $s_i^{\max}$ and set $s = \min \{s, s_i^{\max}\}$
8.	Set $x_{k+1} = x_k + sd_k$
9.	Compute a quasi-Newton update of $B_k$ to get $B_{k+1}$
10.	Test of convergence. If $\ Z_k^T g_k\  \leq \epsilon$ , then stop; otherwise, set $k = k + 1$ go to step 2

◆

Details on this algorithm can be found in Bartholomew-Biggs (2008).

## The Primal-Dual Active-Set Method

In the following, we present the algorithm by Goldfarb and Idnani (1983) for solving the inequality constrained quadratic programming problems

$$\min Q(x) \triangleq \frac{1}{2} x^T G x + c^T x \quad (13.40a)$$

subject to

$$s_i(x) = a_i^T x + b_i = 0, \quad i = 1, \dots, m_e, \quad (13.40b)$$

$$s_i(x) = a_i^T x + b_i \geq 0, \quad i = m_e + 1, \dots, m, \quad (13.40c)$$

$$l \leq x \leq u, \quad (13.40d)$$

where  $G$  is an  $n \times n$  symmetric and positive definite matrix and  $a_i \in \mathbb{R}^n$  for all  $i = 1, \dots, m$ . In general, the algorithms for quadratic programming have two phases. In the first one, a feasible point is obtained, and then, in the second phase, the optimality is achieved while maintaining the feasibility.

The algorithm of Goldfarb and Idnani is of the active-set type. By an active set, we mean a subset of the  $m$  constraints in (13.40) that are satisfied as equalities by the current estimate  $x$  of the solution to the quadratic programming problem (13.40). If we denote the set of indices of the constraints (without simple bounds) by  $I = \{1, \dots, m\}$ , then the active set is denoted by  $A \subseteq I$ . Let  $P(J)$  be the subproblem with the objective function (13.40a) subject only to the subset of the constraints indexed by  $J \subseteq I$ . For example,  $P(\emptyset)$  is the problem of finding the unconstrained minimum of (13.40). If the solution  $x$  of a subproblem  $P(J)$  lies on some linearly independent active set of constraints indexed by  $A \subseteq J$ , then the pair  $(x, A)$  is called a solution. Clearly, if  $(x, A)$  is a solution for the subproblem  $P(J)$ , then it is also a solution for the subproblem  $P(A)$ . By linear independence of a set of constraints, we mean that the normals corresponding to these constraints are linearly independent. In the following, we outline the algorithm given by Goldfarb and Idnani (1983):

1. Assume that we know an arbitrary solution  $(x, A)$ .
2. Choose a violated constraint  $p \in \Delta A$ .
3. If  $P(A \cup \{p\})$  is infeasible, then stop; problem (13.40) is infeasible.
4. Else, obtain a new pair  $(\bar{x}, \bar{A} \cup \{p\})$ , where  $\bar{A} \subseteq A$  and  $Q(\bar{x}) > Q(x)$ , and set  $(x, A) \leftarrow (\bar{x}, \bar{A} \cup \{p\})$ .
5. Continue with step 2 until all the constraints have been satisfied.

Since  $G$  is positive definite, it follows that the unconstrained minimum of the function  $Q(x)$  defined in (13.40a) is  $x^0 = -G^{-1}c$ . Therefore, the above procedure can start with the pair  $(x^0, \emptyset)$ .

For describing the algorithm, we have to introduce some notations like in Goldfarb and Idnani (1983). The matrix of the normal vectors of the constraints in the active set  $A$  is denoted by  $N$ , and the cardinality of  $A$  is denoted by  $q$ . Denote  $A^+ = A \cup \{p\}$ , where  $p \in \Delta A$ , and  $A^-$  is a proper subset of  $A$ , which contains one fewer element than  $A$ .  $N^+$  and  $N^-$  represent the matrices of normals corresponding to  $A^+$  and  $A^-$ , respectively.  $e_i$  represents the  $i$ -th column of the unity matrix, and  $I_k$  is the  $k \times k$  identity matrix.  $n^+$  represents the normal vector  $n_p$  added to  $N$  to obtain  $N^+$ ;  $n^-$  represents the column deleted from  $N$  to get  $N^-$ .

If the columns from  $N$  are linearly independent, then we may introduce the following operators:

$$N^* = (N^T G^{-1} N)^{-1} N^T G^{-1} \quad (13.41)$$

and

$$H = G^{-1} (I - NN^*) = G^{-1} - G^{-1} N (N^T G^{-1} N)^{-1} N^T G^{-1}. \quad (13.42)$$

We can see that  $N^*$  is the pseudo-inverse, or the generalized Moore-Penrose inverse, of  $N$  in the space of variables obtained through the transformation  $y = G^{1/2}x$ . On the other hand,  $H$  is the inverse of the reduced Hessian of the function  $Q(x)$  with respect to the active set of constraints. If  $\hat{x}$  is a point in the  $(n - q)$ -dimensional manifold  $M = \{x \in \mathbb{R}^n : n_i^T x = b_i, i \in A\}$  and  $g(\hat{x}) \equiv \nabla Q(\hat{x}) = G\hat{x} + c$  is the gradient of the objective function  $Q(x)$  at  $\hat{x}$ , then the minimum of  $Q(x)$  over  $M$  is attained in point  $\bar{x} = \hat{x} - Hg(\hat{x})$ . For  $\bar{x}$  to be the optimal solution of the subproblem  $P(A)$ , we must have

$$g(\bar{x}) = Nu(\bar{x}), \quad (13.43)$$

where the vector of Lagrange multipliers  $u(\bar{x}) \geq 0$ . From the definitions of  $N^*$  and  $H$ , it follows that at such a point we have

$$u(\bar{x}) = N^* g(\bar{x}) \geq 0 \quad (13.44)$$

and

$$Hg(\bar{x}) = 0. \quad (13.45)$$

Observe that these conditions are necessary as well as sufficient for  $x$  to be the optimal solution of  $P(A)$ . Let us define  $r = N^* n^+$  as the Lagrange multipliers associated to the infeasible constraints. Define  $H^+$  as in (13.42) with  $N$  replaced by  $N^+$ . A similar notation is used for  $(N^+)^*$  and  $u^+$ .

The algorithm presented below follows the dual approach described above and makes use of the operators  $N^*$  and  $H$  defined in (13.41) and (13.42), respectively (Goldfarb & Idnani, 1983). An efficient implementation does not explicitly compute and does not store these operators. Instead, this implementation stores and updates the matrices  $J = Q^T L^{-1}$  and  $R$  obtained from the Cholesky and QR factorization of  $G = LL^T$  and  $L^{-1}N = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$ .

### Algorithm 13.8 Dual algorithm for quadratic programming

- |    |   |
|----|---|
| 1. | Find the unconstrained minimum of function $Q(x)$ from (13.40a). Compute $x = -G^{-1}c$ , $H = G^{-1}$ , $A = \emptyset$ , and $q = 0$  |
| 2. | Choose a violated constraint, if any, and compute $s_j(x)$ for all $j \in I \setminus A$<br>If $V = \{j \in I \setminus A : s_j(x) < 0\} = \emptyset$ , then stop; the current solution is optimal. Otherwise, choose $p \in V$ and set $n^+ = n_p$ and $u^+ = [u \ 0]^T$<br>If $q = 0$ , then set $u = 0$ . ( $A^+ = A \cup \{p\}$ ) |
| 3. | Determine the search direction.<br>Compute $z = Hn^+$ as the direction in the primal space. If $q > 0$ , then set $r = N^* n^+$ the negative of the direction in the dual space   |
| 4. | Compute the step length.<br>(i) Compute $t_1$ the maximum step in the dual space without violation of the dual feasibility. If $r \leq 0$ or $q = 0$ , then set $t_1 = \infty$ ; otherwise, set   |

$$t_1 = \min \left\{ \frac{u_j^+(x)}{r_j} : r_j > 0, j = 1, \dots, q \right\} = \frac{u_l^+(x)}{r_l}.$$

In step 5 below, the element  $k \in I$  corresponds to the  $l$ -th element from  $A$ .

(ii) Compute  $t_2$  the minimum step in the primal space such that the  $p$ -th constraint becomes feasible. If  $|z| = 0$ , then set  $t_2 = \infty$ ; otherwise, set

$$t_2 = -\frac{s_p(x)}{z^T n^+}.$$

(iii) Step length computation:  $t = \min(t_1, t_2)$

5. Determine the pair and take the step.

(i) If  $t = \infty$ , no step in the primal or dual space is executed and stop. The subproblem  $P(A^+)$  is infeasible and hence problem (13.40) is infeasible.

(ii) If  $t_2 = \infty$ , then set  $u^+ = u^+ + t \begin{bmatrix} -r \\ 1 \end{bmatrix}$  and delete the constraint  $k$ , i.e., set  $A = A \setminus \{k\}$ ,  $q = q - 1$ , update  $H$  and  $N^*$ , and go to step 3.

(iii) Step in the primal and dual space. Set  $x = x + tz$ ,  $u^+ = u^+ + t \begin{bmatrix} -r \\ 1 \end{bmatrix}$ .

If  $t = t_2$ , then set  $u = u^+$  and add the constraint  $p$  to  $A$ , that is,  $A = A \cup \{p\}$ , set  $q = q + 1$ , update  $H$  and  $N^*$ , and go to step 2.

If  $t = t_1$ , then drop the constraint  $k$ , i.e., set  $A = A \setminus \{k\}$ ,  $q = q - 1$ , update  $H$  and  $N^*$ , and go to step 3



Goldfarb and Idnani (1983) prove that the algorithm will solve the quadratic programming problem (13.40) or indicate that it has no feasible solution in a finite number of steps.

The implementation of the above algorithm in a stable manner is based on the Cholesky factorization

$$G = LL^T \quad (13.46)$$

of the symmetric and positive definite matrix  $G$  and the QR factorization

$$B = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1 \ Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix} \quad (13.47)$$

of the  $n \times q$  matrix

$$B = L^{-1}N. \quad (13.48)$$

In these relations,  $L$  is an  $n \times n$  lower triangular matrix,  $R$  is a  $q \times q$  upper triangular matrix, and  $[Q_1 \ Q_2]$  is an  $n \times n$  orthogonal matrix partitioned, so that  $Q_1$  has  $q$  columns. By substituting (13.46)–(13.48) in (13.41) and in (13.42), the operators  $H$  and  $N^*$  can be expressed as

$$H = J_2 J_2^T \quad (13.49)$$

and

$$N^* = R^{-1}J_1^T, \quad (13.50)$$

where

$$[J_1 \ J_2] = [L^{-T}Q_1 \ L^{-T}Q_2] = L^{-T}Q = J. \quad (13.51)$$

Although the algorithm is based on the QR factorization, the orthogonal matrix  $Q$  from (13.47) is not stored; instead, the matrix  $J = L^{-T}Q$  is stored and updated, as whenever  $Q$  is called by the algorithm, it appears in conjunction with  $L^{-T}$ .

In Algorithm 13.8, it is necessary to compute the vectors  $z = Hn^+$  and  $r = N^*n^+$ . If we compute

$$d = J^T n^+ = \begin{bmatrix} J_1^T \\ J_2^T \end{bmatrix} n^+ = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}, \quad (13.52)$$

then, from (13.49) and (13.50), we get

$$z = J_2 d_2, \quad r = R^{-1} d_1. \quad (13.53)$$

In the following, like in Goldfarb and Idnani (1983), let us present some details concerning the updating of the factors  $J$  and  $R$  when a constraint is added to or deleted from the active set.

**Updating the factors  $J$  and  $R$**  Whenever a constraint is added to or deleted from the set of the active constraints, the factors  $J$  and  $R$  must be updated. Numerically stable methods for updating the QR factorization of a matrix when it is modified by the addition or deletion of a single column are well known (Daniel, Graggs, Kaufman, & Stewart, 1976; Gill, Golub, Murray, & Saunders, 1974; Goldfarb, 1975). In the updating scheme described by Goldfarb and Idnani (1983), the Givens matrices  $Q_{ij}$  are used. They are equal to the identity matrix with the elements  $(i, i)$ ,  $(i, j)$ ,  $(j, i)$ , and  $(j, j)$  replaced by  $t$ ,  $s$ ,  $s$ , and  $-t$ , respectively, where  $t = \cos \theta$  and  $s = \sin \theta$  for some value of  $\theta$ . Having in view their special structure, the computations involving the Givens matrices can be illustrated by considering the  $2 \times 2$  matrix

$$\hat{Q} = \begin{bmatrix} t & s \\ s & -t \end{bmatrix}.$$

Since  $t^2 + s^2 = 1$ , it follows that  $\hat{Q}$  is orthogonal. In all the computations,  $\hat{Q}$  is chosen in such a way that the vector  $w = [w_1 \ w_2]^T$  is transformed into the vector  $[\omega \ 0]^T$ , where  $\omega = \pm(w_1^2 + w_2^2)^{1/2}$ . To accomplish this, the following elements are computed:

$$\mu = \max\{|w_1|, |w_2|\}, \quad \omega = \text{sign}(w_1)\mu \left[ \left(\frac{w_1}{\mu}\right)^2 + \left(\frac{w_2}{\mu}\right)^2 \right]^{1/2},$$

$$t = \frac{w_1}{\omega} \quad \text{and} \quad s = \frac{w_2}{\omega}.$$

To compute  $\hat{y} = \hat{Q}y = \hat{Q}^T y$ , where  $y = [y_1 \ y_2]^T$ , the algorithm follows the scheme given in Gill, Golub, Murray, and Saunders (1974) and determines

$$\nu = \frac{w_2}{\omega + w_1} = \frac{s}{1 + t},$$

and

$$\hat{y}_1 = ty_1 + sy_2, \quad \hat{y}_2 = \nu(y_1 + \hat{y}_1) - y_2. \quad (13.54)$$

Observe that if  $\hat{Q}$  (or  $\hat{Q}^T$ ) is applied to a  $2 \times n$  matrix, then the above computing scheme saves  $n - 1$  multiplications, but introduces  $n + 1$  additions over the usual procedure for matrix multiplications. Also, observe that the sign of  $\omega$  is chosen so that there is no cancelation in the computation of  $\nu$ .

**Adding a constraint** When the constraint  $p$  with the normal  $n^+$  is added to the active set  $A$ , then the factorization (13.47) is replaced by

$$B^+ = [B \quad L^{-1}n^+] = Q^+ \begin{bmatrix} R^+ \\ 0 \end{bmatrix}. \quad (13.55)$$

Therefore, from (13.38), (13.51), and (13.52), it follows that

$$Q^T B^+ = \begin{bmatrix} R & d_1 \\ 0 & d_2 \end{bmatrix}.$$

Thus, the factorization (13.55) can be obtained as

$$Q^+ = Q \begin{bmatrix} I_q & 0 \\ 0 & \bar{Q}^T \end{bmatrix} \text{ and } R^+ = \begin{bmatrix} R & d_1 \\ 0 & \delta \end{bmatrix},$$

where  $\delta = \|d_2\|$  and  $\bar{Q} = Q_{1,2}Q_{2,3}\cdots Q_{n-q-1,n-q}$  is the product of the Givens matrices chosen, so that

$$\bar{Q}d_2 = \pm \delta e_1. \quad (13.56)$$

Moreover,

$$J^+ = L^{-T}Q^+ = \begin{bmatrix} J_1 & J_2 \bar{Q}^T \end{bmatrix} = \begin{bmatrix} J_1^+ & J_2^+ \end{bmatrix}, \quad (13.57)$$

where  $J_1$  has  $q$  columns,  $J_2 \bar{Q}^T$  has  $n - q$  columns,  $J_1^+$  has  $q + 1$  columns, and  $J_2^+$  has  $n - q - 1$  columns.

**Dropping a constraint** When the  $l$ -th constraint is deleted from the active set, that is, when the  $l$ -th column of  $N$  is deleted to form  $N^+$ , then

$$Q_1^T B^- = Q_1^T L^{-1} N^- = \begin{bmatrix} R_1 & S \\ 0 & T \end{bmatrix},$$

where the partitioned matrix on the right is equal to  $R$  but with its  $l$ -th column eliminated. If  $l \neq q$ , then  $T$  is a  $(q - l + 1) \times (q - l)$  upper-Hessenberg matrix. Again, a sequence of  $q - l$  Givens matrices can be chosen, so that the product of these matrices  $\bar{Q} = Q_{q-l,q-l+1}\cdots Q_{2,3}Q_{1,2}$  reduces  $T$  to an upper triangular matrix  $R_2$ , i.e.,

$$\bar{Q}T = R_2. \quad (13.58)$$

Thus, the new factors are obtained as

$$R^- = \begin{bmatrix} R_1 & S \\ 0 & R_2 \end{bmatrix} \quad \text{and} \quad J^- = J \begin{bmatrix} I_{l-1} & 0 & 0 \\ 0 & \bar{Q}^T & 0 \\ 0 & 0 & I_{n-q} \end{bmatrix}. \quad (13.59)$$

The matrices  $\bar{Q}$  from (13.56) and (13.58) are not computed. Instead, the Givens matrices from which they are formed, which successively introduce zeros into  $d_2$  and  $T$ , are applied directly to the rows of  $J^T$  using the computational scheme (13.55).

When the step directions  $z$  for the primal variables and  $-r$  for the dual variables given by (13.53) are computed, we need the vector  $d = J^T n^+$ . When a constraint is deleted from the active set, then the same orthogonal transformations that are applied to update  $J^T$  can be applied to  $d$ . If these operations are executed, then the updated  $d$  can be used to compute the new vectors  $z$  and  $r$  after the basis change and to determine the appropriate orthogonal transformation  $\bar{Q}$  in (13.56) needed when the  $p$ -th constraint is finally added to the active set.

Algorithm 13.8 (dual for quadratic programming) is one of the most advanced algorithms for solving quadratic programming problems with equality and inequality constraints. Its efficiency comes from two sources: the dual method and the implementation of an advanced computational scheme based on the QR factorization which is updated by the Givens matrices. One of the first implementations of Algorithm 13.8 was given by Powell (1983) (Fortran package ZQPCVX). A professional extension was given by Schittkowski (2005) (Fortran package QLD).

**Example 13.2** Let us consider the following quadratic programming problem (Andrei, 1999a, p. 792, 2003, p. 329):

$$\min Q(x) \equiv \sum_{i=1}^{15} x_i^2 + \sum_{i=1}^{14} (x_i + x_{i+1})$$

subject to

$$\begin{aligned} 0.1x_1 + 0.1x_7 + 0.3x_8 + 0.2x_9 + 0.2x_{11} - 1 &= 0, \\ 0.1x_2 + 0.2x_8 + 0.3x_9 + 0.4x_{10} + x_{11} - 2 &= 0, \\ 0.1x_3 + 0.2x_8 + 0.3x_9 + 0.4x_{10} + 2x_{11} - 3 &= 0, \\ x_4 + x_8 + 0.5x_9 + 0.5x_{10} + x_{11} - x_{12} - 3 &= 0, \\ 2x_5 + x_6 + 0.5x_7 + 0.5x_8 + 0.25x_9 + 0.25x_{10} + 0.5x_{11} - x_{13} - 4 &= 0, \\ x_4 + x_6 + x_8 + x_9 + x_{10} + x_{11} - x_{14} - 5 &= 0, \\ 0.1x_1 + 1.2x_7 + 1.2x_8 + 1.4x_9 + 1.1x_{10} + 2x_{11} - x_{15} - 6 &= 0, \\ 0 \leq x_i \leq 2, \quad i = 1, \dots, 15. \end{aligned}$$

Table 13.1 contains information on the optimization process given by Algorithm 13.8 in the QLD implementation by Schittkowski (2005). To get a solution, the algorithm needs eight iterations. In Table 13.1, we can see the evolution of the values of the objective function along the iterations. Also

**Table 13.1** Optimization process by QLD

#k	$Q(x_k)$	svr	#nra	t	kkt
1	97.000000	27.800000	7	0	68.760324
2	46.471283	0.266453e-14	7	1	17.341127
3	31.302619	0.155431e-14	7	1	10.560131
4	22.807444	0	7	1	0.740935
5	22.179315	0.199840e-14	7	1	0.168103
6	22.037532	0.666133e-15	7	1	0.232076
7	21.921296	0.111022e-15	7	1	0.670174e-06
8	21.921296	0.888178e-15	7	1	0.303832e-14

In this table, we have  $\#k$  = the number of iterations,  $Q(x_k)$  = the value of the objective function, svr = the sum of the constraints violation,  $\#nra$  = the number of the active constraints,  $t$  = the stepsize, and kkt = the value of the KKT conditions at  $x_k$

**Table 13.2** Initial point, solution, and bounds on variables

	$l$	$x_0$	$x^*$	$u$
1	0	2.1	0.400273	2
2	0	2.1	0.130566	2
3	0	2.1	0	2
4	0	2.1	0	2
5	0	2.1	0.903379	2
6	0	2.1	0.416858	2
7	0	2.1	0	2
8	0	2.1	1.509334	2
9	0	2.1	1.522805	2
10	0	2.1	0.537945	2
11	0	2.1	1.013057	2
12	0	1	0.552766	2
13	0	1	0	2
14	0	1	0	2
15	0	1	0.601009	2

notice the evolution of the sum of the constraints violation, which tends to zero, as well as the evolution of the norm of the KKT conditions. The number of the active constraints is 7; the problem has only equality constraints. The stepsize is equal to 1 along the iterations, like in the Newton method from the unconstrained optimization.

In Table 13.2, we can note the initial point, the bounds on variables, and the solution given by QLD. Observe that the initial point is not feasible.

**Application Q1. DYNAMIC** This application is a representation of the problem concerning the scheduling of three generators to meet the demand for power over a given period of time. The variables  $x_{3k+i}$  denote the output from the generator  $i$  at time  $t_k$ . The constraints of the problem are upper and lower limits on the power available from each generator, bounds on the amount by which the output from a generator can change from time  $t_k$  to  $t_{k+1}$ , as well as the condition that at each time  $t_k$ , the power generated must at least satisfy the demand. The mathematical model is described in Pant, Thangaraj, and Singh (2009) and is as follows:

$$\min \sum_{k=0}^4 (2.3x_{3k+1} + 0.0001x_{3k+1}^2 + 1.7x_{3k+2} + 0.0001x_{3k+2}^2 + 2.2x_{3k+3} + 0.00015x_{3k+3}^2)$$

subject to

$$x_1 + x_2 + x_3 \geq 60,$$

$$x_4 + x_5 + x_6 \geq 50,$$

$$x_7 + x_8 + x_9 \geq 70,$$

$$x_{10} + x_{11} + x_{12} \geq 85,$$

$$x_{13} + x_{14} + x_{15} \geq 100,$$

$$-7 \leq x_{3k+1} - x_{3k-2} \leq 6, \quad k = 1, \dots, 4,$$

$$-7 \leq x_{3k+2} - x_{3k-1} \leq 7, \quad k = 1, \dots, 4,$$

$$-7 \leq x_{3k+3} - x_{3k} \leq 6, \quad k = 1, \dots, 4,$$

**Table 13.3** Dynamic Optimization process by QLD

#k	$Q(x_k)$	svr	#nra	t	kkt
1	1228.3246	0.42e+02	29	0	0.12e+04
2	1115.8453	0	6	1	0.11e+03
3	1006.9820	0	4	1	0.32e+03
4	762.61922	0.11e-13	4	1	0.76e+02
5	708.25997	0.89e-14	8	1	0.26e+02
6	692.67975	0.71e-14	9	1	0.15e+01
7	691.18494	0.36e-14	8	1	0.70e+01
8	684.96921	0	8	1	0.11e+02
9	674.81722	0.18e-13	8	1	0.48e+01
10	670.99302	0.67e-15	9	1	0.28e+01
11	668.92085	0.75e-14	8	1	0.21e+01
12	667.49688	0.53e-14	8	1	0.64e+00
13	666.86138	0.12e-13	7	1	0.32e+01
14	663.69246	0.18e-14	7	1	0.12e+02
15	655.26305	0.18e-14	10	1	0.48e+00
16	654.78703	0.53e-14	10	1	0.78e+00
17	654.12603	0	10	1	0.12e+01
18	653.44270	0	11	1	0.20e-30

The variables are bounded as follows:

$$8 \leq x_1 \leq 21, \quad 0 \leq x_{3k+1} \leq 90, \quad k = 1, \dots, 4,$$

$$43 \leq x_2 \leq 57, \quad 0 \leq x_{3k+2} \leq 120, \quad k = 1, \dots, 4,$$

$$3 \leq x_3 \leq 16, \quad 0 \leq x_{3k+3} \leq 60, \quad k = 1, \dots, 4.$$

Considering the initial point  $x_i^0 = (l_i + u_i)/2$ ,  $i = 1, \dots, 15$ , Algorithm 13.6 gives the following solution:

$$\begin{aligned} x_1^* &= 8, & x_2^* &= 49, & x_3^* &= 3, & x_4^* &= 1, & x_5^* &= 52, \\ x_6^* &= 0, & x_7^* &= 5, & x_8^* &= 59, & x_9^* &= 6, & x_{10}^* &= 7, \\ x_{11}^* &= 66, & x_{12}^* &= 12, & x_{13}^* &= 0, & x_{14}^* &= 95, & x_{15}^* &= 5. \end{aligned}$$

The value of the objective function at this point is  $Q(x^*) = 653.4427$ .

Table 13.3 gives some information about the optimization process with QLD. To get the above solution, the algorithm needs 18 iterations.

### 13.3 Interior Point Methods

In the following, let us present the interior point methods for solving quadratic programming problems. We consider the convex quadratic programs with inequality constraints (Nocedal & Wright, 2006):

$$\begin{aligned} \min_x q(x) &= \frac{1}{2} x^T G x + c^T x, \\ \text{subject to} \\ Ax &\geq b, \end{aligned} \tag{13.60}$$

where  $G$  is symmetric and positive semidefinite and  $A = [a_1, \dots, a_m]^T \in \mathbb{R}^{m \times n}$  and  $b = [b_1, \dots, b_m]^T \in \mathbb{R}^m$ . The KKT optimality conditions for (13.60) are as follows:

$$\begin{aligned} Gx - A^T \lambda + c &= 0, \\ Ax - b &\geq 0, \\ (Ax - b)_i \lambda_i &= 0, \quad i = 1, \dots, m, \\ \lambda &\geq 0. \end{aligned}$$

These conditions may be written as equalities by introducing the slack vector  $y \geq 0$ :

$$Gx - A^T \lambda + c = 0, \tag{13.61a}$$

$$Ax - y - b = 0, \tag{13.61b}$$

$$(Ax - b)_i \lambda_i = 0, \quad i = 1, \dots, m, \tag{13.61c}$$

$$\lambda \geq 0. \tag{13.61d}$$

Observe that the matrix  $G$  is positive semidefinite. Therefore, these KKT conditions are both necessary and sufficient, and consequently, the convex quadratic problem (13.60) can be solved by finding the solutions of the system (13.61).

For this, let us consider the current iterate  $(x, y, \lambda)$  that satisfies  $y > 0$  and  $\lambda > 0$ . Define the *complementarity measure*  $\mu$  as

$$\mu = \frac{y^T \lambda}{m}. \tag{13.62}$$

With this, we define the *path-following primal-dual methods* by considering the *perturbed* KKT conditions

$$F(x, y, \lambda; \sigma\mu) = \begin{bmatrix} Gx - A^T \lambda + c \\ Ax - y - b \\ Y\Lambda e - \sigma\mu e \end{bmatrix} = 0, \tag{13.63}$$

where

$$Y = \text{diag}(y_1, \dots, y_m), \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_m), \quad e = (1, 1, \dots, 1)^T \in \mathbb{R}^m$$

and  $\sigma \in [0, 1]$ . The solutions of (13.63) for all positive values of  $\sigma$  and  $\mu$  define the *central path*, which is a trajectory that leads to the solution of the quadratic problem (13.60) as  $\sigma\mu$  tends to zero.

Now, fixing  $\mu$  and applying the Newton method to (13.63), the following linear system is obtained:

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e + \sigma\mu e \end{bmatrix}, \tag{13.64}$$

where

$$r_d = Gx - A^T\lambda + c, \quad r_p = Ax - y - b. \quad (13.65)$$

The next iteration  $(x^+, y^+, \lambda^+)$  is obtained as

$$(x^+, y^+, \lambda^+) = (x, y, \lambda) + \alpha(\Delta x, \Delta y, \Delta \lambda), \quad (13.66)$$

where  $\alpha$  is chosen in such a way that  $y^+ > 0$  and  $\lambda^+ > 0$ .

Observe that the major computational operation in the interior point method is the solution of the system (13.64). However, by eliminating  $\Delta y = A\Delta x + r_p$  from (13.64), this system may be restated in a more compact form as

$$\begin{bmatrix} G & -A^T \\ A & \Lambda^{-1}Y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p + (-y + \sigma\mu\Lambda^{-1}e) \end{bmatrix}. \quad (13.67)$$

From (13.67), by eliminating  $\Delta \lambda = Y^{-1}\Lambda(-r_p - y + \sigma\mu\Lambda^{-1}e - A\Delta x)$ , a more compact form is obtained as

$$(G + A^T Y^{-1} \Lambda A) \Delta x = -r_d + A^T Y^{-1} \Lambda (-r_p - y + \sigma\mu\Lambda^{-1}e), \quad (13.68)$$

which can be solved by a modified Cholesky algorithm (see Appendix A) to get  $\Delta x$ .

## Stepsize Selection

Like in Nocedal and Wright (2006), suppose that the new iterate is defined as

$$(x^+, y^+) = (x, y) + \alpha^{prim}(\Delta x, \Delta y), \quad (13.69a)$$

$$\lambda^+ = \lambda + \alpha^{dual} \Delta \lambda, \quad (13.69b)$$

where  $\alpha^{prim}$  and  $\alpha^{dual}$  are the stepsizes that ensure the positivity of  $x^+$  and  $\lambda^+$ . From (13.64) and (13.65), it follows that the new residuals satisfy the following relations:

$$r_p^+ = (1 - \alpha^{prim})r_p, \quad (13.70a)$$

$$r_d^+ = (1 - \alpha^{dual})r_d + (\alpha^{prim} - \alpha^{dual})G\Delta x. \quad (13.70b)$$

If  $\alpha^{prim} = \alpha^{dual} = \alpha$ , then  $r_p^+ = (1 - \alpha)r_p$  and  $r_d^+ = (1 - \alpha)r_d$ , that is, both residuals decrease linearly for all  $\alpha \in (0, 1)$ . For different stepsizes, the residual  $r_d^+$  may increase for certain values of  $\alpha^{prim}$  and  $\alpha^{dual}$ . Therefore, the interior point algorithms use equal stepsizes, as in (13.66), and set  $\alpha = \min \{\alpha_\tau^{prim}, \alpha_\tau^{dual}\}$ , where

$$\alpha_\tau^{prim} = \max \{\alpha \in (0, 1] : y + \alpha \Delta y \geq (1 - \tau)y\}, \quad (13.71)$$

$$\alpha_\tau^{dual} = \max \{\alpha \in (0, 1] : \lambda + \alpha \Delta \lambda \geq (1 - \tau)\lambda\}. \quad (13.72)$$

The interior point algorithm for convex quadratic programming is based on the Mehrotra predictor-corrector method, originally developed for linear programming (Mehrotra, 1992). Firstly, an affine scaling step  $(\Delta x^{aff}, \Delta y^{aff}, \Delta \lambda^{aff})$  is computed by setting  $\sigma = 0$  in (13.64). Next, this step is improved by computing a corrector step and the centering parameter  $\sigma = (\mu^{aff}/\mu)^3$ , where

$\mu^{aff} = \left( y + \hat{\alpha}^{aff} \Delta y^{aff} \right)^T \left( \lambda + \hat{\alpha}^{aff} \Delta \lambda^{aff} \right) / m$ . The total step is obtained by solving the following linear system:

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e - \Delta \Lambda^{aff} \Delta Y^{aff} + \sigma \mu e \end{bmatrix}. \quad (13.73)$$

**Algorithm 13.9 Predictor-corrector algorithm for quadratic programming**

- |     |  |
|-----|--|
| 1.  | Compute $(x_0, y_0, \lambda_0)$ such that $y_0 > 0$ and $\lambda_0 > 0$ . Set $k = 0$  |
| 2.  | Test a criterion for stopping the iterations   |
| 3.  | Set $\sigma = 0$ . Set $(x, y, \lambda) = (x_k, y_k, \lambda_k)$ and solve (13.64) subject to $(\Delta x^{aff}, \Delta y^{aff}, \Delta \lambda^{aff})$ |
| 4.  | Compute $\mu = y^T \lambda / m$  |
| 5.  | Compute $\hat{\alpha}^{aff} = \max \{ \alpha \in (0, 1] : (y, \lambda) + \alpha (\Delta y^{aff}, \Delta \lambda^{aff}) \geq 0 \}$                      |
| 6.  | Compute $\mu^{aff} = \left( y + \hat{\alpha}^{aff} \Delta y^{aff} \right)^T \left( \lambda + \hat{\alpha}^{aff} \Delta \lambda^{aff} \right) / m$      |
| 7.  | Set the centering parameter to $\sigma = (\mu^{aff}/\mu)^3$  |
| 8.  | Solve (13.73) subject to $(\Delta x, \Delta y, \Delta \lambda)$  |
| 9.  | Choose $\tau_k \in (0, 1)$ and set $\hat{\alpha} = \min \{ \alpha_{\tau_k}^{prim}, \alpha_{\tau_k}^{dual} \}$  |
| 10. | Set $(x_{k+1}, y_{k+1}, \lambda_{k+1}) = (x_k, y_k, \lambda_k) + \hat{\alpha} (\Delta x, \Delta y, \Delta \lambda)$                                    |
| 11. | Set $k = k + 1$ and go to step 2   |

◆

To accelerate the convergence,  $\tau_k$  can be selected to approach 1 as the iterates approach the solution. The performances of the algorithm are dependent on the initial point. A simple procedure for selecting the initial point is as follows. Firstly, an initial point  $(\bar{x}, \bar{y}, \bar{\lambda})$  is defined and moved far enough from the boundary of the region  $(y, \lambda) \geq 0$  to permit the algorithm to take long steps on the early iterations. Next, the affine scaling step  $(\Delta x^{aff}, \Delta y^{aff}, \Delta \lambda^{aff})$  from the supplied initial point  $(\bar{x}, \bar{y}, \bar{\lambda})$  is computed. Then,  $x_0 = \bar{x}$ ,  $y_0 = \max \{ 1, |y + \Delta y^{aff}| \}$ ,  $\lambda_0 = \max \{ 1, |\bar{\lambda} + \Delta \lambda^{aff}| \}$  are set, where max and the absolute value are applied component wise.

Some comparisons between the active-set and the interior point methods are as follows. The active-set methods require a large number of iterations, in which each search direction is relatively inexpensive to compute. On the other hand, the interior point methods take a smaller number of iterations, each of them being more expensive. Active-set methods are complicated to implement in computing programs, especially if the procedures for updating the matrix factorizations try to take advantage of the sparsity or of the structure of  $G$  and  $A$ . In case of interior point methods, the nonzero structure of the matrix to be factored at each iteration remains the same at all iterations. For very large problems, the interior point methods are more efficient. However, if an estimate of the solution is available, then the active-set methods may rapidly converge in just a few iterations, particularly if the initial value of  $x$  is feasible.

## 13.4 Methods for Convex QP Problems with Equality Constraints

Consider the convex equality constrained quadratic programming problem

$$\begin{aligned} \min q(x) &\equiv \frac{1}{2}x^T Gx + c^T x \\ \text{subject to} \\ Ax &= b, \end{aligned} \tag{13.74}$$

where  $G \in \mathbb{R}^{n \times n}$  is a symmetric and positive definite matrix and  $A \in \mathbb{R}^{m \times n}$  is of full row rank  $m$ , where  $m < n$ . For solving this problem, the following methods can be presented:

### 1. Method Based on KKT Conditions

The first-order necessary optimality conditions for (13.74) are given by

$$Gx^* + c - A^T \lambda^* = 0, \tag{13.75a}$$

$$-Ax^* + b = 0. \tag{13.75b}$$

If  $G$  is positive definite and  $A$  has full row rank, then the matrix

$$\begin{bmatrix} G & -A^T \\ -A & 0 \end{bmatrix}$$

is nonsingular, and the solution  $x^*$  of (13.75) is the unique global minimizer of the convex equality quadratic programming problem (13.74). Therefore, the solution  $x^*$  and the Lagrange multipliers  $\lambda^*$  can be expressed as

$$\lambda^* = (AG^{-1}A^T)^{-1}(AG^{-1}c + b), \tag{13.76a}$$

$$x^* = G^{-1}(A^T\lambda^* - c). \tag{13.76b}$$

However, the solution of the symmetric system (13.75) can be obtained by using numerical methods that are often more efficient and more reliable than the formulae (13.76) (Golub, & Van Loan, 1996).

### 2. Methods Based on Singular Value Decomposition of $A$

Indeed, the solutions of the system  $Ax = b$  from (13.74) have the form  $x = A^+b + V_r z$ , where  $V_r$  is composed of the last  $n - m$  columns of the matrix  $V$ , and  $V$  is obtained from the singular value decomposition (SVD) of  $A$ , namely,  $U\Sigma V^T$  (see Appendix A). With this, using this form of  $x$  in (13.74), the constraints can be eliminated, thus obtaining an unconstrained minimization problem

$$\min \hat{q}(z) = \frac{1}{2}z^T \hat{G}z + \hat{c}^T z, \tag{13.77}$$

where

$$\hat{G} = V_r^T G V_r \quad \text{and} \quad \hat{c} = V_r^T (GA^+b + c). \tag{13.78}$$

If  $G$  is positive definite, then  $\hat{G}$  is also positive definite, and the unique global minimizer of (13.74) is given by

$$x^* = V_r z^* + A^+ b, \quad (13.79)$$

where  $z^*$  is a solution of the linear system equations  $\hat{G}z = -\hat{c}$ .

If  $G$  is positive semidefinite, then  $\hat{G}$  may be either positive definite or positive semidefinite. If  $\hat{G}$  is positive definite, then  $x^*$  given by (13.79) is the unique global minimizer of problem (13.74). If  $\hat{G}$  is positive semidefinite, then there are two possibilities: (a) If  $\hat{c}$  can be expressed as a linear combination of the columns of  $\hat{G}$ , then  $\hat{q}(z)$  has infinitely many global minimizers and so does  $q(x)$ . (b) If  $\hat{c}$  is not a linear combination of the columns of  $\hat{G}$ , then  $\hat{q}(z)$  and also  $q(x)$  have no minimizers.

### 3. Methods Based on QR Decomposition

This method is more economical and is based on the QR decomposition of  $A^T$ , i.e.,

$$A^T = Q \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where  $Q \in \mathbb{R}^{n \times n}$  is an orthogonal matrix and  $R \in \mathbb{R}^{m \times m}$  is an upper triangular matrix (see Appendix A). Therefore, the constraints  $Ax = b$  can be expressed as  $R^T \hat{x}_1 = b$ , where  $\hat{x}_1$  is the vector composed of the first  $m$  components of  $\hat{x}$  with  $\hat{x} = Q^T x$ . Let us denote

$$\hat{x} = \begin{bmatrix} \hat{x}_1 \\ z \end{bmatrix} \quad \text{and} \quad Q = [Q_1 \ Q_2],$$

where  $z \in \mathbb{R}^{n-m}$ ,  $Q_1 \in \mathbb{R}^{n \times m}$ , and  $Q_2 \in \mathbb{R}^{n \times (n-m)}$ . Then,

$$x = Q\hat{x} = Q_1\hat{x}_1 + Q_2z = Q_2z + Q_1R^{-T}b, \quad (13.80)$$

which is equivalent to (13.79).

Therefore, the parameterized solutions (13.80) can be used to convert problem (13.74) to the reduced-size unconstrained minimization problem (13.77), where, this time

$$\hat{G} = Q_2^T G Q_2 \quad \text{and} \quad \hat{c} = Q_2^T (G Q_1 R^{-T} b + c). \quad (13.81)$$

If  $G$  is positive definite, the unique global minimizer of (13.74) can be obtained as

$$x^* = Q_2 z^* + Q_1 R^{-T} b,$$

where  $z^*$  is a solution of the linear system equations  $\hat{G}z = -\hat{c}$ , where  $\hat{G}$  and  $\hat{c}$  are given by (13.81). Of course, in both approaches presented above if  $\hat{G}$  is positive definite, then the linear system of equations  $\hat{G}z = -\hat{c}$  can be efficiently solved by the Cholesky factorization.

## 13.5 Quadratic Programming with Simple Bounds: The Gradient Projection Method

In the following, let us discuss the minimization of the quadratics in which the variables are simple bounded by using the gradient projection approach. This is a very nice application of the optimization with simple bounds on variables, presented in Chap. 12. Unlike the active-set method, the gradient projection method allows the active set to rapidly change from iteration to iteration, thus reducing the number of iterations for getting a solution. Consider the following problem:

$$\begin{aligned} \min q(x) &= \frac{1}{2}x^T G x + c^T x \\ \text{subject to} \\ l \leq x \leq u, \end{aligned} \tag{13.82}$$

where  $G$  is a symmetric matrix and  $l$  and  $u$  are vectors of lower and upper bounds on variables. The feasible region defined by  $l \leq x \leq u$  is called a *box*, because of its rectangular shape. Assume that  $l_i < u_i$  for all  $i$ . It is not necessary to make any positive definiteness assumptions on  $G$ , because the gradient projection method can be applied for both convex and nonconvex problems.

Every iteration of the gradient projection method has two stages. In the first one, from the current point  $x$ , searching is executed along the steepest descent direction, that is, along the direction  $-Gx - c$ . Whenever a bound is encountered, the search direction is bent so that it stays feasible. Therefore, the search continues along a piecewise-linear path until the first local minimizer of  $q$  denoted by  $x^c$  and referred to as the Cauchy point is located. The working set is defined as the set of the bounds constraints that are active at the Cauchy point, denoted by  $A(x^c)$ . In the second stage, a face of the feasible box on which the Cauchy point lies is inspected by solving a subproblem in which the active components  $x_i, i \in A(x^c)$  are fixed at the values  $x_i^c$ .

## The Cauchy Point

The projection of an arbitrary point  $x$  on the feasible region  $l \leq x \leq u$  is defined as a vector whose  $i$ -th component is

$$P(x, l, u)_i = \begin{cases} l_i, & \text{if } x_i < l_i, \\ x_i & \text{if } l_i \leq x_i \leq u_i, \\ u_i & \text{if } x_i > u_i, \end{cases} \tag{13.83}$$

Starting from the current point  $x$ , the piecewise-linear path  $x(t)$  obtained by projecting the steepest descent direction at  $x$  onto the feasible region  $l \leq x \leq u$  is given by

$$x(t) = P(x - tg, l, u), \tag{13.84}$$

where  $g = Gx + c$ . The Cauchy point  $x^c$  is defined as the first local minimizer of the piecewise-quadratic function  $q(x(t))$ , for  $t \geq 0$ . Every line segments that define  $x(t)$  is examined to determine this minimizer point. For this, the values of  $t$  for which each component reaches its bound along  $-g$  are determined. These values  $\bar{t}_i$  are computed as

$$\bar{t}_i = \begin{cases} (x_i - u_i)/g_i, & \text{if } g_i < 0 \text{ and } u_i < +\infty, \\ (x_i - l_i)/g_i, & \text{if } g_i > 0 \text{ and } l_i > +\infty, \\ \infty & \text{otherwise.} \end{cases} \tag{13.85}$$

Therefore, for any  $t$ , the components of  $x(t)$  are as follows:

$$x_i(t) = \begin{cases} x_i - tg_i, & \text{if } t \leq \bar{t}_i, \\ x_i - \bar{t}_i g_i, & \text{otherwise.} \end{cases} \tag{13.86}$$

To determine the first local minimizer along  $P(x - tg, l, u)$ , first eliminate the duplicate values and the zero of  $\bar{t}_i$  from the set  $\{\bar{t}_1, \dots, \bar{t}_n\}$ . Secondly, all these values are sorted, thus obtaining the

reduced set of breakpoints  $\{t_1, \dots, t_l\}$  with  $0 < t_1 < \dots < t_l$ . After that, the intervals  $[0, t_1]$ ,  $[t_1, t_2], \dots$  are examined in this order. Suppose that we have examined these intervals up to  $t_{j-1}$  and no local minimizer has been found. For the interval  $[t_{j-1}, t_j]$ , it follows that

$$x(t) = x(t_{j-1}) + (\Delta t)d^{j-1},$$

where

$$\Delta t = t - t_{j-1} \in [0, t_j - t_{j-1}], \quad (13.87)$$

$$d_i^{j-1} = \begin{cases} -g_i, & \text{if } t_{j-1} < \bar{t}_i, \\ 0, & \text{otherwise.} \end{cases} \quad (13.88)$$

Therefore, on the line segment  $[x(t_{j-1}), x(t_j)]$ , (13.82) can be written as

$$q(x(t)) = c^T(x(t_{j-1}) + (\Delta t)d^{j-1}) + \frac{1}{2}(x(t_{j-1}) + (\Delta t)d^{j-1})^T G(x(t_{j-1}) + (\Delta t)d^{j-1}),$$

which can be rewritten as

$$q(x(t)) = f_{j-1} + f'_{j-1}\Delta t + \frac{1}{2}f''_{j-1}(\Delta t)^2, \quad (13.89)$$

where  $\Delta t \in [0, t_j - t_{j-1}]$ , and the coefficients  $f_{j-1}$ ,  $f'_{j-1}$  and  $f''_{j-1}$  are given by

$$\begin{aligned} f_{j-1} &\triangleq c^T x(t_{j-1}) + \frac{1}{2} x(t_{j-1})^T G x(t_{j-1}), \\ f'_{j-1} &\triangleq c^T d^{j-1} + x(t_{j-1})^T G d^{j-1}, \\ f''_{j-1} &\triangleq (d^{j-1})^T G d^{j-1}. \end{aligned}$$

To determine  $\Delta t^*$ , the minimum of (13.89), we differentiate  $q(x(t))$  with respect to  $\Delta t$  and set  $q'(x(t)) = 0$  to obtain  $\Delta t^* = -f'_{j-1}/f''_{j-1}$ . The following three cases can occur. (i) If  $f'_{j-1} > 0$ , then there is a local minimizer of  $q(x(t))$  at  $t = t_{j-1}$ . (ii) If  $\Delta t^* \in [0, t_j - t_{j-1}]$ , then there is a minimizer at  $t = t_{j-1} + \Delta t^*$ . (iii) In all the other cases, continue the search with the next interval  $[t_j, t_{j+1}]$ . Obviously, for the next interval, the new direction  $d^j$  from (13.88) needs to be computed, with which the new values  $f_j$ ,  $f'_j$ , and  $f''_j$  are computed, etc.

## Subproblem Minimization

After the Cauchy point  $x^c$  has been computed, the active set  $A(x^c) = \{i : x_i^c = l_i, \text{ or } x_i^c = u_i\}$  is determined. As we said, in the second stage of the gradient projection method, the quadratic programming problem obtained by fixing the components  $x_i$  for  $i \in A(x^c)$  at the values  $x_i^c$  is solved. That is, the remaining components  $x_i$  for  $i \notin A(x^c)$  are determined as solution of the subproblem

$$\begin{aligned} \min q(x) &= \frac{1}{2} x^T G x + c^T x \\ \text{subject to} \\ l_i \leq x_i &\leq u_i, \quad i \notin A(x^c) \\ x_i = x_i^c, &\quad i \in A(x^c) \end{aligned} \tag{13.90}$$

Obviously, (13.90) does not need to be solved exactly. To obtain the global convergence of the gradient projection algorithm, it suffices to determine an approximate solution  $x^a$  of (13.90), which satisfies the bounds and has an objective value  $q(x^a) \leq q(x^c)$ . For this, Algorithm 13.2 (preconditioned conjugate gradient for reduced systems) or Algorithm 13.3 (projected conjugate gradient) can be used. Having in view the equalities constraints from (13.90), it follows that the Jacobian and the null-space basis matrix  $Z$  used in Algorithm 13.2 have simple forms. The idea is to apply the conjugate gradient to minimize  $q(x)$  from (13.90) subject to the equality constraints  $x_i = x_i^c$ ,  $i \in A(x^c)$  and to terminate as soon as a bound  $l \leq x \leq u$  is encountered. Another strategy is to continue the iterations by ignoring the bounds and finally to project the obtained solution onto the box constraints.

The gradient projection algorithm for quadratic programming with simple bounds can be presented as follows.

**Algorithm 13.10** *Gradient projection for quadratic programming with box constraints*

1.	Compute a feasible initial point $x_0$
2.	For $k = 0, 1, \dots$ <ul style="list-style-type: none"> <li>(a) If <math>x_k</math> satisfies the KKT optimality conditions for (13.82), stop with solution <math>x^* = x_k</math>; otherwise, set <math>x = x_k</math> and determine the Cauchy point <math>x^c</math></li> <li>(b) Find an approximate solution <math>x^a</math> of (13.90) such that <math>q(x^a) \leq q(x^c)</math> and <math>x^a</math> is feasible</li> <li>(c) Set <math>x_{k+1} = x^a</math></li> </ul> End for

◆

If the strict complementarity holds (Definition 11.23), that is, the Lagrange multipliers associated with all the active bounds are nonzero, then the active sets  $A(x^c)$  generated by Algorithm 13.10 are equal to the optimal active set for all  $k$  sufficiently large.

Of course, Algorithm 13.10 can be applied for minimizing the quadratic forms subject to linear inequality constraints (13.60)  $a_i^T x \geq b_i$ ,  $i = 1, \dots, m$ . However, in this case, the difficulty is to perform the projection on the feasible set. For this, to compute the projection of a given point  $x_k$  onto the set defined by  $a_i^T x \geq b_i$ ,  $i = 1, \dots, m$ , the following convex quadratic programming problem

$$\max_x \|x - x_k\|^2 \text{ subject to } a_i^T x \geq b_i, \quad i = 1, \dots, m,$$

has to be solved. However, this is a difficult problem; the cost of its solving may approach the cost of solving the original problem. Therefore, for minimizing quadratic forms subject to linear inequality constraints, it is not advisable to apply the gradient projection algorithm.

---

## 13.6 Elimination of Variables

Often, quadratic programming with equality constraints can be solved by eliminating the variables. In this method, the equality constraints are used to eliminate the variables, thus obtaining another simplified quadratic programming problem with a smaller number of variables. Consider the problem

$$\begin{aligned} \min q(x) &\equiv \frac{1}{2} x^T G x + c^T x \\ \text{subject to} \\ Ax + b &= 0, \end{aligned} \tag{13.91}$$

where  $G$  is an  $n \times n$  symmetric matrix,  $A \in \mathbb{R}^{m \times n}$  with  $\text{rank}(A) = m$  and  $m < n$ . The method of eliminating the variables can be implemented in different manners, as follows:

### 1. The Elimination of Variables Method

Suppose that for the problem (13.91),  $\text{rank}(A) = m$ , that is,  $A$  has full column rank, where  $m < n$ . Then, the matrix  $A$  can be partitioned as  $A = [B \ N]$ , where  $B \in \mathbb{R}^{m \times m}$  is a nonsingular submatrix. Therefore, the system of constraints  $Ax + b = 0$  can be written as  $Bx_B + Nx_N + b = 0$ , where  $x = [x_B \ x_N]^T$  is the corresponding partition of  $x$ , with  $x_B \in \mathbb{R}^m$  and  $x_N \in \mathbb{R}^{n-m}$ , respectively. With this,  $x_B = -B^{-1}(Nx_N + b)$ .

Therefore, the objective function (13.91) can be written as

$$q(x) = \frac{1}{2} \begin{bmatrix} x_B^T & x_N^T \end{bmatrix} \begin{bmatrix} G_B & G_{BN} \\ G_{NB} & G_N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} + \begin{bmatrix} c_B^T & c_N^T \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix}, \tag{13.92}$$

where  $G_B \in \mathbb{R}^{m \times m}$ ,  $G_N \in \mathbb{R}^{(n-m) \times (n-m)}$ ,  $G_{BN} \in \mathbb{R}^{m \times (n-m)}$ ,  $G_{NB} = G_{BN}^T$ ,  $c_B \in \mathbb{R}^m$ , and  $c_N \in \mathbb{R}^{n-m}$ . Now, substituting  $x_B$  into the objective function (13.92), we obtain a reduced quadratic programming problem in  $x_N$ , i.e.,

$$\min_{x_N \in \mathbb{R}^{n-m}} \frac{1}{2} x_N^T \widehat{G}_N x_N + \widehat{c}_N^T x_N \tag{13.93}$$

where  $\widehat{G}_N$  and  $\widehat{c}_N$  are obtained from the algebraic manipulation of the above substitution, that is,

$$\begin{aligned} \widehat{G}_N &= G_N + N^T B^{-T} G_B B^{-1} N - G_{NB} B^{-1} N - N^T B^{-T} G_{BN}, \\ \widehat{c}_N &= c_N - N^T B^{-T} c_B + (N^T B^{-T} G_B - G_{NB}) B^{-1} b. \end{aligned}$$

If  $\widehat{G}_N$  is positive definite, then  $x_N^* = -\widehat{G}_N^{-1} \widehat{c}_N$ , which is unique. Obviously,  $x_B^* = -B^{-1}(Nx_N^* + b)$ . Therefore, the solution of problem (13.91) is

$$x^* = \begin{bmatrix} x_B^* \\ x_N^* \end{bmatrix} = \begin{bmatrix} -B^{-1}b \\ 0 \end{bmatrix} + \begin{bmatrix} B^{-1}N \\ -I \end{bmatrix} \widehat{G}_N^{-1} \widehat{c}_N. \tag{13.94}$$

The Lagrange multipliers  $\lambda^*$  at  $x^*$  are obtained from the equation  $c + Gx^* = A\lambda^*$ , using the above partition of vectors and matrices.

If  $\widehat{G}_N$  is positive semidefinite, then, when

$$(I + \widehat{G}_N \widehat{G}_N^+) \widehat{c}_N = 0, \tag{13.95}$$

i.e., when  $\widehat{c}_N \in R(\widehat{G}_N)$ , the minimization problem (13.93) has a solution, and its solution is  $x_N^* = -\widehat{G}_N^+ \widehat{c}_N + (I - \widehat{G}_N^+ \widehat{G}_N) \tilde{x}$ , where  $\tilde{x} \in \mathbb{R}^{n-m}$  is any vector and  $\widehat{G}_N^+$  is the generalized inverse of  $\widehat{G}_N$ . If (13.95) does not hold, then the problem (13.93) has no lower bound, and thus the problem (13.91) also has no lower bound, i.e., it has no finite solution.

If  $\widehat{G}_N$  has negative eigenvalues, it is clear that the minimization problem (13.93) has no lower bound.

This method of elimination of variables is simple, but when  $B$  is close to a singular matrix, then computing the solution (13.94) is numerically unstable.

## 2. Generalized Elimination of Variables: The Null-Space Method

A generalized elimination of variables method is exactly the null-space method which uses the reduced Hessian and the reduced gradient. Let  $\{y_1, \dots, y_m\}$  be a set of  $m$  linearly independent vectors in  $R(A)$ , the range of  $A$ , and  $\{z_1, \dots, z_{n-m}\}$  a set of  $n-m$  linearly independent vectors in  $N(A^T)$ , the null space of  $A^T$ . Consider the matrices

$$Y = [y_1, \dots, y_m] \in \mathbb{R}^{n \times m} \quad \text{and} \quad Z = [z_1, \dots, z_{n-m}] \in \mathbb{R}^{n \times (n-m)}.$$

Obviously,  $R(Y) = R(A)$ ,  $R(Z) = N(A^T)$ , and  $[Y \ Z]$  is nonsingular. Besides,  $AY$  is a nonsingular matrix and  $AZ = 0$ .

With this, we can write

$$x = Y\bar{x} + Z\hat{x},$$

where  $\bar{x} \in \mathbb{R}^m$  and  $\hat{x} \in \mathbb{R}^{n-m}$ .

Therefore, the constraints  $Ax + b = 0$  of the problem (13.91) can be written as  $AY\bar{x} = -b$ . Since  $AY$  is nonsingular, it follows that  $\bar{x} = -(AY)^{-1}b$ . Hence,

$$x = -Y(AY)^{-1}b + Z\hat{x}. \quad (13.96)$$

Substituting (13.96) in (13.91), we get

$$\min_{\hat{x} \in \mathbb{R}^{n-m}} \frac{1}{2} \hat{x}^T Z^T G Z \hat{x} + \left( Z^T c - Z^T G Y (AY)^{-1} b \right)^T \hat{x}. \quad (13.97)$$

Supposing that  $Z^T G Z$  is positive definite, then the solution of (13.97) can be written as

$$\hat{x}^* = -\left( Z^T G Z \right)^{-1} \left( Z^T c - Z^T G Y (AY)^{-1} b \right). \quad (13.98)$$

Now, from (13.96) the solution of the problem (13.91) is

$$x^* = -Y(AY)^{-1}b - Z\left(Z^T G Z\right)^{-1} \left( Z^T c - Z^T G Y (AY)^{-1} b \right). \quad (13.99)$$

Moreover, from the KKT condition it follows that  $c + Gx^* = A^T \lambda^*$ . Therefore, since  $AY$  is nonsingular, we have  $\lambda^* = (AY)^{-T} Y^T (c + Gx^*)$ , where  $x^*$  is given by (13.99).

Observe that the generalized elimination of variables uses the linear independent column vectors  $\{z_1, \dots, z_{n-m}\}$  which form the matrix  $Z$ , which is a base of the null space of  $A^T$ . This transforms the quadratic programming problem (13.91) into an unconstrained minimization problem (13.97). That is why this method is called the null-space method, in which the matrix  $Z^T G Z$  is the reduced Hessian and the vector  $Z^T(c - GY(AY)^{-1}b)$  is the reduced gradient. The null-space method is very useful for small- and medium-scale problems and when the computation of the null space and the factorization of the reduced Hessian are not very expensive.

Another approach for eliminating the variables is based on the *elimination of the balance constraints* (Andrei & Bărbulescu, 1993; Andrei, 2011d). In the system of constraints from (13.91),

a particular constraint  $i$  is called a balance constraint if  $b_i = 0$ . By eliminating the balance constraints, a new quadratic programming problem is obtained, with a reduced number of constraints.

### Notes and References

The quadratic programming problem has its importance in the nonlinear optimization area, many optimization problems being modeled as quadratic programs. The most famous is the portfolio optimization formulated by Markowitz (1952). This chapter is based on Nocedal and Wright (2006) and Sun and Yuan (2006). Methods for general quadratic programming are presented in Gill, Murray, Saunders, and Wright (1984) and Gould (1991). A quadratic programming bibliography is given by Gould and Toint (2012). Quadratic programming is intensively used in sequential quadratic programming, one of the most efficient methods in nonlinear optimization. We have considered the active-set methods and the interior point for quadratic programming. Interior point methods for convex quadratic programming can be found in Wright (1997) and Vanderbei (1994, 1996). The numerical comparison of the active-set and the interior point methods for convex quadratic programming shows that the interior point methods are generally much faster on large problems (Gould & Toint, 2002a). However, if a warm start is available, the active-set methods may be generally preferable. Active-set methods for convex quadratic programming are implemented in many codes: QPOPT (Gill & Murray, 1978), BQPD (Fletcher, 2000), QPA (Gould & Toint, 2002b), DONLP (Spellucci, 1998). The interior point solvers for quadratic programming are CPLEX (ILOG CPLEX, 2002), XPRESS-MP (Guéret, Prins, & Sevaux, 2002), MOSEK (Andersen & Andersen, 2000), etc. The problem of determining whether a feasible point for a nonconvex quadratic program is an NP-hard problem was proven in Murty and Kabadi (1987). Also, the problem of determining whether a given point is a local minimizer is NP-hard (Vavasis, 1990).



# Penalty and Augmented Lagrangian Methods

14

This chapter introduces two very important concepts in the constrained nonlinear optimization. These are *penalty* and *augmented Lagrangian*. Both concepts replace the original problem by a sequence of subproblems in which the constraints are expressed by terms added to the objective function. The penalty concept is implemented in two different methods. The *quadratic penalty method* adds a multiple of the square of the violation of each constraint to the objective function and solves a sequence of unconstrained optimization subproblems. Although simple and intuitive, this approach has some major deficiencies. The *nonsmooth exact penalty method*, on the other hand, solves a single unconstrained optimization problem. In this approach, a popular function is the  $l_1$  penalty function. The problem is that the nonsmoothness may create complications in numerical implementations. Finally, the second concept is the *multipliers method* or the *augmented Lagrangian method*, which explicitly uses Lagrange multiplier estimates in order to avoid the ill-conditioning of the quadratic penalty method.

The modern and best known methods for solving nonlinear optimization problems combine the penalty concept with the augmented Lagrangian in a penalty barrier with quadratic approximation of the inequality constraints (SPENBAR), or the minimization of a modified augmented Lagrangian subject to the linearized constraints (MINOS), or the minimization of the augmented Lagrangian subject to the simple bounds on variables (LANCELOT). In this chapter, we present the theory behind the algorithms SPENBAR (Andrei, 1996a, b, c, 1998a, 2001, 2015a, 2017c) and MINOS (Murtagh & Saunders, 1978, 1980, 1982, 1987, 1995) together with some numerical results for solving some applications from the LACOP collection.

## 14.1 The Quadratic Penalty Method

Let us consider the equality constrained optimization problem

$$\min \{f(x) : h_i(x) = 0, i \in E\} \quad (14.1)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i \in E \triangleq \{1, \dots, p\}$ , are continuously differentiable functions defined on  $\mathbb{R}^n$ . The *quadratic penalty function*  $Q(x, \sigma)$  for the problem (14.1) is

$$Q(x, \sigma) \triangleq f(x) + \frac{\sigma}{2} \sum_{i \in E} h_i^2(x), \quad (14.2)$$

where  $\sigma > 0$  is the *penalty parameter*. Observe that if  $\sigma$  is increased to  $\infty$ , the constraint violations are penalized. Therefore, the idea is to consider a sequence of  $\{\sigma_k\}$  with  $\sigma_k \rightarrow \infty$  as  $k \rightarrow \infty$  and to seek the approximate minimum  $x_k$  of  $Q(x, \sigma_k)$  for each  $k$ . Since the penalty term in (14.2) is smooth, any unconstrained optimization method can be used for  $x_k$  computation. Therefore, a sequence of unconstrained minimization subproblems  $\min_{x \in \mathbb{R}^n} Q(x, \sigma_k)$  is solved for each value of the penalty parameter  $\sigma_k$ . Clearly, in searching for  $x_k$ , the approximate minimum  $x_{k-1}$  of the previous subproblem may be used.

Now, let us consider the general nonlinear optimization problem

$$\min \{f(x) : c_i(x) \geq 0, i \in I, h_i(x) = 0, i \in E\}, \quad (14.3)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i \in I \triangleq \{1, \dots, m\}$  and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i \in E \triangleq \{1, \dots, p\}$ , which contains both inequality and equality constraints. For this problem, the following quadratic penalty function may be defined:

$$Q(x, \sigma) \triangleq f(x) + \frac{\sigma}{2} \sum_{i \in E} h_i^2(x) + \frac{\sigma}{2} \sum_{i \in I} ([c_i(x)]^-)^2, \quad (14.4)$$

where  $[y]^- = \max\{-y, 0\}$ . In this case, the quadratic penalty function may be less smooth than the original objective and constraints functions of problem (14.1).

The quadratic penalty method is based on the minimization of the quadratic penalty function in the following framework.

#### Algorithm 14.1 Quadratic penalty method

- |    |   |
|----|---|
| 1. | Consider an initial starting point $x_0^s$ . Select an initial value $\sigma_0 > 0$ for the penalty parameter and a nonnegative sequence $\{\tau_k\}$ with $\tau_k \rightarrow 0$ . Set $k = 0$   |
| 2. | Find an approximate minimum $x_k$ of the sub-problem $\min_{x \in \mathbb{R}^n} Q(x, \sigma_k)$ starting at $x_k^s$ , and terminate the iterations when $\ \nabla_x Q(x, \sigma_k)\  \leq \tau_k$ |
| 3. | If a test of convergence of the method is satisfied, then stop with $x_k$ as an approximate solution of the problem   |
| 4. | Choose a new penalty parameter $\sigma_{k+1} > \sigma_k$  |
| 5. | Choose a new starting point $x_{k+1}^s$ , set $k = k + 1$ and go to step 2  |

Some remarks are as follows:

1. The sequence of the penalty parameters  $\{\sigma_k\}$  may be adaptively selected with respect to the difficulty in minimizing the penalty function at each iteration  $k$ . When the minimization of  $Q(x, \sigma_k)$  is too expensive, then  $\sigma_{k+1}$  can be chosen near  $\sigma_k$ , for example,  $\sigma_{k+1} = 1.5\sigma_k$ . On the other hand, if the minimization of  $Q(x, \sigma_k)$  is relatively easy to be done, then  $\sigma_{k+1}$  can be selected larger than  $\sigma_k$ , for example,  $\sigma_{k+1} = 10\sigma_k$ . Of course, some other schemes for enlarging  $\sigma_k$  may be adopted.
2. In case the problem has only equality constraints, then  $Q(x, \sigma_k)$  is smooth. Therefore, the algorithms for the unconstrained optimization (in step 2) can be used to find an approximate solution  $x_k$ . However, the minimization of  $Q(x, \sigma_k)$  becomes more difficult as  $\sigma_k$  becomes larger.

Near the minimum, the Hessian  $\nabla^2 Q(x, \sigma_k)$  becomes ill-conditioned, and in this case, the quasi-Newton or the conjugate gradient methods perform very poorly.

**Theorem 14.1** Suppose that  $x_k$  for any  $k \geq 0$  is the exact global minimum of  $Q(x, \sigma_k)$  defined in (14.2), determined by Algorithm 14.1. Suppose that  $\sigma_k \rightarrow \infty$ . Then, every limit point  $x^*$  of the sequence  $\{x_k\}$  is a global solution of the problem (14.1).

**Proof** Let  $\bar{x}$  be a global solution of (14.1), that is,  $f(\bar{x}) \leq f(x)$  for all  $x$  satisfying  $h_i(x) = 0$ ,  $i \in E$ . Since  $x_k$  is a minimum of  $Q(x, \sigma_k)$  for each  $k$ , we have  $Q(x_k, \sigma_k) \leq Q(\bar{x}, \sigma_k)$ , which leads to the inequality

$$f(x_k) + \frac{\sigma_k}{2} \sum_{i \in E} h_i^2(x_k) \leq f(\bar{x}) + \frac{\sigma_k}{2} \sum_{i \in E} h_i^2(\bar{x}) = f(\bar{x}). \quad (14.5)$$

By rearranging the terms of (14.5), we get

$$\sum_{i \in E} h_i^2(x_k) \leq \frac{2}{\sigma_k} [f(\bar{x}) - f(x_k)]. \quad (14.6)$$

Suppose that  $x^*$  is a limit point of  $\{x_k\}$  so that there is an infinite sequence  $K$ , such that  $\lim_{k \in K} x_k = x^*$ . By taking the limit as  $k \rightarrow \infty$ ,  $k \in K$ , on both sides of (14.6), we obtain

$$\sum_{i \in E} h_i^2(x^*) = \lim_{k \rightarrow \infty, k \in K} \sum_{i \in E} h_i^2(x_k) \leq \lim_{k \rightarrow \infty, k \in K} \frac{2}{\sigma_k} [f(\bar{x}) - f(x_k)] = 0,$$

where the last equality follows from the fact that  $\sigma_k \rightarrow \infty$ . Therefore, we have  $h_i(x^*) = 0$  for all  $i \in E$ , so that  $x^*$  is feasible. Additionally, by taking the limit as  $k \rightarrow \infty$  for  $k \in K$  in (14.5) and by the nonnegativity of  $\sigma_k$  and of each  $h_i^2(x_k)$ , we get

$$f(x^*) \leq f(x^*) + \lim_{k \rightarrow \infty, k \in K} \frac{\sigma_k}{2} \sum_{i \in E} h_i^2(x_k) \leq f(\bar{x}).$$

Since  $x^*$  is a feasible point whose objective function is not larger than that of the global solution  $\bar{x}$ , it follows that  $x^*$  is also a global solution, as claimed.  $\blacklozenge$

Observe that this result requires finding the global minimum of each subproblem. In general, this property of convergence to the global solution of (14.1) cannot be attained. Nocedal and Wright (2006) prove the convergence of the sequence  $\{x_k\}$  when the inexact minimization of  $Q(x, \sigma_k)$  is allowed, as in the following theorem.

**Theorem 14.2** Suppose that the tolerances and the penalty parameters satisfy  $\tau_k \rightarrow 0$  and  $\sigma_k \rightarrow \infty$ . Then, if a limit point  $x^*$  of the sequence  $\{x_k\}$  is infeasible, then there is a stationary point of the function  $\|h(x)\|^2$ . On the other hand, if a limit point  $x^*$  is feasible and the constraint gradients  $\nabla h_i(x^*)$ ,  $i \in E$ , are linearly independent, then  $x^*$  is a KKT point for the problem (14.1). For such points, for any infinite sequence  $K$  such that  $\lim_{k \in K} x_k = x^*$ , we have

$$\lim_{k \rightarrow \infty, k \in K} -\sigma_k h_i(x_k) = \lambda_i^*, \quad (14.7)$$

for all  $i \in E$ , where  $\lambda^* = [\lambda_1^*, \dots, \lambda_p^*]$  is the multiplier vector that satisfies the KKT optimality conditions (11.21) and (11.22) for the equality constrained problem (14.1).

**Proof** From (14.2), we have

$$\nabla Q(x_k, \sigma_k) = \nabla f(x_k) + \sum_{i \in E} \sigma_k h_i(x_k) \nabla h_i(x_k). \quad (14.8)$$

Therefore, from the termination criterion in step 2 of Algorithm 14.1, it follows that

$$\left\| \nabla f(x_k) + \sum_{i \in E} \sigma_k h_i(x_k) \nabla h_i(x_k) \right\| \leq \tau_k. \quad (14.9)$$

Having in view that  $\|a\| - \|b\| \leq \|a + b\|$ , from (14.9), we get

$$\left\| \sum_{i \in E} h_i(x_k) \nabla h_i(x_k) \right\| \leq \frac{1}{\sigma_k} [\tau_k + \|\nabla f(x_k)\|]. \quad (14.10)$$

Let  $x^*$  be a limit point of the sequence  $\{x_k\}$ . Then, there is a subsequence  $K$  such that  $\lim_{k \in K} x_k = x^*$ . When  $k \rightarrow \infty$  for  $k \in K$ ,  $\tau_k + \|\nabla f(x_k)\| \rightarrow \|\nabla f(x_k)\|$ . Since  $\sigma_k \rightarrow \infty$ , the right-hand side of (14.10) approaches zero. Therefore, from the corresponding limit of the left-hand side of (14.10), we have

$$\sum_{i \in E} h_i(x^*) \nabla h_i(x^*) = 0. \quad (14.11)$$

Now, if the constraints gradients  $\nabla h_i(x^*)$  are linearly dependent, then from (14.11),  $h_i(x^*) \neq 0$ . In this case,  $x^*$  is a stationary point of the function  $\|h(x)\|^2$ . On the other hand, if the constraint gradients  $\nabla h_i(x^*)$  are linearly independent at the limit point  $x^*$ , from (14.11), it follows that  $h_i(x^*) = 0$  for all  $i \in E$ , i.e.,  $x^*$  is feasible. Hence, the KKT condition (11.13b) is satisfied. We need to check (11.13a) and to show that (11.15) holds.

Let  $A(x)^T = [\nabla h_i(x)]_{i \in E}$  be the matrix of the constraint gradients and  $\lambda_k = -\sigma_k h(x_k)$ . From (14.9), it follows that

$$A(x_k)^T \lambda_k = \nabla f(x_k) - \nabla Q(x_k, \sigma_k), \quad \|\nabla Q(x_k, \sigma_k)\| \leq \tau_k. \quad (14.12)$$

For all  $k \in K$  sufficiently large, the matrix  $A(x_k)$  is of full rank; therefore,  $A(x_k)A(x_k)^T$  is nonsingular. From the first relation of (14.12), we get

$$\lambda_k = \left[ A(x_k)A(x_k)^T \right]^{-1} A(x_k) [\nabla f(x_k) - \nabla Q(x_k, \sigma_k)]. \quad (14.13)$$

Taking the limit when  $k \in K$  goes to  $\infty$ , we obtain

$$\lim_{k \rightarrow \infty, k \in K} \lambda_k = \lambda^* = \left[ A(x^*)A(x^*)^T \right]^{-1} A(x^*) \nabla f(x^*). \quad (14.14)$$

Therefore, taking the limit in (14.9), we get

$$\nabla f(x^*) - A(x^*)^T \lambda^* = 0, \quad (14.15)$$

i.e.,  $\lambda^*$  satisfies the first KKT condition (11.13a) for the problem (14.1). In conclusion,  $x^*$  is a KKT point for (14.1) with the unique Lagrange multiplier vector  $\lambda^*$ .  $\blacklozenge$

Observe that the quantities  $\sigma_k h_i(x_k)$  may be used as estimates of the Lagrange multipliers  $\lambda_i^*$ ,  $i \in E$ . This is important for the analysis of the augmented Lagrangian method.

Let us study the ill-conditioning of the Hessian  $\nabla^2 Q(x, \sigma_k)$ . This analysis is extremely important in choosing effective algorithms for the minimization in step 2 of Algorithm 14.1, as well as the linear algebra calculations at each iteration  $k$ . From (14.2), we get

$$\nabla Q(x, \sigma_k) = \nabla f(x) + \sum_{i \in E} \sigma_k h_i(x) \nabla h_i(x). \quad (14.16)$$

$$\nabla^2 Q(x, \sigma_k) = \nabla^2 f(x) + \sum_{i \in E} \sigma_k h_i(x) \nabla^2 h_i(x) + \sigma_k A(x)^T A(x), \quad (14.17)$$

where  $A(x)^T = [\nabla h_i(x)]_{i \in E}$ . In the conditions of Theorem 14.2, from (14.7), it follows that the sum of the first two terms on the right-hand side of (14.17) is approximately equal to the Hessian of the Lagrangian function. Therefore, when  $x$  is close to the minimum point of  $Q(x, \sigma_k)$ , it follows that

$$\nabla^2 Q(x, \sigma_k) \approx \nabla^2 L(x, \lambda^*) + \sigma_k A(x)^T A(x). \quad (14.18)$$

Note that  $\nabla^2 Q(x, \sigma_k)$  is the sum of two matrices: a matrix whose elements are independent of  $\sigma_k$  and a matrix of rank  $\text{card}(E)$  whose nonzero eigenvalues are of order  $\sigma_k$ . Usually, the number of constraints  $\text{card}(E)$  is smaller than the number of variables  $n$ . In this case, the last term in (14.18) is singular. Therefore, the whole matrix  $\nabla^2 Q(x, \sigma_k)$  has some eigenvalues approaching a constant, while the others are of order  $\sigma_k$ . Since  $\sigma_k \rightarrow \infty$ , it follows that the ill-conditioning of the matrix  $\nabla^2 Q(x, \sigma_k)$  is increasing as  $k$  is increasing. *This is the major difficulty associated to the quadratic penalty method.* Note that this ill-conditioning is responsible for a possible inaccuracy in the computations of the Newton step  $p$  for the subproblem  $\min_{x \in \mathbb{R}^n} Q(x, \sigma_k)$ , which is obtained by solving the linear algebraic system

$$\nabla^2 Q(x, \sigma_k) p = -\nabla Q(x, \sigma_k). \quad (14.19)$$

Clearly, the poor conditioning of this system will lead to significant errors in the value of  $p$ , regardless of the computational technique used to solve (14.19).

## 14.2 The Nonsmooth Penalty Method

As already seen, the quadratic penalty function is not exact. Its minimum point is generally not the same as the solution of the nonlinear optimization problem for any positive value of the penalty parameter  $\sigma$ . This penalty function is called *inexact*. In this context, the *exact penalty function* is introduced, in which for certain choices of its penalty parameter, a single minimization with respect to  $x$  can yield the exact solution of the nonlinear optimization problem (Han & Mangasarian, 1979). In this section, we consider the *nonsmooth exact penalty functions* (Nocedal & Wright, 2006).

For the general nonlinear optimization problem (14.3), the  $l_1$  penalty function is defined by

$$\varphi_1(x, \sigma) = f(x) + \sigma \sum_{i \in E} |h_i(x)| + \sigma \sum_{i \in I} [c_i(x)]^-, \quad (14.20)$$

where  $[y]^- = \max \{0, -y\}$ . The following result is proved by Han and Mangasarian (1979).

**Theorem 14.3** Suppose that  $x^*$  is a strict local solution of the problem (14.3) at which the first-order necessary optimality conditions (11.21) hold with the Lagrange multipliers  $\mu_i^*$ ,  $i \in I$ , and  $\lambda_i^*$ ,  $i \in E$ . Then,  $x^*$  is a local minimum of  $\varphi_1(x, \sigma)$  for all  $\sigma > \sigma^*$ , where

$$\sigma^* = \max \left\{ \max_{i \in I} \{|\mu_i^*|\}, \max_{i \in E} \{|\lambda_i^*|\} \right\}. \quad (14.21)$$

If, in addition, the second-order sufficient conditions (11.22) hold and  $\sigma > \sigma^*$ , then  $x^*$  is a strict local minimum of  $\varphi_1(x, \sigma)$ .  $\blacklozenge$

The following definition characterizes the stationary points of  $\varphi_1$ , even if  $\varphi_1$  is not differentiable.

**Definition 14.1** A point  $\bar{x} \in \mathbb{R}^n$  is a stationary point of the penalty function  $\varphi_1(x, \sigma)$  if

$$D(\varphi_1(\bar{x}, \sigma); p) \geq 0, \quad (14.22)$$

for all  $p \in \mathbb{R}^n$ . Similarly,  $\bar{x}$  is a stationary point of the measure of infeasibility

$$g(x) = \sum_{i \in E} |h_i(x)| + \sum_{i \in I} [c_i(x)]^- \quad (14.23)$$

if  $D(g(\bar{x}), p) \geq 0$  for all  $p \in \mathbb{R}^n$ .

If a point is infeasible for (14.3) but stationary with respect to the infeasibility measure  $g$ , then it is an infeasible stationary point.  $\blacklozenge$

The following theorem shows that under some mild assumptions, the stationary points of  $\varphi_1(x, \sigma)$  correspond to the KKT points of the constrained nonlinear optimization problem (14.3).

**Theorem 14.4** Suppose that  $\bar{x}$  is a stationary point of the penalty function  $\varphi_1(x, \sigma)$  for all  $\sigma > \bar{\sigma} > 0$ , where  $\bar{\sigma}$  is a certain threshold. Then, if  $\bar{x}$  is feasible for the nonlinear optimization problem (14.3), it satisfies the KKT optimality conditions (11.22) for (14.3). If  $\bar{x}$  is not feasible for (14.3), then it is an infeasible stationary point.

**Proof** Suppose that  $\bar{x}$  is feasible. Therefore,

$$D(\varphi_1(\bar{x}, \sigma); p) = \nabla f(\bar{x})^T p + \sigma \sum_{i \in E} |\nabla h_i(\bar{x})^T p| + \sigma \sum_{i \in I \cap A(\bar{x})} [\nabla c_i(\bar{x})^T p]^-,$$

where  $A(\bar{x})$  is the set of active constraints at  $\bar{x}$ . Consider any direction  $p$  that satisfies  $\nabla h_i(\bar{x})^T p = 0$  for all  $i \in E$  and  $\nabla c_i(\bar{x})^T p \geq 0$  for all  $i \in I \cap A(\bar{x})$ . Therefore,

$$\sum_{i \in E} |\nabla h_i(\bar{x})^T p| + \sum_{i \in I \cap A(\bar{x})} [\nabla c_i(\bar{x})^T p]^- = 0. \quad (14.24)$$

By the stationarity assumption on  $\varphi_1(\bar{x})$ , it follows that

$$D(\varphi_1(\bar{x}, \sigma); p) = \nabla f(\bar{x})^T p \geq 0,$$

for all  $p$  satisfying (14.24). By Theorem A4.2 (Farkas), it follows that

$$\nabla f(\bar{x}) = \sum_{i \in A(x^*)} \bar{\lambda}_i \nabla c_i(\bar{x}),$$

for some coefficients  $\bar{\lambda}_i$  with  $\bar{\lambda}_i \geq 0$ ,  $i \in I \cap A(\bar{x})$ . However, this expression implies that the KKT optimality conditions (11.21) hold. The second part of the proof is very simple and it is omitted here.  $\blacklozenge$

With these results, the following algorithm based on the classical  $l_1$  penalty function may be presented.

**Algorithm 14.2**  $l_1$  penalty method

- |    |  |
|----|--|
| 1. | Consider an initial starting point $x_0^s$ . Select an initial value $\sigma_0 > 0$ for the penalty parameter and a tolerance $\tau > 0$ . Set $k = 0$ |
| 2. | Starting with $x_k^s$ , find an approximate minimum $x_k$ of $\varphi_1(x, \sigma_k)$  |
| 3. | If $g(x_k) \leq \tau$ , then stop, with $x_k$ as an approximate solution of the problem  |
| 4. | Choose a new penalty parameter $\sigma_{k+1} > \sigma_k$   |
| 5. | Choose a new starting point $x_{k+1}^s$ , set $k = k + 1$ , and go to step 2   |

$\blacklozenge$

Because of nonsmoothness, the minimization of  $\varphi_1(x, \sigma_k)$  in step 2 above is the most difficult operation in Algorithm 14.2. A practical  $l_1$  penalty method will be presented below.

The scheme for updating the penalty parameter  $\sigma_k$  in step 4 is to increase it by an arbitrary constant factor, for example,  $\sigma_{k+1} = 5\sigma_k$ . If  $\sigma_k$  is too small, then many cycles of Algorithm 14.2 may be required. On the other hand, if  $\sigma_k$  is too large, then the penalty function will be difficult to minimize.

Obviously,  $\varphi_1(x, \sigma_k)$  is nonsmooth. The gradient of  $\varphi_1(x, \sigma_k)$  is not defined at any  $x$  for which  $c_i(x) = 0$  for some  $i \in I$  and  $h_i(x) = 0$  for some  $i \in E$ . In order to avoid the difficulties for minimizing  $\varphi_1(x, \sigma_k)$ , the strategy is to consider a *simplified model* of this function and try to find the minimum of this model. The most direct idea is to define the model by linearizing the constraints and by replacing the objective function with a quadratic function, as follows:

$$\begin{aligned} q(p; \sigma) = & f(x) + \nabla f(x)^T p + \frac{1}{2} p^T W p \\ & + \sigma \sum_{i \in E} |h_i(x) + \nabla h_i(x)^T p| + \sigma \sum_{i \in I} [c_i(x) + \nabla c_i(x)^T p]^- \end{aligned} \quad (14.25)$$

where  $W$  is a symmetric matrix which contains second derivatives information about the functions of the problem (14.3). The model (14.25) is not smooth, but it can be reformulated as a *smooth quadratic programming problem* by introducing the artificial variables  $r_i, s_i, i \in E$ , and  $t_i, i \in I$ , as follows:

$$\begin{aligned}
& \min_{p,r,s,t} f(x) + \nabla f(x)^T p + \frac{1}{2} p^T W p + \sigma \sum_{i \in E} (r_i + s_i) + \sigma \sum_{i \in I} t_i \\
& \text{subject to:} \\
& h_i(x) + \nabla h_i(x)^T p = r_i - s_i, \quad i \in E, \\
& c_i(x) + \nabla c_i(x)^T p \geq -t_i, \quad i \in I, \\
& r, s, t \geq 0.
\end{aligned} \tag{14.26}$$

This problem can be solved by any standard quadratic programming solver.

In this algorithm, by using the quadratic programming (14.26), the updating of the penalty parameter  $\sigma_k$  remains a crucial problem. Discussions about this aspect of the algorithm are presented by Nocedal and Wright (2006). An idea is to choose an initial value of the penalty parameter and to increase it until the feasibility has been attained. Another variant is to update the penalty parameter so that  $\sigma_k$  is greater than an estimate of the Lagrange multipliers computed at  $x_k$ .

The development of the filter method (Fletcher & Leyffer, 2002), which does not require any penalty parameter, placed the nonsmooth penalty methods in a cone of penumbra. However, these methods have not been completely abandoned, partly because of their ability to handle degenerate problems (see Remark 11.8).

### 14.3 The Augmented Lagrangian Method

This section discusses the augmented Lagrangian method, known as the *method of multipliers*. Hestenes (1969) and Powell (1969) independently proposed these methods as a possibility of reducing the ill-conditioning by introducing explicit Lagrange multiplier estimates into the function to be minimized, known as the *augmented Lagrange function*. In the following, we consider the problems with equalities constraints:  $\min\{f(x) : h_i(x) = 0, i \in E\}$ . As seen in Theorem 14.2, the approximate minimum  $x_k$  of  $Q(x, \sigma_k)$  does not satisfy the feasibility constraints  $h_i(x) = 0, i \in E$ . Instead, the constraints are perturbed (see Eq. (14.7)) so that

$$h_i(x_k) \approx -\lambda_i^*/\sigma_k, \tag{14.27}$$

for all  $i \in E$ . Indeed, as  $\sigma_k \rightarrow \infty$ ,  $h_i(x_k) \rightarrow 0$ . However, is it possible to avoid this systematic perturbation of  $Q(x, \sigma_k)$ , that is, to make the approximate minimizers better satisfy the constraints  $h_i(x) = 0$ , even for moderate values of  $\sigma_k$ ?

The augmented Lagrangian function

$$L_A(x, \lambda, \sigma) \triangleq f(x) - \sum_{i \in E} \lambda_i h_i(x) + \frac{\sigma}{2} \sum_{i \in E} h_i^2(x), \tag{14.28}$$

achieves this goal by including the explicit estimate of the Lagrange multipliers  $\lambda_i$ . Observe that the augmented Lagrangian (14.28) is a combination of the Lagrangian function and of the quadratic penalty function. An algorithm based on the augmented Lagrangian function at iteration  $k$  fixes the penalty parameter  $\sigma$  to a value  $\sigma_k > 0$  and fixes  $\lambda$  at the current estimate  $\lambda^k$ . Thus, a minimization of the augmented Lagrangian with respect to  $x$  is performed. Let  $x_k$  be the approximate minimum of  $L_A(x, \lambda^k, \sigma_k)$ . Therefore, by the optimality conditions for unconstrained optimization (see Theorem 11.3), we have

$$\nabla L_A(x_k, \lambda^k, \sigma_k) = \nabla f(x_k) - \sum_{i \in E} [\lambda_i^k - \sigma_k h_i(x_k)] \nabla h_i(x_k) = 0. \quad (14.29)$$

Now, comparing this result with the optimality condition (14.15) for (14.1), it follows that

$$\lambda_i^* \approx \lambda_i^k - \sigma_k h_i(x_k), \quad (14.30)$$

for all  $i \in E$ . From (14.30), we get

$$h_i(x_k) \approx -\frac{1}{\sigma_k} (\lambda_i^* - \lambda_i^k),$$

for all  $i \in E$ . Therefore, if  $\lambda^k$  is close to the optimal multiplier vector  $\lambda^*$ , the infeasibility at  $x_k$  will be much smaller than  $(1/\sigma_k)$ , more than being proportional to  $(1/\sigma_k)$  as in (14.27). The relation (14.30) suggests a formula for updating the current estimate  $\lambda^k$  of the Lagrange multiplier vector by using the approximate minimum  $x_k$  already calculated as

$$\lambda_i^{k+1} = \lambda_i^k - \sigma_k h_i(x_k), \quad i \in E. \quad (14.31)$$

With these theoretical developments, the augmented Lagrangian algorithm for nonlinear optimization problems with equality constraints can be presented as follows.

#### Algorithm 14.3 Augmented Lagrangian method: Equality constraints

- |    |  |
|----|--|
| 1. | Consider the initial starting points $x_0^s$ and $\lambda^0$ . Select an initial value $\sigma_0 > 0$ for the penalty parameter and a tolerance $\tau_0 > 0$ . Set $k = 0$   |
| 2. | Find an approximate minimum $x_k$ of the $L_A(x, \lambda^k, \sigma_k)$ starting at $x_k^s$ . If $\ \nabla L_A(x_k, \lambda^k, \sigma_k)\  \leq \tau_k$ , stop the iterations |
| 3. | If a convergence test for (14.1) is satisfied, then stop with $x_k$ as an approximate solution of the problem  |
| 4. | Update the Lagrange multipliers as in (14.31) to get $\lambda^{k+1}$   |
| 5. | Choose a new penalty parameter $\sigma_{k+1} \geq \sigma_k$  |
| 6. | Set the starting point for the new iteration as $x_{k+1}^s = x_k$  |
| 7. | Choose the tolerance $\tau_{k+1}$ , set $k = k + 1$ , and go to step 2   |

◆

It was proved that the convergence of this method is assured without increasing  $\sigma$  indefinitely. In other words, the ill-conditioning of this method is reduced; thus, the choice of the starting point  $x_{k+1}^s$  is less critical than in the previous algorithms based on the quadratic or the nonsmooth penalty methods. The tolerance  $\tau_k$  could be chosen by using the infeasibility measure  $\sum_{i \in E} |h_i(x_k)|$ , and the penalty parameter  $\sigma$  may be updated (increased) if the reduction in this infeasibility measure is insufficient at the current iteration (Nocedal & Wright, 2006).

**Theorem 14.5** *Let  $x^*$  be a local solution of (14.1) at which the gradients  $\nabla h_i(x^*)$ ,  $i \in E$ , are linearly independent and the second-order sufficient conditions given in Theorem 11.13 are satisfied for  $\lambda = \lambda^*$ . Then, there is a threshold  $\bar{\sigma}$  such that for all  $\sigma \geq \bar{\sigma}$ ,  $x^*$  is a strict local minimum of the augmented Lagrangian  $L_A(x, \lambda^*, \sigma)$ .*

**Proof** We show that  $x^*$  satisfies the second-order sufficient conditions for a strict local minimum of  $L_A(x, \lambda^*, \sigma)$  given in Theorem 11.6, for all  $\sigma$  sufficiently large, i.e.,

$$\nabla L_A(x^*, \lambda^*, \sigma) = 0, \quad (14.32)$$

$$\nabla^2 L_A(x^*, \lambda^*, \sigma) \text{ is positive definite.} \quad (14.33)$$

Now, as  $x^*$  is a local solution of (14.1) at which LICQ is satisfied, from [Theorem 11.11](#) (see [Remark 11.3](#)), it follows that  $\nabla L(x^*, \lambda^*) = 0$  and  $h_i(x^*) = 0, i \in E$ . Therefore,

$$\begin{aligned} \nabla L_A(x^*, \lambda^*, \sigma) &= \nabla f(x^*) - \sum_{i \in E} [\lambda_i^* - \sigma h_i(x^*)] \nabla h_i(x^*) \\ &= \nabla f(x^*) - \sum_{i \in E} \lambda_i^* \nabla h_i(x^*) = \nabla L(x^*, \lambda^*) = 0 \end{aligned}$$

verifies (14.32), independently of  $\sigma$ . Further on, to verify (14.33), consider

$$\nabla^2 L_A(x^*, \lambda^*, \sigma) = \nabla^2 L(x^*, \lambda^*) + \sigma A(x^*)^T A(x^*),$$

where  $A(x)^T = [\nabla h_i(x)]_{i \in E}$ .

If  $\nabla^2 L_A(x^*, \lambda^*, \sigma)$  is not positive definite, then, for each integer  $k \geq 1$ , we could choose a vector  $w_k$  with  $\|w_k\| = 1$ , such that

$$0 \geq w_k^T \nabla^2 L_A(x^*, \lambda^*, k) w_k = w_k^T \nabla^2 L(x^*, \lambda^*) w_k + k \|A(x^*) w_k\|_2^2, \quad (14.34)$$

and therefore,

$$\|A(x^*) w_k\|_2^2 \leq -\frac{1}{k} w_k^T \nabla^2 L(x^*, \lambda^*) w_k \rightarrow 0, \quad (14.35)$$

as  $k \rightarrow \infty$ . Since the vectors  $\{w_k\}$  lie in a compact set (the surface of the unit sphere), they have an accumulation point  $w$ . From (14.35), it follows that  $A(x^*) w = 0$ . Now, from (14.34), we have

$$w_k^T \nabla^2 L(x^*, \lambda^*) w_k \leq -k \|A(x^*) w_k\|_2^2 \leq 0,$$

So, by taking the limits, we have  $w^T \nabla^2 L(x^*, \lambda^*) w \leq 0$ . However, this inequality contradicts the second-order condition in [Theorem 11.13](#), which states that we must have  $w^T \nabla^2 L(x^*, \lambda^*) w > 0$  for all nonzero vectors  $w$  with  $A(x^*) w = 0$ . Therefore, (14.33) holds for all  $\sigma$  sufficiently large.  $\blacklozenge$

The following theorem, proved by Bertsekas (1999), gives the conditions under which there is a minimum of the augmented Lagrangian  $L_A(x, \lambda, \sigma)$  that lies close to  $x^*$ , conditions that specify some error bounds on both  $x_k$  and  $\lambda^{k+1}$ , solutions of the subproblem at iteration  $k$ .

**Theorem 14.6** (Bertsekas) Suppose that the assumptions of [Theorem 14.5](#) are satisfied at  $x^*$  and  $\lambda^*$ . Let  $\bar{\sigma}$  be chosen as in the theorem. Then, there exist positive scalars  $\delta$ ,  $\varepsilon$ , and  $M$ , such that the following claims hold:

(a) For all  $\lambda^k$  and  $\sigma_k$  satisfying

$$\|\lambda^k - \lambda^*\| \leq \sigma_k \delta, \quad \sigma_k \geq \bar{\sigma} \quad (14.36)$$

the problem  $\min_{x \in \mathbb{R}^n} L_A(x, \lambda^k, \sigma_k)$  subject to  $\|x - x^*\| \leq \varepsilon$  has a unique solution  $x_k$ . Moreover,

$$\|x_k - x^*\| \leq M \|\lambda^k - \lambda^*\| / \sigma_k. \quad (14.37)$$

(b) For all  $\lambda^k$  and  $\sigma_k$  satisfying (14.36), we have

$$\|\lambda^{k+1} - \lambda^*\| \leq M\|\lambda^k - \lambda^*\|/\sigma_k, \quad (14.38)$$

where  $\lambda^{k+1}$  is given by (14.31).

(c) For all  $\lambda^k$  and  $\sigma_k$  satisfying (14.36), the matrix  $\nabla^2 L_A(x_k, \lambda^k, \sigma_k)$  is positive definite, and the constraint gradients  $\nabla h_i(x_k)$ ,  $i \in E$ , are linearly independent. ♦

The theorem of Bertsekas (1999) is important. It illustrates some salient properties of the augmented Lagrangian approach. The bound (14.37) shows that  $x_k$  will be close to  $x^*$  if  $\lambda^k$  is accurate or if the penalty parameter  $\sigma_k$  is large. Therefore, in this approach of the augmented Lagrangian method, there are two ways of improving the accuracy of  $x_k$ , whereas in the quadratic penalty approach, there is only one: increase  $\sigma_k$ . The bound (14.37) shows that the accuracy of the multipliers can be increased by choosing a sufficiently large value of  $\sigma_k$ . Finally, the last observation of the theorem shows that under the given conditions, the second-order sufficient optimality conditions for the unconstrained minimization are satisfied for the  $k$ -th subproblem. Therefore, very good numerical performances can be expected by applying any standard unconstrained minimization technique.

**Example 14.1** Let us solve the following optimization problem by using the augmented Lagrangian method

$$\begin{aligned} \min f(x) &\equiv x_1^2 + 8x_2^2 \\ \text{subject to :} \\ h_1(x) &\equiv x_1 + x_2 - 4 = 0. \end{aligned}$$

Observe that the augmented Lagrange function is

$$L_A(x, \lambda, \sigma) = x_1^2 + 8x_2^2 - \lambda(x_1 + x_2 - 4) + \sigma(x_1 + x_2 - 4)^2.$$

Consider the initial point  $x_0 = [0 \ 4]^T$ , the initial values for the penalty parameter  $\sigma_0 = 1$ , and the Lagrange multiplier  $\lambda^0 = 0$ . By using the Newton method for minimizing the augmented Lagrange function in step 2 of Algorithm 14.3, we get the results from Table 14.1.

As a comparison, in Table 14.2, we present the optimization process for solving the problem by using the quadratic penalty method.

This example shows that the inclusion of the Lagrange multiplier term in the augmented Lagrangian  $L_A(x, \lambda, \sigma)$  represents a significant improvement of the quadratic penalty method.

**Table 14.1** Augmented Lagrangian method

$k$	$\sigma$	$\lambda$	$x_1$	$x_2$	$f(x_k)$	$h_1(x_k)$
0	1	0	0	4	128	0
1	10	3.767400	1.882353	0.235294	3.986159	-1.88235
2	100	6.837930	3.418960	0.427370	13.15051	-0.153661
3	1000	7.108704	3.554350	0.444290	14.21260	-0.00135
4	10,000	7.1111	3.555555	0.444444	14.22222	-0.120e-5

**Table 14.2** Quadratic penalty method

$k$	$\sigma$	$x_1$	$x_2$	$f(x_k)$	$h_1(x_k)$
0	1	0	4	128	0
1	10	1.882353	0.235294	3.986151	-1.882350
2	100	3.265306	0.408163	11.99500	-0.326530
3	1000	3.524229	0.440528	13.97271	-0.035242
4	10,000	3.552390	0.444049	14.19697	-0.003552
5	100,000	3.555240	0.444404	14.21969	-0.000355
6	1,000,000	3.555524	0.444440	14.22197	-0.355e-4
7	10,000,000	3.555555	0.444444	14.22222	-0.355e-5

## 14.4 Criticism of the Penalty and Augmented Lagrangian Methods

Let us see the fundamentals of the penalty and of the augmented Lagrangian methods. For this, we consider the nonlinear optimization problem with equality constraints:

$$\min \{f(x) : h_i(x) = 0, i \in E\}, \quad (14.39)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i \in E \triangleq \{1, \dots, p\}$ , are continuously differentiable on  $\mathbb{R}^n$ . Define  $h(x) = [h_1(x), \dots, h_p(x)]$ . A local solution  $(x^*, \lambda^*)$  satisfies the first-order KKT necessary optimality conditions

$$\nabla h(x)^T \lambda = \nabla f(x), \quad (14.40a)$$

$$h(x) = 0, \quad (14.40b)$$

where  $\lambda \in \mathbb{R}^p$  is the vector of the Lagrange multipliers.

It is common knowledge that in everyday life we always try to find a balance (an equilibrium) between what is desirable (to minimize the objective function) and what is legally achievable (to satisfy the constraints). This multiple objective point of view represents the essence of the penalty methods. For example, the quadratic penalty method combines these two requirements and solves a sequence of unconstrained optimization subproblems

$$\min_{x \in \mathbb{R}^n} Q(x, \sigma) = f(x) + \frac{1}{2} \sigma \|h(x)\|^2, \quad (14.41)$$

parameterized by the penalty parameter  $\sigma > 0$ . Thus, a trajectory of the points  $x^*(\sigma)$  is obtained as solutions of (14.41), as well as an increasing sequence of penalty parameters. We must let  $\sigma$  become large in order to nearly achieve feasibility, but at least the penalty function should be *smooth*. Therefore, we may apply the Newton method or the quasi-Newton methods for solving (14.41). Let us introduce the Lagrange function

$$L(x, \lambda) = f(x) - \lambda^T h(x). \quad (14.42)$$

The derivatives of the quadratic penalty function are

$$\nabla Q(x, \sigma) = \nabla f(x) + \nabla h(x)^T \lambda, \quad (14.43)$$

$$\nabla^2 Q(x, \sigma) = \nabla^2 f(x) + \sum_{i \in E} \lambda_i \nabla^2 h_i(x) + \sigma \nabla h(x)^T \nabla h(x), \quad (14.44)$$

where  $\lambda_i = \sigma h_i(x)$ ,  $i \in E$ , and  $\lambda = [\lambda_1, \dots, \lambda_p]$ .

Observe that for  $x^*(\sigma)$ ,  $\nabla Q(x^*(\sigma), \sigma) = 0$ . Defining  $\lambda^*(\sigma) = \sigma h(x^*(\sigma))$ , then  $(x^*(\sigma), \lambda^*(\sigma))$  is the *exact solution* of a perturbed form of the problem (14.39)

$$\min \{f(x) : h(x) = h(x^*(\sigma))\}. \quad (14.45)$$

If the Jacobian matrix  $\nabla h(\bar{x})$  has full row rank and  $\bar{x}$  is a unique local minimizer for (14.39), we can show that the full Hessian  $\nabla^2 Q(x, \sigma)$  is positive definite at  $(x^*(\sigma), \lambda^*(\sigma))$  for sufficiently large  $\sigma$ . Thus, the penalty function is *convex* for large  $\sigma$  and the minimum  $x^*(\sigma)$  exists. Therefore, the problem (14.41) can be very well solved by the Newton method.

We note that the Newton method for minimizing (14.41) would generate a search direction  $d$  by solving the linear system  $\nabla^2 Q(x, \sigma)d = -\nabla Q(x, \sigma)$ , i.e.,

$$\left( \nabla^2 L(x, \lambda) + \sigma \nabla h(x)^T \nabla h(x) \right) d = -\left( \nabla f(x) + \sigma \nabla h(x)^T h(x) \right), \quad (14.46)$$

where  $\nabla^2 L(x, \lambda)$  is defined with  $\lambda = \sigma h(x)$ . The difficulty with this approach is that the system (14.46) is ill-conditioned for large  $\sigma$  (assuming  $p < n$ ). This is one reason why the quadratic penalty approach (convenient enough as smoothness) proved unsuccessful for solving nonlinear optimization problems. A solution for eliminating this deficiency was given by Gould (1986), who introduced the vector  $q = \sigma(h(x) + \nabla h(x)d)$  at the current point  $x$ . With this, the Newton system (14.46) is equivalent to

$$\begin{bmatrix} \nabla^2 L(x, \lambda) & \nabla h(x)^T \\ \nabla h(x) & -\frac{1}{\sigma} I \end{bmatrix} \begin{bmatrix} d \\ q \end{bmatrix} = -\begin{bmatrix} \nabla f(x) \\ h(x) \end{bmatrix}, \quad (14.47)$$

which contains no large numbers and may be preferable for computing the search direction  $d$ . If  $(x, \lambda)$  is close to a local optimum  $(x^*, \lambda^*)$  and  $\sigma$  is large, any ill-conditioning in (14.47) reflects the sensitivity of  $(x^*, \lambda^*)$  to perturbations in the data of the problem.

Unfortunately, although  $d$  can be reliably computed from (14.47) when  $\sigma$  is large, this does not save the quadratic penalty method. When  $h(x)$  is not very small, then  $d$  leads away from the linearization of the constraints  $h(x) = 0$  at the current point  $x$ , and therefore, the Newton method is likely to be too slow. It is obvious that the algebraic trick used does not eliminate the deficiency of the quadratic Newton approach.

Let us now continue to study the problem (14.39) with equality constraints, bearing in mind the difficulties encountered with the quadratic penalty method when  $\sigma$  becomes very large. Let  $(x^*, \lambda^*)$  be a local minimizer, and assume that the Jacobian matrix  $\nabla h(x)$  has full row rank at  $x^*$ , i.e.,  $\text{rank}(\nabla h(x^*)) = p$ . In this minimum point, the gradient of the Lagrange function must be zero. Therefore, the required solution  $(x^*, \lambda^*)$  is a *stationary point* of the Lagrangian. In general, we cannot find  $x^*$  by minimizing  $L(x, \lambda)$  as a function of  $x$ , even if we set  $\lambda = \lambda^*$ . The second-order optimality condition for  $(x^*, \lambda^*)$  to be an isolated local minimizer is that the Hessian of the Lagrange function  $\nabla^2 L(x^*, \lambda^*)$  should be positive definite within the null space of  $\nabla h(x^*)$ . That is, the Hessian should satisfy the condition  $z^T \nabla^2 L(x^*, \lambda^*) z > 0$  for all nonzero vectors  $z$  satisfying  $\nabla h(x^*)^T z = 0$ . The following result, proved by Debreu (1952) on quadratic forms, is relevant.

**Theorem 14.7** (Debreu). Let  $H$  be an  $n \times n$  symmetric matrix and  $J$  an  $m \times n$  matrix with  $m \leq n$ . If  $z^T Hz > 0$  for every nonzero  $z$  satisfying  $Jz = 0$ , then for all  $\sigma$  sufficiently large  $H + \sigma J^T J$  is positive definite.  $\blacklozenge$

This result suggests that we should add to the Lagrangian function a term whose Hessian is  $\sigma \nabla h(x)^T \nabla h(x)$ . It is exactly what we already did when we introduced the augmented Lagrangian

$$L_A(x, \lambda, \sigma) = f(x) - \lambda^T h(x) + \frac{1}{2} \sigma \|h(x)\|^2. \quad (14.48)$$

This may be looked upon as a modification of the Lagrangian or as a shifted quadratic penalty function. For a given  $\lambda$  and  $\sigma$ , the derivatives of  $L_A(\cdot)$  are

$$\nabla L_A(x, \lambda, \sigma) = \nabla f(x) - \nabla h(x)^T \hat{\lambda}, \quad (14.49a)$$

$$\nabla^2 L_A(x, \lambda, \sigma) = \nabla^2 f(x) - \sum_{i \in E} \hat{\lambda}_i \nabla^2 h_i(x) + \sigma \nabla h(x)^T \nabla h(x), \quad (14.49b)$$

where  $\hat{\lambda} = \lambda + \sigma h(x)$ . Observe that (14.49b) illustrates Debreu's theorem.

As already seen, the augmented Lagrangian method for solving (14.39) proceeds by choosing  $\lambda$  and  $\sigma$  as judiciously as possible and then by minimizing  $L_A(x, \lambda, \sigma)$  with respect to  $x$ . The resulting  $x$ , solution of this minimizing problem is used to choose a new  $\lambda$  and  $\sigma$  and the process repeats itself. The auxiliary vector  $\hat{\lambda}$  simplifies the above process and proves to be useful in its own right.

Observe that if  $\sigma$  is reasonably large, minimizing  $L_A$  will tend to make  $\|h(x)\|$  small for any values of  $\lambda$ . Also, the Hessian  $\nabla^2 L_A(x, \lambda, \sigma)$  will tend to have positive curvature in the null space of  $\nabla h(x)$ . Therefore, the problem  $\min L_A(x, \lambda, \sigma)$  has a solution. On the other hand, since the minimization of  $L_A$  makes  $\|\nabla L_A\|$  small if  $\lambda$  is close to  $\lambda^*$ , it follows that  $(x, \lambda) \approx (x, \lambda^*)$  almost satisfies (14.40a). Moreover, if  $\|h(x)\|$  also happens to be small (because  $\sigma$  is large enough), then  $(x, \lambda)$  will almost satisfy (14.40b) as well. Therefore, the strategy is to check that  $\|h(x)\|$  is suitably small after each approximate minimization of  $L_A$ . If so,  $\lambda$  is updated as  $\lambda = \lambda - \sigma h(x)$ . If not,  $\sigma$  is judiciously increased and  $\lambda$  remains the same. Under favorable conditions, we hope that  $(x, \lambda) \rightarrow (x^*, \lambda^*)$  before  $\sigma$  becomes too large (Saunders, 2015a).

This weakness of the augmented Lagrangian method (judicious selection of  $\sigma$ , favorable conditions, etc.) determined some modifications of it. Two algorithms based on the modifications of the augmented Lagrangian are presented in this chapter. One is SPENBAR (Breitfeld, & Shanno, 1994a, b, c; Andrei, 1996a, b, c, 1998a, 2017c). In it, the inequality constraints are embedded into a modified logarithmic penalty function. The other one is MINOS (Murtagh, & Saunders, 1978, 1980, 1982, 1987, 1995). It uses a combination of successive linear programming with a modification of the augmented Lagrangian.

## 14.5 A Penalty-Barrier Algorithm (SPENBAR)

Let us consider the following general nonlinear optimization problem:

$$\begin{aligned} & \min f(x) \\ & \text{subject to:} \\ & \quad c_i(x) \geq 0, \quad i \in I_c, \\ & \quad c_i(x) = 0, \quad i \in E, \\ & \quad x_j \geq l_j, \quad j \in I_l, \\ & \quad x_j \leq u_j, \quad j \in I_u, \end{aligned} \quad (14.50)$$

where  $x \in \mathbb{R}^n$ ,  $E \triangleq \{1, \dots, m_e\}$  and  $I_c \triangleq \{1, \dots, m\}$ . The functions  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i \in I_c \cup E$ , are assumed to be twice continuously differentiable on  $\mathbb{R}^n$ .  $I_l, I_u \subseteq \{1, \dots, n\}$ . To simplify the presentation of the algorithm, the simple bounds on the variables are also denoted  $c_i(x)$ . Define  $I_{sb}$  as the set of indices, such that for all  $j \in I_l \cup I_u$  there is an  $i \in I_{sb}$  with the property

$$c_i(x) = x_j - l_j \geq 0, \quad \text{for } j \in I_l, \quad \text{or} \quad (14.51a)$$

$$c_i(x) = u_j - x_j \geq 0, \quad \text{for } j \in I_u. \quad (14.51b)$$

The set of all the inequality constraints including the simple bounds is denoted by  $I = I_c \cup I_{sb}$ . Let  $|I| = p$ , i.e.,  $I$  has  $p$  elements. Suppose that

- (i) The domain  $X_B = \{x \in \mathbb{R}^n : l_j \leq x_j, j \in I_l, \quad x_j \leq u_j, j \in I_u\}$ , which is a compact set, has a nonempty interior.
- (ii) The functions  $f(x)$ ,  $c_i(x)$ ,  $i \in E \cup I_c$  are twice continuously differentiable on  $X_B$ .

The main idea of the penalty-barrier method to find a solution for the problem (14.50) is to solve a sequence of unconstrained minimization subproblems where the solution  $x_k$  of the subproblem at iteration  $k$  is considered as an initial point for the subproblem at the next iteration. The suggested algorithm *combines* the augmented Lagrangian with a penalty function (a composite function which includes the augmented Lagrangian and the log-barrier function) in which the parameters are updated in such a way as to obtain a KKT point for problem (14.50). The constraints are treated separately as follows.

The equality constraints  $c_i(x) = 0$ ,  $i \in E$ , are embedded into the *augmented Lagrangian*. Therefore, to minimize  $f(x)$  subject to  $c_i(x) = 0$ ,  $i \in E$ , the augmented Lagrangian is defined as

$$L(x, \sigma, \lambda) \triangleq f(x) + \sum_{i=1}^{m_e} \lambda_i c_i(x) + \frac{1}{2\sigma} \sum_{i=1}^{m_e} c_i(x)^2, \quad (14.52)$$

where  $\lambda_i$ ,  $i = 1, \dots, m_e$ , are the estimates of the Lagrange multipliers and  $\sigma > 0$  is the penalty parameter. The reason for introducing the augmented Lagrangian function is to make sure that the penalty parameter  $\sigma$  does not need to be enlarged too much as to avoid the ill-conditioning of the augmented Lagrangian.

The inequality constraints  $c_i(x) \geq 0$ ,  $i \in I_c$ , are embedded into a *logarithmic penalty function*. Therefore, to minimize  $f(x)$  subject to  $c_i(x) \geq 0$ ,  $i \in I_c$ , we consider the log-barrier function

$$B(x, \sigma) \triangleq f(x) - \sigma \sum_{i=1}^m \log(c_i(x)), \quad (14.53)$$

where  $\sigma > 0$  is the penalty parameter. For solving the problem  $\min\{f(x) : c_i(x) \geq 0, i = 1, \dots, m\}$ , the following algorithm can be presented.

#### Algorithm 14.4 General barrier

- |    |  |
|----|--|
| 1. | Select an initial feasible point $x_0$ and a value for the penalty parameter $\sigma_0 > 0$ . Set $k = 0$                  |
| 2. | Solve the subproblem $\min_{x \in \mathbb{R}^n} B(x, \sigma_k)$ obtaining a local solution $x_{k+1}$                       |
| 3. | If $\sigma_k$ is small enough, then stop. Otherwise, select $\sigma_{k+1} < \sigma_k$ , set $k = k + 1$ , and go to step 2 |

Observe that when  $\sigma_k \rightarrow 0$ , the Hessian of  $B(x_k, \sigma_k)$  is ill-conditioned. On the other hand, it is difficult to select a good value for  $\sigma_{k+1}$ . The first-order minimizing condition for  $B(x, \sigma_k)$  is

$$\nabla f(x) - \sum_{i=1}^m \frac{\sigma_k}{c_i(x)} \nabla c_i(x) = 0. \quad (14.54)$$

If  $\sigma_k \rightarrow 0$ , then  $\sigma_k/c_i(x) \rightarrow \lambda_i$ , which is the Lagrange multiplier associated to the  $i$ -th inequality constraint. Therefore, the estimates of the Lagrange multipliers associated to the active constraints are computed as the ratio of two quantities, both of them tending to zero. More exactly, observe that when  $1 > c_i(x) > 0$ ,  $i \in I_c$ , then  $\log(c_i(x)) < 0$ . Hence, the second term on the right-hand side of (14.53) implies  $B(x, \sigma) \gg f(x)$  when any of the constraint functions is small and positive. Suppose that the inequality constraint problem  $\min\{f(x) : c_i(x) \geq 0, i \in I_c\}$  has a solution  $(x^*, \lambda^*)$ . Also suppose that  $\bar{\sigma}$  is a positive constant such that for all  $\sigma_k < \bar{\sigma}$ , the Hessian matrix  $\nabla^2 B(x, \sigma_k)$  of the barrier function (14.53) is positive definite for all feasible  $x$ . If  $x_k$  denotes the solution of the unconstrained problem  $\min\{B(x, \sigma_k)\}$ , then  $x_k \rightarrow x^*$  as  $\sigma_k \rightarrow 0$ . Moreover, from (14.54),  $\sigma_k/c_i(x_k) \rightarrow \lambda_i^*$ ,  $i \in I_c$ , as  $\sigma_k \rightarrow 0$ .

Jittorntrum and Osborne (1980) modified the log-barrier function (14.53) by introducing the estimates of the Lagrange multipliers as

$$J(x, \sigma, \lambda) \triangleq f(x) - \sigma \sum_{i=1}^m \lambda_i \log(c_i(x)). \quad (14.55)$$

However, this modified log-barrier function presents the same numerical difficulties as those of function  $B(x, \sigma)$ .

Polyak (1992) suggested the following *modified log-barrier function*:

$$M(x, \sigma, \lambda) \triangleq f(x) - \sigma \sum_{i \in I} \lambda_i \log \left( 1 + \frac{c_i(x)}{\sigma} \right), \quad (14.56)$$

where  $\lambda_i$ ,  $i = 1, \dots, p$ , are nonnegative estimates of the Lagrange multipliers associated to the inequality constraints and  $\sigma > 0$  is the barrier parameter. For this modified log-barrier function, Polyak (1992) established the convergence properties similar to those given by Bertsekas (1982b) for the multiplier method. Under reasonably mild conditions, Polyak showed that there exists a threshold value  $\bar{\sigma} > 0$  of the barrier parameter, such that for any fixed  $\sigma < \bar{\sigma}$ , i.e., by only updating the Lagrange multipliers  $\lambda_i$ ,  $i = 1, \dots, p$ , the sequence  $(x^k, \lambda^k)$  converges, at least at a linear rate, to a local solution  $(x^*, \lambda^*)$  of the problem (14.50). Very encouraging computational results based on the log-barrier function (14.53) were reported in Breitfeld and Shanno (1994a); Ben-Tal, Yuzefovich, and Zibulevsky (1992); and Nash, Polyak, and Sofer (1994).

The simple bounds  $x_j \geq l_j$ ,  $j \in I_l$ ,  $x_j \leq u_j$ ,  $j \in I_u$ , are handled by means of the classical logarithmic barrier terms.

Therefore, the classical log-barrier function for inequality constraints is defined by

$$C(x, \sigma) \triangleq f(x) - \sigma \sum_{i \in I} \log(c_i(x)), \quad (14.57)$$

where  $\sigma > 0$  is the barrier parameter. The log-barrier function (14.57) was introduced by Frisch (1955) and developed by Fiacco and McCormick (1968).

Our approach considers a combination of all the above functions which define a *general penalty-barrier function*. This is embedded into the general scheme of the penalty method for which the

global convergence was proved. Mainly, the algorithm considers a sequence of simple bounded minimization subproblems which are solved by means of the simple bound minimization techniques (see Chap. 12).

## The Penalty-Barrier Method

Having in view the above developments, for the general nonlinear optimization problem (14.50), the following penalty-barrier function is defined:

$$\begin{aligned} F(x, \sigma_k, \lambda^k, s^k, \beta^k) = & f(x) - \sigma_k \sum_{i \in I} \lambda_i^k \Psi^k(c_i(x)) \\ & + \sum_{i=1}^{m_e} \lambda_i^k c_i(x) + \frac{1}{2\sigma_k} \sum_{i=1}^{m_e} c_i(x)^2, \end{aligned} \quad (14.58)$$

where  $k \geq 0$  and

$$\Psi^k(c_i(x)) = \log \left( s_i^k + \frac{c_i(x)}{\sigma_k} \right), \quad \text{for } c_i(x) \geq -\beta^k \sigma_k s_i^k \text{ and } i \in I_c, \quad (14.59a)$$

$$\Psi^k(c_i(x)) = Q^k(c_i(x)), \quad \text{for } c_i(x) < -\beta^k \sigma_k s_i^k \text{ and } i \in I_c, \quad (14.59b)$$

$$\Psi^k(c_i(x)) = \log \left( \frac{c_i(x)}{\sigma_k} \right), \quad \text{for } i \in I_{sb}, \quad (14.59c)$$

where  $Q^k(c_i(x))$  is a quadratic extrapolation function defined as

$$Q(c_i(x)) = \frac{1}{2} q_i^a (c_i(x))^2 + q_i^b c_i(x) + q_i^c. \quad (14.60)$$

The coefficients  $q_i^a$ ,  $q_i^b$ , and  $q_i^c$  are determined in such a way so that the function value as well as the first and the second derivatives of  $Q(c_i(x))$  match the corresponding values of the logarithmic terms (14.59a) at  $c_i(\hat{x}) = -\beta^k \sigma_k s_i^k$ . A simple computation shows that

$$q_i^a = \frac{-1}{(s_i^k \sigma_k (1 - \beta^k))^2}, \quad (14.61)$$

$$q_i^b = \frac{1 - 2\beta^k}{s_i^k \sigma_k (1 - \beta^k)^2}, \quad (14.62)$$

$$q_i^c = \frac{\beta^k (2 - 3\beta^k)}{2(1 - \beta^k)^2} + \log [s_i^k (1 - \beta^k)]. \quad (14.63)$$

The values of the penalty parameter  $\sigma_k$  are positive scalars which tend to zero as  $k$  goes to infinity. For the shifts  $s_i^k$  and the extrapolation parameter  $\beta^k$ , we impose

$$0 \leq \beta^k \leq \beta_u < 1, \quad (14.64a)$$

$$1 \leq s_i^k \leq s_u < +\infty, \quad i \in I_c, \quad (14.64b)$$

where  $\beta_u \geq 0$  and  $s_u \geq 1$  for all  $i \in I_c$  and  $k \geq 0$ .

The algorithm finds an approximate local minimum of (14.50) by repeatedly solving the  $k$ -th subproblem

$$\min_{x \in X_B} F(x, \sigma_k, \lambda^k, s^k, \beta^k), \quad (14.65)$$

such that

$$\|\nabla_x F(x_{k+1}, \sigma_k, \lambda^k, s^k, \beta^k)\|_2 \leq \varepsilon_k, \quad (14.66)$$

where  $\varepsilon_k$  is the convergence tolerance corresponding to the  $k$ -th subproblem and  $\varepsilon_k \rightarrow 0$  as  $k \rightarrow +\infty$ . Observe that (14.65) is a simple bounded constrained optimization.

Extrapolating the modified log-barrier terms was suggested by Ben-Tal, Yuzefovich, and Zibulevsky (1992). The quadratic extrapolating terms are defined beyond the singularities of the logarithmic terms (14.59a). This allows an efficient line-search for the unconstrained minimization and the reduction of the penalty-barrier parameter  $\sigma$  without restrictions. To ensure that the logarithmic terms (14.59a) are well defined in case the quadratic extrapolation was not used, the shifts  $s^k$  are selected such that

$$\frac{c_i(x)}{\sigma_k} - s^k > 0 \quad (14.67)$$

for all  $i \in I$ ,  $k \geq 0$ .  $x = x_k$  is the starting point for the  $k$ -th subproblem.

Updating the multiplier estimates  $\lambda^k$  and the penalty-barrier parameters  $\sigma_k$  is made in such a way so that the iterates are convergent to a stationary point  $x^*$  of the problem (14.50), i.e., there exists  $\lambda^*$  such that  $(x^*, \lambda^*)$  satisfies the first-order optimality conditions

$$\nabla f(x^*) - \sum_{i \in E \cup I} \lambda_i^* \nabla c_i(x^*) = 0, \quad (14.68a)$$

$$\sum_{i \in E \cup I} \lambda_i^* c_i(x^*) = 0, \quad (14.68b)$$

$$c_i(x^*) \geq 0 \text{ and } \lambda_i^* \geq 0, \quad i \in I, \quad (14.68c)$$

$$c_i(x^*) = 0, \quad i \in E. \quad (14.68d)$$

The initialization of the estimates of the Lagrange multipliers is given by a vector  $\lambda^0$  computed as solution of the problem

$$\min_{\lambda_I \geq 0} \left\| \nabla f(x_0) - \sum_{i \in E \cup I} \lambda_i \nabla c_i(x_0) \right\|_2, \quad (14.69)$$

where  $\lambda_I = (\lambda_i)_{i \in I}$  and  $x_0$  is the initial estimation of  $x$ . Having in view the combinatorial nature of solving (14.69), a procedure is to compute a solution  $\hat{\lambda}$  of (14.69) by neglecting the restrictions  $\lambda_I \geq 0$  and then setting

$$\lambda_i^0 = \begin{cases} \min \left\{ \max \left\{ \lambda_l, \hat{\lambda}_i \right\}, \lambda_u \right\}, & \text{if } i \in I \text{ or } i = 1, \dots, m_e, \quad \hat{\lambda}_i \geq 0, \\ \max \left\{ \min \left\{ -\lambda_l, \hat{\lambda}_i \right\}, -\lambda_u \right\}, & \text{if } i = 1, \dots, m_e, \quad \hat{\lambda}_i < 0, \end{cases} \quad (14.70)$$

where  $\lambda_l$  and  $\lambda_u$  are the lower and upper bounds for the initial estimates of the multipliers, which are imposed by numerical reasons. Usually,  $\lambda_l = 1$  and  $\lambda_u = 100$ .

The first derivative of  $F(x, \sigma_k, \lambda^k, s^k, \beta^k)$  is

$$\begin{aligned}\nabla_x F(x, \sigma_k, \lambda^k, s^k, \beta^k) &= \nabla f(x) - \sigma_k \sum_{i \in I} \lambda_i^k \Psi'(c_i) \nabla c_i(x) \\ &\quad - \sum_{i=1}^{m_e} \left[ \lambda_i^k - \frac{c_i(x)}{\sigma_k} \right] \nabla c_i(x),\end{aligned}\tag{14.71}$$

where  $\Psi'(c_i)$  is the first derivative of  $\Psi^k$  with respect to  $c_i$ , i.e.,

$$\Psi'(c_i(x)) = \begin{cases} \frac{1}{\sigma_k s_i^k + c_i(x)}, & c_i(x) \geq -\beta^k \sigma_k s_i^k, \quad i \in I_c, \\ q_i^a c_i(x) + q_i^b, & c_i(x) < -\beta^k \sigma_k s_i^k, \quad i \in I_c, \\ \frac{1}{c_i(x)}, & i \in I_{sb}, \end{cases}\tag{14.72}$$

where  $q_i^a$  and  $q_i^b$  are given by (14.61) and (14.62), respectively. The first-order optimality conditions (14.68) and (14.71) suggest the following scheme for updating the Lagrange multiplier estimates:

$$\lambda_i^{k+1} = \begin{cases} \lambda_i^k - \frac{c_i(x_{k+1})}{\sigma_k}, & i \in E, \\ \sigma_k \lambda_i^k \Psi'(c_i(x_{k+1})), & i \in I, \end{cases}\tag{14.73}$$

where  $x_{k+1}$  is the solution of the  $k$ -th subproblem (14.65) satisfying (14.66). Observe that (14.73) requires that the components  $\lambda_i^0$ ,  $i \in I$ , should be positive; otherwise, if there is a  $j \in I$  with  $\lambda_j^0 = 0$ , then for all  $k \geq 1$ ,  $\lambda_j^k = 0$ .

The penalty-barrier parameter  $\sigma$  is initialized by a positive scalar  $\sigma_0$  and then updated by

$$\sigma_{k+1} = \frac{\sigma_k}{\gamma},\tag{14.74}$$

for  $k \geq 0$ , where  $\gamma > 1$  ensures that  $\sigma_k \rightarrow 0$  as  $k \rightarrow +\infty$ . ( $\sigma_0 = 10^{-1}$  and  $\gamma = 10$ .)

The parameter  $\beta$  defines how close to singularities the logarithmic terms (14.59a) are extrapolated. Breitfeld and Shanno (1994c) show that if  $\beta$  is too small ( $0 \leq \beta^k < 0.5$ ), then the minimum of an individual penalty term (14.59a) or (14.59b) might be very far from the feasibility domain of (14.50), and therefore, this term does not impose a penalty for being infeasible. On the other hand, if  $\beta$  is too close to one ( $\beta = 0.99$ ), then the big function values of the logarithmic terms are no longer avoided, and therefore, the ill-conditioning is introduced into the penalty-barrier function. Hence,  $\beta$  is initialized by  $\beta^0$  and updated as

$$\beta^{k+1} = \beta^k \theta^k,\tag{14.75}$$

for  $k \geq 0$ , where  $\beta^0$  and  $\theta^k$  are scalars such that  $0 \leq \beta^k \leq \beta_u$  for all  $k \geq 0$  with  $\beta_u < 1$ . ( $\beta^0 = 0.9$  and  $\theta^k = 1$ , for all  $k \geq 0$ .)

The shifting parameters  $s$  allow the scaling of the constraints in order to avoid the ill-conditioning of the terms (14.59b). We can show it by introducing the general shifts  $s$  into the modified barrier terms. The scaling of the inequality constraints, such that  $\|c(x)\| = 1$ , can be achieved without reformulating the problem (14.50). To see this, consider  $x$  for which  $\|c(x)\| > 0$  and let  $s = \|c(x)\| > 0$ . Then, the scaled constraint is defined as  $\tilde{c}(x) = c(x)/s \geq 0$ . Clearly,  $\|\tilde{c}(x)\| = 1$ . With this, Polyak's modified log-barrier term (14.56) for  $\tilde{c}(x)$  is

$$\log \left( 1 + \frac{\hat{c}(x)}{\sigma} \right) = \log \left( 1 + \frac{c(x)}{s\sigma} \right) = \log \left( s + \frac{c(x)}{\sigma} \right) - \log(s).$$

The last term,  $\log(s)$ , is constant with respect to  $x$ , so it can be ignored in the minimization of the penalty-barrier function. Therefore, the scaling of the inequality constraints can be readily achieved by simply choosing the appropriate shifts  $s$ .

On the other hand, in order to avoid the ill-conditioning of the penalty-barrier function introduced by the quadratic extrapolation, we set

$$s_i^k = \min \{ \max \{ 1, -c_i(x_k) \}, s_u \}, \quad (14.76)$$

for all  $i \in I_c$  and  $k \geq 0$ , which basically scales the quadratic terms with respect to the constraint values. Usually,  $s_u = 10^3$ .

Concerning the stopping criteria, we take the advantage of estimating the Lagrange multipliers along the iterations for solving (14.65). Therefore, a KKT point can be identified, and we terminate the penalty-barrier method for a  $k \geq 0$  for which

$$v_1^k \leq \tau, \quad (14.77)$$

or

$$v_2^k \leq \tau \text{ and } v_3^k \leq \tau_f, \quad (14.78)$$

where  $\tau > 0$  is the convergence tolerance,  $\tau_f = 10^{-2}\tau$  and

$$v_1^k = \max \{ \max_{i=1,\dots,m_e} \{ |c_i(x_k)| \}, -\min_{i \in I} \{ c_i(x_k) \}, \\ \frac{\sum_{i \in E \cup I} |\lambda_i^k c_i(x_k)|}{1 + \|x_k\|_2}, \frac{\left\| \nabla f(x_k) - \sum_{i \in E \cup I} \lambda_i^k \nabla c_i(x_k) \right\|_\infty}{1 + \|x_k\|_2} \}, \\ v_2^k = \max \{ \max_{i \in E} \{ |c_i(x_k)| \}, -\min_{i \in I} \{ c_i(x_k) \} \}, \\ v_3^k = \frac{|f(x_k) - f(x_{k-1})|}{1 + |f(x_k)|}.$$

Observe that if  $v_1^k \leq \tau$ , then the feasibility, the scaled complementarity, and the scaled first-order optimality conditions are satisfied, i.e., the current point is a KKT point with accuracy  $\tau$ .

Finally, for having the penalty-barrier algorithm, we must specify the optimization method used for solving (14.65) for each set of parameters  $(\sigma_k, \lambda^k, s^k, \beta^k)$ . Since (14.65) is a simple bounded constrained optimization problem, we can apply any method for solving this type of problems (the spectral projected gradient algorithm by Birgin, Martínez, & Raydan, 2000, 2001); the limited-memory BFGS with gradient projection by Byrd, Lu, and Nocedal (1995a); or the truncated Newton with simple bounds (Nash, 1984a, b, 1985). In our implementation of the algorithm, we have used the truncated Newton with simple bounds (TNBC) (Nash, 1984a, b, 1985). The approximate solution  $x_k$  is used as the starting point for the next subproblem. Now, the following penalty-barrier algorithm with quadratic extrapolation of the inequality constraints can be presented (Andrei, 1996a, b, c, 1998a).

**Algorithm 14.5** *Penalty-barrier—SPENBAR—Andrei*

1. Choose:  $x_0 \in \text{int}(X_B)$ ,  $\tau > 0$  and the sequences  $\{\varepsilon_k\}$  and  $\{\theta_k\}$ . Determine  $\lambda^0$  as in (14.70) and  $s_i^0 = \min \{ \max \{1, -c_i(x_0)\}, s_u \}$ ,  $i \in I$ . Set  $k = 0$
2. Test of convergence. If  $v_1^k \leq \tau$  or if  $v_2^k \leq \tau$  and  $v_3^k \leq \tau_f$ , stop. Otherwise, go to step 3
3. Using  $x_k$  as the initial point, determine  $x_{k+1} \in \text{int}(X_B)$  for which
$$\frac{\|\nabla_x F(x, \sigma_k, \lambda^k, s^k, \beta^k)\|_2}{1 + \|x\|_2} \leq \varepsilon_k$$
4. Update the parameters
$$\begin{cases} \lambda_i^{k+1} = \begin{cases} \lambda_i^k - \frac{c_i(x_{k+1})}{\sigma_k}, & i \in E, \\ \sigma_k \lambda_i^k \Psi'(c_i(x_k)), & i \in I, \end{cases} \\ \sigma_{k+1} = \sigma_k / \gamma, \\ \beta^{k+1} = \beta^k \theta^k, \\ s_i^k = \min \{ \max \{1, -c_i(x_{k+1})\}, s_u \}, \quad i \in I. \end{cases}$$
Set  $k = k + 1$  and go to step 2

◆

We notice that SPENBAR has two types of iterates. The first one is the so-called major (external) iteration, in which the test of convergence in step 2 is checked. The second is the *minor (internal) iteration*, in which the simple bounded optimization subproblem (14.65) in step 3 is solved.

## Global Convergence

Let  $\{x_k\}$  be the sequence generated by the penalty-barrier algorithm. The global convergence is proved under the assumption that the sequence  $\{x_k\}$  converges. To prove the global convergence of  $\{x_k\}$ , suppose that the following three additional assumptions hold:

- (i) The set  $X_B$  is bounded.
- (ii) The gradients of the active constraints at the limit point of the sequence  $\{x_k\}$  generated by the algorithm are linearly independent.
- (iii) The initial estimates of the Lagrange multipliers corresponding to the inequalities and to the simple bounds are positive.

The following two propositions proved by Breitfeld and Shanno (1994c) refer to the estimates of the Lagrange multipliers that correspond to the inequality constraints including the simple bounds, under the updating formula (14.73).

**Proposition 14.1** (*Positivity of the Lagrange Multipliers*). *Suppose that (i), (ii), and (v) hold. Under the updating scheme (14.73), the estimates of the Lagrange multipliers  $\lambda_i^k$  are positive for all  $i \in I$  and  $k \geq 0$ .*

**Proof** For all  $i \in I$ , by assumption,  $\lambda_i^0 > 0$ . The proposition is proved by induction, and suppose that  $\lambda_i^k > 0$  for all  $i \in I$  and some  $k \geq 0$ . For the case (14.59a) and for all  $i \in I_c$  and  $k \geq 0$ , from (14.73), we get

$$\lambda_i^{k+1} = \frac{\lambda_i^k}{s_i^k + c_i(x_{k+1})/\sigma_k} > 0.$$

The positivity follows from  $c_i(x_{k+1}) \geq -\beta^k \sigma_k s_i^k$  and from

$$s_i^k + \frac{c_i(x_{k+1})}{\sigma_k} \geq s_i^k(1 - \beta^k) \geq 1 - \beta_u > 0,$$

where  $\beta_u < 1$  is the upper bound of  $\beta$ . For the case (14.59b), from (14.61), since  $c_i(x_{k+1}) < -\beta^k \sigma_k s_i^k$ , it follows that

$$\begin{aligned} \lambda_i^{k+1} &= \sigma_k \lambda_i^k \left( \frac{-c_i(x_{k+1})}{(s_i^k \sigma_k (1 - \beta^k))^2} + \frac{1 - 2\beta^k}{s_i^k \sigma_k (1 - \beta^k)^2} \right) \\ &> \lambda_i^k \left( \frac{\beta^k}{s_i^k (1 - \beta^k)^2} + \frac{1 - 2\beta^k}{s_i^k (1 - \beta^k)^2} \right) = \frac{\lambda_i^k}{s_i^k (1 - \beta^k)} \geq \frac{\lambda_i^k}{s_u} > 0 \end{aligned}$$

for all  $i \in I_c$  and  $k \geq 0$ , where  $s_u$  is the upper bound of the shifts  $s$ . For the simple bounds, since  $\sigma_k > 0$  and  $x_{k+1}$  is strictly feasible with respect to the simple bounds, it follows that

$$\lambda_i^{k+1} = \lambda_i^k \frac{\sigma_k}{c_i(x_{k+1})} > 0,$$

for all  $i \in I_{sb}$  and  $k \geq 0$ . ◆

**Proposition 14.2 (Complementarity Slackness).** Suppose that (i), (ii), and (v) hold and the sequence  $\{x_k\}$  generated by the algorithm converges to the point  $\bar{x}$ . Let  $I_{in}(\bar{x}) = \{i : c_i(\bar{x}) > 0, i \in I\}$  be the set of indices of the inactive inequality constraints at  $\bar{x}$ . Then,

$$\{\lambda_i^k\}_{k \geq 0} \rightarrow \bar{\lambda}_i = 0$$

for all  $i \in I_{in}(\bar{x})$ .

**Proof** For every  $i \in I_{in}(\bar{x})$ , there is a  $\delta > 0$  such that  $c_i(\bar{x}) \geq \delta$ . Therefore, for all  $k \geq \bar{k}$  and  $i \in I_{in}(\bar{x})$ , there exists a  $\bar{k} \geq 0$  such that

$$c_i(x_k) \geq \frac{\delta}{2} > 0. \quad (14.79)$$

Observe that for the positive general constraints, the logarithmic terms (14.59a) are utilized. For estimating the multipliers of the general constraints, from (14.73), (14.79), and  $s_i^k \geq 1$ , it follows that

$$\begin{aligned} \lambda_i^{k+1} &= \frac{\lambda_i^k}{s_i^k + c_i(x_{k+1})/\sigma_k} \\ &\leq \lambda_i^k \frac{\sigma_k}{\sigma_k + c_i(x_{k+1})} \leq \lambda_i^k \frac{\sigma_k}{\sigma_k + \delta/2}, \end{aligned} \quad (14.80)$$

for all  $i \in I_{in}(\bar{x}) \cap I_c$  and  $k \geq \bar{k}$ . From (14.80), we see that  $\lambda_i^{k+1} < \lambda_i^k$  for  $k \geq \bar{k}$ . Therefore,

$$\lambda_i^{k+1} = \lambda_i^k \frac{\sigma_k}{\sigma_k + \delta/2}, \quad (14.81)$$

for all  $i \in I_{in}(\bar{x}) \cap I_c$  and  $k \geq \bar{k}$ . On the other hand, for the simple bounds,

$$\lambda_i^{k+1} = \lambda_i^k \frac{\sigma_k}{c_i(x_{k+1})} \leq \lambda_i^k \frac{\sigma_k}{\delta/2} \quad (14.82)$$

for all  $i \in I_{in}(\bar{x}) \cap I_{sb}$  and  $k \geq \bar{k}$ . Since  $\sigma_k \rightarrow 0$ , there is a  $\hat{k} \geq \bar{k}$  such that  $\sigma_k \leq \delta/2$  for all  $k \geq \hat{k}$ , and therefore, (14.82) implies  $\lambda_i^{k+1} < \lambda_i^k$  for  $k \geq \hat{k}$ . Using this together with (14.81), we get

$$\lambda_i^{k+1} \leq \lambda_i^k \frac{\sigma_k}{\delta/2} \quad (14.83)$$

for all  $k \geq \hat{k}$  and  $i \in I_{in}(\bar{x})$ . Since  $\sigma_k \rightarrow 0$  as  $k \rightarrow +\infty$ ,  $\lambda_i^k$  is positive from Proposition 14.1 and since  $\lambda_i^k$  is finite, it follows that  $\{\lambda_i^k\}_{k \geq 0} \rightarrow 0$  for all  $i \in I_{in}(\bar{x})$ , which proves the proposition.  $\blacklozenge$

The following result establishes the convergence of the sequence  $\{x_k\}$  and of the estimates sequence of the Lagrange multipliers to a KKT point which satisfies the first-order optimality conditions (14.68). We follow the proof given by Breitfeld and Shanno (1994c).

**Theorem 14.8 (KKT Theorem).** Suppose that  $\{x_k\}$  converges to a point  $\bar{x}$  and the assumptions (i)–(v) hold. Then, the sequence of the multiplier estimates  $\{\lambda^k\}$  converges to a limit point  $\bar{\lambda}$  such that  $(\bar{x}, \bar{\lambda})$  satisfies the first-order optimality conditions (14.68), i.e., it is a KKT point of the problem (14.50).

**Proof** The assumptions (i) and (ii) guarantee that  $F(x, \sigma_k, \lambda^k, s^k, \beta^k)$  is well defined for all  $x \in \text{int}(X_B)$ ,  $k \geq 0$ , and there is an  $\hat{x} \in \text{int}(X_B)$  for which

$$F(\hat{x}, \sigma_k, \lambda^k, s^k, \beta^k) < \infty. \quad (14.84)$$

Now, let us define

$$F(x, \sigma_k, \lambda^k, s^k, \beta^k) = \infty \text{ for } x \in X_B \setminus \text{int}(X_B). \quad (14.85)$$

Observe that  $F(x, \sigma_k, \lambda^k, s^k, \beta^k)$  is continuous and  $X_B$  is a compact set because it is bounded and closed. Therefore, by Theorem 11.1 (Weierstrass),  $F(x, \sigma_k, \lambda^k, s^k, \beta^k)$  attains its minimum in  $X_B$ , which together with (14.84) and (14.85) imply that the approximate minimum  $x_{k+1} \in \text{int}(X_B)$ , for all  $k \geq 0$ . Therefore  $\bar{x} \in X_B$ .

Now, let us show that the estimates of the Lagrange multipliers corresponding to the active constraints are convergent. For simplicity, let us denote  $F(x_k, \sigma_{k-1}, \lambda^{k-1}, s^{k-1}, \beta^{k-1})$  by  $F^k$ . Therefore, from (14.71) and (14.73), we have

$$\nabla_x F^k = \nabla f(x_k) - \sum_{i \in I} \lambda_i^k \nabla c_i(x_k) - \sum_{i \in E} \lambda_i^k \nabla c_i(x_k), \quad (14.86)$$

for all  $k \geq 1$ . In the following, we consider the multiplier estimates of the inactive constraints at  $\bar{x}$  separately from the others. The vectors or the matrices corresponding to the constraints that are inactive at  $\bar{x}$  are denoted by the subscript  $in$ . For example, the vector of the multiplier estimates of the inactive constraints at  $\bar{x}$  is denoted by  $\lambda_{in}$  with  $\lambda_{in} = (\lambda_i)_{i \in I_{in}(\bar{x})}$ . On the other hand, the vectors and the matrices corresponding to the remaining constraints are denoted by the subscript  $ac$ . These are all equalities and inequalities that are active at  $\bar{x}$ , i.e., all the indices  $i \in E \cup I \setminus I_{in}(\bar{x})$ . With this, (14.86) can be rewritten as

$$\nabla_x F^k = \nabla f(x_k) - \nabla C_{ac}(x_k)^T \lambda_{ac}^k - \nabla C_{in}(x_k)^T \lambda_{in}^k, \quad (14.87)$$

where  $\nabla C_{ac}(\cdot)$  is the Jacobian matrix whose rows are the gradients of the constraints  $c_i, i \in E \cup I \setminus I_{in}(\bar{x})$  and  $\nabla C_{in}(\cdot)$  is the Jacobian matrix corresponding to the inactive constraints at  $\bar{x}$ . Observe that (iv) implies that  $\nabla C_{ac}(\bar{x})$  has full rank. Therefore, there is a  $\bar{k} \geq 0$  such that  $\nabla C_{ac}(x_k)$  has full rank for all  $k \geq \bar{k}$ . Thus, (14.87) can be rewritten as

$$\lambda_{ac}^k = \left[ \nabla C_{ac}(x_k) \nabla C_{ac}(x_k)^T \right]^{-1} \nabla C_{ac}(x_k) \left[ \nabla f(x_k) - \nabla C_{in}(x_k)^T \lambda_{in}^k - \nabla_x F^k \right]$$

for all  $k \geq \bar{k}$ . Since  $x \in \text{int}(X_B)$ , it results that  $\nabla C_{in}(x)$  is a bounded matrix (by assumption (ii)),  $\lambda_i^k \rightarrow 0$  for  $i \in I_{in}(\bar{x})$  (by Proposition 14.2), and  $\|\nabla_x F^k\| \rightarrow 0$ , so it follows that if  $k \rightarrow +\infty$ , then

$$\lambda_{ac}^k \rightarrow \left[ \nabla C_{ac}(\bar{x}) \nabla C_{ac}(\bar{x})^T \right]^{-1} \nabla C_{ac}(\bar{x}) \nabla f(\bar{x}). \quad (14.88)$$

Now, let us define

$$\bar{\lambda}_{ac} = \left[ \nabla C_{ac}(\bar{x}) \nabla C_{ac}(\bar{x})^T \right]^{-1} \nabla C_{ac}(\bar{x}) \nabla f(\bar{x}). \quad (14.89)$$

Observe that by (ii), the components of  $\bar{\lambda}_{ac}$  are finite. From (14.87) and (14.89), since  $\|\nabla_x F^k\| \rightarrow 0$  and  $\lambda_i^k = 0$  for all  $i \in I_{in}(\bar{x})$ , we get

$$\nabla f(\bar{x}) - \sum_{i \in E \cup I} \bar{\lambda}_i \nabla c_i(\bar{x}) = 0.$$

From Proposition 14.2 and from (14.88), we can see that there exist  $\delta > 0$  and  $\bar{k} \geq 0$  with  $\bar{\lambda}_i - \delta \leq \lambda_i^k \leq \bar{\lambda}_i + \delta$  for all  $i \in E \cup I$  and  $k \geq \bar{k}$ , which means that the sequence  $\{\lambda^k\}_{k \geq \bar{k}}$  is bounded. From (14.73), we get

$$\lambda_i^{k+1} = \lambda_i^k - \frac{c_i(x_{k+1})}{\sigma_k}, \quad (14.90)$$

for all  $i \in E$  and  $k \geq 0$ . This, together with the boundedness of  $\lambda^k$ , implies that the sequence  $\{c_i(x_{k+1})/\sigma_k\}_{k \geq \bar{k}}$  is bounded for all  $i \in E$ . Since  $\sigma_k \rightarrow 0$ , we must have  $c_i(\bar{x}) = 0$  for all  $i \in E$ . Therefore, the feasibility of  $\bar{x}$  with respect to the equality constraints is established.

Now, let us show that  $\bar{x}$  is feasible with respect to the inequality constraints, i.e.,  $c_i(\bar{x}) \geq 0$  for all  $i \in I$ . Observe that it is necessary to consider the general constraints  $c_i, i \in I_c$ , because the simple bounds are always strictly feasible. Let us assume that there is an index  $j \in I_c$  with  $c_j(\bar{x}) = -\delta < 0$ . Then, there exists a  $\bar{k} \geq 0$  such that

$$c_j(x_k) \leq -\frac{\delta}{2} \quad (14.91)$$

for all  $k \geq \bar{k}$ . Now, from (14.64), it follows that  $-\beta^k \sigma_k s_j^k \rightarrow 0$  when  $\sigma_k \rightarrow 0$ . Thus, there is a  $\hat{k} \geq \bar{k}$  with

$$c_j(x_k) \leq -\frac{\delta}{2} < -\beta^k \sigma_k s_j^k \quad (14.92)$$

for all  $k \geq \hat{k}$ , and hence, the quadratic extrapolation is used. Consequently, (14.64) and (14.91) imply that

$$\begin{aligned} \lambda_j^{k+1} &= \sigma_k \lambda_j^k \left( \frac{-c_j(x_{k+1})}{(s_j^k \sigma_k (1 - \beta^k))^2} + \frac{1 - 2\beta^k}{s_j^k \sigma_k (1 - \beta^k)^2} \right) \\ &\geq \lambda_j^k \left( \frac{\delta/2}{\sigma_k (s_j^k (1 - \beta^k))^2} - \frac{1}{s_j^k (1 - \beta^k)^2} \right) \geq \lambda_j^k \left( \frac{\delta/2}{\sigma_k s_u^2} - \frac{1}{(1 - \beta_u)^2} \right) \end{aligned} \quad (14.93)$$

for all  $k \geq \hat{k}$ , where  $\hat{k}$  is chosen as in (14.92) with the additional requirement that

$$\sigma_k \leq \frac{\delta}{2s_u^2 \left( 1 + 1/(1 - \beta_u)^2 \right)}.$$

Then, from (14.93), it follows that  $\lambda_j^{k+1} \geq \lambda_j^k$  for  $k \geq \hat{k}$ , and therefore,

$$\lambda_j^{k+1} \geq \hat{\lambda}_j^k \left( \frac{\delta/2}{\sigma_k s_u^2} - \frac{1}{(1 - \beta_u)^2} \right) \quad (14.94)$$

for  $k \geq \hat{k}$ . As  $\sigma_k \rightarrow 0$ , (14.94) implies that  $\lambda_j^k \rightarrow +\infty$ , which contradicts the finiteness of the multipliers  $\bar{\lambda}_j$ ,  $j \in I_c$  (see (14.89) and Proposition 14.2). Therefore,  $\bar{x}$  is feasible.

Finally, let us argue that the complementary slackness holds at  $(\bar{x}, \bar{\lambda})$ , i.e.,

$$\bar{\lambda}_i c_i(\bar{x}) = 0 \quad (14.95)$$

for all  $i \in E \cup I$ . If  $c_i(\bar{x}) = 0$  for some  $i \in E \cup I$ , it is clear that (14.95) is satisfied. Otherwise,  $c_i(\bar{x}) = \delta > 0$ , from Proposition 14.2, it follows that  $\bar{\lambda}_i = 0$ . Thus, (14.95) holds. Furthermore, from Proposition 14.1, it follows that  $\bar{\lambda}_i \geq 0$  for all  $i \in I$ .

In conclusion, we have proved that  $(x_k, \lambda^k)$  converges to a point  $(\bar{x}, \bar{\lambda})$  satisfying (14.68). Hence, the limit point is a KKT point of the problem (14.50), which completes the proof of the theorem.  $\blacklozenge$

The convergence of the penalty-barrier algorithm without assuming that the sequence of iterates  $\{x_k\}$  is convergent is proved in Breitfeld and Shanno (1994c). The convergence of the algorithm when  $\{x_k\}$  is not convergent is proved under the assumptions that the gradients of the active constraints at a limit point of  $\{x_k\}_{k \in K}$  are linearly independent, where  $K$  is a subset of  $\{k : k \geq 0\}$  and the sequence of the Lagrange multiplier estimates  $\{\lambda^k\}_{k \geq 0}$  stays bounded. The proof is quite technical and it is not presented here.

For the convex programming, the convergence results under mild conditions are given by Ben-Tal and Zibulevsky (1993). For the modified log-barrier method, under the convexity assumption, the convergence is given in Jensen and Polyak (1992). Moreover, Polyak (1992) proves the convergence for the modified log-barrier method applied to non-convex nonlinear programs under nondegeneracy and second-order sufficiency assumptions. The above theoretical results assume the exactness of the solutions to the unconstrained subproblems. For an augmented Lagrangian algorithm, Conn, Gould,

and Toint (1991a) establish the global convergence, allowing the inexact minimization of the augmented Lagrangian without the assumption that the sequence of the Lagrange multiplier estimates stays bounded. A similar approach used in the SPENBAR algorithm is considered by Conn, Gould, and Toint (1992a). The Lagrangian barrier function comprises similar modified barrier terms, as (14.59a). Conn, Gould, and Toint (1992a) remark that the convergence theory extends to a composite Lagrangian barrier, i.e., the augmented Lagrangian algorithm. In this algorithm, the update strategy also allows to show that the penalty parameter is bounded away from zero, and therefore, the potential ill-conditioning of the augmented Lagrangian function can be avoided. Also, Powell (1969) suggests updating the parameters in a similar way, depending on the reduction in the infeasibility of the constrained problem.

In the following, let us present some examples illustrating the numerical performances of SPENBAR, in which the subproblems (14.65) are solved by the truncated Newton with simple bounds (TNBC).

**Example 14.2** (Schittkowski, 1987, p. 186)

$$\begin{aligned} \min & [-5(x_1 + x_2) - 4x_3 - x_1x_3 - 6x_4 - \frac{5x_5}{1+x_5} \\ & - \frac{8x_6}{1+x_6} - 10(1 - 2\exp(-x_7) + \exp(-2x_7))] \end{aligned}$$

subject to

$$\begin{aligned} & 2x_4 + x_5 + 0.8x_6 + x_7 - 5 = 0, \\ & x_2^2 + x_3^2 + x_5^2 + x_6^2 - 5 = 0, \\ & 10 - x_1 - x_2 - x_3 - x_4 - x_5 - x_6 - x_7 \geq 0, \\ & 5 - x_1 - x_2 - x_3 - x_4 \geq 0, \\ & 5 - x_1 - x_3 - x_5 - x_6^2 - x_7^2 \geq 0, \\ & x_i \geq 0, \quad i = 1, \dots, 7. \end{aligned}$$

Consider the initial point  $x_0 = [0.1, 0.1, \dots, 0.1]$ , for which  $f(x_0) = -3.282377$ . Table 14.3 presents the results given by SPENBAR.

**Table 14.3** Iterations generated by SPENBAR.  $\sigma_0 = 0.1$ ,  $\beta = 0.9$ ,  $\tau = 10^{-8}$

$k$	1	2	3	4	5
$\sigma_k$	0.1	0.01	0.001	0.0001	0.00001
#itin	40	41	39	8	2
#nf	200	162	175	42	9
$x_1$	1.460992	1.46663	1.468803	1.468808	1.468808
$x_2$	1.940633	1.983322	1.983970	1.983971	1.983971
$x_3$	0.464774	0.354626	0.351885	0.351878	0.351878
$x_4$	1.202136	1.195411	1.195342	1.195342	1.195342
$x_5$	0.577817	0.570095	0.569399	0.569396	0.569396
$x_6$	0.774228	0.784720	0.784745	0.784746	0.784746
$x_7$	1.379950	1.411455	1.412121	1.412122	1.412122
$f(x)$	-37.68229	-37.41242	-37.41296	-37.41296	-37.41296

In this table, #itin and #nf represent the number of iterations and the number of function evaluations, respectively, in the truncated Newton method for solving subproblem (14.65).

Note that towards the end of the solving process, the number of iterations *itin* is smaller and smaller. This is a characteristic of SPENBAR (Andrei, 1996a, b, c). Observe that SPENBAR takes 5 major iterations, at each of them one subproblem (14.65) being solved, and a total of 130 minor iterations for solving (14.65) by the truncated Newton with simple bounds. The total number of function evaluations is 588. Observe that along the iterations, the values of the penalty parameter  $\sigma_k$  are quite reasonable. In SPENBAR, the Jacobian matrices  $\nabla c(x)$  are considered as sparse matrices. Therefore, it is able to solve large-scale nonlinear optimization problems.

**Example 14.3** (*Optimization of a heavy body*) (Brown & Bartholomew-Biggs, 1987; Andrei, 2003, Application A10, p. 356).

$$\begin{aligned} & \min (x_1 + 0.5x_3 + x_4) \\ & \text{subject to} \\ & x_1 - x_6^2 - 0.1 = 0, \\ & x_3 - x_7^2 = 0, \\ & x_4 - x_8^2 = 0, \\ & z^2 + y^2 - 4 = 0, \\ & zu + vy = 0, \\ & uy - vz - 0.7 = 0, \end{aligned}$$

where

$$\begin{aligned} z &= 0.1(x_1 + x_3 + x_4) + 0.01(x_1^2 + 2x_1x_3 + 2x_1x_4) \cos x_2 + 0.01x_4^2 \cos x_5, \\ y &= 1 + 0.1(x_1 + x_3 + x_4) + 0.01(x_1^2 + 2x_1x_3 + 2x_1x_4) \sin x_2 + 0.01x_4^2 \sin x_5, \\ u &= 0.1 + 0.02(x_1 \cos x_2 + x_4 \cos x_5), \\ v &= 0.1 + 0.02(x_1 \sin x_2 + x_4 \sin x_5). \end{aligned}$$

Considering the initial point  $x_0 = [5.67, 5.23, 11.96, 23.88, -2.61, 2.34, 3.45, 4.89]$ , for which  $f(x_0) = 35.53$ , Table 14.4 presents the results of the optimization process, as well as the solution.

**Table 14.4** Iterations generated by SPENBAR.  $\sigma_0 = 0.1$ ,  $\beta = 0.9$ ,  $\tau = 10^{-8}$ . Optimization of a heavy body

$k$	1	2	3	4	5	6	7
$\sigma_k$	0.1	0.01	0.001	0.0001	1.e-5	1.e-6	1.e-7
#itin	44	117	190	290	5	4	1
#nf	264	665	1018	1412	20	17	6
$x_1$	0.203143	0.007214	0.305438	0.39424	0.39417	0.39417	0.39417
$x_2$	7.237269	5.48815	7.430341	7.42649	7.4266	7.4266	7.4266
$x_3$	7.695956	2.232681	13.48405	13.70247	13.703	13.703	13.703
$x_4$	0.081001	9.94434	23.58007	24.03176	24.0325	24.0325	24.0325
$x_5$	-0.90282	-1.23978	-1.91034	-1.91831	-1.9183	-1.9183	1.9183
$x_6$	0.305193	-0.18e-8	0.543286	0.619871	0.61981	0.61981	0.61981
$x_7$	-2.75607	1.49421	3.67206	3.701685	3.70175	3.70175	3.70175
$x_8$	-0.32e-9	-3.15316	-4.85593	-4.90222	-4.9023	-4.9023	-4.9023
$f(x)$	4.13212	11.06789	30.62753	31.27724	31.2782	31.2782	31.2782

**Table 14.5** Performances of SPENBAR.  $\sigma_0 = 0.1$ ,  $\beta = 0.9$ ,  $\tau = 10^{-8}$ . Application DES. Thomson problem

$n$	$me$	$mc$	#itM	#itm	#nf	#qe	cpu	vfo
150	50	0	6	402	4095	0	2.23	1055.182314
225	75	0	6	638	6600	0	7.47	2454.369689
300	100	0	6	838	9663	0	18.78	4448.350634
450	150	0	6	1138	1,4508	0	61.30	10236.26400
525	175	0	7	1356	18,206	0	103.73	14034.95857
600	200	0	7	1680	25,854	0	190.41	18438.975151
900	300	0	7	2679	37,903	0	618.82	42131.673280
1200	400	0	7	3876	52,568	0	1006.89	75583.037847

In this table, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #itM = the number of the major iterations; #itm = the number of the minor iterations in the truncated Newton method for solving (14.65), #nf = the total number of evaluations of functions in the truncated Newton method for solving (14.65), #qe = the total number of quadratic extrapolations, cpu = the cpu time (seconds) to get a local solution to the problem, and vfo = the optimal value of the objective function.

**Table 14.6** Performances of SPENBAR.  $\sigma_0 = 0.1$ ,  $\beta = 0.9$ ,  $\tau = 10^{-8}$ . Application HANG. Dog curve

$n$	$me$	$mc$	#itM	#itm	#nf	#qe	cpu	vfo
50	1	0	3	1338	11,341	0	0.50	5.068577
100	1	0	3	3191	29,213	0	2.42	5.068505
200	1	0	3	3539	32,715	0	12.37	5.068486
400	1	0	4	4873	43,951	0	13.06	5.068481
800	1	0	3	15,896	147,351	0	86.43	5.068480
1000	1	0	3	21,465	196,275	0	144.29	5.068480
2000	1	0	5	102,285	780,770	0	1084.85	5.088175

For solving this problem, SPENBAR needs 7 major iterations and a total of 651 minor iterations used by the truncated Newton method. The total number of function evaluations is 3402. The final value of the objective at the local optimum is  $f(x^*) = 31.27823435$ .  $\diamond$

**Application DES** (*Distribution of Electrons on a Sphere*). (Dolan, Moré, & Munson, 2004; Andrei, 2015a). This is application L10 from the LACOP collection, known as the Thomson problem, as described in Appendix C. In Table 14.5, we present the performances of SPENBAR for solving this application for different numbers of electrons on the sphere.

**Application HANG** (*Hanging Chain*). This is application L11 from the LACOP collection presented in Appendix C. Table 14.6 contains the performances of SPENBAR for solving this application for different numbers of discretization points, with  $a = 1$ ,  $b = 3$ , and  $L = 4$ .

### Numerical Study—SPENBAR: Solving Applications from the LACOP Collection

In Appendix C, we have presented the LACOP collection, which contains a number of 18 real nonlinear optimization applications. In Table 14.7 of this numerical study, we can see the numerical performances of SPENBAR for solving nine applications.

**Table 14.7** Performances of SPENBAR for solving nine applications from the LACOP collection.  $\sigma_0 = 0.1$ ,  $\beta = 0.9$ ,  $\tau = 10^{-8}$

	<i>n</i>	<i>me</i>	<i>mc</i>	#it <i>M</i>	#itm	#nf	#qe	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	6	215	1109	0	0.04	-47.761090
ALKI	10	3	8	8	419	2064	1290	0.06	-1768.8070
PREC	8	0	6	5	87	403	39	0.01	3.9511635
PPSE	9	6	0	9	137	643	0	0.03	5055.0118
MSP3	13	0	15	9	1434	9687	8128	0.33	97.587531
MSP5	16	0	21	9	1256	9189	8413	0.5	174.787136
POOL	34	20	0	8	1282	11,620	0	1.0	2785.8000
TRAFO	6	0	2	6	171	744	173	0.02	135.075955
LATHE	10	1	14	10	781	5079	2296	0.19	-4430.5793

## 14.6 The Linearly Constrained Augmented Lagrangian (MINOS)

This section presents one of the most respectable algorithms and software for solving general nonlinear optimization problems, elaborated by Murtagh and Saunders (1978, 1980, 1982, 1995). The main idea behind this method is to generate a step by minimizing the Lagrangian or the augmented Lagrangian, subject to the linearizations of the constraints.

The origins of the linearly constrained augmented Lagrangian method can be found in the papers of Robinson (1972) and Rosen and Kreuser (1972). Let us consider the general nonlinear optimization problem with the equality constraints

$$\min \{f(x) : h(x) = 0, l \leq x \leq u\} \quad (14.96)$$

For a given point  $(x_k, \lambda_k)$ , Robinson's method defines the following:

- The linear approximation to  $h(x)$ :  $\bar{h}_k(x) = h(x_k) + \nabla h(x_k)(x - x_k)$ ,
- The departure from linearity:  $d_k(x) = h(x) - \bar{h}_k(x)$ ,
- The modified Lagrangian:  $M_k(x) = f(x) - \lambda_k^T d_k(x)$ .

Thus, the next iteration of Robinson's method is obtained as solution of the following subproblem:

$$\min_{x \in \mathbb{R}^n} \{M_k(x) : \bar{h}_k(x) = 0, l \leq x \leq u\}. \quad (14.97)$$

Under suitable conditions, Robinson (1972) proves that the sequence of the subproblem solutions  $\{(x_k, \lambda_k)\}$  converges *quadratically* to a solution of (14.96).

On the other hand, in MINOS, as described in Murtagh and Saunders (1982), the *penalty* term of the *augmented* Lagrangian is included in the subproblem objective in an attempt to improve the convergence from arbitrary starting points. Therefore, a *modified augmented* Lagrangian subproblem is obtained:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} \left\{ f(x) - \lambda_k^T d_k(x) + \frac{1}{2} \sigma_k \|d_k(x)^2\| \right\} \\ & \text{subject to} \\ & \bar{h}_k(x) = 0, \quad l \leq x \leq u. \end{aligned} \quad (14.98)$$

Observe that this is equivalent to the normal augmented Lagrangian because  $d_k(x)$  and  $h(x)$  are the same when the linearized constraints are satisfied ( $\bar{h}_k(x) = 0$ ). An important benefit is the following: if  $h(x)$  involves only some of the variables nonlinearly, i.e., it is not hard nonlinear, then  $M_k(x)$  has the same property, whereas  $\|h(x)\|^2$  appears to be more nonlinear.

MINOS uses the simplex or the reduced-gradient iterations to satisfy the linearized constraints for each subproblem. It limits the number of “minor iterations” performed on (14.98) in a heuristic way in order to avoid excessive optimization within a wrong subspace. MINOS also monitors  $\|x_{k+1} - x_k\|$  and  $\|\lambda_{k+1} - \lambda_k\|$ , and if they seem to be large, then the step towards  $(x_{k+1}, \lambda_{k+1})$  is heuristically shortened. Only if  $\|h(x)\|$  has increased substantially should  $\sigma_k$  be increased.

## MINOS for Linear Constraints

As described in Murtagh and Saunders (1982), MINOS is a particular implementation of Wolfe’s reduced gradient algorithm (1967) (see Chap. 16). It is designed to solve large-scale problems with nonlinear objective functions, expressed in the following standard format:

$$\min f(x_N) + c^T x, \quad (14.99a)$$

subject to

$$Ax = b, \quad (14.99b)$$

$$l \leq x \leq u, \quad (14.99c)$$

where  $A$  is an  $m \times n$  matrix with  $m \leq n$ . The vector  $x$  is partitioned into the linear portion  $x_L$  and the nonlinear portion  $x_N$ :

$$x = \begin{bmatrix} x_N \\ x_L \end{bmatrix}.$$

The components of  $x_N$  are called the *nonlinear variables*. On the other hand, the components of  $x_L$  are the *linear variables*. Observe that in the objective function, the nonlinear variables are incorporated in a nonlinear part of it. Possibly, the part of  $c^T x$  involving  $x_N$  may be included into  $f(x_N)$ . It is quite clear that if  $f(x_N) = 0$ , then (14.99) is a linear programming problem. MINOS is an extension of the revised simplex method (Dantzig, 1963; Vanderbei, 2001) and is able to deal with nonlinear terms by using quasi-Newton procedures.

As known, in the simplex method, a basic solution is characterized by having at most  $m$  *basic variables* lying between their bounds, while the rest of  $n - m$  *nonbasic variables* are equal to one bound or to another. An associated nonsingular square basis matrix  $B$  is extracted from the columns of the constraint matrix  $A$ , and along the iterations of the simplex algorithm, the columns of  $B$  are replaced one at a time.

In the case of nonlinear optimization problems, we cannot expect an optimal point to be a basic solution. However, if the number of nonlinear variables is small, then it seems reasonable to suppose that an optimal solution will be *nearly* basic. Thus, as described in Murtagh and Saunders (1978), as a generalization of the simplex algorithm, the concept of *superbasic variables* is introduced. Therefore, partitioning the vector  $x$  as  $x = [x^B \ x^S \ x^N]^T$ , the linear constraints (14.99b) may be written as

$$Ax = [B \quad S \quad N] \begin{bmatrix} x^B \\ x^S \\ x^N \end{bmatrix} = b. \quad (14.100)$$

The matrix  $B$  is square and nonsingular, as in the simplex algorithm. Therefore, it is invertible. The matrix  $S$  is  $m \times s$  with  $0 \leq s \leq n - m$  and  $N$  is given by the remaining columns of  $A$ . Clearly, the associated variables  $x^B$ ,  $x^S$ , and  $x^N$  are called the *basics*, *superbasics*, and *nonbasics*, respectively. As in the simplex algorithm, both the basics and superbasics are free to vary between their bounds. The superbasic variables may be moved in any direction in order to improve the objective value. On the other hand, the basic variables are obliged to change in a definite way to maintain the feasibility of the constraints  $Ax = b$ .

The following theorem shows that the optimal solutions of a nonlinear optimization problem are often near basic (Murtagh & Saunders, 1978).

**Theorem 14.9** *Let us suppose that a nonlinear optimization problem has  $t$  variables occurring nonlinearly in either the objective function or in the constraints. Then, there is an optimal solution in which the number of superbasic variables  $s$  satisfies  $s \leq t$ .*

**Proof** Let us assume that the nonlinear variables are fixed at their optimal values. The remaining problem is a linear programming one for which a basic solution exists ( $s = 0$ ). The conclusion of the theorem follows immediately if the nonlinear variables are now regarded as superbasics in the original problem. Observe that at the very beginning  $s = t$ , but if any of the nonlinear variables are actually on a bound, then we can label them as nonbasic, i.e.,  $s < t$ .  $\blacklozenge$

Let us now detail the method of MINOS for linear constraints. Suppose that  $f$  is twice continuously differentiable. Therefore, it can be expanded in Taylor's series with remainder of second-order

$$f(x + \Delta x) = f(x) + g(x)^T \Delta x + \frac{1}{2} \Delta x^T H(x + \gamma \Delta x) \Delta x, \quad (14.101)$$

where  $0 \leq \gamma \leq 1$ ,  $g(x) = \nabla f(x)$ , and  $H(x + \gamma \Delta x)$  is the Hessian of  $f$  evaluated at some point between  $x$  and  $x + \Delta x$ . Observe that if  $f(x)$  is a quadratic function, then  $H$  is a constant matrix. Now, let us partition  $\Delta x$  and  $g(x)$  corresponding to the partition of  $A$ :  $\Delta x = [\Delta x^B \quad \Delta x^S \quad \Delta x^N]^T$  and  $g = [g_B \quad g_S \quad g_N]^T$ .

If  $f(x)$  is quadratic, then a constrained stationary point at  $x + \Delta x$  is obtained by requiring that the following two properties of the step  $\Delta x$  should hold:

### Property 1 (Feasibility)

$$\begin{bmatrix} B & S & N \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \Delta x^B \\ \Delta x^S \\ \Delta x^N \end{bmatrix} = 0. \quad (14.102)$$

This property says that the step remains on the surface given by the intersection of the active constraints, i.e., the new point  $x + \Delta x$  is feasible. From (14.102), observe that  $\Delta x^N = 0$ .

**Property 2 (Optimality)**

$$\begin{bmatrix} g_B \\ g_S \\ g_N \end{bmatrix} + H \begin{bmatrix} \Delta x^B \\ \Delta x^S \\ \Delta x^N \end{bmatrix} = \begin{bmatrix} B^T & 0 \\ S^T & 0 \\ N^T & I \end{bmatrix} \begin{bmatrix} \mu \\ \lambda \end{bmatrix}, \quad (14.103)$$

i.e., the gradient at  $x + \Delta x$ , given by the left-hand side of (14.103), is orthogonal to the surface of the active constraints and thus is expressed as a linear combination of the active constraint normals. Moreover, for the optimality of  $x + \Delta x$ , the negative gradient must be orthogonal outside of the feasibility domain. For (14.99), we impose that  $\lambda_j \leq 0$  if  $x_j^N = u_j$ , or  $\lambda_j \geq 0$  if  $x_j^N = l_j$ ,  $j = 1, \dots, n - m - s$ .  $\mu$  and  $\lambda$  are the Lagrange multipliers.

For a general function  $f(x)$ , the step  $\Delta x$  may not lead directly to a stationary point, but the Properties 1 and 2 may be used to get a feasible descent direction. Now, from (14.102), we get

$$\begin{aligned} B\Delta x^B + S\Delta x^S &= 0, \\ \Delta x^N &= 0. \end{aligned} \quad (14.104)$$

Therefore,

$$\Delta x^B = -W\Delta x^S, \quad (14.105)$$

where

$$W = B^{-1}S. \quad (14.106)$$

Thus,

$$\Delta x = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x^S.$$

So, we can work only with  $\Delta x^S$ . The matrix  $W$  is not explicitly computed because  $B^{-1}$  is represented as the product form of the inverse or as the elimination form of the inverse (Andrei, 2011d).

Observe that (14.103) may be simplified when it is multiplied to the left by the matrix

$$\begin{bmatrix} I & 0 & 0 \\ -W^T & I & 0 \\ 0 & 0 & I \end{bmatrix}. \quad (14.107)$$

With this, from the first row of (14.103) multiplied by (14.107), we get an expression for the estimates of the Lagrange multipliers for the general constraints

$$g_B + [I \ 0 \ 0]H \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x^S = B^T \mu. \quad (14.108)$$

Observe that if  $\Delta x^S = 0$ , which means that  $x$  is stationary, we have

$$B^T \mu = g_B. \quad (14.109)$$

Let  $u$  be the solution of (14.109), which, as we can see, is analogous to the *pricing vector* in the revised simplex algorithm.

Considering now the third row of (14.103) multiplied by (14.107), we get

$$g_N + [0 \ 0 \ I]H \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x^S = N^T \mu + \lambda. \quad (14.110)$$

Again, if  $\Delta x^S = 0$ , then this equation reduces to

$$\lambda = g_N - N^T \mu, \quad (14.111)$$

which is analogous to the vector of the *reduced costs* in linear programming.

Finally, considering the second row of (14.103) multiplied by (14.107), we obtain

$$[-W^T \ I \ 0]H \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x^S = -h, \quad (14.112)$$

where

$$h = [-W^T \ I \ 0]g = g_S - S^T u. \quad (14.113)$$

The form of the equation (14.112) suggests that

$$[-W^T \ I \ 0]H \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \quad (14.114)$$

can be regarded as the *reduced Hessian* and  $h = [-W^T \ I \ 0]g$  as the *reduced gradient*. Observe that (14.112) gives a Newton step in the independent variables  $\Delta x^S$ . Therefore,  $\|h\| = 0$  becomes a necessary condition for a stationary point on the current set of the active constraints, which, if the reduced Hessian is nonsingular, implies that  $\|\Delta x^S\| = 0$ .

Practically, an  $R^T R$  factorization of the reduced Hessian (14.114) is computed, where  $R$  is an upper triangular matrix, such that the step  $\Delta x^S$  is obtained from the system

$$(R^T R) \Delta x^S = -h \quad (14.115)$$

by forward and backward substitution, like in the simplex algorithm with the LU factorization of the basis.

Gill and Murray (1974b) considered a class of algorithms in which the search direction along the surface of the active constraints is characterized as being in the range of a matrix  $Z$  which is orthogonal to the matrix of the constraint normals. Thus, if  $\widehat{A}x = \widehat{b}$  is the current set of  $n - s$  active constraints, then  $Z$  is an  $n \times s$  matrix, such that

$$\widehat{A}Z = 0. \quad (14.116)$$

Using this idea, the main steps to be executed at each iteration to get a feasible direction  $p$  are as follows:

- (i) Compute the reduced gradient  $\bar{g} = Z^T g$ .
- (ii) Compute an approximation to the reduced Hessian  $\bar{H} = Z^T H Z$ .
- (iii) Compute an approximate solution to the system  $\bar{H}\bar{p} = -\bar{g}$ .
- (iv) Compute the search direction  $p = Z\bar{p}$ .
- (v) Perform a line-search to find an approximation to the stepsize  $\alpha^*$

$$f(x + \alpha^* p) = \min_{\alpha} \{f(x + \alpha p) : x + \alpha p \text{ feasible}\}.$$

This algorithm is very general. Two conditions are imposed on the matrix  $Z$ : to be of full rank on columns and to verify (14.116). Therefore,  $Z$  may have any form which verifies the above conditions. Particularly, in MINOS,  $Z$  has the following form:

$$Z = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}. \quad (14.117)$$

This is a very convenient representation which will be used both in MINOS with linear constraints and in MINOS with nonlinear constraints. Note that this matrix is not computed. The algorithm works only with  $S$  and the triangular (LU) factorization of  $B$ .

Now, let us present the MINOS algorithm with linear constraints. For this, suppose that we have the following elements:

1. A feasible vector  $x$  satisfying  $[B \quad S \quad N]x = b$  and  $l \leq x \leq u$
2. The function value  $f(x)$  and the gradient  $g(x) = [g_B \quad g_S \quad g_N]^T$
3. The number of superbasic variables  $s$ , where  $0 \leq s \leq n - m$
4. An LU factorization of the  $m \times m$ -basis matrix  $B$
5. An  $R^T R$  factorization of the quasi-Newton approximation to the matrix  $Z^T H Z$
6. A vector  $u$ , solution of the system  $B^T u = g_B$
7. The reduced gradient  $h = g_S - S^T u$
8. Small positive convergence tolerances  $\epsilon_h$  and  $\epsilon_\lambda$

With these developments, the MINOS algorithm for solving nonlinear optimization problems with linear constraints is as follows.

#### Algorithm 14.6 MINOS–linear constraints–Murtagh and Saunders

1.	Initialization of the algorithm with the above described elements. Set $k = 1$ , $x_k = x$ , $f(x_k) = f_k$ , and $g(x_k) = g_k$
2.	Test for convergence. If $\ h\  > \epsilon_h$ , then go to step 4; otherwise, continue with step 3
3.	Compute an estimation of the Lagrange multipliers. Modify the number of superbasics: (a) Compute $\lambda = g_N - N^T u$ (b) Since for the verification of the KKT conditions all the variables fixed at the upper (lower) bound must have the Lagrange multipliers negative (positive), determine: $\lambda_1 = \max \{\lambda_j : \lambda_j > 0\} \text{ for } x_j^N \text{ fixed at its upper bound } u_j,$

	$\lambda_2 = \min \{\lambda_j : \lambda_j < 0\}$ for $x_i^N$ fixed at its lower bound $l_j$ . If $\lambda_1 \leq \varepsilon_\lambda$ and $\lambda_2 \leq \varepsilon_\lambda$ , stop, the current solution is optimal (c) Otherwise, choose $x_i^N$ corresponding to $ \lambda_i  = \max \{ \lambda_1 ,  \lambda_2 \}$ as a new superbasic variable (1) Augment the matrix $S$ with the column corresponding to the variable $x_i^N$ (2) Augment the reduced gradient $h$ with $\lambda_i$ (3) Add a suitable new column to $R$ (d) Update $s = s + 1$
4.	Compute the search direction $p$ : (a) Solve the system $(R^T R)p_S = -h$ (b) Solve the system $(LU)p_B = -Sp_S$ (c) Set $p = [p_B \ p_S \ 0]^T$
5.	Determine the maximum value of the stepsize: (a) Find $\alpha_{\max} \geq 0$ as the greatest positive value of $\alpha$ for which one component of the vector $[x^B + \alpha p_B \ x^S + \alpha p_S]^T$ has reached one of its bounds (b) If $\alpha_{\max} = 0$ , then go to step 10; otherwise, continue with step 6
6.	Line-search. Find $\alpha^*$ such that $f(x_k + \alpha^* p) = \min \{f(x_k + \alpha p) : 0 < \alpha \leq \alpha_{\max}\}$
7.	Update the solution as $x_{k+1} = x_k + \alpha^* p$ and set $k = k + 1$
8.	Compute $f(x_k) = f_k$ and $g(x_k) = g_k$
9.	Compute the reduced gradient: (a) Solve the system $(U^T L^T)u = g_B$ (b) Compute the new reduced gradient $\bar{h} = g_S - S^T u$ (c) Using $\alpha, p$ and the change in the reduced gradient $\bar{h} - h$ , modify $R$ corresponding to a quasi-Newton update of $R^T R$ (d) Set $h = \bar{h}$ (e) If $\alpha = \alpha_{\max}$ , that is one component of $x^B$ or $x^S$ has reached one of their bounds, then continue with step 10; otherwise ( $\alpha < \alpha_{\max}$ ) go to step 2
10.	Here $\alpha_{\max} = 0$ . At this step of the algorithm, a basic variable or a superbasic variable, let us say with index $i$ , has reached one of its bounds (a) If the variable $i$ is basic, then: (1) Interchange the $i$ -th basic variable $x_i^B$ (i.e. the column corresponding to $x_i^B$ from $B$ ) with the $j$ -th superbasic variable $x_j^S$ (i.e. the column corresponding to $x_j^S$ from $S$ ). The index $j$ is chosen to keep $B$ nonsingular (2) Update the $L$ and $U$ factors, the matrix $R$ and the vector $u$ (3) Compute the new reduced gradient $h = g_S - S^T u$ and go to step 10.c) (b) If the $i$ variable is superbasic, then define $j = i - m$ (c) Make the $j$ -th variable in $S$ nonbasic at the appropriate bound: (1) Delete column $j$ from $S$ and place it in $N$ (2) Delete column $j$ from $R$ (3) Delete the $j$ -th component from the vectors $x^S$ and $h$ (4) Restore $R$ to the triangular form (d) Set $s = s - 1$ and go to step 2 <div style="text-align: right;">◆</div>

An iteration of the MINOS algorithm for linear constraints is roughly equivalent to an iteration of the revised simplex algorithm on an  $m \times n$  linear programming problem, plus an iteration of a quasi-Newton algorithm on an unconstrained optimization problem with  $s$  variables. MINOS with linear constraints is an advanced software, written in Fortran, which takes the sparsity of  $A$  and uses an efficient LU factorization of the basis matrix  $B$ , based on the “*bump and spike*” algorithm (Hellerman & Rarick, 1971, 1972). Murtagh and Saunders (1978) present plenty of details on the following: update of the sparse matrix factorizations, quasi-Newton updates, basis change, removal of one superbasic variable, addition of one superbasic variable, convergence tests, use of the first and of the second derivatives, forward and backward transformations of vectors, quadratic programs, etc.

**Table 14.8** MINOS for solving some large-scale linear programs

	Characteristics			Optimization		
	<i>m</i>	<i>n</i>	<i>nz</i>	#iter	<i>z</i>	<i>cpu</i>
cq9	9451	13,778	157,598	104,150	0.5055445e6	449.47
cre-b	9649	72,447	328,542	183,473	0.2312964e8	678.23
cre-d	8927	69,980	312,626	240,460	0.2445497e8	829.64
ge	10,339	11,098	53,763	15,524	0.5581281e7	51.39
ken-11	14,694	21,349	70,354	15,307	-0.69723e10	71.21
ken-13	28,633	42,659	139,834	45,461	-0.10257e10	440.56
pds-10	16,559	48,763	140,063	74,814	0.267270e11	404.45
stocfor3	16,675	15,695	74,004	14,082	-0.399767e5	72.55
osa-07	1118	23,949	167,643	1915	0.5357225e6	2.02
osa-14	2338	52,460	367,220	4116	0.1106462e7	7.91
co5	5878	7993	92,788	30,938	0.7144723e6	81.28
cq5	5149	7530	83,564	37,280	0.4001338e6	85.06
p05	5090	9500	68,455	1675	0.5560002e6	3.82
r05	5190	9500	113,455	1712	0.5578318e6	5.37

Table extracted from Andrei (2011d, p. 325))

In this table, we have:  $m$  = the number of linear constraints,  $n$  = the number of variables,  $nz$  = the number of nonzeros in the matrix of linear constraints, #iter = the number of iterations necessary to get the optimal solution,  $z$  = the value of the objective, and  $cpu$  = the CPU computing time (seconds) for solving the problem. The numerical experiments were run on a Workstation Intel Pentium 4 with 1.8 GHz.

## Numerical Study: MINOS for Linear Programming

To see the performance of MINOS for solving linear programming problems, in Table 14.8, we present the results of solving some large-scale linear programming problems (the terms  $f(x_N)$  in (14.99a) is zero). These are taken from the Netlib collection (Gay, 1985), or the collection of linear programs described in Carolan, Hill, Kennington, Niemi, and Wochmann (1990) (see Andrei, 2011d, p. 325).

## MINOS for Nonlinear Constraints

In the following, we shall extend MINOS for solving general nonlinear optimization problems of the form

$$\min f^0(x) + c^T x + d^T y \quad (14.118a)$$

$$\text{subject to} \\ f(x) + A_1 y = b_1, \quad (14.118b)$$

$$A_2 x + A_3 y = b_2, \quad (14.118c)$$

$$l \leq \begin{bmatrix} x \\ y \end{bmatrix} \leq u, \quad (14.118d)$$

where  $f(x) = [f^1(x) \ \cdots \ f^{m_1}(x)]$ ,  $b_1 \in \mathbb{R}^{m_1}$ ,  $b_2 \in \mathbb{R}^{m_2}$ , which emphasizes the *linear part* as well as the *nonlinear part* of the problem. Define  $m = m_1 + m_2$ . The first  $n_1$  variables  $x$  are called the

*nonlinear variables.* They occur nonlinearly in either the objective function or in the first  $m_1$  constraints. The variables  $y$  represent the *linear variables*. They include the slack variables. The problem may have purely linear constraints (14.118c). Suppose that the functions  $f^i(x)$ ,  $i = 0, \dots, m_1$ , are twice continuously differentiable with the gradients  $g^i(x)$ ,  $i = 0, \dots, m_1$ , and there exists a local minimum  $x^*$  with the corresponding Lagrange multipliers  $\lambda^*$ , such that the first- and second-order KKT conditions hold.

As described by Murtagh and Saunders (1982), the solution process of this problem consists of a sequence of *major iterations*, each one involving a linearization of the nonlinear constraints at some current point  $x_k$  corresponding to a first-order Taylor's series approximation

$$f^i(x) = f^i(x_k) + g^i(x_k)^T(x - x_k) + O(\|x - x_k\|^2).$$

Define

$$\tilde{f}(x, x_k) \triangleq f(x_k) + J(x_k)(x - x_k), \quad (14.119)$$

or  $\tilde{f} = f_k + J_k(x - x_k)$ , where  $J(x)$  is the  $m_1 \times n_1$ -Jacobian matrix of function  $f(x)$ . Observe that

$$f - \tilde{f} = (f - f_k) - J_k(x - x_k) \quad (14.120)$$

contains the high-order nonlinear terms in the Taylor's expansion of  $f(x)$  about the current point  $x_k$ .

At the  $k$ -th major iteration, the following *linearly constrained subproblem* is formed:

$$\begin{aligned} \min_{x, y} L(x, y, x_k, \lambda_k, \rho) = \\ f^0(x) + c^T x + d^T y - \lambda_k^T (f - \tilde{f}) + \frac{1}{2} \rho (f - \tilde{f})^T (f - \tilde{f}) \end{aligned} \quad (14.121a)$$

subject to

$$\tilde{f} + A_1 y = b_1, \quad (14.121b)$$

$$A_2 x + A_3 y = b_2, \quad (14.121c)$$

$$l \leq \begin{bmatrix} x \\ y \end{bmatrix} \leq u. \quad (14.121d)$$

The objective function  $L(\cdot)$  given by (14.121a) is a *modified augmented Lagrangian* in which  $f - \tilde{f}$  is used instead of the conventional constraint violation  $f + A_1 y - b_1$ . The partial derivatives of  $L(\cdot)$  are

$$\begin{aligned} \frac{\partial L(x, y)}{\partial x} &= g^0(x) + c - (J - J_k)^T [\lambda_k - \rho(f - \tilde{f})], \\ \frac{\partial L(x, y)}{\partial y} &= d. \end{aligned} \quad (14.122)$$

Observe that the nonlinearities in  $L(\cdot)$  involve  $x$  but not  $y$ , which means that the subproblem has the same nonlinear variables as the original one. The modified Lagrangian was used by Robinson (1972) with  $\rho = 0$ . The use of a penalty term ensures that the augmented Lagrangian maintains a positive definite Hessian in the appropriate subspace. It was suggested by Arrow and Sollow (1958) and later considered by Hestenes (1969), Powell (1969), and Sargent and Murtagh (1973).

In the following, in order to specify the MINOS algorithm for nonlinear constraints, it is necessary to define the procedures for the choice of the Lagrange multipliers  $\lambda_k$  and of the penalty parameter  $\rho$ .

**Choice of  $\lambda_k$**  The best choice is  $\lambda_k = \lambda^*$ , but the optimal value of the multipliers is not known. Therefore, the simplest choice is  $\lambda_k = \widehat{\lambda}$ , where  $\widehat{\lambda}$  is the vector of the multipliers corresponding to the linearized constraints at the solution of the previous subproblem. Let us assume that the problem does not have linear constraints. Then,  $\widehat{\lambda}$  is the solution of the system  $B^T \widehat{\lambda} = g_B$  from the end of the previous iteration. Moreover,  $\widehat{\lambda}$  also verifies the system  $S^T \widehat{\lambda} = g_S$ . As we know,  $g$  is zero for all the slack variables, and it follows that  $\widehat{\lambda}_i = 0$  if the  $i$ -th linearized constraint is inactive. Therefore, the choice  $\lambda_k = \widehat{\lambda}$  ensures that an inactive nonlinear constraint will be excluded from the Lagrangian term  $\lambda_k^T (f - \widetilde{f})$  in the next subproblem.

**Choice of  $\rho$**  As known,  $x^*$  need not be a local minimum of the Lagrangian function. If we assume that  $J(x^*)$  is of full rank, then  $\lambda^*$  exists, so that

$$L(x, \lambda) = f^0(x) + c^T x + d^T y - \lambda^T (f + A_1 y - b_1),$$

is stationary at  $(x^*, \lambda^*)$ , but  $L(x^*, \lambda^*)$  may have a negative curvature in  $x$  at  $x^*$ . If we consider that the constraints are satisfied at  $x^*$  as equalities and ignore the inactive constraints, then the necessary (sufficient) conditions for  $x^*$  to be a local minimum are

$$Z(x^*)^T \frac{\partial L(x^*, \lambda^*)}{\partial x} = 0$$

and

$$Z(x^*)^T \frac{\partial^2 L(x^*, \lambda^*)}{\partial x^2} Z(x^*)$$

to be positive semidefinite (positive definite), where  $Z(x^*)$  is defined in (14.117). Therefore, if the search is restricted to the linearly constrained subspace defined by  $Z(x^*)$ , then we do seek a minimum of the Lagrangian. We may expect that when  $x_k$  is sufficiently close to  $x^*$  for  $J(x_k)$  to be close to  $J(x^*)$ , we may minimize (14.121a) with  $\rho = 0$ . As it is discussed in Murtagh and Saunders (1982), the difficulty arises when  $x_k$  is far away from  $x^*$ , since, in this case, the linearized constraints may define a subspace where perhaps a saddle point would be closer to  $x^*$  than a minimum would. Successive minima of (14.121) with  $\rho = 0$  may therefore fail to converge to  $x^*$ . The addition of a penalty term  $\rho(f - \widetilde{f})^T (f - \widetilde{f})$  imposes the correct curvature properties on (14.121a) for a sufficiently large  $\rho > 0$ .

To illustrate the importance of the penalty term in (14.121), let us consider the following problem with equality constraints:

$$\begin{aligned} & \min f^0(x) \\ & \text{subject to} \\ & f(x) = 0, \end{aligned} \tag{14.123}$$

where the functions are twice continuously differentiable with bounded Hessians. Assume that at some point  $x^*$  the Jacobian  $J(x^*)$  is of full rank, that there is a  $\lambda^*$  such that  $\partial f^0 / \partial x = J(x^*)^T \lambda^*$ , and that the reduced Hessian

$$Z(x^*)^T \left[ \frac{\partial^2 L(x^*, \lambda^*)}{\partial x^2} \right] Z(x^*)$$

is positive definite, i.e., the sufficient conditions of optimality are satisfied for  $x^*$  to be a local minimum. As in Murtagh and Saunders (1982), the following theorems can be presented.

**Theorem 14.10** Let  $(x_k, \lambda_k)$  be an approximate solution to (14.123) and let  $(\hat{x}, \hat{\lambda})$  be a solution to the linearized subproblem

$$\begin{aligned} & \min f^0(x) - \lambda_k^T (f - \tilde{f}) + \frac{1}{2} \rho (f - \tilde{f})^T (f - \tilde{f}) \\ & \text{subject to} \\ & \quad f(x, x_k) = 0, \end{aligned} \tag{14.124}$$

where  $\tilde{f}(x, x_k)$  is given by (14.119). If  $\hat{\lambda} - \lambda_k = \varepsilon_1$  and  $f(\hat{x}) = \varepsilon_2$ , then  $(\hat{x}, \hat{\lambda})$  is also a solution to the perturbed problem

$$\begin{aligned} & \min f^0(x) + (\varepsilon_1 + \rho \varepsilon_2)^T (f - \tilde{f}) \\ & \text{subject to} \\ & \quad f(x) = \varepsilon_2, \end{aligned} \tag{14.125}$$

for  $\varepsilon_1$  and  $\varepsilon_2$  sufficiently small.

**Proof** If  $(\hat{x}, \hat{\lambda})$  is a solution of (14.124), then we must have  $\tilde{f} = 0$  and

$$g^0(\hat{x}) - (\hat{J} - J_k)^T \lambda_k + \rho (\hat{J} - J_k)^T (f - \tilde{f}) = J_k^T \hat{\lambda},$$

where  $J_k$  is the Jacobian at  $x_k$ , but  $\hat{J}$ ,  $f$ , and  $\tilde{f}$  are evaluated at  $\hat{x}$ . Now, adding  $(\hat{J} - J_k)^T \hat{\lambda}$  to both sides of the above equality and inserting the expression for  $\varepsilon_1$  and  $\varepsilon_2$ , we get

$$g^0(\hat{x}) - (\hat{J} - J_k)^T \varepsilon_1 + \rho (\hat{J} - J_k)^T \varepsilon_2 = \hat{J}^T \hat{\lambda},$$

which shows that  $(\hat{x}, \hat{\lambda})$  also satisfies the conditions for a stationary point of (14.125). Observe that the Hessians for the Lagrangian function of (14.124) and (14.125) differ only in the amount  $\rho (\hat{J} - J_k)^T (\hat{J} - J_k)$  at the solution of (14.125), which is of order  $\rho \|\Delta x_k\|$ , where  $\Delta x_k = \hat{x} - x_k$ . Hence, for sufficiently small  $\varepsilon_1$ ,  $\varepsilon_2$ , and  $\Delta x_k$ , if the reduced Hessian of (14.124) is positive definite at  $\hat{x}$ , then, by continuity, the reduced Hessian of (14.125) will also be positive definite, thus satisfying the sufficient conditions for a local minimum of (14.125) at  $\hat{x}$ .  $\blacklozenge$

**Theorem 14.11** Let  $(x_k, \lambda_k)$  be an approximate solution of (14.123) and let  $(\hat{x}, \hat{\lambda})$  be a solution of the linearized subproblem

$$\begin{aligned} & \min f^0(x) - \lambda_k^T (f - \tilde{f}) + \frac{1}{2} \rho f^T f \\ & \text{subject to} \\ & \quad f(x, x_k) = 0. \end{aligned} \tag{14.126}$$

If  $\hat{\lambda} - \lambda_k = \varepsilon_1$  and  $f(\hat{x}) = \varepsilon_2$ , then  $(\hat{x}, \hat{\lambda})$  is also a solution to the perturbed problem

$$\begin{aligned}
& \min f^0(x) + \varepsilon_1^T (\hat{f} - f) + \rho \varepsilon_2^T f \\
& \text{subject to} \\
& f(x) = \varepsilon_2.
\end{aligned} \tag{14.127}$$

**Proof** The proof is similar to the proof of Theorem 14.10.  $\blacklozenge$

If  $\varepsilon_1$  and  $\varepsilon_2$  are sufficiently small, then  $\rho$  can safely be reduced to zero (Murtagh & Saunders, 1982). Apparently, problem (14.125) is less sensitive to deviations from its optimum than (14.127). Thus, let  $\Delta x$  be an arbitrary small change to the solution  $\hat{x}$  of (14.125). Then, the objective function for (14.125) differs from the true objective  $f^0(x)$  by an amount  $\delta_1 = (\varepsilon_1 + \rho \varepsilon_2)^T (\hat{f} - f)$ . Clearly,  $|\delta_1| \leq (\|\varepsilon_1\| + \rho \|\varepsilon_2\|) O(\|\Delta x\|^2)$ . On the other hand, for (14.127), the analogous deviation is

$$\delta_2 = \varepsilon_1^T (\hat{f} - f) + \rho \varepsilon_2^T f = \varepsilon_1^T (\hat{f} - f) + \rho \varepsilon_2^T (\hat{f} + \hat{J} \Delta x + O(\|\Delta x\|^2)).$$

In this case,  $|\delta_2| \leq (\|\varepsilon_1\| + \rho \|\varepsilon_2\|) O(\|\Delta x\|^2) + \rho \|\varepsilon_2\|^2 + \rho \|\hat{J}^T \varepsilon_2\| \|\Delta x\|$ . Since  $\delta_1$  is of order  $\|\Delta x\|^2$  while  $\delta_2$  is of order  $\|\Delta x\|$ , it follows that the modified penalty term in (14.124) has a theoretical advantage over the conventional penalty term of (14.126).

Having in view the above theoretical developments, the MINOS algorithm for solving the general nonlinear optimization problems is as follows.

#### Algorithm 14.7 MINOS–nonlinear constraints–Murtagh and Saunders

1.	Consider $k = 0$ . Choose the initial estimates $x_k$ , $y_k$ , and $\lambda_k$ . Select a value for the penalty parameter $\rho > 0$ as well as a convergence tolerance $\varepsilon_c > 0$
2.	Solve the linear subproblem: <ol style="list-style-type: none"> <li>Given <math>x_k</math>, <math>y_k</math>, <math>\lambda_k</math>, and <math>\rho</math>, using the Algorithm 14.6 (MINOS for linear constraints), solve the linear constrained subproblem (14.121) obtaining <math>x_{k+1}</math>, <math>y_{k+1}</math> and <math>u</math>, where <math>u</math> is the vector of the Lagrange multipliers associated to the constraints from (14.121)</li> <li>Form the vector <math>\lambda_{k+1}</math> where its first <math>m_1</math> components are those of <math>u</math></li> </ol>
3.	Test for convergence. If $(x_k, y_k)$ satisfies the KKT optimality conditions, then stop
4.	If $\frac{\ f(x_{k+1}) + A_1 y_{k+1} - b_1\ }{1 + \ x_{k+1} - y_{k+1}\ } \leq \varepsilon_c, \text{ and } \frac{\ \lambda_{k+1} - \lambda_k\ }{1 + \ \lambda_{k+1}\ } \leq \varepsilon_c,$ then set $\rho = 0$ ; otherwise, set $\rho = 10\rho$
5.	Consider a re-linearization of the constraints in $x_{k+1}$ , set $k = k + 1$ , and go to step 2 $\blacklozenge$

In Murtagh and Saunders (1982), we find some details on the algorithm concerning sparse matrices techniques for solving linear algebraic systems, infeasible subproblems, user options, verification of the gradients option, Jacobian options (dense or sparse), evaluation options (the constraints and the gradients are evaluated only once per major iteration or they are evaluated as often as the objective), convergence conditions (major iterations, minor iterations, radius of convergence, row tolerance), etc. Let us present some numerical examples.

**Example 14.4** Consider the nonlinear optimization problem (Wright (No.4), 1976; Murtagh & Saunders, 1982)

$$\begin{aligned} & \min (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4 \\ & \text{subject to} \end{aligned}$$

$$\begin{aligned} x_1 + x_2^2 + x_3^3 &= 2 + 3\sqrt{2}, \\ x_2 - x_3^2 + x_4 &= -2 + 2\sqrt{2}, \\ x_1 x_5 &= 2. \end{aligned}$$

Considering the initial points

$$\begin{aligned} A &= [1 \ 1 \ 1 \ 1 \ 1]^T, \quad B = [2 \ 2 \ 2 \ 2 \ 2]^T, \quad C = [-1 \ 3 \ -0.5 \ -2 \ -3]^T, \\ D &= [-1 \ 2 \ 1 \ -2 \ -2]^T, \quad E = [-2 \ -2 \ -2 \ -2 \ -2]^T, \end{aligned}$$

Algorithm 14.7 (MINOS with nonlinear constraints) gives the following results:

$$\begin{aligned} A, B \rightarrow \quad x_1^* &= [1.11663 \quad 1.22043 \quad 1.53779 \quad 1.97277 \quad 1.79110]^T, \\ f(x_1^*) &= 0.0029307, \\ C \rightarrow \quad x_2^* &= [-0.703393 \quad 2.63570 \quad -0.0963618 \quad -1.79799 \quad -2.84336]^T, \\ f(x_2^*) &= 44.022089, \\ D \rightarrow \quad x_3^* &= [-1.27305 \quad 2.41035 \quad 1.19486 \quad -0.154239 \quad -1.57103]^T, \\ f(x_3^*) &= 27.8718714, \\ E \rightarrow \quad x_4^* &= [-2.79087 \quad -3.00414 \quad 0.20538 \quad 3.87474 \quad -0.71662]^T, \\ f(x_4^*) &= 607.03036. \end{aligned}$$

In Table 14.9, we can see the characteristics of the optimization process, where  $\#itM$  is the number of the major iterations,  $\#itm$  is the number of minor iterations necessary for solving the subproblems with linear constraints, and  $\#evf$  is the number of evaluations of the objective and of the constraints, including their gradients.

**Example 14.5** Consider the problem (Wright (No.9), 1976; Murtagh & Saunders, 1982)

$$\begin{aligned} & \min 10x_1 x_4 - 6x_2^2 x_3 + x_1^3 x_2 + 9 \sin(x_5 - x_3) + x_2^3 x_4^2 x_5^4 \\ & \text{subject to} \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 \leq 20, \\ & x_1 x_3 + x_4 x_5 \geq -2, \\ & x_2^2 x_4 + 10x_1 x_5 \geq 5. \end{aligned}$$

**Table 14.9** MINOS with different initializations. Example 14.4

	A	B	C	D	E
#itM	8	9	7	6	8
#itm	30	21	14	17	21
#evf	64	49	35	35	55

**Table 14.10** MINOS with different initializations. Example 14.5

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
#itM	10	8	20	9
#itm	51	25	105	46
#evf	136	64	195	97

For the initial points

$$A = [1 \ 1 \ 1 \ 1 \ 1]^T, \quad B = [1 \ -3 \ 1 \ -1 \ 2]^T, \quad C = [5 \ -5 \ 1 \ -3 \ 1]^T,$$

$$D = [10 \ -1 \ 1 \ 10 \ 2]^T,$$

Algorithm 14.7 gives the following results:

$$A \rightarrow x_1^* = [-0.08145 \ 3.69238 \ 2.48741 \ 0.37713 \ 0.17398]^T,$$

$$f(x_1^*) = -210.407817,$$

$$B, C \rightarrow x_2^* = [1.47963 \ -2.63661 \ 1.05467 \ -1.61151 \ 2.67388]^T,$$

$$f(x_2^*) = -2500.584488,$$

$$D \rightarrow x_3^* = [-0.0774 \ -2.58139 \ 0.01143 \ 2.10766 \ 2.98229]^T,$$

$$f(x_3^*) = -6043.539113.$$

Table 14.10 contains the characteristics of the optimization process initialized in different points.

### Numerical Study—MINOS: Solving Applications from the LACOP Collection

In Appendix C, we have presented the LACOP collection, which includes a number of 18 real nonlinear optimization applications. In Tables 14.11 and 14.12 of this numerical study, we can see the numerical performances of MINOS for solving 12 small-scale nonlinear optimization applications and 6 large-scale nonlinear optimization applications of different dimensions, respectively.

**Table 14.11** Performances of MINOS for solving 12 applications from the LACOP collection. Small-scale nonlinear application

	<i>n</i>	<i>me</i>	<i>mc</i>	#itM	#itm	#nf	#nr	#s	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	1	29	84	1	7	0.02	-47.761090
ALKI	10	3	8	12	29	74	74	1	0.02	-1768.8070
PREC	8	0	6	17	71	171	171	3	0.02	4.286758
PPSE	9	6	0	6	13	17	17	1	0.01	5055.0118
MSP3	13	0	15	7	60	95	95	1	0.01	97.587531
MSP5	16	0	21	10	89	139	139	0	0.01	174.787136
POOL	34	20	0	5	60	133	133	3	0.01	2569.800
TRAFO	6	0	2	8	41	83	83	4	0.00	135.075962
LATHE	10	1	14	23	150	327	327	0	0.02	-4430.08793
DES	150	50	0	66	2344	5838	5838	100	2.75	1055.182314
CSTC	303	200	0	6	160	320	320	100	0.19	-3.480074
DIFF	396	324	0	1	48	0	0	0	0.02	0.00

**Table 14.12** Performances of MINOS for solving six applications from the LACOP collection. Large-scale nonlinear applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#itM	#itm	#nf	#s	cpu	vfo
HANG	1002	501	0	94	3681	8642	496	4.46	5.06855
	2002	1001	0	209	8252	19712	987	27.49	5.06851
FLOW	1182	734	0	1	997	14	0	0.19	0.00
FLOWO	1556	1005	0	1	709	2	0	0.25	0.00
POL	4004	3000	0	28	1002	3173	0	5.29	14.14484
	6004	4500	0	40	1502	4821	0	12.87	14.14341
	8004	6000	0	53	2000	6524	0	22.45	14.12985
	10,004	7500	0	66	2499	8489	0	36.52	14.14179
CAT	3003	2000	0	11	272	1183	0	2.18	-0.04805
	6003	4000	0	18	536	2430	0	9.06	-0.04805
	9003	6000	0	23	796	3703	0	20.73	-0.04804
CONT	2505	2000	0	1	506	663	1	0.09	1.013239
	5005	4000	0	1	1005	1314	2	0.20	1.005922
	7500	6000	0	1	1509	1978	1	0.69	1.004561
	10,005	8000	0	1	2005	2627	0	1.33	1.004072

**Table 14.13** Performances of MINOS for solving the HANG application from the LACOP collection. Large-scale nonlinear applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#itM	#itm	#nf	#s	cpu	vfo
HANG	4002	2001	0	447	17,791	42,955	1950	205.64	5.06849
	8002	4001	0	704	28,153	68,784	3385	1024.63	5.06893

In Tables 14.11–14.13 we have:  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #itM = the number of major iterations, #itm = the number of minor iterations, #nf = the number of evaluations of the objective function, #nr = the number of evaluations of constraints, #s = the number of superbasic variables,  $cpu$  = the CPU computing time for solving the problem (seconds), and  $vfo$  = the value of the objective function.

Table 14.13 shows the performances of MINOS for solving the application HANG with a large number of variables.

## Notes and References

The penalty and the augmented Lagrangian are two very important concepts in nonlinear optimization. The content of this chapter is based on the books of Nocedal and Wright (2006) and Bertsekas (1982b). The criticism of penalty and of the augmented Lagrangian methods is taken from Saunders (2015a) and Andrei (2015a). The quadratic penalty function was first proposed by Courant (1943). The quadratic penalty method is often used in practice for its simplicity, although it has a major disadvantage, mainly because of the ill-conditioning of the Hessian of the quadratic penalty function. In the nonsmooth exact penalty methods, a single unconstrained optimization problem rather than a sequence replaces the original constrained problem. However, nonsmoothness may create complications. A well-known function of this type is the  $l_1$  penalty function. This computational scheme based on the exact penalty method sometimes works very well in practice, but it can be inefficient. If the initial penalty parameter  $\sigma_0$  in Algorithm 14.2 is too small, many cycles may be needed. Besides, the iterates may move away from the solution in the initial cycles. If  $\sigma_0$  is large, then the penalty function will be difficult to minimize by requiring a large number of iterations. A different

kind of exact penalty approach is the augmented Lagrangian method, which avoids the ill-conditioning by using the explicit Lagrange multiplier estimate.

The augmented Lagrangian was independently proposed by Hestenes (1969) and Powell (1969). The  $l_1$  penalty method was developed by Fletcher. All these concepts are implemented in different combinations in very efficient nonlinear optimization packages. For example, the  $l_1$  penalty method that uses linear programming subproblems has been implemented as part of the KNITRO/ACTIVE software package (see Chap. 15).

The augmented Lagrangian methods solve the nonlinear constrained optimization problems (14.3) by a sequence of subproblems that minimize the augmented Lagrangian, either subject to a linearization of the constraints or as a bound-constrained problem. The best known software implementing the penalty or the augmented Lagrangian are as follows:

ALGENCAN (Andreani, Birgin, Martínez, & Schuverdt, 2007, 2008; Birgin & Martínez, 2014) is a general algorithm of the augmented Lagrangian type, in which the subproblems are solved by using GENCAN. GENCAN (included in ALGENCAN) is a Fortran code for minimizing a smooth function with a potentially large number of variables and box constraints. The subproblem is solved by using a quasi-Newton method. The penalty on each constraint is increased if the previous iteration does not yield a better point.

GALAHAD (Gould, Orban, & Toint, 2004) contains a range of solvers for the large-scale nonlinear optimization. It also contains a presolve routine for quadratic programming (Gould & Toint, 2004), as well as some other support routines. It includes LANCELOT B, an augmented Lagrangian method with a nonmonotone descent condition; FILTRANE, a solver for feasibility problems based on a multidimensional filter; and an interior point and active-set methods for solving large-scale quadratic programs.

LANCELOT (Conn, Gould, & Toint, 1992b) is a large-scale implementation of a bound-constrained augmented Lagrangian method. In LANCELOT, the simple bounds are treated explicitly, and all the other constraints are converted to equations and incorporated into an augmented Lagrangian function. It approximately solves a sequence of bound-constrained augmented Lagrangian subproblems by using a trust-region approach. In LANCELOT, each trust-region subproblem is solved approximately, firstly by identifying a Cauchy point to ensure the global convergence and then by applying conjugate gradient steps to accelerate the local convergence. Although this approach proves to be robust (Conn, Gould, & Toint, 1996a), it has a number of drawbacks. One of them is that the augmentation may not be the ideal way to treat the linear constraints. A more attractive approach is to handle all the linear constraints explicitly (Conn, Gould, Sartenaer, & Toint, 1996b). The methods that use the linearly constrained subproblems are described by Rosen and Kreuser (1972), Robinson (1972, 1974), and Murtagh and Saunders (1978).

MINOS (Murtagh & Saunders, 1987, 1995) uses a projected augmented Lagrangian for solving general nonlinear problems. At each “major iteration,” a linearly constrained nonlinear subproblem is solved, where the linear constraints constitute all the linear constraints of (14.3) and the linearizations of the nonlinear constraints. The objective function of this subproblem is an augmented Lagrangian which contains the departure from linearity of the nonlinear constraints. The subproblems are solved by using a reduced gradient algorithm along with a quasi-Newton algorithm. The quasi-Newton algorithm provides a search direction along which a line-search is performed to improve the value of the objective function and to reduce the infeasibilities. MINOS integrates a lot of computational ingredients from linear algebra. The most important is LUSOL, a set of procedures for computing and updating the LU factors of a general sparse matrix  $A$ . The original LUSOL procedures are described in Gill, Murray, Saunders, and Wright (1987). The main factorization uses a traditional Markowitz strategy with threshold partial pivoting (Markowitz, 1957). It is designed to allow  $A$  to be square or

rectangular with arbitrary rank, to factorize  $A = LU$  directly by finding suitable row and column orderings, to replace a column or a row of  $A$ , to add or delete a column or a row of  $A$ , etc. LUSOL continues to evolve with the addition of rank-revealing LU factorizations for sparse matrices, using either threshold rook pivoting or threshold complete pivoting (Saunders, 2015a, b). MINOS is embedded in the GAMS technology (Andrei, 2017c). Both LANCELOT and MINOS are suitable for solving large-scale nonlinear optimization problems. However, they differ significantly in the formulation of the step-computation subproblems and in the techniques used to solve these subproblems. MINOS implements the linearly constrained Lagrangian, followed by a reduced-space approach to handle the linearized constraints, and it employs a dense quasi-Newton approximation to the Hessian of the Langrangian. On the other hand, LANCELOT implements the bound-constrained Lagrangian method. Numerical experiments have shown an advantage of MINOS over LANCELOT for problems with linear constraints. Our description of MINOS is based on the papers of Murtagh and Saunders (1978, 1980, 1982, 1995).

SPENBAR is another approach, based on the logarithmic penalty function. In this approach, the logarithmic terms prevent the feasible iterates from moving too close to the boundary of the feasible region of the problem. SPENBAR is designed for solving large-scale constrained optimization problems. Only the nonzero elements of the Jacobian of the equalities and of the inequalities are stored separately. In SPENBAR, the subproblem is solved by the truncated Newton with simple bounds (TNBC). The SPENBAR algorithm, presented in Andrei (1996a, b, c, 1998a, 2001), combines the augmented Lagrangian with a log-barrier function (a composite function which includes the augmented Lagrangian and the classical log-barrier function) in which the parameters are updated in such a way as to obtain a KKT point for the considered nonlinear optimization problem. The equality and inequality constraints are treated separately. In this variant, the penalty-barrier function includes only the equality and the inequality constraints. Thus, the algorithm reduces to minimizing a sequence of simple bound optimization subproblems. The simple bounds on variables can also be introduced in the composite function, thus obtaining another variant of the penalty-barrier algorithm, in which a sequence of unconstrained minimization subproblems is used. For both variants, the convergence theory is the same (Ben-Tal & Zibulevsky, 1993; Conn, Gould, & Toint, 1991a, 1992a). However, the numerical performances could be different. SPENBAR illustrates the importance of modifying the classical augmented Lagrangian in order to avoid the ill-conditioning. Also see Vassiliadis and Floudas (1997).

PENNON (Kocvara & Stingl, 2003) is an augmented Lagrangian penalty-barrier method. It solves a sequence of unconstrained optimization subproblems in which the inequality constraints are included in the barrier functions and the equality constraints in the penalty functions. Every unconstrained minimization subproblem is solved with Newton's method.

A new penalty method for solving the general nonlinear optimization problems subject to inequality constraints was presented by Xavier (2001).



# Sequential Quadratic Programming

15

Sequential quadratic programming (SQP) methods are very effective for solving optimization problems with significant nonlinearities in constraints. These are active-set methods and generate steps by solving quadratic programming subproblems at every iteration. These methods are used both in the line-search and in the trust-region paradigm. In this chapter, we consider both the equality quadratic programming and the inequality quadratic programming. There are some differences between these approaches. In the inequality quadratic programming, at every iteration, a general inequality-constrained quadratic programming problem is solved for computing a step and for generating an estimate of the optimal active-set. On the other hand, in the equality quadratic programming, these are separated. Firstly, they compute an estimate of the optimal active-set, then they solve an equality-constrained quadratic programming problem to find the step. Our presentation proceeds to develop the theory of the sequential quadratic programming approach for the step computation, followed by practical line-search and trust-region methods that achieve the convergence from remote starting points. At the same time, a number of three implementations of these methods are discussed, which are representative of the following: an SQP algorithm for large-scale-constrained optimization (SNOPT), an SQP algorithm with successive error restoration (NLPQLP), and the active-set sequential linear-quadratic programming (KNITRO/ACTIVE). Their performances are illustrated in solving large-scale problems from the LACOP collection.

## Equality-Constrained Problems

Let us consider the equality-constrained nonlinear optimization problem

$$\min f(x) \quad (15.1a)$$

$$\begin{aligned} & \text{subject to} \\ & h(x) = 0, \end{aligned} \quad (15.1b)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are smooth functions, where  $h(x) = [h_1(x), \dots, h_m(x)]^T$ . The idea of the SQP method is to model (15.1) at the current point  $x_k$  by a quadratic programming subproblem and then to use the solution of this subproblem to define a new iterate  $x_{k+1}$ . The most challenging is how to design the quadratic subproblem so that it yields a good approximation of (15.1). One of the simplest derivations of the SQP methods is to apply Newton's method to the KKT optimality conditions for the nonlinear problem (15.1).

The Lagrange function for (15.1) is

$$L(x, \lambda) = f(x) - \lambda^T h(x), \quad (15.2)$$

where  $\lambda \in \mathbb{R}^m$  is the vector of the Lagrange multipliers associated to the constraints of the problem. Let us define  $A(x)$  as the Jacobian matrix of the constraints, that is

$$A(x)^T = [\nabla h_1(x), \dots, \nabla h_m(x)] \in \mathbb{R}^{n \times m}, \quad (15.3)$$

where  $h_i(x)$ ,  $i = 1, \dots, m$ , are the components of the vector  $h(x)$ . The first-order optimality conditions of the equality-constrained problem can be written as a system of  $n + m$  equations with  $n + m$  unknowns  $x$  and  $\lambda$ :

$$F(x, \lambda) = \begin{bmatrix} \nabla f(x) - A(x)^T \lambda \\ h(x) \end{bmatrix} = 0. \quad (15.4)$$

Observe that any solution  $(x^*, \lambda^*)$  of (15.1) for which  $A(x^*)$  has full rank satisfies (15.4). To find such a point, the solution of the nonlinear system (15.4), the Newton method is the most suitable. The Jacobian of (15.4) with respect to  $x$  and  $\lambda$  is given by

$$F'(x, \lambda) = \begin{bmatrix} \nabla_{xx}^2 L(x, \lambda) & -A(x)^T \\ A(x) & 0 \end{bmatrix}, \quad (15.5)$$

where  $\nabla_{xx}^2 L(x, \lambda) = \nabla^2 f(x) - \sum_{i=1}^m \lambda_i \nabla^2 h_i(x)$ . The Newton step from the iterate  $(x_k, \lambda_k)$  is given by

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} d_x \\ d_\lambda \end{bmatrix}, \quad (15.6)$$

where  $d_x$  and  $d_\lambda$  are solutions of the Newton system

$$\begin{bmatrix} \nabla_{xx}^2 L(x_k, \lambda_k) & -A(x_k)^T \\ A(x_k) & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) + A(x_k)^T \lambda_k \\ -h(x_k) \end{bmatrix}. \quad (15.7)$$

The Newton iteration (15.6) is well defined when the KKT matrix in (15.7) is nonsingular. As we know (see Chap. 11), this matrix is nonsingular if the following two assumptions hold:

C1. *The Jacobian  $A(x)$  of the constraints has full row rank.*

C2. *The matrix  $\nabla_{xx}^2 L(x, \lambda)$  is positive definite on the tangent space of the constraints, that is,*

$$d^T \nabla_{xx}^2 L(x, \lambda) d > 0 \text{ for all } d \neq 0 \text{ for which } A(x)d = 0.$$

Observe that the first assumption is exactly the linear independence constraint qualification (LICQ) discussed in Chap. 11 (see Remark 11.2). The second assumption holds whenever  $(x, \lambda)$  is close to the optimum point  $(x^*, \lambda^*)$  and the second-order sufficient condition is satisfied at the solution (see Theorem 11.13).

This approach, which theoretically is very well justified, has an alternative that *illustrates the motivation of the sequential quadratic programming*. Let us focus on the Newton iteration given by (15.6) and (15.7) and let us assume that at the iteration  $(x_k, \lambda_k)$  the problem (15.1) is modeled as a quadratic program

$$\min_{d \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d, \quad (15.8a)$$

subject to

$$A(x_k)d + h(x_k) = 0. \quad (15.8b)$$

If the assumptions (C1) and (C2) hold, then this problem has a unique solution  $(d_k, l_k)$  that satisfies

$$\nabla_{xx}^2 L(x_k, \lambda_k) d_k + \nabla f(x_k) - A(x_k)^T l_k = 0, \quad (15.9a)$$

$$A(x_k) d_k + h(x_k) = 0. \quad (15.9b)$$

But the vectors  $d_k$  and  $l_k$  can be immediately identified as the solution of the Newton system (15.7). Now, if we subtract  $A(x_k)^T \lambda_k$ , from both sides of the first equation in (15.7), we get

$$\begin{bmatrix} \nabla_{xx}^2 L(x_k, \lambda_k) & -A(x_k)^T \\ A(x_k) & 0 \end{bmatrix} \begin{bmatrix} d_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ -h(x_k) \end{bmatrix}. \quad (15.10)$$

Therefore, having in view the nonsingularity of the coefficient matrix of (15.10), it follows that  $\lambda_{k+1} = l_k$  and  $d_k$  represents a solution of the quadratic program (15.8) and of the system (15.7).

In conclusion, *the new iterate  $(x_{k+1}, \lambda_{k+1})$  can be defined either as the solution of the quadratic programming problem (15.8) or as the iterate generated by the Newton method (15.6) and (15.7) applied to the optimality conditions of the problem (15.1)*. Both these approaches are useful. The Newton point of view is suitable for analysis, whereas the sequential quadratic programming enables us to derive practical algorithms and to extend the technique to the inequality-constrained case.

Based on these theoretical developments, the following sequential quadratic programming algorithm for equality constraints can be presented.

### Algorithm 15.1 Sequential quadratic programming—equality constraints

- |    |  |
|----|--|
| 1. | Choose an initial point $(x_0, \lambda_0)$ and set $k = 0$   |
| 2. | If a test for stopping the iterations is satisfied, stop; otherwise, go to step 3                  |
| 3. | Evaluate: $f(x_k)$ , $\nabla f(x_k)$ , $\nabla_{xx}^2 L(x_k, \lambda_k)$ , $h(x_k)$ , and $A(x_k)$ |
| 4. | Solve the quadratic subproblem (15.8) to obtain a solution $(d_k, l_k)$                            |
| 5. | Set $x_{k+1} = x_k + d_k$ and $\lambda_{k+1} = l_k$ . Set $k = k + 1$ and go to step 2             |

◆

It is obvious that in the objective function (15.8a), the linear term  $\nabla f(x_k)^T d$  can be replaced by  $\nabla_x L(x_k, \lambda_k)^T d$ , since the constraints (15.8b) make the two choices equivalent. In this case, (15.8a) is exactly a quadratic approximation of the Lagrangian function. This is the main motivation for the choice of the quadratic model (15.8): *first replace the nonlinear optimization problem (15.1) by the problem of minimizing the Lagrangian subject to the equality constraints (15.1b), then make a quadratic approximation to the Lagrangian and a linear approximation to the constraints to obtain the quadratic program (15.8)*.

Now, let us present the conditions that guarantee the local convergence of Algorithm 15.1. Consider that the algorithm uses exact second derivatives.

**Theorem 15.1** Assume that the point  $x^*$  is a local solution of the problem (15.1) at which the following conditions hold:

- (i) The functions  $f$  and  $h$  are twice continuously differentiable in a neighborhood of  $x^*$  with Lipschitz continuous second derivatives.
- (ii) The linear independence constraint qualification (see Remark 11.2) holds at  $x^*$ . This condition implies that the KKT conditions (11.21) (see Theorem 11.15) are satisfied for a certain vector of multipliers  $\lambda^*$ .
- (iii) The second-order sufficient conditions (see Theorem 11.16) hold at  $(x^*, \lambda^*)$ .

Then, if  $(x_0, \lambda_0)$  is sufficiently close to  $(x^*, \lambda^*)$ , the pairs  $(x_k, \lambda_k)$  generated by Algorithm 15.1 quadratically converge to  $(x^*, \lambda^*)$ .

**Proof** This result follows directly from the convergence of the Newton method applied to the nonlinear system (15.4).  $\blacklozenge$

Algorithm 15.1 can be specified in some variants. A version of the algorithm uses a quasi-Newton estimate of  $\nabla_{xx}^2 L(x_k, \lambda_k)$  rather than calculating it from the second derivatives (see Wilson (1963), Han (1976), and Powell (1978a)). In step 5 of the algorithm, we can use the line-search to determine  $x_{k+1} = x_k + ad_k$  ensuring  $P(x_{k+1}) < P(x_k)$ , where  $P$  denotes a certain penalty function. Different penalty functions can be tried. The line-search is important because it forces a reduction in a composite function, involving both the objective  $f$  and the constraints  $h$ . It ensures that the new point  $x_{k+1}$  is, in a measurable way, an improvement on  $x_k$  and therefore provides a basis for a proof of the convergence.

### Inequality-Constrained Problems

The above-described sequential quadratic programming approach can be extended to the general nonlinear optimization problem

$$\min f(x) \quad (15.11a)$$

$$\begin{aligned} \text{subject to} \\ c_i(x) = 0, \quad i \in E, \end{aligned} \quad (15.11b)$$

$$c_i(x) \geq 0, \quad i \in I, \quad (15.11c)$$

where  $E \triangleq \{1, \dots, m_e\}$  and  $I \triangleq \{m_e + 1, \dots, m\}$  are sets on indices for the equality and inequality constraints, respectively. Assume that all the functions of the problem (15.11) are twice continuously differentiable.

As in the case of the equality-constrained problems, at the current point  $x_k$ , the problem (15.11) is modeled as a quadratic program by linearizing both the equality and the inequality constraints to obtain

$$\min_{d \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d \quad (15.12a)$$

$$\begin{aligned} \text{subject to} \\ \nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in E, \end{aligned} \quad (15.12b)$$

$$\nabla c_i(x_k)^T d + c_i(x_k) \geq 0, \quad i \in I. \quad (15.12c)$$

For solving the subproblem (15.12), we can use the algorithms for quadratic programming described in Chap. 13. The new iterate is given by  $(x_k + d_k, \lambda_{k+1})$ , where  $d_k$  and  $\lambda_{k+1}$  are the solution and the corresponding Lagrange multiplier of (15.12). The algorithm is as follows:

**Algorithm 15.2** *Sequential quadratic programming—inequality constraints*

- |    |   |
|----|---|
| 1. | Choose an initial point $(x_0, \lambda_0)$ and set $k = 0$  |
| 2. | If a test for stopping the iterations is satisfied, stop; otherwise, go to step 3                                     |
| 3. | Evaluate: $f(x_k), \nabla f(x_k), \nabla^2_{xx} L(x_k, \lambda_k), c_i(x_k)$ and $\nabla c_i(x_k)$ , $i \in E \cup I$ |
| 4. | Solve the quadratic problem (15.12) to obtain a solution $(d_k, l_k)$   |
| 5. | Set $x_{k+1} = x_k + d_k$ and $\lambda_{k+1} = l_k$ . Set $k = k + 1$ and go to step 2                                |

◆

In this approach, the set of active constraints at the solution of (15.12) is the guess of the active-set at the solution of the nonlinear problem (15.11). If the SQP method is able to correctly identify the optimal active-set (i.e., the active-set at the solution of (15.11)), then the method works like a Newton method for the equality-constrained optimization. In this case, the convergence will be rapid. The following result, given by Robinson (1974), presents the conditions under which this behavior of the algorithm holds.

**Theorem 15.2** *Assume that  $x^*$  is a local solution of (15.11) at which the KKT conditions are satisfied for some  $\lambda^*$ . Also, assume that the linear independence constraint qualification (Remark 11.2), the strict complementarity (Definition 11.22) and the second-order sufficient conditions (11.13) hold at  $(x^*, \lambda^*)$ . Then, if  $(x_k, \lambda_k)$  is sufficiently close to  $(x^*, \lambda^*)$ , there is a local solution of the subproblem (15.12) whose active-set is the same as the active-set of the nonlinear program (15.11) at  $x^*$ .*

**Remark 15.1** To design SQP algorithms for solving the general nonlinear optimization problem (15.11), two methods are known. The first, which has been described above, solves the quadratic subproblem (15.12) at each iteration by taking the active-set at the solution of this subproblem as a guess of the optimal active-set. This approach is called IQP (*inequality-constrained QP*). It has proved to be quite effective and successful in practice. Its main disadvantage is the expense of solving the general quadratic subproblem (15.12), which can be high when the problem is large. However, an improvement of this situation could be as follows: as the iterates converge to the solution, solving the quadratic subproblem (15.12) becomes economical if the information from the previous iteration is used to make a good guess of the optimal solution to the current subproblem. This strategy is called *warm-start*. The second approach selects a subset of constraints at each iteration to be the so-called *working set* and solves only equality-constrained subproblems of the form (15.8), where the constraints in the working set are imposed as equalities and all the other constraints are ignored. This approach is called EQP (*equality-constrained QP*). In it, the working set is updated at every iteration by rules based on the Lagrange multiplier estimates or by solving an auxiliary subproblem. The EQP is more effective because the equality-constrained quadratic subproblems are less expensive to solve than (15.12), when the problem is large. An example of an EQP method is the sequential linear-quadratic programming (SLQP) method, which is to be discussed in Sect. 15.6. Another successful EQP is the gradient projection method in the context of the bound-constrained quadratic programming.

All these ideas will be developed in the next sections of this chapter. We note that Algorithms 15.1 and 15.2 are successful for solving nonlinear optimization problems. However, these algorithms in the form in which they were shown represent a simple approach of the sequential quadratic

programming method. Some other ingredients must be introduced in their structure, in order to have efficient and robust SQP algorithms. In the following, we try to be more specific and detail the SQP for inequality-constrained problems. After that, some important ingredients are introduced in the SQP approach for having modern, efficient, and robust algorithms, able to solve complex large-scale nonlinear optimization problems.  $\blacklozenge$

## 15.1 A Simple Approach to SQP

The SQP Algorithms 15.1 and 15.2 have good local properties. However, they represent a rudimentary approach of this method. As a matter of fact, it can be shown that Algorithm 15.2 converges quadratically if: (i) the initial point is sufficiently close to a local minimizer  $x^*$ , (ii) the coefficient matrix in (15.7) or in (15.10) for  $(x_0, \lambda_0)$  is nonsingular, (iii) the second-order sufficient conditions hold for  $(x^*, \lambda^*)$  with  $\text{rank}(A(x^*)) = m$ , and (iv) for  $(x_0, \lambda_0)$  the quadratic programming problem (15.12) has a unique solution. (See Fletcher 1987, Chap. 12).

To be more specific, let us consider the inequality-constrained problem

$$\min f(x) \quad (15.13a)$$

subject to

$$c_i(x) \geq 0, \quad i \in I, \quad (15.13b)$$

where  $I \triangleq \{1, \dots, m\}$  is the set of indices for the inequality constraints. Suppose that the functions  $f(x)$  and  $c_i(x)$ ,  $i = 1, \dots, m$ , are twice continuously differentiable and the feasible region determined by the constraints (15.13b) is nonempty. The Lagrange function associated to (15.13) is

$$L(x, \lambda) = f(x) - \sum_{i=1}^m \lambda_i c_i(x). \quad (15.14)$$

The KKT conditions are as follows:

$$\nabla_x L(x, \lambda) = 0, \quad (15.15a)$$

$$c_i(x) \geq 0, \quad i = 1, \dots, m, \quad (15.15b)$$

$$\lambda \geq 0, \quad (15.15c)$$

$$\lambda_i c_i(x) = 0, \quad i = 1, \dots, m. \quad (15.15d)$$

Now, for the iterate  $(x_k, \lambda_k)$  an increment  $(\delta_x, \delta_\lambda)$  needs to be found such that the next iterate  $(x_{k+1}, \lambda_{k+1}) = (x_k + \delta_x, \lambda_k + \delta_\lambda)$  approximates the KKT conditions (15.15) in the sense that

$$\nabla_x L(x_{k+1}, \lambda_{k+1}) \approx \nabla_x L(x_k, \lambda_k) + \nabla_{xx}^2 L(x_k, \lambda_k) \delta_x + \nabla_{x\lambda}^2 L(x_k, \lambda_k) \delta_\lambda = 0, \quad (15.16a)$$

$$c_i(x_k + \delta_x) \approx c_i(x_k) + \delta_x^T \nabla c_i(x_k) \geq 0, \quad i = 1, \dots, m, \quad (15.16b)$$

$$(\lambda_{k+1})_i \geq 0, \quad i = 1, \dots, m, \quad (15.16c)$$

$$[c_i(x_k) + \delta_x^T \nabla c_i(x_k)] (\lambda_{k+1})_i = 0, \quad i = 1, \dots, m. \quad (15.16d)$$

From (15.14) it follows that

$$\nabla_x L(x_k, \lambda_k) = \nabla f(x_k) - \sum_{i=1}^m (\lambda_k)_i \nabla c_i(x_k) \equiv g_k - A_k^T \lambda_k, \quad (15.17a)$$

$$\nabla_{xx}^2 L(x_k, \lambda_k) = \nabla^2 f(x_k) - \sum_{i=1}^m (\lambda_k)_i \nabla^2 c_i(x_k) \equiv W_k, \quad (15.17b)$$

$$\nabla_{x\lambda}^2 L(x_k, \lambda_k) = -A_k^T, \quad (15.17c)$$

where

$$A_k = \begin{bmatrix} \nabla^T c_1(x_k) \\ \vdots \\ \nabla^T c_m(x_k) \end{bmatrix}. \quad (15.17d)$$

With this, the approximate KKT conditions can be expressed as

$$W_k \delta_x + g_k - A_k^T \lambda_{k+1} = 0, \quad (15.18a)$$

$$A_k \delta_x \geq -c_k, \quad (15.18b)$$

$$\lambda_{k+1} \geq 0, \quad (15.18c)$$

$$(\lambda_{k+1})_i (A_k \delta_x + c_k)_i = 0, \quad i = 1, \dots, m, \quad (15.18d)$$

where

$$c_k = [c_1(x_k), c_2(x_k), \dots, c_m(x_k)]^T \quad (15.18e)$$

is the vector of the constraints from (15.13).

Given  $(x_k, \lambda_k)$ , then (15.18) can be interpreted as the exact KKT conditions of the quadratic problem

$$\begin{aligned} & \min \frac{1}{2} \delta_x^T W_k \delta_x + \delta_x^T g_k \\ & \text{subject to} \\ & A_k \delta_x \geq -c_k. \end{aligned} \quad (15.19)$$

If  $\delta_x$  is a regular solution of (15.19), that is the gradients of the constraints that are active at  $x_k$  are linearly independent, then (15.18a) can be written as

$$W_k \delta_x + g_k - A_{ak}^T l_{k+1} = 0,$$

where the rows of  $A_{ak}$  are the rows of  $A_k$  satisfying the equality  $(A_k \delta_k + c_k)_i = 0$  and  $l_{k+1}$  is the associated Lagrange multiplier vector. Therefore,  $l_{k+1}$  can be computed as

$$l_{k+1} = (A_{ak} A_{ak}^T)^{-1} A_{ak} (W_k \delta_x + g_k). \quad (15.20)$$

Of course, from the complementarity conditions (15.18d) it follows that the Lagrange multiplier  $\lambda_{k+1}$  can be obtained by inserting zeros where necessary in  $l_{k+1}$ . With this, the following algorithm can be presented.

**Algorithm 15.3** Sequential quadratic programming—inequality constraints. Variant

- |    |  |
|----|--|
| 1. | Choose an initial point $(x_0, \lambda_0)$ and set $k = 0$ . $x_0$ and $\lambda_0$ are chosen such that $c_i(x_0) \geq 0$ , $i = 1, \dots, m$ , and $\lambda_0 \geq 0$ . Choose the sufficiently small convergence tolerance $\varepsilon > 0$ |
| 2. | Evaluate: $g_k = \nabla f(x_k)$ , $c(x_k)$ , $A_k$ and $W_k$   |
| 3. | Solve the quadratic problem (15.19) to obtain a solution $\delta_x$ and from (15.20) compute the multiplier $l_{k+1}$  |
| 4. | Set $x_{k+1} = x_k + \delta_x$ . If $\ \delta_x\  \leq \varepsilon$ , then output $x^* = x_{k+1}$ and stop. Otherwise, set $k = k + 1$ and go to step 2  |
- ◆

Algorithm 15.3 is very close to the Algorithms 15.1 and 15.2. The main disadvantage of the Algorithms 15.1, 15.2, and 15.3 is that, they may fail to converge when the initial point  $x_0$  is far away from  $x^*$ . Another disadvantage is that they need to evaluate the Hessian of the Lagrangian which combines the Hessian of the minimizing function with the Hessians of the constraints. Therefore, these algorithms would require a large number of function evaluations, even for problems of moderate size.

To ameliorate and to save their behavior, two modifications could be operational. The first one uses a line-search method proposed by Han (1977), which enables the Algorithms 15.1, 15.2, and 15.3 to converge from arbitrary initial points. The second one approximates the Hessian of the Lagrangian by using the BFGS updating formula. As in the BFGS algorithm for unconstrained optimization, when starting with an initial positive definite approximation for the Hessian, all the subsequent approximations will remain positive definite under a certain mild condition.

**SQP with Line-Search Step**

Consider the problem (15.13). Then a line-search step can be introduced in Algorithm 15.3 by generating the  $(k + 1)$ -th iteration as

$$x_{k+1} = x_k + \alpha_k \delta_x, \quad (15.21)$$

where  $\delta_x$  is the solution of the quadratic problem (15.19) and  $\alpha_k$  is the stepsize obtained by a line-search. Han (1977) suggested that the scalar  $\alpha_k$  should be computed by minimizing the one-variable exact penalty function

$$\varphi(\alpha) = f(x_k + \alpha \delta_x) + r \sum_{i=1}^m [c_i(x_k + \alpha \delta_x)]^- \quad (15.22)$$

on an interval  $[0, \delta]$ , where the length  $\delta$  of the interval and the parameter  $r > 0$  are fixed throughout the minimization process and  $[c_i(x_k + \alpha \delta_x)]^- = \max \{0, -c_i(x_k + \alpha \delta_x)\}$ . Observe that the second term in the right-hand side of (15.22) is always nonnegative and contains only those constraints that violate the nonnegativity condition in (15.13). The term “penalty function” used for  $\varphi(\alpha)$  is related to the fact that the value of  $\varphi(\alpha)$  depends on how many constraints from (15.13) are violated at  $\alpha$  and on the degree of violation. Choosing an appropriate value for  $r$ , a line-search algorithm applied to  $\varphi(\alpha)$  will yield a value  $\alpha_k$  such that the objective function at  $x_k + \alpha_k \delta_x$  is reduced with fewer violated constraints and a reduced degree of violation. The difficulty with this approach is that the penalty function  $\varphi(\alpha)$  is not differentiable. However, efficient search methods for the minimization of the nondifferentiable one-dimensional functions are available (see the direct methods from Chap. 9).

An alternative to this method for the determination of  $\alpha_k$  was suggested by Powell (1978c). In this method, an inexact line-search is applied to the Lagrangian (15.14) with  $x = x_k + \alpha \delta_x$  and  $\lambda = \lambda_{k+1}$  to obtain the following one-variable function

$$\psi(\alpha) = f(x_k + \alpha\delta_k) - \sum_{i=1}^m (\lambda_{k+1})_i c_i(x_k + \alpha\delta_k), \quad (15.23)$$

where  $(\lambda_{k+1})_i$  is the  $i$ -th component of the vector  $\lambda_{k+1}$ . Observe that if  $\lambda_{k+1} \geq 0$ , then the second term in the right-hand side (15.23) acts as a penalty term since an  $\alpha$  for which  $c_i(x_k + \alpha\delta_k) < 0$  will increase the value of  $\psi(\alpha)$ . Powell suggested a method for determining  $\alpha_k$  by minimizing  $\psi(\alpha)$  on the interval  $[0,1]$ . Firstly, a line-search is performed to find the minimizer  $\alpha_1^*$  of  $\psi(\alpha)$  on  $[0,1]$ , that is

$$\alpha_1^* = \arg \min_{0 \leq \alpha \leq 1} \psi(\alpha).$$

Observe that  $\psi(\alpha)$  is differentiable and hence efficient gradient algorithms can be used to determine  $\alpha_1^*$ . Next, from the transversality condition of the optimality conditions (15.18d), observe that, if for a particular index  $i$ ,  $c_i(x_k) + \delta_k^T \nabla c_i(x_k) = 0$ , then  $(\lambda_{k+1})_i = 0$  and in this case the term  $(\lambda_{k+1})_i c_i(x_k + \alpha\delta_k)$  is not present in function  $\psi(\alpha)$  from (15.23). In other words, the constraints  $c_i(x) \geq 0$  for which  $(\lambda_{k+1})_i = 0$  need to be dealt separately. Now, let us define the index set  $I = \{i : (\lambda_{k+1})_i = 0\}$  and evaluate

$$\alpha_2^* = \min_{i \in I} \max \{\alpha : c_i(x_k + \alpha\delta_k) \geq 0\}.$$

Therefore, the value of  $\alpha_k$  in (15.21) is computed as

$$\alpha_k = 0.95 \min \{\alpha_1^*, \alpha_2^*\}.$$

With this stepsize  $\alpha_k$ , in order to compute  $l_{k+1}$  and  $\lambda_{k+1}$  the increment  $\delta_x$  in (15.20) needs to be modified as  $\alpha_k \delta_x$ . In addition, in step 4 of Algorithm 15.3,  $x_{k+1} = x_k + \delta_k$  is modified as  $x_{k+1} = x_k + \alpha_k \delta_x$ . With this line-search included, the modified SQP algorithm turns out to be more robust in the sense that it converges from arbitrary initial points.

### SQP with Approximate Hessian

The BFGS updating formula applied to the Lagrangian (15.14) is given by

$$W_{k+1} = W_k - \frac{W_k \delta_x \delta_x^T W_k}{\delta_x^T W_k \delta_x} + \frac{y_k y_k^T}{y_k^T \delta_x}. \quad (15.24)$$

where  $\delta_x = x_{k+1} - x_k$  and

$$y_k = \nabla_x L(x_{k+1}, \lambda_{k+1}) - \nabla_x L(x_k, \lambda_k) = (g_{k+1} - g_k) - (A_{k+1} - A_k)^T \lambda_{k+1}. \quad (15.25)$$

If  $W_k$  is positive definite, then  $W_{k+1}$  obtained from (15.24) is also positive definite if and only if  $y_k^T \delta_x > 0$ . Obviously, this condition does not hold when the Lagrangian has a negative curvature for the iterate  $(x_{k+1}, \lambda_{k+1})$ . Powell proposed a method for overcoming this difficulty by replacing  $y_k$  in (15.24) by

$$\eta_k = \theta y_k + (1 - \theta) W_k \delta_x,$$

where  $y_k$  is given by (15.25) and

$$\theta = \begin{cases} 1, & \text{if } \delta_x^T y_k \geq 0.2 \delta_x^T W_k \delta_x, \\ \frac{0.8 \delta_x^T W_k \delta_x}{\delta_x^T W_k \delta_x - \delta_x^T y_k}, & \text{otherwise.} \end{cases}$$

Algorithm 15.3 can be easily adapted to include both the search step (15.21) and the BFGS approximate to the Hessian (15.24).

### Equality Constraints

Now, let us consider the nonlinear optimization problems with equality constraints

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & h_i(x) = 0, \quad i \in E, \end{aligned}$$

where  $E \triangleq \{1, \dots, me\}$  is the set of indices for the equality constraints. Suppose that the functions  $f(x)$  and  $h_i(x)$ ,  $i = 1, \dots, me$ , are twice continuously differentiable and the feasible region determined by the constraints is nonempty. The Lagrange function associated to this problem is

$$L(x, \lambda) = f(x) - \sum_{i=1}^{me} \lambda_i h_i(x).$$

The first-order optimality condition is  $\nabla L(x, \lambda) = 0$ . Then the formula for the Newton method is

$$\begin{aligned} x_{k+1} &= x_k + d_k, \\ \lambda_{k+1} &= \lambda_k + v_k, \end{aligned}$$

where  $d_k$  and  $v_k$  are solutions to the linear system

$$\begin{bmatrix} \nabla_{xx}^2 L(x_k, \lambda_k) & -\nabla h(x_k) \\ -\nabla h(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} d_k \\ v_k \end{bmatrix} = \begin{bmatrix} -\nabla_x L(x_k, \lambda_k) \\ h(x_k) \end{bmatrix},$$

where  $v_k$  is the vector of the Lagrange multipliers. Observe that this system of linear equations represents the first-order optimality conditions for the following optimization problem:

$$\begin{aligned} & \min \frac{1}{2} d^T [\nabla_{xx}^2 L(x_k, \lambda_k)] d + d^T [\nabla_x L(x_k, \lambda_k)] \\ & \text{subject to} \\ & [\nabla h(x_k)]^T d + h(x_k) = 0. \end{aligned}$$

But this is a quadratic program which is the minimization of a quadratic function subject to linear constraints. In a sequential quadratic programming method, at each iteration, a quadratic program is solved to get  $(d_k, v_k)$ . This solution is used to update  $(x_k, \lambda_k)$ , and the process is repeated at the new point.

Computing the updates  $d_k$  and  $v_k$  by solving the quadratic program corresponds to applying the Newton method to the optimality conditions associated to the original problem. If the initial point is *close enough* to the solution, then the method is quadratic convergent, provided that  $\nabla_{xx}^2 L(x^*, \lambda^*)$  is nonsingular. The Hessian of the Lagrangian will be nonsingular if the regularity conditions (see Remark 11.4) and the second-order sufficient conditions (see Theorem 11.13) for the original optimization problem are satisfied, that is if  $\nabla h(x^*)$  is of full rank and if  $Z^T \nabla_{xx}^2 L(x^*, \lambda^*) Z$  is positive definite, where  $Z$  is a basis matrix for the null space of  $\nabla h(x^*)^T$ .

As in the case of nonlinear optimization problems with inequality constraints, for problems with equality constraints, it is very easy to introduce the SQP with line-search step and the SQP with approximate Hessian. The convergence results for the sequential quadratic programming method are

based on the idea that the approximation  $(x_{k+1}, \lambda_{k+1})$  is a *better* estimate of the solution than  $(x_k, \lambda_k)$ . To determine the progress of the algorithm along the iterations, a *merit function*  $\varphi(\cdot)$  is used. Usually, a merit function is the sum of two terms: the objective function and the amount of infeasibility of the constraints. If the new point reduces both the objective function and the infeasibility, then the value of the merit function will decrease. However, for many problems, improvements in the objective value come to increase the infeasibility and vice versa, so the merit function must balance these two goals.

Ideally, the merit function would be chosen in such a way that the solution  $(x^*, \lambda^*)$  would be a local minimizer of the merit function if and only if it was a local solution of the optimization problem. If this is the case, then a line-search with respect to the merit function could be performed as

$$\begin{aligned} x_{k+1} &= x_k + \alpha d_k, \\ \lambda_{k+1} &= \lambda_k + \alpha v_k, \end{aligned}$$

where  $\alpha$  is chosen such that  $\varphi(x_{k+1}, \lambda_{k+1}) < \varphi(x_k, \lambda_k)$ . To be successful, the search direction from the quadratic program would have to be a descent direction for the merit function. Unfortunately, it is rarely possible to guarantee that the local minimizers of the merit function coincide with the local solutions of the optimization problem.

Another point is when a quasi-Newton approximation to the Hessian is used to define the quadratic program. Then the positive definiteness of the reduced Hessian is often guaranteed by the choice of the quasi-Newton update formula. If the Newton method is used so that the Hessian in the quadratic program is exactly  $\nabla_{xx}^2 L(x_k, \lambda_k)$ , then it is necessary to test if the Hessian is positive definite and, if the case, to modify it by using, for example, the methods presented in Sect. 4.5 (Gill and Murray modification of the Newton method). In the constrained nonlinear optimization, testing whether the Hessian is positive definite is more complicated than in the unconstrained case.

Normally, near the solution, we would like to take a step  $\alpha = 1$  in the line-search so that the quadratic convergence rate of the Newton method could be achieved. Therefore, the merit function should be chosen so that a step of  $\alpha = 1$  is guaranteed to be accepted in the limit as the solution is approached. However, this is not true for some merit functions.

Therefore, as we can see this naive or rudimentary approach of the sequential quadratic programming has a number of deficiencies which preclude its applicability for solving large-scale problems.

Along the years, there has been considerable interest in understanding and improving the design and implementation of algorithms based on the idea of sequential quadratic programming. Some of these improvements, which are to be discussed in the next sections of this chapter, refer to enforcing the convergence from remote starting points, handling inconsistent linearizations, full quasi-Newton approximations, damped BFGS updating, reduced-Hessian quasi-Newton approximations, the merit function in sequential quadratic programming using nonsmooth penalty functions or augmented Lagrangians, second-order correction to overcome the Maratos effect, etc. In the following, we describe some of these points needed to produce *practical* SQP algorithms able to solve large-scale nonlinear optimization problems.

## 15.2 Reduced-Hessian Quasi-Newton Approximations

Let us consider the equality-constrained problem (15.1). By examining the KKT system (15.10), we can observe that the part of step  $d_k$  in the range space of  $A(x_k)^T$  is completely determined by  $A(x_k)d_k = -h(x_k)$ . The Lagrangian Hessian  $\nabla_{xx}^2 L(x_k, \lambda_k)$  affects only the part of  $d_k$  in the orthogonal subspace, i.e., the null space of  $A_k$ . Therefore, it seems reasonable to consider the quasi-Newton methods that find approximations to only that part of  $\nabla_{xx}^2 L(x_k, \lambda_k)$  which affects the component of  $d_k$

in the null space of  $A_k$ . This is the idea of the reduced-Hessian quasi-Newton approximations, which we are to be detailed in this section. For this, we consider the solution of the step equations (15.10) by means of the null-space approach presented in Sect. 13.1.

Let us define the matrices  $Y_k$  and  $Z_k$  whose columns span the range space of  $A(x_k)^T$  and the null space of  $A(x_k)$ , respectively. Using these matrices,  $d_k$  can be written as

$$d_k = Y_k d_Y + Z_k d_Z, \quad (15.26)$$

where  $d_Y$  is the *normal component* and  $d_Z$  is the *tangential component* of  $d_k$ . By substituting  $d_k$  from (15.26) in (15.10) the following system is obtained:

$$[A(x_k)Y_k]d_Y = -h(x_k), \quad (15.27a)$$

$$[Z_k^T \nabla_{xx}^2 L(x_k, \lambda_k)Z_k]d_Z = -Z_k^T \nabla_{xx}^2 L(x_k, \lambda_k)Y_k d_Y - Z_k^T \nabla f(x_k), \quad (15.27b)$$

to get the components  $d_Y$  and  $d_Z$ , in this order. Observe that from the first line of (15.10) the Lagrange multipliers  $\lambda_{k+1}$  can be computed by solving the linear system

$$[A(x_k)Y_k]^T \lambda_{k+1} = Y_k^T [\nabla f(x_k) + \nabla_{xx}^2 L(x_k, \lambda_k)d_k]. \quad (15.28)$$

In practical implementations, some simplifications are often considered. The idea is to avoid the computation of  $\nabla_{xx}^2 L(x_k, \lambda_k)$  by introducing several approximations in the null-space approach.

One simplification in the null-space approach is to remove the cross term  $Z_k^T \nabla_{xx}^2 L(x_k, \lambda_k)Y_k d_Y$ , in (15.27b), thus obtaining the more simple system

$$[Z_k^T \nabla_{xx}^2 L(x_k, \lambda_k)Z_k]d_Z = -Z_k^T \nabla f(x_k). \quad (15.29)$$

Dropping the cross term is motivated when  $Z_k^T \nabla_{xx}^2 L(x_k, \lambda_k)Z_k$  is replaced by a quasi-Newton approximation because the normal component  $d_Y$  usually converges to zero faster than the tangential component  $d_Z$ , thereby making (15.29) a good approximation of (15.27b).

Another simplification consists in deleting the term involving  $d_k$  from the right-hand side of (15.28), thus decoupling the computations of  $d_k$  and  $\lambda_{k+1}$ . A motivation of this simplification is that  $d_k$  converges to zero as we approach the solution, whereas  $\{\nabla f(x_k)\}$  does not. If we choose  $Y_k = A(x_k)^T$  (when  $A(x_k)$  has full row rank), we obtain

$$\hat{\lambda}_{k+1} = [A(x_k)A(x_k)^T]^{-1} A(x_k) \nabla f(x_k). \quad (15.30)$$

### 15.3 Merit Functions

Often, the SQP methods use a merit function to see whether a trial step should be accepted. In the line-search methods, the merit function controls the size of the step. In the trust-region methods, it determines whether the step is accepted or rejected and whether the trust-region radius should be adjusted. A variety of the merit functions was proposed in literature, including the nonsmooth penalty functions ( $l_1$  merit function) and the augmented Lagrangians. For the step computation and evaluation of a merit function, the inequality constraints are converted to equalities by introducing the slack variables. Therefore, in the following, we consider only the nonlinear equality-constrained problem (15.1).

The  $l_1$  merit function for (15.1) is defined by

$$P(x, \sigma) = f(x) + \sigma \|h(x)\|_1. \quad (15.31)$$

Another merit function was proposed by Wright (1976) and Schittkowski (1981, 1983) as augmented Lagrangian

$$L(x, \sigma, \lambda) = f(x) - \lambda^T h(x) + \frac{1}{2} \sigma h(x)^T h(x), \quad (15.32)$$

where  $\lambda$  is an estimation of the Lagrange multipliers and  $\sigma > 0$  is the penalty parameter.

In a line-search method, a step  $\alpha_k d_k$  is accepted if the following sufficient decrease condition holds:

$$P(x_k + \alpha_k d_k, \sigma_k) \leq P(x_k, \sigma_k) + \eta \alpha_k P'_{d_k}(x_k, \sigma_k), \quad (15.33)$$

where  $P'_{d_k}(x_k, \sigma_k)$  is the directional derivative of  $P(\cdot)$  in the direction  $d_k$  and  $\eta$  is a parameter sufficiently small,  $\eta \in (0, 1)$ . If  $d_k$  is a descent direction, that is  $P'_{d_k}(x_k, \sigma_k) < 0$ , then the reduction (15.33) resembles the rule of Armijo used in the unconstrained optimization. The following theorem shows that if  $\sigma$  is chosen sufficiently large, then the descent condition holds.

**Theorem 15.3** *Let  $d_k$  and  $\lambda_{k+1}$  be generated by the system (15.10). Then, the directional derivative of  $P(\cdot)$  in the direction  $d_k$  satisfies*

$$P'_{d_k}(x_k, \sigma_k) = \nabla f(x_k)^T d_k - \sigma \|h(x_k)\|_1. \quad (15.34)$$

Moreover, we have

$$P'_{d_k}(x_k, \sigma_k) \leq -d_k^T \nabla^2_{xx} L(x_k, \lambda_k) d_k - (\sigma - \|\lambda_{k+1}\|_\infty) \|h(x_k)\|_1. \quad (15.35)$$

**Proof** By applying Taylor's theorem we obtain

$$\begin{aligned} P(x_k + \alpha d, \sigma) - P(x_k, \sigma) &= f(x_k + \alpha d) - f(x_k) + \sigma \|h(x_k + \alpha d)\|_1 - \sigma \|h(x_k)\|_1 \\ &\leq \alpha \nabla f(x_k)^T d + \gamma \alpha^2 \|d\|^2 + \sigma \|h(x_k) + \alpha A(x_k) d\|_1 - \sigma \|h(x_k)\|_1, \end{aligned}$$

where  $\gamma$  is a positive constant which bounds the second derivative terms. If  $d = d_k$  is given by (15.10), then  $A(x_k) d_k = -h(x_k)$ , so that for  $\alpha \leq 1$  it follows that

$$P(x_k + \alpha d_k, \sigma) - P(x_k, \sigma) \leq \alpha \left[ \nabla f(x_k)^T d_k - \sigma \|h(x_k)\|_1 \right] + \alpha^2 \gamma \|d_k\|^2.$$

Similarly, we obtain the following lower bound:

$$P(x_k + \alpha d_k, \sigma) - P(x_k, \sigma) \geq \alpha \left[ \nabla f(x_k)^T d_k - \sigma \|h(x_k)\|_1 \right] - \alpha^2 \gamma \|d_k\|^2.$$

Taking limits, we conclude that

$$P'_{d_k}(x_k, \sigma_k) = \nabla f(x_k)^T d_k - \sigma \|h(x_k)\|_1,$$

which proves (15.34).

Now, since  $d_k$  satisfies the first equation from the system (15.10), then

$$P'_{d_k}(x_k, \sigma_k) = -d_k^T \nabla_{xx}^2 L(x_k, \lambda_k) d_k + d_k^T A(x_k)^T \lambda_{k+1} - \sigma \|h(x_k)\|_1. \quad (15.36)$$

But from the second equation of the system (15.10), the term  $d_k^T A(x_k)^T \lambda_{k+1}$  from (15.36) can be replaced by  $-h(x_k)^T \lambda_{k+1}$ . Having in view the Hölder inequality (see Appendix A), we have  $-h(x_k)^T \lambda_{k+1} \leq \|h(x_k)\|_1 \|\lambda_{k+1}\|_\infty$ , then, from (15.36), we obtain (15.35).  $\blacklozenge$

A strategy for selecting  $\sigma$ , which is appropriate for both line-search and trust-region algorithms, has in view the effect of the step on a model of the merit function. In (Nocedal, & Wright, 2006) a quadratic model for function  $P(\cdot)$  is defined as

$$q_\sigma(d) = f(x_k) + \nabla f(x_k)^T d + \frac{\mu}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d + \sigma m(d), \quad (15.37)$$

where  $m(d) = \|h(x_k) + A(x_k)d\|_1$  and  $\mu$  is a parameter to be defined. The strategy is as follows. After computing a step  $d_k$ , the penalty parameter  $\sigma$  is chosen large enough such that

$$q_\sigma(0) - q_\sigma(d_k) \geq \rho \sigma [m(0) - m(d_k)], \quad (15.38)$$

for some value of the parameter  $\rho \in (0, 1)$ . Now, from (15.37) and (15.8b) it follows that the inequality (15.38) is satisfied for

$$\sigma \geq \frac{\nabla f(x_k)^T d_k + (\mu/2) d_k^T \nabla_{xx}^2 L(x_k, \lambda_k) d_k}{(1 - \rho) \|h(x_k)\|_1}. \quad (15.39)$$

If the value of  $\sigma$  from the previous iteration of the SQP method satisfies (15.39), then it is left unchanged. Otherwise,  $\sigma$  is increased so that it satisfies this inequality with some margins. The constant  $\mu$  is used to handle the situations in which the Hessian  $\nabla_{xx}^2 L(x_k, \lambda_k)$  is not positive definite. A simple strategy for  $\mu$  is

$$\mu = \begin{cases} 1, & \text{if } d_k^T \nabla_{xx}^2 L(x_k, \lambda_k) d_k > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (15.40)$$

## 15.4 Second-Order Correction (Maratos Effect)

In the constrained nonlinear optimization, we practically face two purposes. One is to minimize the objective function and another one is to satisfy the constraints. A way to balance these competing goals is to use the merit functions or the filters. The idea is that a step generated by an algorithm will be accepted only if it leads to a sufficient reduction in the merit function or if it is acceptable to the filter. However, the following situation may appear. Suppose that an algorithm for solving nonlinear optimization problems generates a step that reduces the objective function value, but increases the violation of the constraints, a phenomenon known as the *Maratos effect* (1978). The following example due to Powell (1986b) illustrates this situation.

**Example 15.1** (*Maratos effect*) Let us consider the problem

$$\begin{aligned} \min f(x_1, x_2) &= 2(x_1^2 + x_2^2 - 1) - x_1 \\ \text{subject to} \\ h(x_1, x_2) &= x_1^2 + x_2^2 - 1 = 0. \end{aligned} \quad (15.41)$$

The solution of the problem is  $x_1^* = 1$ ,  $x_2^* = 0$  and the Lagrange multiplier is  $\lambda^* = 3/2$ . Moreover,  $\nabla_{xx}^2 L(x^*, \lambda^*) = I$ .

Consider an iterate  $x_k$  as  $x_k = [\cos\theta \ \sin\theta]^T$ . We immediately see that  $x_k$  is feasible for any value of  $\theta$ . Suppose that our algorithm computes the following step:

$$d_k = \begin{bmatrix} \sin^2\theta \\ -\sin\theta\cos\theta \end{bmatrix}, \quad (15.42)$$

which determines the next point (iterate)

$$x_k + d_k = \begin{bmatrix} \cos\theta + \sin^2\theta \\ (1 - \cos\theta)\sin\theta \end{bmatrix}.$$

Now, we can see that

$$\|x_k + d_k - x^*\|_2 = 2\sin^2(\theta/2) \text{ and } \|x_k - x^*\|_2 = 2|\sin(\theta/2)|.$$

Hence,

$$\frac{\|x_k + d_k - x^*\|_2}{\|x_k - x^*\|_2^2} = \frac{1}{2},$$

i.e., this step tends Q-quadratically to the solution of the problem. However, if we insist and compute, we find

$$\begin{aligned} f(x_k + d_k) &= \sin^2\theta - \cos\theta > -\cos\theta = f(x_k), \\ h(x_k + d_k) &= \sin^2\theta > h(x_k) = 0. \end{aligned}$$

Therefore, even if the considered step determines a Q-quadratic convergence, the values of the objective function increase, and the constraints are more and more unsatisfied. This behavior occurs for any value of  $\theta$  although the initial point is arbitrarily close to the solution.

For this example, let us consider the sequential quadratic programming algorithm in which  $d_k$  is determined as solution of the quadratic program (15.8), in which  $\nabla_{xx}^2 L(x_k, \lambda_k)$  is replaced by  $\nabla_{xx}^2 L(x^*, \lambda^*) = I$ . Hence, consider the current point  $x_k = [\cos\theta \ \sin\theta]^T$ . Since

$$f(x_k) = -\cos\theta, \quad \nabla f(x_k) = \begin{bmatrix} 4\cos\theta - 1 \\ 4\sin\theta \end{bmatrix}, \quad A(x_k)^T = \begin{bmatrix} 2\cos\theta \\ 2\sin\theta \end{bmatrix},$$

the quadratic subproblem (15.8) is

$$\begin{aligned} & \min_d (4 \cos \theta - 1)d_1 + 4d_2 \sin \theta + \frac{1}{2}d_1^2 + \frac{1}{2}d_2^2 \\ & \text{subject to} \\ & d_2 + d_1 \operatorname{ctg} \theta = 0. \end{aligned}$$

By solving this subproblem, we get the direction

$$d_k = \begin{bmatrix} \sin^2 \theta \\ -\sin \theta \cos \theta \end{bmatrix},$$

which coincides with (15.42), showing the failure of the sequential quadratic programming algorithm.  $\blacklozenge$

The Maratos effect shows that for many merit functions a superlinear convergent step may not be accepted, thus preventing the algorithm from a fast convergence. There are three ways to overcome the Maratos effect. The first is to relax the line-search conditions. The second is to use a second-order correction step  $\hat{d}_k$  which satisfies  $P(x_k + d_k + \hat{d}_k) < P(x_k)$ . In this way,  $d_k + \hat{d}_k$  is an acceptable step and it is still a superlinear convergent step. The third is to use the smooth exact penalty functions as merit functions.

As we said, the technique for avoiding the Maratos effect is to introduce the second-order corrections, as shown by Coleman and Conn (1982a, 1982b), Fletcher (1982), Mayne and Polak (1982), Fukushima (1986), etc.

Suppose that the SQP algorithm has computed a step  $d_k$  as solution of the subproblem (15.12). If this step yields an increase in the merit function  $P(x_k, \sigma)$ , then a possible cause is that the linear approximation of the constraints given by (15.12b) and (15.12c) is not accurate enough. To overcome this situation, a natural solution is to re-solve (15.12) with the linear terms  $c_i(x_k) + \nabla c_i(x_k)^T d$  replaced by the quadratic approximations

$$c_i(x_k) + \nabla c_i(x_k)^T d + \frac{1}{2} d^T \nabla^2 c_i(x_k) d. \quad (15.43)$$

However, the resulting subproblem with quadratic constraints is too difficult to solve. Instead, the following procedure is used. The constraints are evaluated in the new point  $x_k + d_k$  and then we make use of their approximation by Taylor's theorem

$$c_i(x_k + d_k) \cong c_i(x_k) + \nabla c_i(x_k)^T d_k + \frac{1}{2} d_k^T \nabla^2 c_i(x_k) d_k. \quad (15.44)$$

Assuming that the second-order step  $d$ , which is not known, will not be too different from  $d_k$ , then the last term in (15.43) can be approximated as

$$d^T \nabla^2 c_i(x_k) d = d_k^T \nabla^2 c_i(x_k) d_k. \quad (15.45)$$

Therefore, by making this substitution in (15.43) and by using the approximation (15.44), the following *second-order correction subproblem* is obtained:

$$\min_{d \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2 L(x_k, \lambda_k) d \quad (15.46a)$$

subject to

$$\nabla c_i(x_k)^T d + \left( c_i(x_k + d_k) - \nabla c_i(x_k)^T d_k \right) = 0, \quad i \in E, \quad (15.46b)$$

$$\nabla c_i(x_k)^T d + \left( c_i(x_k + d_k) - \nabla c_i(x_k)^T d_k \right) \geq 0, \quad i \in I. \quad (15.46c)$$

Let  $\hat{d}_k$  be the solution of (15.46). Observe that the second-order correction step implies the evaluation of the constraints in the point  $x_k + d_k$ , i.e.,  $c_i(x_k + d_k)$ ,  $i \in E \cup I$ , which can be a laborious activity. Therefore, a strategy is to use this correction step only if the increase in the merit function is concomitant with an increase in the constraint norm. When the step  $d_k$  is generated by the SQP algorithm based on (15.12), then, near a solution satisfying the second-order sufficient conditions, the algorithm based on (15.46) takes either the full step  $d_k$  or the corrected step  $d_k + \hat{d}_k$ .

## 15.5 The Line-Search SQP Algorithm

Taking into consideration the above developments, we can see that there is a large variety of line-search SQP algorithms that differ in many respects, such as the way in which the Hessian approximation is computed (limited-memory BFGS that is suitable for large-scale problems), the step acceptance mechanism using different techniques to avoid the Maratos effect, and the use of the merit functions or of the filters. In the following, we present a *practical* line-search SQP algorithm for solving the general nonlinear optimization problem (15.11). To keep the description as simple as possible, the algorithm below does not include details to ensure the feasibility of the subproblem or the second-order correction mechanisms. Instead, the algorithm is simply obtained by solving the subproblem (15.12). Assume that the quadratic subproblem (15.12) is convex. Therefore, it can be solved by means of the active-set methods for quadratic programming described in Chap. 13.

### Algorithm 15.4 Line-search SQP algorithm

- |     |   |
|-----|---|
| 1.  | Choose an initial pair $(x_0, \lambda_0)$ , as well as the parameters $\eta \in (0, 1/2)$ and $\tau \in (0, 1)$ . Evaluate $f(x_0)$ , $\nabla f(x_0)$ , $c(x_0)$ , and $A(x_0)$ . If a quasi-Newton approximation of the Hessian of the Lagrange function is used, then choose an initial $n \times n$ symmetric and positive definite approximation $B_0$ ; otherwise, compute $\nabla_{xx}^2 L(x_0, \lambda_0)$ . Set $k = 0$ |
| 2.  | Test a criterion for stopping the iterations  |
| 3.  | Compute the search direction $d_k$ as solution of the quadratic subproblem (15.12). Let $\hat{\lambda}$ be the corresponding Lagrange multipliers   |
| 4.  | Set $d_\lambda = \hat{\lambda} - \lambda_k$   |
| 5.  | Choose $\sigma_k$ to satisfy (15.39) with $\mu = 1$ . Set $\alpha_k = 1$  |
| 6.  | Inner while loop. Select $\eta \in (0, 1)$ and $\tau_\alpha \in (0, \tau]$ . While $P(x_k + \alpha_k d_k, \sigma_k) > P(x_k, \sigma_k) + \eta \alpha_k P'_{d_k}(x_k, \sigma_k)$ , reset $\alpha_k = \tau_\alpha \alpha_k$ ; otherwise, go to step 7   |
| 7.  | Set $x_{k+1} = x_k + \alpha_k d_k$ and $\lambda_{k+1} = \lambda_k + \alpha_k d_\lambda$   |
| 8.  | Evaluate $f(x_{k+1})$ , $\nabla f(x_{k+1})$ , $c(x_{k+1})$ , and $A(x_{k+1})$ . If the case, evaluate $\nabla_{xx}^2 L(x_{k+1}, \lambda_{k+1})$   |
| 9.  | If a quasi-Newton approximation method is used, then set<br>$s_k = \alpha_k d_k$ and $y_k = \nabla_x L(x_{k+1}, \lambda_{k+1}) - \nabla_x L(x_k, \lambda_{k+1})$<br>and obtain $B_{k+1}$ by updating $B_k$ using a quasi-Newton method (BFGS)   |
| 10. | Set $k = k + 1$ and go to step 2  |



An efficient implementation of Algorithm 15.4 is to use the *warm start* procedure. The working set for each quadratic programming subproblem is initialized with the working set of the previous SQP iteration. In step 6 of the algorithm, where the inner while loop is implemented instead of a merit function, a filter may be used. In step 9, the limited memory BFGS updating approach can be used, especially for large-scale problems.

## 15.6 The Trust-Region SQP Algorithm

The easiest way to formulate a trust-region SQP method is to add a trust-region constraint to the subproblem (15.12), as

$$\min_{d \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d \quad (15.47a)$$

subject to

$$\nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in E, \quad (15.47b)$$

$$\nabla c_i(x_k)^T d + c_i(x_k) \geq 0, \quad i \in I. \quad (15.47c)$$

$$\|d\| \leq \Delta_k. \quad (15.47d)$$

Even if the constraints (15.47b) and (15.47c) are compatible, it is often possible that the subproblem (15.47) may not always have a solution because of the trust-region constraint (15.47d). To solve this possible conflict between the linear constraints (15.47b), (15.47c), and (15.47d), it is not appropriate to increase  $\Delta_k$  until the set of the steps  $d$  satisfying the linear constraints has intersected the trust-region. A more appropriate viewpoint is not to exactly satisfy the linearized constraints at every step; instead, we try to improve the feasibility of these constraints at each step and to exactly satisfy them only if the trust-region constraint permits it. This point of view can be implemented in three ways: by relaxation methods, penalty methods, and filter methods.

### A Relaxation Method for the Equality-Constrained Optimization

Consider the equality-constrained nonlinear optimization problem (15.1). At the iterate  $x_k$ , the SQP step is computed by solving the subproblem

$$\min_{d \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d \quad (15.48a)$$

subject to

$$A(x_k)d + h(x_k) = r_k, \quad (15.48b)$$

$$\|d\|_2 \leq \Delta_k, \quad (15.48c)$$

where  $r_k \in \mathbb{R}^m$  is the *relaxation vector*. The relaxation vector is selected as the smallest vector such that (15.48b) and (15.48c) are consistent for some reduced value of the trust-region radius  $\Delta_k$ . To achieve this, first solve the subproblem

$$\min_y \|A(x_k)y + h(x_k)\|_2^2 \quad (15.49a)$$

subject to

$$\|y\|_2 \leq 0.8\Delta_k. \quad (15.49b)$$

Let  $y_k$  be the solution of (15.49). Define the relaxation vector as

$$r_k = A(x_k)y_k + h(x_k). \quad (15.50)$$

In the following, the step  $d_k$  is computed by (15.48) and then the new iterate is defined as  $x_{k+1} = x_k + d_k$ . The new multiplier estimate  $\lambda_{k+1}$  is computed as

$$\lambda_{k+1} = [A(x_k)A(x_k)^T]^{-1}A(x_k)\nabla f(x_k). \quad (15.51)$$

Now, for computing an approximate solution  $d_k$  of (15.48), one method is the projected conjugate gradient. The idea is to apply this algorithm to (15.48a, and 15.48b), monitoring the satisfaction of the trust-region constraint (15.48c) and stopping the algorithm if the boundary of this region has been reached or if a negative curvature has been detected. A merit function that takes into consideration all these aspects is the nonsmooth  $l_2$  function  $P_2(x, \sigma) = f(x) + \sigma\|h(x)\|_2$ . This merit function is modeled by

$$q_\sigma(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2}d^T \nabla_{xx}^2 L(x_k, \lambda_k)d + \sigma m(d), \quad (15.52)$$

where

$$m(d) = \|h(x_k) + A(x_k)d\|_2. \quad (15.53)$$

The penalty parameter  $\sigma$  is selected large enough so as to satisfy (15.38). The acceptability of the step  $d_k$  is monitored by the ratio

$$\rho_k = \frac{\text{ared}_k}{\text{pred}_k} = \frac{P_2(x_k, \sigma) - P_2(x_k + d_k, \sigma)}{q_\sigma(0) - q_\sigma(d_k)}. \quad (15.54)$$

A description of the trust-region SQP algorithm for solving the equality-constrained nonlinear optimization problem (15.1),  $\min\{f(x) : h(x) = 0\}$ , is as follows Omojokun (1989), Lalee, Nocedal, and Plantenga (1998), Nocedal and Wright (2006).

### Algorithm 15.5 Trust-region SQP algorithm

1.	Choose the initial point $x_0$ and the initial trust-region radius $\Delta_0 > 0$ . Choose the constants $\varepsilon > 0$ and $\eta, \gamma \in (0, 1)$ . Set $k = 0$
2.	Evaluate $f(x_k)$ , $\nabla f(x_k)$ , $h(x_k)$ , and $A(x_k)$
3.	Compute the Lagrange multipliers: $\lambda_{k+1} = [A(x_k)A(x_k)^T]^{-1}A(x_k)\nabla f(x_k)$
4.	If $\ \nabla f(x_k) - A(x_k)^T \lambda_k\ _\infty < \varepsilon$ and $\ h(x_k)\ _\infty < \varepsilon$ , then stop. $x_k$ is the solution of the problem
5.	Solve the subproblem (15.49) with respect to $y_k$
6.	Compute the relaxation parameter $r_k = A(x_k)y_k + h(x_k)$
7.	Compute $\nabla_{xx}^2 L(x_k, \lambda_k)$ or a quasi-Newton approximation of it
8.	Compute $d_k$ as solution of the quadratic programming subproblem (15.48)
9.	Choose $\sigma$ , large enough to satisfy (15.39)
10.	Compute the ratio $\rho_k = \text{ared}_k/\text{pred}_k$
11.	If $\rho_k > \eta$ , then set $x_{k+1} = x_k + d_k$ , choose $\Delta_{k+1}$ to satisfy $\Delta_{k+1} \geq \Delta_k$ and go to step 12. If $\rho_k \leq \eta$ , then set $x_{k+1} = x_k$ , compute $\Delta_{k+1} = \gamma\ d_k\ $ and go to step 12
12.	Set $k = k + 1$ and continue with step 2



We can simply note that a second-order correction can be introduced to avoid the Maratos effect. The main computational cost of this algorithm is given by the projected conjugate gradient iteration for solving the quadratic programming subproblem (15.48).

### Sequential $l_1$ Quadratic Programming for the Equality and Inequality Constraints

In this method, the linearized constraints (15.47b) and (15.47c) are placed into the objective of the quadratic program as an  $l_1$  penalty term. Thus, the following subproblem is obtained:

$$\begin{aligned} \min_{d} q_\sigma(d) &\triangleq f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d \\ &+ \sigma \sum_{i \in E} |c_i(x_k) + \nabla c_i(x_k)^T d| + \sigma \sum_{i \in I} [c_i(x_k) + \nabla c_i(x_k)^T d]^- \\ \text{subject to} \quad & \|d\|_\infty \leq \Delta_k, \end{aligned} \quad (15.55)$$

where  $\sigma$  is the penalty parameter and  $[y]^- = \max \{0, -y\}$ . When introducing the slack variables, this problem is reformulated as

$$\begin{aligned} \min_{d, v, w, t} & f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d + \sigma \sum_{i \in E} (v_i + w_i) + \sigma \sum_{i \in I} t_i \\ \text{subject to} \quad & \nabla c_i(x_k)^T d + c_i(x_k) = v_i - w_i, \quad i \in E, \\ & \nabla c_i(x_k)^T d + c_i(x_k) \geq -t_i, \quad i \in I, \\ & v, w, t \geq 0, \\ & \|d\|_\infty \leq \Delta_k. \end{aligned} \quad (15.56)$$

The constraints of this problem are always consistent. Moreover, since the trust-region has been defined by using the  $l_\infty$  norm, (15.56) is a smooth quadratic program which can be solved by means of a quadratic programming algorithm.

To determine the acceptance of the step, the algorithm uses the  $l_1$  merit function

$$P(x, \sigma) = f(x) + \sigma \sum_{i \in E} |c_i(x)| + \sigma \sum_{i \in I} [c_i(x)]^- . \quad (15.57)$$

Observe that  $q_\sigma$  defined in (15.55) can be viewed as a model of the merit function (15.57) at  $x_k$ , in which we approximate each constraint function  $c_i$  by its linearization and in which  $f$  is replaced by a quadratic function whose curvature term includes information from both the objective and the constraints.

After computing the step  $d_k$  as solution of (15.56), from (15.54) the ratio  $\rho_k$  is computed by using this time the above merit function and defining  $q_\sigma$  by (15.55). The step is accepted or rejected according to the standard trust-region rules, as described in Algorithm 15.5. Clearly, a second-order correction step can be introduced in order to prevent the Maratos effect.

## 15.7 Sequential Linear-Quadratic Programming (SLQP)

As we have already seen, the SQP methods presented above require the solution of a general quadratic programming problem with equality and inequality constraints at each iteration. However, this is a difficult task, especially when the second derivative information is incorporated into the algorithm. This approach imposes a limit on the size of the problem that can be solved in practice by these methods.

The sequential linear-quadratic programming method, which we are going to briefly describe, tries to overcome this situation by computing the step in two phases. In the first one, a linear programming problem (LP) is formed and solved to identify a working set  $W$ . The second one consists of an equality-constrained quadratic programming phase in which the constraints in the working set  $W$  are imposed as equalities. The total step is a combination of the steps obtained in the linear programming and equality-constrained phases.

In the LP phase, the following problem is solved:

$$\min_{d \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T d \quad (15.58a)$$

subject to

$$\nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in E, \quad (15.58b)$$

$$\nabla c_i(x_k)^T d + c_i(x_k) \geq 0, \quad i \in I, \quad (15.58c)$$

$$\|d\|_\infty \leq \Delta_k^{LP}. \quad (15.58d)$$

Observe that this linear programming problem differs from the standard SQP subproblem (15.47) only in the fact that the second-order term  $d^T \nabla_{xx}^2 L(x_k, \lambda_k) d$  has been omitted and that a  $\ell_\infty$  norm has been used to define the trust-region. Since the constraints of (15.58) may be inconsistent, instead of (15.58), a  $\ell_1$  penalty reformulation is defined as

$$\begin{aligned} \min_d l_\sigma(d) &\triangleq f(x_k) + \nabla f(x_k)^T d \\ &+ \sigma \sum_{i \in E} |c_i(x_k) + \nabla c_i(x_k)^T d| + \sigma \sum_{i \in I} [c_i(x_k) + \nabla c_i(x_k)^T d]^- \\ \text{subject to} \\ \|d\|_\infty &\leq \Delta_k^{LP}. \end{aligned} \quad (15.59)$$

Introducing the slack variables, (15.59) can be reformulated as a linear programming problem. Let  $d^{LP}$  be the solution of (15.59). From this solution, the following explicit estimate of the optimal active-set can be obtained:

$$A_k(d^{LP}) = \left\{ i \in E : c_i(x_k) + \nabla c_i(x_k)^T d^{LP} = 0 \right\} \cup \left\{ i \in I : c_i(x_k) + \nabla c_i(x_k)^T d^{LP} = 0 \right\}.$$

Similarly, the set  $V_k$  of the violated constrained is defined as

$$V_k(d^{LP}) = \left\{ i \in E : c_i(x_k) + \nabla c_i(x_k)^T d^{LP} \neq 0 \right\} \cup \left\{ i \in I : c_i(x_k) + \nabla c_i(x_k)^T d^{LP} < 0 \right\}.$$

The working set  $W_k$  is defined as some linearly independent subset of the active-set  $A_k(d^{LP})$ . To ensure the progress of the algorithm on the penalty function (15.57), the Cauchy step is defined as

$$d^C = \alpha^{LP} d^{LP}, \quad (15.60)$$

where  $\alpha^{LP} \in [0, 1]$  is the stepsize that provides sufficient decrease in the model  $q_\sigma$  defined in (15.55). With the working set  $W_k$  an equality-constrained quadratic programming is solved, where the constraints in  $W_k$  are considered as equalities, ignoring all the others. Thus, the following subproblem is obtained:

$$\begin{aligned} & \min df(x_k) + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d + \left( \nabla f(x_k) + \sigma_k \sum_{i \in V_k} \gamma_i \nabla c_i(x_k) \right)^T d \\ & \text{subject to} \\ & \quad \nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in E \cap W_k, \\ & \quad \nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in I \cap W_k, \\ & \quad \|d\|_2 \leq \Delta_k, \end{aligned} \quad (15.61)$$

where the scalars  $\gamma_i$  represent the algebraic sign of the  $i$ -th violated constraint. Observe that the trust-region constraint in (15.61) is spherical and  $\Delta_k$  is different from the trust-region radius  $\Delta_k^{LP}$  used in (15.59). Let  $d^Q$  be the solution of (15.61). The total step  $d_k$  of the SLQP method is computed as

$$d_k = d^C + \alpha^Q (d^Q - d^C),$$

where  $\alpha^Q \in [0, 1]$  is the step size that approximately minimizes the model  $q_\sigma$  defined in (15.55). Byrd, Gould, Nocedal, and Waltz (2004a) argue that the choice of the radius  $\Delta_{k+1}^{LP}$  for the LP phase is more delicate, since it influences our guess of the optimal active-set. The value of  $\Delta_{k+1}^{LP}$  should be set to be a little larger than the total step  $d_k$ , subject to some other restrictions. The Lagrange multiplier estimates  $\lambda_k$  used in the evaluation of the Hessian  $\nabla_{xx}^2 L(x_k, \lambda_k)$  are computed as in (15.51) using the working set  $W_k$  and modified so that  $\lambda_i \geq 0, i \in I$ . The advantage of using SLQP is that well-established algorithms for solving large-scale versions of linear programming and equality quadratic subproblems are readily available.

## 15.8 A SQP Algorithm for Large-Scale-Constrained Optimization (SNOPT)

The algorithm described in this section, implements the above discussions as well as some other ideas. It was elaborated by Gill, Murray, and Saunders (2002, 2005) to solve nonlinear optimization problems of the following form:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ & \text{subject to} \\ & \quad l \leq \begin{bmatrix} x \\ c(x) \\ Ax \end{bmatrix} \leq u, \end{aligned} \quad (15.62)$$

where  $f$  is the objective function (linear or nonlinear),  $c(x)$  is the vector of nonlinear constraints,  $A$  is the matrix corresponding to the linear constraints, and  $l$  and  $u$  are bounds on variables and on constraints. Suppose that the nonlinear functions are smooth, their first derivatives are available and the Jacobian of the constraints is a sparse matrix.

SNOPT is the implementation of a particular SQP algorithm that exploits the sparsity in the constraint Jacobian and maintains a limited-memory quasi-Newton approximation  $B_k$  to the Hessian

of the Lagrange function. To update  $B_k$  in the presence of the negative curvature, a new method is used. The quadratic programming subproblems are solved using an inertia-controlling reduced-Hessian active-set method (SQOPT). Other features include the treatment of infeasible nonlinear constraints using elastic programming, the use of a well-conditioned nonorthogonal basis for the null space of the quadratic programming working set, early termination of the quadratic programming subproblems, and the finite-difference estimates of the missing gradients. The method used by the quadratic programming solver SQOPT is based on solving a sequence of linear systems involving the reduced Hessian  $Z^T B_k Z$ , where  $Z$  is implicitly defined by using the sparse LU factorization.

In the following, we present a technique for treating the infeasible constraints and then we show how SNOPT is working for solving the general nonlinear optimization problem with the inequality constraints  $\min\{f(x) : c(x) \geq 0\}$ . Finally, some particularizations of these developments will be applied to problems with linear and nonlinear constraints (including simple bounds) given in (15.62).

### Infeasible Constraints

In SNOPT, the infeasible constraints are treated by means of the  $l_1$  penalty functions. At the very beginning, the infeasible linear constraints are identified by solving a problem of the following form:

$$\begin{aligned} & \min_{x,v,w} e^T(v + w) \\ & \text{subject to} \\ & l \leq \begin{bmatrix} x \\ Ax - v + w \end{bmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \end{aligned} \tag{15.63}$$

where  $e$  is a vector with all the components equal to one. The problem (15.63) is to minimize the one-norm of the general linear constraint violations subject to the simple bounds. This problem is often called *elastic programming* (Conn, 1976), (Bartels, 1980). If the linear constraints are infeasible ( $v \neq 0$  and  $w \neq 0$ ), then SNOPT stops without computing the nonlinear functions. Otherwise, all the subsequent iterates satisfy the linear constraints. SNOPT then solves (15.62) by using the quadratic programming subproblem based on the linearization of the nonlinear constraints. If a quadratic programming subproblem is infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters the *nonlinear elastic mode* and solves the problem

$$\begin{aligned} & \min_{x,v,w} f(x) + \gamma e^T(v + w) \\ & \text{subject to} \\ & l \leq \begin{bmatrix} x \\ c(x) - v + w \\ Ax \end{bmatrix} \leq u, \\ & v \geq 0, \quad w \geq 0, \end{aligned} \tag{15.64}$$

where  $f(x) + \gamma e^T(v + w)$  is known as the composite objective function and the penalty parameter  $\gamma$  ( $\gamma \geq 0$ ) may take a finite sequence of increasing values. If (15.62) has a feasible solution and  $\gamma$  is large enough, then the solutions of (15.62) and (15.64) are identical. On the other hand, if (15.62) has no feasible solution, then (15.64) will tend to determine an infeasible point if  $\gamma$  is sufficiently large.

### The SQP Iteration for the General Inequality Nonlinear Optimization

The SQP approach was popularized mainly by Biggs (1972), Han (1976), Powell (1977, 1978b), and further developed by Schittkowski (1986) (NLPQL), Gill, Murray, Saunders and Wright (1986) (NPSOL), and Spellucci (1998) (DONLP). Under mild conditions, these solvers typically find a local optimum for the continuous general nonlinear optimization problems from arbitrary starting points, and they require relatively few evaluations of the functions defining the problem and their gradients compared, for example, to MINOS (Murtagh, & Saunders, 1978, 1995). For large-scale problems with *equality* constraints, the method of Lalee, Nocedal, and Plantenga (1998) uses either the exact Hessian of the Lagrangian or a limited-memory quasi-Newton approximation. The methods by Biegler, Nocedal, and Schmid (1995) use the reduced Hessian method. They maintain a dense approximation to the reduced Hessian based on quasi-Newton updates. For large problems with general *inequality* constraints as in (15.49), the SQP methods were proposed by Eldersveld (1991) (LSSQP), Fletcher, and Leyffer (1998, 2002) (filterSQP). In the following, we shall present the SNOPT algorithm based on the works of Gill, Murray, and Saunders (2005). Consider the problem

$$\begin{aligned} & \min_x f(x) \\ & \text{subject to} \\ & c(x) \geq 0, \end{aligned} \quad (15.65)$$

where  $x \in \mathbb{R}^n$ ,  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  are twice continuously differentiable. Suppose that a KKT point  $(x^*, \lambda^*)$  exists, where the first-order optimality conditions are satisfied

$$c(x^*) \geq 0, \quad \lambda^* \geq 0, \quad c(x^*)^T \lambda^* = 0, \quad \nabla c(x^*)^T \lambda^* = \nabla f(x^*). \quad (15.66)$$

The sequential quadratic programming method involves *major* and *minor iterations*. The major iterations generate a sequence of iterates  $(x_k, \lambda_k)$  that converge to  $(x^*, \lambda^*)$ . At each iteration, a quadratic programming subproblem is solved to generate a search direction towards the next iteration  $(x_{k+1}, \lambda_{k+1})$ . Solving the quadratic subproblems is based on an iterative procedure involving the so-called minor iterations.

Let  $x_k$  and  $\lambda_k$  be estimates of  $x^*$  and  $\lambda^*$ . Gill, Murray, and Saunders (2005) use the following modified Lagrange function associated to (15.65):

$$L(x, x_k, \lambda_k) = f(x) - \lambda_k^T d_L(x, x_k), \quad (15.67)$$

which is defined in terms of *constraint linearization* and *departure from linearity*

$$c_L(x, x_k) = c(x_k) + \nabla c(x_k)(x - x_k), \quad (15.68a)$$

$$d_L(x, x_k) = c(x) - c_L(x, x_k). \quad (15.68b)$$

This idea is taken from the MINOS algorithm. In fact, SNOPT is an improved extension of MINOS (Saunders, 2015a, 2015b). The first and the second derivatives of the modified Lagrangian are as follows:

$$\nabla L(x, x_k, \lambda_k) = \nabla f(x) - (\nabla c(x) - \nabla c(x_k))^T \lambda_k, \quad (15.69a)$$

$$\nabla^2 L(x, x_k, \lambda_k) = \nabla^2 f(x) - \sum_{i=1}^m (\lambda_k)_i \nabla^2 c_i(x). \quad (15.69b)$$

Observe that for  $x = x_k$  we have  $L(x_k, x_k, \lambda_k) = f(x_k)$  and  $\nabla L(x_k, x_k, \lambda_k) = \nabla f(x_k)$ .

Now, let

$$L_q(x, x_k, \lambda_k) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 L(x_k, x_k, \lambda_k)(x - x_k)$$

be the quadratic approximation of  $L$  at  $x_k$ . If  $(x_k, \lambda_k) = (x^*, \lambda^*)$ , then the optimality conditions for the quadratic program

$$\begin{aligned} & \min_x L_q(x, x_k, \lambda_k) \\ & \text{subject to} \\ & c_L(x, x_k) \geq 0 \end{aligned} \quad (15.70)$$

are identical to those of the original problem (15.65). This suggests that if  $B_k$  is an approximation to  $\nabla^2 L$  at the point  $(x_k, \lambda_k)$ , then an improved estimate of the solution may be obtained from  $(\hat{x}_k, \hat{\lambda}_k)$  as the solution of the following quadratic programming subproblem:

$$\begin{aligned} & \min_x f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k(x - x_k) \\ & \text{subject to} \\ & c(x_k) + \nabla c(x_k)(x - x_k) \geq 0. \end{aligned} \quad (15.71)$$

The optimality conditions of (15.71) are

$$\begin{aligned} c(x_k) + \nabla c(x_k)(\hat{x}_k - x_k) &= \hat{s}_k, & \hat{\lambda}_k \geq 0, & \hat{s}_k \geq 0, \\ \nabla f(x_k) + B_k(\hat{x}_k - x_k) &= \nabla c(x_k)^T \hat{\lambda}_k, & \hat{\lambda}_k^T \hat{s}_k &= 0, \end{aligned}$$

where  $\hat{s}_k$  is the vector of the slack variables for the linearized constraints. The triplet  $(\hat{x}_k, \hat{\lambda}_k, \hat{s}_k)$  is an estimate of  $(x^*, \lambda^*, s^*)$ , where the slack variables  $s^*$  satisfy  $c(x^*) - s^* = 0$ ,  $s^* \geq 0$ . The vector  $\hat{s}_k$  is needed in the line-search procedure.

**The working set matrix** The working set is important for both major and minor iterations and represents the set of constraints that are binding at a solution (see Definition 11.12). If the subproblem (15.71) has been solved, then the algorithm which solves this problem returns an independent set of constraints that are active at the solution. This is the *optimal working set* for the subproblem (15.71). The same constraint indices define the working set of (15.65) and of the next subproblem (15.71) at the iteration  $k + 1$ . The corresponding gradients of these constraints form the rows of the *working set matrix*  $W_k$ , an  $n_Y \times n$  full rank submatrix of the Jacobian  $\nabla c(x_k)$ . Let  $Z_k$  be an  $n \times n_Z$  full rank matrix which is a basis of the null space of  $W_k$ . Then  $W_k Z_k = 0$  and  $n_Z = n - n_Y$ . In SNOPT,  $Z_k$  is defined by the LU factorization of  $W_k$ . The matrix  $Z_k$  is useful both in theoretical developments and in the construction of the *reduced Hessian*  $Z_k^T B_k Z_k$  and of the *reduced gradient*  $Z_k^T \nabla f(x_k)$ .

**The merit function and the line-search** Let  $(x_k, \lambda_k, s_k)$  be the current solution of (15.71). As soon as we have a solution  $(\hat{x}_k, \hat{\lambda}_k, \hat{s}_k)$  for (15.71), a new estimation for the solution of (15.65) can be computed by means of a linear search from  $(x_k, \lambda_k, s_k)$  towards the optimal solution of (15.71). The purpose of the linear search is a sufficient decrease of a merit function based on the augmented Lagrangian

$$M_\sigma(x, \lambda, s) = f(x) - \lambda^T(c(x) - s) + \frac{1}{2} \sum_{i=1}^m \sigma_i (c_i(x) - s_i)^2, \quad (15.72)$$

where  $\sigma = [\sigma_1, \dots, \sigma_m]^T \in \mathbb{R}^m$  is the vector of the penalty parameters. For the stepsize  $\alpha \in (0, 1]$ , let  $v(\alpha)$  be the following line:

$$v(\alpha) = \begin{bmatrix} x_k \\ \lambda_k \\ s_k \end{bmatrix} + \alpha \begin{bmatrix} \hat{x}_k - x_k \\ \hat{\lambda}_k - \lambda_k \\ \hat{s}_k - s_k \end{bmatrix}$$

and  $\varphi_\sigma(\alpha) = M_\sigma(v(\alpha))$ , which represents  $M_\sigma$  as a function by  $\alpha$ . Observe that  $\varphi'_\sigma(0)$  is the directional derivative of the merit function at the base point  $\alpha = 0$  for a given  $\sigma$ . To determine the values of the penalty parameters, Gill, Murray, and Saunders (2005) recommend the following procedure. Let  $\sigma^*$  be the solution of the following least-squares problem:

$$\begin{aligned} & \min \|\sigma\|^2 \\ & \text{subject to} \\ & \varphi'_\sigma(0) = -\frac{1}{2} (\hat{x}_k - x_k)^T B_k (\hat{x}_k - x_k), \quad \sigma \geq 0. \end{aligned} \quad (15.73)$$

The solution of this problem can be obtained analytically and it can be shown that for any  $\sigma \geq \sigma^*$ ,  $\varphi'_\sigma(0) \leq -\frac{1}{2} (\hat{x}_k - x_k)^T B_k (\hat{x}_k - x_k)$ , (Eldersveld, 1991), (Gill, Murray, Saunders, and Wright, 1992).

It is important to allow the penalty parameters to decrease during the early major iterations. The reduction scheme involves a parameter  $\Delta_\sigma \geq 1$ . Let  $\bar{\sigma}$  be the value of the penalty parameter at the start of the iterate  $k$ . Then define the new parameter  $\bar{\sigma}_i$  as the geometric mean of  $\sigma_i$  and  $\sigma_i^*$  as long as this mean is sufficiently positive and not too close to  $\sigma_i$ :

$$\bar{\sigma}_i = \max \{\sigma_i^*, \hat{\sigma}_i\}, \quad (15.74)$$

where

$$\hat{\sigma}_i = \begin{cases} \sigma_i, & \text{if } \sigma_i < 4(\sigma_i^* + \Delta_\sigma), \\ \sqrt{\sigma_i(\sigma_i^* + \Delta_\sigma)}, & \text{otherwise.} \end{cases}$$

Initially, for  $k = 0$ ,  $\Delta_\sigma = 1$ . This value of  $\Delta_\sigma$  can be increased by a factor of two in order to ensure a positive value for  $\bar{\sigma}_i$  and not too close to  $\sigma_i$ .

Now, considering  $\sigma = \bar{\sigma}$  in the merit function (15.72), a safeguarded line-search is used to find  $\alpha_{k+1}$  that reduces  $M_\sigma$  to give the next solution estimate  $v(\alpha_{k+1}) = (x_{k+1}, \lambda_{k+1}, s_{k+1})$ . With this, prior to the solution of (15.71) at the iteration  $k + 1$ ,  $s_{k+1}$  is redefined to minimize the merit function as a function of  $s$ , (Gill, Murray, Saunders, and Wright, 1986), (Eldersveld, 1991).

**Bounding the constraint violation** In the line-search, the following condition is enforced for some vector  $b > 0$ :

$$c(x_k + \alpha_k(\hat{x}_k - x_k)) \geq -b.$$

SNOPT uses  $b_i = \tau_v \max \{1, -c_i(x_0)\}$ , where  $\tau_v$  is a constant, for example,  $\tau_v = 10$ . This defines a region in which the objective function is expected to be defined and bounded below. If  $\alpha_k = 0$ , then

the objective is considered to be “unbounded” in the expanded region. In this case, the elastic program is introduced.

**The approximate Hessian** As soon as the line-search has been finished, the change in  $x$  and the gradient of the modified Lagrangian are defined as

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla L(x_{k+1}, x_k, \lambda) - \nabla L(x_k, x_k, \lambda),$$

for some vector  $\lambda$ . Then, an estimate of the curvature of the modified Lagrangian along  $s_k$  is incorporated in the BFGS quasi-Newton update

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}. \quad (15.75)$$

When  $B_k$  is positive definite, then  $B_{k+1}$  is positive definite if and only if the curvature  $y_k^T s_k$  is positive. From (15.67) observe that

$$\begin{aligned} y_k &= \nabla L(x_{k+1}, x_k, \lambda_{k+1}) - \nabla L(x_k, x_k, \lambda_{k+1}) \\ &= \nabla f(x_{k+1}) - \nabla f(x_k) - (\nabla c(x_{k+1}) - \nabla c(x_k))^T \lambda_{k+1}. \end{aligned}$$

**Maintaining the positive definiteness of the Hessian** Since the Hessian of the modified Lagrangian need not be positive definite at a local minimizer, the approximate curvature  $y_k^T s_k$  can be negative or very small at points arbitrarily close to  $(x^*, \lambda^*)$ . The curvature is considered not sufficiently positive if

$$y_k^T s_k < \rho_k, \quad \rho_k = \alpha_k (1 - \eta) d_k^T B_k d_k, \quad (15.76)$$

where  $\eta \in (0, 1)$  is a given constant ( $0 < \eta < 1$ ) and  $d_k = \hat{x}_k - x_k$  is the search direction defined by the quadratic programming subproblem (15.71). In such cases, if the problem has nonlinear constraints, two attempts are made to modify the update: the first is modifying  $s_k$  and  $y_k$ , the second is modifying only  $y_k$ . If neither modification provides a sufficiently positive approximate curvature, then no update is made.

**The first modification** Define a new point  $z_k$  and evaluate the nonlinear functions in order to obtain the new values for  $s_k = x_{k+1} - z_k$  and  $y_k = \nabla L(x_{k+1}, x_k, \lambda_{k+1}) - \nabla L(z_k, x_k, \lambda_{k+1})$ . The point  $z_k$  is chosen as the first feasible point  $\bar{x}_k$  of (15.71). The search direction may be written as

$$d_k = (\bar{x}_k - x_k) + (\hat{x}_k - \bar{x}_k) \equiv d_R + d_N.$$

Consider  $z_k = x_k + \alpha_k d_R$ , thus obtaining  $s_k = \alpha_k d_N$  and

$$y_k^T s_k = \alpha_k y_k^T d_N \cong \alpha_k^2 d_N^T \nabla^2 L(x_k, x_k, \lambda_k) d_N,$$

so that  $y_k^T s_k$  approximates the curvature along  $d_N$ . If  $W_k$  is the final working set of (15.71) at  $x_k$ , it is also the working set at  $\bar{x}_k$ , then  $W_k d_N = 0$ . Therefore,  $y_k^T s_k$  approximates the curvature for the reduced Hessian, which must be positive semi-definite at the minimizer of (15.65).

**The second modification** If  $(x_k, \lambda_k)$  is not close to  $(x^*, \lambda^*)$ , then the modified approximate curvature  $y_k^T s_k$  may not be sufficiently positive. Hence, the second modification may be necessary as follows. Choose  $\Delta y_k$  so that  $(y_k + \Delta y_k)^T s_k = \rho_k$  and redefine  $y_k$  as  $y_k + \Delta y_k$ . This approach was suggested by Powell (1978b), who proposed to redefine  $y_k$  as a linear combination of  $y_k$  and  $B_k s_k$ .

To get  $\Delta y_k$ , consider the augmented modified Lagrangian

$$L_A(x, x_k, \lambda_k) = f(x) - \lambda_k^T d_L(x, x_k) + \frac{1}{2} d_L(x, x_k)^T \Omega d_L(x, x_k), \quad (15.77)$$

where  $\Omega = \text{diag}(\omega_i)$ ,  $\omega_i \geq 0$ ,  $i = 1, \dots, m$ , is a matrix of the parameters to be determined. The perturbation

$$\Delta y_k = (\nabla c(x_{k+1}) - \nabla c(x_k))^T \Omega d_L(x_{k+1}, x_k)$$

is equivalent to redefining the gradient difference as

$$y_k = \nabla L_A(x_{k+1}, x_k, \lambda_{k+1}) - \nabla L_A(x_k, x_k, \lambda_{k+1}). \quad (15.78)$$

The elements  $\omega_i$ ,  $i = 1, \dots, m$ , are determined by solving the linearly constrained least-squares problem

$$\begin{aligned} & \min_{\omega} \|\omega\|_2^2 \\ & \text{subject to} \\ & a^T \omega = \beta, \quad \omega \geq 0, \end{aligned} \quad (15.79)$$

where  $\beta = \rho_k - y_k^T s_k$  and  $a_i = v_i w_i$ ,  $i = 1, \dots, m$ , with  $v = (\nabla c(x_{k+1}) - \nabla c(x_k))^T s_k$  and  $w = d_L(x_{k+1}, x_k)$ . The idea is to choose the smallest  $\omega_i$  that increases  $y_k^T s_k$  to  $\rho_k$  (see (15.76)). If there is no solution for the problem (15.79) or if  $\|\omega\|$  is very large, no update of the Hessian is made. It is worth mentioning that the second modification is not required in the neighborhood of a solution because  $\nabla^2 L_A$  converges to  $\nabla^2 L$  when  $x \rightarrow x^*$  and the first modification will have already been successful. The second modification is related to updating an approximation of the Hessian of the Lagrangian suggested by Han (1976) and Tapia (1974).

**Convergence tests** A point  $(x, \lambda)$  is a satisfactory solution for (15.65) if it satisfies the first-order optimality conditions (15.66) within a certain tolerance. In SNOPT, two sufficiently small constants  $\tau_P$  and  $\tau_D$  are used, which define the quantities  $\tau_x = \tau_P(1 + \|x\|_\infty)$  and  $\tau_\lambda = \tau_D(1 + \|\lambda\|_\infty)$ . The SQP algorithm terminates if

$$c_i(x) \geq -\tau_x, \quad \lambda_i \geq -\tau_\lambda, \quad c_i(x)\lambda_i \leq \tau_\lambda, \quad |d_j| \leq \tau_\lambda, \quad (15.80)$$

where  $d = \nabla f(x) - \nabla c(x)^T \lambda$ .

These conditions cannot be satisfied if (15.65) is infeasible. However, in these situations, the problem (15.65) can be transformed in its elastic variant

$$\begin{aligned} & \min_{x,v} f(x) + \gamma e^T v \\ & \text{subject to} \\ & c(x) + v \geq 0, \quad v \geq 0, \end{aligned} \quad (15.81)$$

where  $\gamma$  takes the increasing values  $\{\gamma_l\}$  up to some maximum value. The first-order optimality conditions for (15.81) include

$$0 \leq \lambda_i \leq \gamma, \quad (c_i(x) + v_i)\lambda_i = 0, \quad v_i(\gamma - \lambda_i) = 0. \quad (15.82)$$

The fact that  $\|\lambda^*\|_\infty \leq \gamma$  at the solution of (15.81) leads us to initiate the elastic mode if  $\|\lambda_k\|_\infty$  exceeds some value, let us say  $\gamma_1$ , or if (15.71) is infeasible. SNOPT uses

$$\gamma_1 \triangleq \gamma_0 \|\nabla f(x_{k_1})\|_\infty, \quad \gamma_l = \gamma_1 10^{l(l-1)/2}, \quad l = 2, 3, \dots, \quad (15.83)$$

where  $\gamma_0 = 10^4$  and  $x_{k_1}$  is the iterate at which  $\gamma$  is first needed.

### The Quadratic Programming Solver SQOPT

We now return to our problem given by (15.62) considered by SNOPT. The form of this problem requires solving (at every major iteration) a quadratic programming subproblem with  $f(x)$  replaced by a convex quadratic function and with  $c(x)$  replaced by its current linearization in the current point

$$\begin{aligned} & \min_x f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k(x - x_k) \\ & \text{subject to} \\ & l \leq \begin{bmatrix} x \\ c(x_k) + \nabla c(x_k)(x - x_k) \\ Ax \end{bmatrix} \leq u. \end{aligned} \quad (15.84)$$

In SNOPT, this subproblem is solved by the package SQOPT (Gill, Murray, and Saunders, 1997), which employs a two-phase active-set algorithm and implicitly implements the elastic programming when necessary. The Hessian  $B_k$  may be positive semi-definite and is defined by a routine for forming the products  $B_k v$  for a given vector  $v$ .

At each minor iteration, when (15.84) is solved, the *active-set method* determines a search direction  $d$  satisfying the KKT system

$$\begin{bmatrix} B_k & W^T \\ W & 0 \end{bmatrix} \begin{bmatrix} d \\ y \end{bmatrix} = - \begin{bmatrix} g_q \\ 0 \end{bmatrix}, \quad (15.85)$$

where  $W$  is the current working set matrix and  $g_q$  is the gradient of the objective function from (15.84). SQOPT implements several null-space methods, as described in (Gill, Murray, and Saunders, 2005).

**The null-space approach** One way to obtain a solution for the linear system (15.85) is to solve the reduced Hessian system

$$Z^T B_k Z d_Z = -Z^T g_q, \quad d = Z d_Z, \quad (15.86)$$

where  $Z$  is a basis for the null space of  $W$ . SQOPT maintains  $Z$  in the reduced-gradient form as in MINOS. The idea is to use the sparse LU factors of a square nonsingular matrix  $B$ , called basis, whose columns change as the working set  $W$  changes

$$W = \begin{bmatrix} B & S & N \\ 0 & 0 & I \end{bmatrix} P, \quad Z = P^T \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}, \quad (15.87)$$

where  $P$  is a permutation matrix such that the selected basis  $B$  is nonsingular. The variables associated with  $B$  and  $S$  are called *basic* and *superbasic*, respectively. The other variables are called *nonbasic*. The number of the superbasic variables  $n_Z$  is exactly the column dimension of  $S$  and  $Z$ . The products  $Zv$  and  $Z^T g$  are obtained by solving the corresponding systems with  $B$  or  $B^T$ , respectively. If  $n_Z$  is

small, then SQOPT uses the dense Cholesky factorization  $Z^T B_k Z = R^T R$ . As the major iterations converge, the quadratic programming subproblems require fewer changes to their working set, and with a warm start they are eventually solved in one or two minor iterations. Hence, the work required by SQOPT becomes dominant by the computation of the reduced Hessian  $Z^T B_k Z$  and its factor  $R$ . For this reason, SQOPT can optionally maintain a quasi-Newton approximation  $Z^T B_k Z \cong R^T R$ , as in MINOS.

**The conjugate gradient approach** By construction, the Hessians  $B_k$  of (15.84) are positive definite or positive semi-definite. Therefore, the conjugate gradient method is a natural tool for solving the very large systems (15.85). SQOPT includes a conjugate gradient option for finding approximate solutions to

$$(Z^T B_k Z + \delta^2 I) d_Z = -Z^T g_q, \quad (15.88)$$

where  $\delta \cong 10^{-3}$  is a small regularization parameter to allow for the singular  $Z^T B_k Z$ . When  $Z$  has many columns, then it is possible for the conjugate gradient methods to require many iterations to get a useful approximation to  $d_Z$ . The conjugate gradient methods require some sort of preconditioning, which is much dependent on the system to be solved. However, in SQOPT no preconditioner is used. The explanation is that, when looking at (15.88), we see that both  $B_k$  and  $Z^T Z$  have similar structures: “a diagonal matrix plus a matrix of small rank.”

**The initial point** To use a good starting point  $x_0$ , the algorithm SQOPT is applied to one of the proximal-point problems for initialization

$$\min_x \left\{ \|\bar{x} - \bar{x}_0\|_1 : \bar{l} \leq \begin{bmatrix} x \\ Ax \end{bmatrix} \leq \bar{u} \right\} \quad (15.89)$$

or

$$\min_x \left\{ \|\bar{x} - \bar{x}_0\|_2^2 : \bar{l} \leq \begin{bmatrix} x \\ Ax \end{bmatrix} \leq \bar{u} \right\} \quad (15.90)$$

where  $\bar{l}$  and  $\bar{u}$  are the corresponding bounds for the linear constraints and the simple bounds from (15.84).  $\bar{x}$  and  $\bar{x}_0$  correspond to the nonlinear variables in  $x$  and  $x_0$ . In practice, (15.89) is preferred because it is linear and can use SQOPT’s implicit elastic bounds. The solution of (15.89) (or (15.90)) defines a new starting point  $x_0$  for the SQP iteration.

SNOPT, as described by Gill, Murray, and Saunders (2005), is a complex algorithm with plenty of details which are not presented here. They refer to the following: null-space computation, choices for multipliers, large-scale Hessians in case of problems with few nonlinear variables, dense Hessians, limited-memory procedure to the update Hessians, elastic bounds, inertia control, unbounded quadratic programming subproblems, basis handling in SQOPT, threshold pivoting in the LU factorization, basis repair in the singular case, basis repair in the rectangular case, undefined functions, early termination of quadratic programming subproblems, and linearly constrained problems. Suppose that a starting point  $(x_0, \lambda_0)$  is available and that the reduced-Hessian quadratic programming solver SQOPT is being used. The main steps of SNOPT are as follows.

**Algorithm 15.6 SNOPT—Gill, Murray, and Saunders**

1.	Apply the quadratic programming solver to the problem (15.89) (or (15.90)) to find a point close to $x_0$ satisfying the linear constraints and the simple bounds on variables. If (15.89) is infeasible, then the problem (15.62) is infeasible. Otherwise, a working set matrix $W_0$ is obtained. Set $k = 0$ and evaluate the functions and the gradients at $x_0$ . Initialize the penalty parameters $\sigma_i = 0$ , $i = 1, \dots, m$
2.	Factorize $W_k$
3.	Define $s_k$ to minimize the merit function (15.72) as a function of the slacks $s$
4.	Determine $\bar{x}_k$ as a feasible solution for (15.84). This is an intermediate point for the SQOPT solver, which also provides a working set matrix $\bar{W}_k$ as well as its null space $\bar{Z}_k$ . If no feasible point exists, initiate the elastic mode and restart the SQOPT solver
5.	Compute the reduced Hessian $\bar{Z}_k^T B_k \bar{Z}_k$ and compute its Cholesky factor
6.	Using SQOPT, continue solving the quadratic programming subproblem (15.84) to find $(\hat{x}_k, \hat{\lambda}_k)$ . Observe that in step 4, only a feasible solution was obtained for (15.84). SQOPT provides a working set $\hat{W}_k$ as well as its null-space matrix $\hat{Z}_k$
	If the elastic mode has not been initiated, but $\ \hat{\lambda}_k\ _\infty$ is “large,” then enter the elastic mode and restart the SQOPT solver If the subproblem (15.84) is unbounded and $x_k$ satisfies the nonlinear constraints, the problem (15.62) is unbounded ( <i>fis</i> unbounded below in the feasible region). Otherwise, if (15.84) is unbounded, go to step 8
7.	If the solution $(x_k, \lambda_k)$ satisfies the convergence tests for (15.62) analogously to (15.80), then this is the optimal solution of the problem; stop If the convergence tests similar to (15.80) are satisfied for the subproblem (15.81), then go to step 8. Otherwise, go to step 9
8.	If the elastic mode of (15.84) has not been initiated, then consider the elastic variant of the problem and repeat step 6. Otherwise, if $\gamma$ has not reached its maximum value, increase $\gamma$ and repeat step 6. Otherwise, declare the problem (15.62) infeasible
9.	Update the penalty parameters as in (15.74)
10.	Find a stepsize $\alpha_k$ that gives a sufficient reduction in the merit function (15.72). Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$ and $\lambda_{k+1} = \lambda_k + \alpha_k(\hat{\lambda}_k - \lambda_k)$ . Evaluate the functions and the gradients at the new point $x_{k+1}$
11.	Compute $s_k = x_{k+1} - x_k$ , $y_k = \nabla L(x_{k+1}, x_k, \lambda_{k+1}) - \nabla L(x_k, x_k, \lambda_{k+1})$ . If $y_k^T s_k < \rho_k$ , where $\rho_k$ is computed as in (15.76), then re-compute $s_k$ and $y_k$ with $x_k$ redefined as $x_k + \alpha_k(\bar{x}_k - x_k)$ . (This requires an extra evaluation of the derivatives.) If necessary, increase $y_k^T s_k$ (if possible) by increasing $y_k$ with $\Delta y_k$ and by redefining the difference of the gradients of the augmented Lagrangian (see (15.77))
12.	If $y_k^T s_k \geq \rho_k$ , then apply the BFGS formula to update $B_k$ by using the pair $(B_k s_k, y_k)$
13.	Using $\hat{W}_k$ , redefine $W_k$ , set $k = k + 1$ and go to step 2 ◆

For both linearly and nonlinearly constrained problems, SNOPT applies a sparse sequential quadratic programming method (Gill, Murray, and Saunders, 2005), using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for the stepsize computation is an augmented Lagrangian, as in the dense SQP solver NPSOL (Gill, Murray, Saunders, & Wright, 1986). The computational effort is in steps 2 and 5, where the matrix  $W_k$  is factorized, and the reduced Hessian  $\bar{Z}_k^T B_k \bar{Z}_k$  is formed, and Cholesky factorized.

**Numerical study—SNOPT: Solving Applications from the LACOP Collection**

Table 15.1 contains the performances of SNOPT for solving 12 applications of nonlinear optimization from the LACOP collection presented in Appendix C.

**Table 15.1** Performances of SNOPT for solving 12 applications from the LACOP collection. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	# <i>itM</i>	# <i>itm</i>	# <i>nf</i>	# <i>nr</i>	# <i>s</i>	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	19	49	54	0	7	0.01	-47.761090
ALKI	10	3	8	16	47	28	28	1	0.01	-1768.8070
PREC	8	0	6	53	119	202	202	4	0.01	3.951163
PPSE	9	6	0	9	17	20	20	1	0.01	5055.01167
MSP3	13	0	15	71	141	367	367	1	0.02	97.5875581
MSP5	16	0	21	54	152	106	106	2	0.02	174.910868
POOL	34	20	0	5	36	8	8	4	0.01	2569.8000
TRAFO	6	0	2	30	38	36	36	4	0.01	135.075962
LATHE	10	1	14	33	86	55	55	0	0.01	-4430.0882
DES	150	50	0	104	204	130	130	100	0.16	1055.18231
CSTC	303	200	0	10	111	15	15	100	0.03	3.48007467
DIFF	396	324	0	0	60	0	0	0	0.01	0.0

**Table 15.2** Performances of SNOPT for solving 6 applications from the LACOP collection. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	# <i>itM</i>	# <i>itm</i>	# <i>nf</i>	# <i>s</i>	<i>cpu</i>	<i>vfo</i>
HANG	1002	501	0	59	1032	93	499	3.37	5.0685783
	2002	1001	0	60	2028	125	999	18.80	5.0685126
FLOW	1163	735	0	0	457	2	0	0.01	0.0
FLOWO	1556	1005	0	0	329	2	0	0.01	0.0
POL	4004	3000	0	57	1562	114	0	0.92	13.480213
	6004	4500	0	63	2272	128	0	1.86	13.469429
	8004	6000	0	63	2992	125	0	3.06	13.638381
	10004	7500	0	73	3392	145	0	4.51	13.210731
CAT	3003	2000	0	11	272	1183	0	2.28	-0.048055
	6003	4000	0	18	536	2430	0	8.75	-0.048052
	9003	6000	0	23	796	3703	0	20.47	-0.048047
CONT	2505	2000	0	3	696	5	0	0.05	1.01323
	5005	4000	0	3	1394	5	2	0.11	1.00592
	7505	6000	0	3	2094	5	0	0.27	1.00456
	10005	8000	0	3	1424	5	0	0.44	1.00407

In Tables 15.1 and 15.2, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #*itM* = the number of major iterations, #*itm* = the number of minor iterations, #*nf* = the number of evaluations of the objective function, #*nr* = the number of evaluations of the constraints, #*s* = the number of superbasic variables, *cpu* = the CPU computing time to solve the problem (seconds), and *vfo* = the value of the objective function at the optimal point.

Table 15.2 presents the performances of SNOPT for solving 6 large-scale nonlinear optimization applications from the LACOP collection of different dimensions.

SNOPT uses many concepts defined in MINOS. A comparison between these two algorithms shows the importance of the sequential quadratic programming implemented in SNOPT versus the minimization of the augmented Lagrangian subject to the linearization of the constraints used in MINOS. For all these 15 large-scale numerical experiments shown in Tables 14.12 (MINOS) and 15.2 (SNOPT), respectively, Table 15.3 shows the total number of major iterations (#*itMt*), the total

**Table 15.3** Comparison: MINOS versus SNOPT for solving 15 large-scale applications from the LACOP collection. Large-scale nonlinear optimization applications

	#itMt	#itmt	cput
MINOS	548	27271	143.80
SNOPT	439	21276	64.91

number of minor iterations (#itmt), and the total CPU computing time (cput) in seconds, to get a solution for all these applications.

## 15.9 A SQP Algorithm with Successive Error Restoration (NLPQLP)

Let us consider the general nonlinear optimization problem with equality and inequality constraints

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_i(x) = 0, \quad i = 1, \dots, m_e, \\ & c_i(x) \geq 0, \quad i = m_e + 1, \dots, m, \end{aligned} \tag{15.91}$$

where it is assumed that the functions  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , and  $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , are twice continuously differentiable. Also, assume that

$$X = \{x \in \mathbb{R}^n : c_i(x) = 0, \quad i = 1, \dots, m_e, \quad c_i(x) \geq 0, \quad i = m_e + 1, \dots, m\} \neq \emptyset.$$

For solving this problem, we shall present the sequential quadratic programming with the successive error restoration algorithm NLPQLP, elaborated by Schittkowski (1986, 2002, 2005, 2009, and 2010). Since the sequential quadratic programming is sensitive to the accuracy by which the partial derivatives are provided, NLPQLP is stabilized by a non-monotone line-search and by internal and external restarts in case of errors when computing the search direction due to inaccurate derivatives.

### Search Direction

The basic idea is to formulate and solve a sequence of quadratic programming subproblems obtained by the linearization of the constraints and by approximating the Lagrangian function

$$L(x, u) = f(x) - \sum_{i=1}^m u_i c_i(x) \tag{15.92}$$

quadratically, where  $x \in \mathbb{R}^n$  is the vector of the primal variables and  $u = [u_1, \dots, u_m]^T \in \mathbb{R}^m$  is the vector of the dual variables or the vector of the Lagrange multipliers.

Let  $v_k \in \mathbb{R}^m$  be an approximation of the Lagrange multipliers. To formulate the quadratic programming subproblem at the current iterate  $x_k$ , the gradients  $\nabla f(x_k)$  and  $\nabla c_i(x_k)$ ,  $i = 1, \dots, m$ , as well as an approximation  $B_k \in \mathbb{R}^{n \times n}$  of the Hessian to the Lagrangian are computed. Then the quadratic programming subproblem is formed and solved

$$\begin{aligned}
& \min \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d \\
& \text{subject to} \\
& \quad \nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i = 1, \dots, m_e, \\
& \quad \nabla c_i(x_k)^T d + c_i(x_k) \geq 0, \quad i = m_e + 1, \dots, m.
\end{aligned} \tag{15.93}$$

Let  $d_k$  be the optimal solution and  $u_k$  the corresponding Lagrange multipliers of (15.93). A new iteration is computed as

$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \alpha_k \begin{bmatrix} d_k \\ u_k - v_k \end{bmatrix}, \tag{15.94}$$

where  $\alpha_k \in (0, 1]$  is the stepsize.

Although we can guarantee that the matrix  $B_k$  is positive definite, it is quite possible for (15.93) not to be solvable because its constraints are inconsistent. To remedy this situation, Schittkowski (1986) suggested introducing an additional variable  $\delta \in \mathbb{R}$ , thus transforming (15.93) into a modified quadratic subproblem with  $(n + 1)$  variables, with consistent constraints.

Another numerical difficulty with (15.93) is that the gradients of all the constraints must be re-evaluated at each iteration. But if  $x_k$  is close to the solution, then the computation of the gradients of the inactive constraints is redundant. To avoid this redundant computational effort, define the set  $I \triangleq \{i : m_e < i \leq m\}$ , and, given a constant  $\varepsilon > 0$  small enough, define the sets

$$\bar{I}_1^{(k)} = \left\{ i \in I : c_i(x_k) \leq \varepsilon \quad \text{or} \quad v_k^{(i)} > 0 \right\}, \quad \bar{I}_2^{(k)} = I \setminus \bar{I}_1^{(k)}, \tag{15.95}$$

where  $v_k = [v_k^{(1)}, \dots, v_k^{(m)}]^T$ . With these developments, at each iteration the following modified quadratic programming subproblem is solved:

$$\begin{aligned}
& \min_{d \in \mathbb{R}^n, \delta \in [0,1]} \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d + \frac{1}{2} \rho_k \delta^2 \\
& \text{subject to} \\
& \quad \nabla c_i(x_k)^T d + (1 - \delta)c_i(x_k) = 0, \quad i = 1, \dots, m_e, \\
& \quad \nabla c_i(x_k)^T d + (1 - \delta)c_i(x_k) \geq 0, \quad i \in \bar{I}_1^{(k)}, \\
& \quad \nabla c_i(x_{\kappa(k,i)})^T d + c_i(x_k) \geq 0, \quad i \in \bar{I}_2^{(k)}.
\end{aligned} \tag{15.96}$$

In (15.96),  $\kappa(k, i) \leq k$  represents the previous iterations where the corresponding gradient has been last evaluated. We start with  $\bar{I}_1^{(0)} = I$  and  $\bar{I}_2^{(0)} = \emptyset$ . At the subsequent iterations, only the constraint gradients belonging to the active-set  $\bar{I}_1^{(k)}$  are reevaluated. The remaining rows of the Jacobian matrix remain filled with the previously computed gradients. In (15.96),  $\rho_k > 0$  is a parameter sufficiently small.

Let  $(d_k, \delta_k, u_k)$  be the solution of (15.96), where  $u_k$  is the vector of the multipliers and  $\delta_k$  is the additional variable introduced to prevent the inconsistency of the linear constraints. If the linear independency of the constraints qualification (LICQ) is satisfied (see Remark 11.2), then  $\delta_k < 1$ . The matrix  $B_k$  is a positive definite approximation of the Hessian to the Lagrange function. For the global convergence analysis, any choice of  $B_k$  is appropriate as long as its eigenvalues are bounded away from zero. However, to guarantee a superlinear convergence rate,  $B_k$  is updated by the BFGS quasi-Newton method modified by a stabilization procedure to ensure the positive definiteness. The penalty parameter  $\rho_k$  is required to reduce the perturbation of the search direction by the additional variable  $\delta$  as much as possible. A suitable choice is given in Schittkowski (1983).

### Stepsize Computation

The stepsize parameter  $\alpha_k$  is required in (15.94) in order to enforce the global convergence of the SQP method when the initial point  $x_0$  is arbitrarily selected and  $v_0 = 0$ ,  $B_0 = I$ . The stepsize  $\alpha_k$  should satisfy at least a sufficient decrease condition of a merit function  $\Phi_\sigma(\alpha)$  given by

$$\Phi_\sigma(\alpha) \triangleq P_\sigma\left(\begin{bmatrix} x \\ v \end{bmatrix} + \alpha \begin{bmatrix} d \\ u - v \end{bmatrix}\right), \quad (15.97)$$

where  $P_\sigma(x, v)$  is a suitable penalty function, for example, the augmented Lagrange function

$$P_\sigma(x, v) = f(x) - \sum_{i \in J} \left( v_i c_i(x) - \frac{1}{2} \sigma_i c_i(x)^2 \right) - \frac{1}{2} \sum_{i \in K} \frac{v_i^2}{\sigma_i}, \quad (15.98)$$

where the sets  $J$  and  $K$  are defined as (Schittkowski, 1983)

$$J = \{1, \dots, m_e\} \cup \{i : m_e < i \leq m, c_i(x) \leq v_i/\sigma_i\} \quad \text{and} \quad K = \{1, \dots, m\} \setminus J.$$

The objective function is penalized as soon as an iterate has left the feasible region. The penalty parameters  $\sigma_i$ ,  $i = 1, \dots, m$ , control the degree of the constraint violation. To guarantee a descent direction of the merit function,  $\sigma_i$  are chosen such that

$$\Phi'_{\sigma_k}(0) = \nabla P_{\sigma_k}(x_k, v_k)^T \begin{bmatrix} d_k \\ u_k - v_k \end{bmatrix} < 0. \quad (15.99)$$

Usually, the stepsize  $\alpha_k$  is computed by means of the Armijo rule, i.e., a sufficient descent condition of the merit function (15.98) which guarantees the convergence to a stationary point. However, to take into consideration the curvature of the merit function, we need some kind of compromise between a polynomial interpolation, typically a quadratic one, and a reduction of the stepsize by a given factor until a stopping criterion has been reached (Schittkowski, 2010). The determination of the stepsize  $\alpha_k$  is very important in the economy of the NLPQLP algorithm. The purpose is not to have too many function calls in the procedure for the  $\alpha_k$  computation. Moreover, the behavior of the merit function becomes irregular in the case of the constrained optimization because of the very steep slopes at the border caused by the large penalty terms. The implementation of a procedure for the  $\alpha_k$  computation is more complex if the linear constraints and the simple bounds on variables are to be satisfied during the line-search. Since  $\Phi_\sigma(0)$ ,  $\Phi'_\sigma(0)$  and  $\Phi_\sigma(\alpha_i)$  are known ( $\alpha_i$  is the current value of the stepsize), then a procedure based on the quadratic interpolation for the  $\alpha_{k+1}$  computation is as in the following fragment of the algorithm:

#### Algorithm 15.7 Linear search in NLPQLP

- |    |  |
|----|--|
| 1. | Choose the parameters $0 < \beta < 1$ and $0 < \mu < 1/2$ . Consider $\sigma = \sigma_k$ . Set $\alpha_0 = 1$ and $i = 0$                  |
| 2. | If $\Phi_\sigma(\alpha_i) < \Phi_\sigma(0) + \mu \alpha_i \Phi'_\sigma(0)$ , then stop; otherwise, go to step 3                            |
| 3. | Compute: $\bar{\alpha}_i = \frac{1}{2} \frac{\alpha_i \Phi'_\sigma(0)}{\alpha_i \Phi'_\sigma(0) - \Phi_\sigma(\alpha_i) + \Phi_\sigma(0)}$ |
| 4. | Let $\alpha_i = \max \{\beta \alpha_i, \bar{\alpha}_i\}$ . Set $i = i + 1$ and go to step 2  |

◆

The algorithm is well known, see for example (Powell, 1978b).  $\bar{\alpha}_i$  is the minimizer of the quadratic interpolation polynomial. For checking the termination, the Armijo descent property is used. The computation of  $\alpha_i$  in step 4 of the algorithm as  $\alpha_i = \max \{\beta \alpha_i, \bar{\alpha}_i\}$  is required to avoid the irregular

values, since the minimizer of the quadratic interpolation polynomial can be outside the feasible region  $[0, 1]$ . Schittkowski mentions that Algorithm 15.7 is equipped with some safeguards to prevent, for example, the violation of the bounds. In Algorithm 15.7, it is supposed that  $\Phi_\sigma(1)$  is known before calling it. Moreover, the algorithm stops if sufficient descent is not observed after a certain number of iterations. If the tested stepsize falls below the machine precision, then the merit function cannot decrease further on. Algorithm 15.7 could break down because of too many iterations. In this case, Schittkowski recommends proceeding from a descent direction of the merit function when  $\Phi'_\sigma(0)$  is extremely small. In order to avoid the interruption of the whole iteration process of NLPQLP, another criterion is used instead of the criterion in step 2 of Algorithm 15.7. The idea is to repeat the line-search with another stopping criterion. Instead of testing (15.98), the stepsize  $\alpha_k$  is accepted as soon as the inequality

$$\Phi_{\sigma_k}(\alpha_k) \leq \max_{k-t(k) \leq j \leq k} \{\Phi_{\sigma_j}(0)\} + \mu \alpha_k \Phi'_{\sigma_k}(0), \quad (15.100)$$

has been satisfied, where  $t(k) = \min\{k, \bar{t}\}$ , and  $\bar{t}$  is a given parameter. Thus, an increase of the reference value  $\Phi_{\sigma_j}$  is allowed, i.e., an increase of the merit function value. To implement (15.100), it is necessary to store the previous values of the merit function from the first  $t(k)$  iterations. The nonmonotone search goes back to Grippo, Lampariello, and Lucidi (1986) and was extended to the trust-region methods for the constrained optimization by Bonnans, Panier, Tits, and Zhou (1992), Deng, Xiao, and Zhou (1993), Grippo, Lampariello, and Lucidi (1989, 1991), Raydan (1993, 1997), Panier and Tits (1991), Toint (1996, 1997).

### Updating the Matrix $B_k$

To avoid the calculation of the second derivatives and to obtain a final superlinear convergence rate, the standard approach used in NLPQLP is to update  $B_k$  by the quasi-Newton BFGS formula

$$B_{k+1} = B_k + \frac{q_k q_k^T}{s_k^T q_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}, \quad (15.101)$$

where  $q_k = \nabla_x L(x_{k+1}, u_k) - \nabla_x L(x_k, u_k)$  and  $s_k = x_{k+1} - x_k$ . The algorithm is equipped with special safeguards guarantee that ensures the curvature condition  $s_k^T q_k > 0$ . In this case, if  $B_0$  is positive definite, then all the matrices  $B_k$  generated by (15.101) remain positive definite along the iterations. The algorithm contains a scaling and a restart procedure to replace an actual  $B_k$  by  $\gamma_k I$  before performing the update (15.101), where  $\gamma_k = s_k^T q_k / s_k^T s_k$ , (Liu, & Nocedal, 1989).

The main steps of the NLPQLP algorithm are as follows.

### Algorithm 15.8 NLPQLP—Schittkowski

- |    |  |
|----|--|
| 1. | Choose an initial point $x_0$ , an initial approximation to the Hessian of the Lagrange function $B_0$ and the tolerances $\epsilon_1$ and $\epsilon_2$ . Set $k = 0$  |
| 2. | Compute $f(x_k)$ , $c(x_k)$ , $\nabla f(x_k)$ , and $\nabla c(x_k)$  |
| 3. | Compute $d_k$ as solution of the quadratic programming subproblem (15.96)  |
| 4. | Compute the stepsize by means of Algorithm 15.7  |
| 5. | If the KKT optimality conditions are satisfied with the tolerance $\epsilon_1$ and the norm of the gradient of the Lagrange function is smaller than the tolerance $\epsilon_2$ , then stop; otherwise, go to step 6 |
| 6. | Using (15.101), update $B_k$   |
| 7. | Compute a new Cholesky factorization of $B_k$  |
| 8. | Set $k = k + 1$ and continue with step 2   |

◆

**Table 15.4** Performances of NLPQLP for solving 8 applications from the LACOP collection. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#iter	#nf	#ng	#nq	KKT	cpu	vfo
ELCH	10	3	0	21	30	21	21	0.11e-8	0.03	-47.76109
ALKI	10	3	8	53	69	53	53	0.13e-11	0.03	-1768.807
PREC	8	0	6	19	20	19	19	0.36e-8	0.02	3.9511635
PPSE	9	6	0	9	10	9	9	0.21e-8	0.03	5055.0118
MSP3	13	0	15	146	264	146	146	0.62e-8	0.04	79.631357
POOL	34	20	0	22	22	22	22	0.64e-13	0.07	2785.800
TRAFO	6	0	2	19	19	19	19	0.19e-10	0.03	135.0759
LATHE	10	1	14	19	19	19	19	0.27e-9	0.03	-4430.087

In Table 15.4, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #iter = the number of iterations to get a local optimal solution, #nf = the number of evaluations of the functions of the problem, #ng = the number of the evaluations of the gradients of the functions, #nq = the number of the quadratic programming subproblems solved to get a local optimal solution of the problem, KKT = the norm of the KKT conditions,  $cpu$  = the CPU computing time for solving the problem (seconds), and  $vfo$  = the value of the objective function at the solution.

**Table 15.5** Performances of NLPQLP for solving the ETA-MACRO application

<i>n</i>	<i>me</i>	<i>mc</i>	#iter	#nf	#ng	#nq	KKT	cpu	<i>vfo</i>
60	1	41	31	31	31	31	0.74e-8	0.17	5.3700593
120	1	81	103	103	103	13	0.83e-8	3.69	7.9532086
300	1	201	823	823	823	823	0.86e-8	454.90	9.3301684

### Numerical Study—NLPQLP: Solving Applications from the LACOP Collection

Table 15.4 presents the performances of NLPQLP for solving 8 nonlinear optimization applications from the LACOP collection, described in Appendix C.

#### Application P1: ETA-MACRO

ETA-MACRO simulates a market economy through a dynamic nonlinear optimization process. To describe the production relationships within this economy, two dynamic submodels are incorporated: (a) ETA, a process analysis for the energy technology assessment and (b) a macroeconomic growth model providing for the substitution between capital, labor, and energy inputs. The application is described in (Manne, 1977) (see also (Bergman, 1988), (Murtagh, & Saunders, 1995, pp. 98)). Table 15.5 contains the performances of NLPQLP for solving the application ETA-MACRO for a different number  $n$  of variables.

---

## 15.10 Active-Set Sequential Linear-Quadratic Programming (KNITRO/ACTIVE)

KNITRO represents one of the most elaborated algorithms (and Fortran packages) for solving general large-scale nonlinear optimization problems (Byrd, Gould, Nocedal, & Waltz, 2004a). This is characterized by great flexibility and robustness, integrating two very powerful and complementary algorithmic approaches for nonlinear optimization: the *active-set sequential linear-quadratic approach* and the *interior-point approach*. KNITRO includes a number of much-studied algorithms for linear algebra, very carefully implemented in computing programs, able to solve a large variety of

nonlinear optimization problems like special cases of unconstrained optimization, systems of nonlinear equations, least square problems, and linear and nonlinear programming problems.

KNITRO has two algorithmic options. The first one, known as KNITRO/ACTIVE, uses a new active-set method based on the sequential linear-quadratic programming (SLQP) and the projected conjugate gradient iteration. The second, known as KNITRO/INTERIOR, uses the interior-point methods in two implementations: KNITRO/INTERIOR-CG, in which the algorithmic step is computed by means of an iterative conjugate gradient method, and KNITRO/INTERIOR-DIRECT, in which the step is (usually) computed via a direct factorization of the corresponding linear systems. These two approaches, KNITRO/ACTIVE and KNITRO/INTERIOR, communicate by the so-called *crossover* technique. This crossover procedure, used for the first time by Megiddo (1989) in linear programming, is implemented internally by switching to the ACTIVE algorithm after the INTERIOR-DIRECT or the INTERIOR-CG algorithm has solved the problem to a requested tolerance.

The difference between these two approaches is as follows. The active-set sequential linear-quadratic programming algorithm is similar in nature to a sequential quadratic programming method, but it uses linear programming subproblems to estimate the active-set at each iteration. This active-set approach may be useful when a good initial point can be provided. On the other hand, in the interior-point methods, also known as barrier methods, the nonlinear programming problem is replaced by a sequence of barrier subproblems controlled by a barrier parameter. The algorithm uses the trust-region and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem, then decreases the barrier parameter and repeats the process until the problem has been solved to the desired accuracy.

In this chapter, we discuss only KNITRO/ACTIVE. This algorithm is described in (Byrd, Gould, Nocedal, & Waltz, 2004a), and its global convergence is presented in (Byrd, Gould, Nocedal, & Waltz, 2002).

KNITRO/INTERIOR will be described in Chap. 17, where the crossover technique is also presented.

Let us consider the general nonlinear optimization problem

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_E(x) = 0, \\ & c_I(x) \geq 0, \end{aligned} \tag{15.102}$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_E: \mathbb{R}^n \rightarrow \mathbb{R}^l$  and  $c_I: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are twice continuously differentiable functions. Define:  $E \triangleq \{1, \dots, l\}$  and  $I \triangleq \{1, \dots, m\}$ .

The active-set method implemented in KNITRO/ACTIVE does not follow a SQP approach because the cost of solving linear-constrained quadratic programming subproblems generally imposes limitations on the size of problems that can be solved in practice. In addition, the incorporation of the second derivative information in the SQP methods has proved to be difficult in practice. Instead, in KNITRO/ACTIVE a sequential linear-quadratic programming (SLQP) method is used (see Sect. 15.7). SLQP computes a step in two phases. Firstly, a linear programming problem (LP) is solved in order to identify a working set  $W$ . This is followed by an equality-constrained quadratic programming phase, in which the constraints in the working set  $W$  are imposed as equalities. The total step of the algorithm is a combination of the previous steps obtained in linear programming and equality-constrained quadratic programming phases.

### KNITRO/ACTIVE Algorithm

To achieve progress on both feasibility and optimality, the algorithm is designed to reduce the  $l_1$  penalty function

$$P(x, \sigma) \triangleq f(x) + \sigma \sum_{i \in E} |c_i(x)| + \sigma \sum_{i \in I} \max \{0, -c_i(x)\}, \quad (15.103)$$

where  $c_i, i \in E$ , represents the components of vector  $c_E$ , and  $c_i, i \in I$ , represents the components of  $c_I$ . The penalty parameter  $\sigma$  is chosen by an adaptive procedure described later.

In the LP phase, given an estimate  $x_k$  of the solution of the nonlinear optimization problem (15.102), the following linear programming problem is formed and solved:

$$\min_d \nabla f(x_k)^T d \quad (15.104a)$$

$$\text{subject to} \quad \nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in E, \quad (15.104b)$$

$$\nabla c_i(x_k)^T d + c_i(x_k) \geq 0, \quad i \in I, \quad (15.104c)$$

$$\|d\|_\infty \leq \Delta_k^{LP}, \quad (15.104d)$$

where  $\Delta_k^{LP} > 0$  is the trust-region radius. Observe that (15.104) differs from the classical subproblem used in the SQP methods only in that (15.104) does not contain the quadratic term  $\frac{1}{2}d^T B_k d$ , where  $B_k$  is an approximation to the Hessian of the Lagrangian of (15.102). Now, since the constraints of (15.104) may be inconsistent, instead of it we solve the  $l_1$  penalty reformulation of (15.104) as

$$\begin{aligned} \min_d l_\sigma(d) \triangleq & \nabla f(x_k)^T d + \sigma_k \sum_{i \in E} |\nabla c_i(x_k)^T d + c_i(x_k)| \\ & + \sigma_k \sum_{i \in I} \max \left\{ 0, -\nabla c_i(x_k)^T d - c_i(x_k) \right\} \end{aligned} \quad (15.105)$$

subject to

$$\|d\|_\infty \leq \Delta_k^{LP}.$$

The solution of this linear program, denoted by  $d^{LP}$ , is computed by the simplex algorithm. Based on this solution, we define the working set  $W$  as a linearly independent subset of the active-set at the LP solution  $p^{LP}$

$$\begin{aligned} A(d^{LP}) = & \left\{ i \in E : \nabla c_i(x_k)^T d^{LP} + c_i(x_k) = 0 \right\} \\ & \cup \left\{ i \in I : \nabla c_i(x_k)^T d^{LP} + c_i(x_k) = 0 \right\}. \end{aligned} \quad (15.106)$$

Similarly, define the set  $V$  of the violated constraints as

$$V(d^{LP}) = \left\{ i \in E : \nabla c_i(x_k)^T d^{LP} + c_i(x_k) \neq 0 \right\} \cup \left\{ i \in I : \nabla c_i(x_k)^T d^{LP} + c_i(x_k) < 0 \right\}. \quad (15.107)$$

To ensure the progress of the algorithm subject to the penalty function  $P(x, \sigma)$  defined in (15.103), define the Cauchy step

$$d^C = \alpha^{LP} d^{LP}, \quad (15.108)$$

where  $\alpha^{LP} \in [0, 1]$  is the stepsize that provides sufficient decrease in the following piece-wise quadratic model of the penalty function  $P(x, \sigma)$

$$q_k(d) = l_\sigma(d) + \frac{1}{2} d^T B(x_k, \lambda_k) d. \quad (15.109)$$

In (15.109),  $l_\sigma(d)$  is defined by (15.105) and  $B(x_k, \lambda_k)$  is the Hessian of the Lagrange function or an approximation of it.

Given the working set  $W_k$ , the following equality-constrained quadratic program in the variable  $d$  is solved, treating the constraints in  $W_k$  as equalities and ignoring all the other constraints

$$\min \frac{1}{2} d^T B(x_k, \lambda_k) d + \left( \nabla f(x_k) + \sigma_k \sum_{i \in V} \gamma_i \nabla c_i(x_k) \right)^T d \quad (15.110a)$$

subject to

$$\nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in E \cap W_k, \quad (15.110b)$$

$$\nabla c_i(x_k)^T d + c_i(x_k) = 0, \quad i \in I \cap W_k, \quad (15.110c)$$

$$\|d\|_2 \leq \Delta_k, \quad (15.110d)$$

where  $\gamma_i$  is the algebraic sign of the  $i$ -th constraint violated in  $x_k$ . Observe that (15.110d) is spherical and is distinct from the trust-region radius  $\Delta_k^{LP}$  used in (15.104d). The solution of (15.110), denoted by  $d^Q$ , is obtained by the projected conjugate gradient algorithm described later.

The total step  $d$  of the SLQP method is computed as

$$d = d^C + \alpha^Q (d^Q - d^C), \quad (15.111)$$

where  $\alpha^Q \in [0, 1]$  is the stepsize that approximately minimizes the model function (15.109). Now, we can present the KNITRO/ACTIVE algorithm as follows:

#### Algorithm 15.9 KNITRO/ACTIVE—Byrd, Gould, Nocedal, and Waltz

1.	Choose an initial point $x_0 \in \mathbb{R}^n$ , as well as the parameters $\Delta_0 > 0$ , $\Delta_0^{LP} > 0$ , $0 < \eta < 1$ . Set $k = 0$
2.	If a stopping test for the nonlinear problem (15.102) is satisfied, stop; otherwise, continue with step 3
3.	Update the penalty parameter $\sigma_k$ and solve the linear program (15.105) to obtain the step $d_k^{LP}$ and the working set $W_k$
4.	Compute $\alpha_k^{LP} \in [0, 1]$ as an approximate minimizer of the quadratic model $q(\alpha d_k^{LP})$ given by (15.109) such that $\alpha_k^{LP} \ d_k^{LP}\  \leq \Delta_k$ . Set $d_k^C = \alpha_k^{LP} d_k^{LP}$
5.	Solve the equality quadratic programming subproblem (15.110) to obtain the solution $d_k^Q$
6.	Compute $d_k^{CE} = d_k^Q - d_k^C$ as the segment line from the Cauchy point to the solution of the equality quadratic programming subproblem
7.	Compute $\alpha_k^Q \in [0, 1]$ as an approximate minimizer of $q_k(d_k^C + \alpha d_k^{CE})$ with respect to $\alpha$
8.	Compute $d_k = d_k^C + \alpha_k^Q d_k^{CE}$ and set $x_T = x_k + d_k$
9.	Compute the standard ratio $\rho_k = \frac{P(x_k, \sigma_k) - P(x_T, \sigma_k)}{q_k(0) - q_k(d_k)}.$ If $\rho_k \geq \eta$ , then set $x_{k+1} = x_T$ ; otherwise, set $x_{k+1} = x_k$ and go to step 10
10.	Update $\Delta_{k+1}^{LP}$ and $\Delta_{k+1}$ . Set $k = k + 1$ and go to step 2

The trust-region radius  $\Delta_k$  is updated based on the standard ratio  $\rho_k$ . The choice of  $\Delta_k^{LP}$  is important because this determines the working set. The estimates of the multipliers  $\lambda_k$  used in the Hessian are the least square estimates using the working set  $W_k$  and modified so that  $\lambda_i \geq 0$  for  $i \in I$ .

### Strategy for Penalty Parameter Update

The algorithm KNITRO/ACTIVE in step 3 requires an updating of the penalty parameter  $\sigma_k$ . This is done in the first phase, where the linear program (15.104) is solved. For this, a piecewise linear model of the constraint violation at the current point  $x_k$  is defined as

$$m_k(d) = \sum_{i \in E} |\nabla c_i(x_k)^T d + c_i(x_k)| + \sum_{i \in I} \max \left\{ 0, -\nabla c_i(x_k)^T d - c_i(x_k) \right\}, \quad (15.112)$$

so that the objective (15.105) of the LP subproblem can be written as

$$l_\sigma(d) = \nabla f(x_k)^T d + \sigma_k m_k(d). \quad (15.113)$$

Given a value  $\sigma_k$ , then the solution of the LP subproblem (15.105) is denoted as  $d^{LP}(\sigma_k)$  to emphasize its dependence on the penalty parameter. Let  $d^{LP}(\sigma_\infty)$  be the minimizer of  $m_k(d)$  subject to the trust-region constraint  $\|d\|_\infty \leq \Delta_k^{LP}$  from (15.105). Then, the following algorithm describes the computation of the LP step  $d_k^{LP}$  and the updating of the penalty parameter  $\sigma_k$ .

#### Algorithm 15.10 Penalty update algorithm

1.	Initialize the data: $x_k$ , $\sigma_{k-1} > 0$ and $\Delta_k^{LP} > 0$ . Choose the parameters $\varepsilon_1, \varepsilon_2 \in (0, 1]$
2.	Solve the subproblem (15.105) with $\sigma = \sigma_{k-1}$ to obtain $d^{LP}(\sigma_{k-1})$
3.	If $m_k(d^{LP}(\sigma_{k-1})) = 0$ , then set $\sigma^+ = \sigma_{k-1}$ ; otherwise, compute $d^{LP}(\sigma_\infty)$
4.	If $m_k(d^{LP}(\sigma_\infty)) = 0$ , then find $\sigma^+ > \sigma_{k-1}$ such that $m_k(d^{LP}(\sigma^+)) = 0$ . Otherwise, find $\sigma^+ \geq \sigma_{k-1}$ such that $m_k(0) - m_k(d^{LP}(\sigma^+)) \geq \varepsilon_1[m_k(0) - m_k(d^{LP}(\sigma_\infty))]$
5.	If necessary, increase the value of $\sigma^+$ to satisfy $l_{\sigma^+}(0) - l_{\sigma^+}(d^{LP}(\sigma^+)) \geq \varepsilon_2 \sigma^+ [m_k(0) - m_k(d^{LP}(\sigma^+))].$ Set $\sigma_k = \sigma^+$ and $d_k^{LP} = d^{LP}(\sigma^+)$ <span style="float: right;">◆</span>

The selection of  $\sigma^+ > \sigma_{k-1}$  is achieved by the successive increase of the current trial value by 10, for example, and by resolving the LP problem. The algorithm is simple and easy to be implemented when the *warm start* strategy is used. The penalty update algorithm above guarantees that  $\sigma$  is chosen large enough to ensure the convergence to a stationary point (Byrd, Gould, Nocedal, & Waltz, 2002).

Since Algorithm 15.10 is a penalty method, it can naturally deal with infeasibility. If a problem is infeasible, then the penalty parameter will be driven to infinity. Moreover, if the algorithm is converging to a stationary point, for our infeasibility measure we have

$$m_k(0) - m_k(d^{LP}(\sigma_\infty)) \rightarrow 0$$

during the penalty update procedure, providing a clear indication of local infeasibility.

### Iteration of Projected Conjugate Gradient Algorithm

One of the main modules of Algorithm 15.9 is solving the equality quadratic programming subproblem (15.110) in step 5. This module is shared by all the algorithms implemented in KNITRO (with active-set or interior-point). In KNITRO, the equality quadratic programming subproblems (15.110)

are solved by using the projected conjugate gradient method (Gould, Hribar, & Nocedal, 2001), (Keller, Gould, & Wathen, 2000), which is well suited for large-scale problems. This is also suitable in case of large-scale subproblems and can treat the situations in which the negative curvature appears without modifications of the Hessian of the Lagrange function. A variant of this iteration in the context of the sequential linear-quadratic programming is as follows.

Let us consider the following quadratic program

$$\min \frac{1}{2}x^T Gx + h^T x \quad (15.114a)$$

subject to

$$Ax = b, \quad (15.114b)$$

where  $G$  is supposed to be symmetric and positive definite on the null space of  $A \in \mathbb{R}^{n \times l}$ . One way to solve (15.114) is to apply a special form of the conjugate gradient (CG) iteration to the KKT system associated to (15.114), which is given by

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix}. \quad (15.115)$$

Although the coefficient matrix is not positive definite, we can apply the CG method to (15.115), provided that we precondition and project the CG method so that it effectively solves the positive definite reduced problem within the feasible manifold (15.114b). Denote the preconditioning matrix by  $P$ . The following algorithm gives the iterations of the projected conjugate gradient.

#### Algorithm 15.11 Preconditioned projected conjugate gradient algorithm

1.	Choose an initial point $x_0$ satisfying $Ax_0 = b$ , as well as the tolerance $\varepsilon > 0$ . Set $x = x_0$ and compute $r = Gx + h$ . Set $z = Pr$ and $d = -z$
2.	If $\ z\  \leq \varepsilon$ , then stop; otherwise, go to step 3
3.	Compute (in this order): $\alpha = r^T z / d^T Gd, \quad x = x + \alpha d,$ $r^+ = r + \alpha Gd, \quad z^+ = Pr^+,$ $\beta = (r^+)^T z^+ / r^T z, \quad d = -z^+ + \beta d,$ $z = z^+, \quad r = r^+$ and continue with step 2



Algorithm 15.11 has exactly the same form as the standard preconditioned CG algorithm for solving symmetric and positive definite systems of linear equations (Golub, & Van Loan, 1996). The difference is that  $P$  is normally a symmetric and positive definite matrix, whereas in our case it represents a *projection and preconditioning matrix* defined as follows.

Given a vector  $r$ , compute  $z = Pr$  as solution of the system

$$\begin{bmatrix} D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad (15.116)$$

where  $D$  is a symmetric and positive definite matrix on the null space of  $A$  and  $w$  is an auxiliary vector. To be a practical preconditioning matrix,  $D$  must be a sparse matrix so that solving (15.116) is

significantly less costly than solving (15.115). By construction,  $z = Pr$  is in the null space of  $A$  and so are all the search directions generated by Algorithm 15.11. Since  $Ax_0 = b$ , all the subsequent iterates  $x$  also satisfy these linear constraints. Let  $Z \in \mathbb{R}^{n \times (n-l)}$  be a basis for the null space of  $A$  from (15.114b). Then, in the null space of  $A$  the solution of (15.114) may be expressed as  $x = x_0 + Zu$ , where  $u \in \mathbb{R}^{n-l}$  is the solution of the linear system

$$(Z^T G Z) u = Z^T (Gx_0 + h). \quad (15.117)$$

The iterations generated by Algorithm 15.11 are given by  $x = x_0 + Zu$ , where  $u$  are the iterations of the preconditioned CG method of the system (15.117) using the matrix  $Z^T D Z$  as a preconditioner. Therefore, Algorithm 15.11 is a standard preconditioned CG algorithm as soon as  $G$  and  $D$  are positive definite on the null space of  $A$  (Gould, Hribar, & Nocedal, 2001).

Algorithm 15.11 has two advantages over the reduced conjugate gradient approach. The first is that there is no need to compute a null space basis and consequently no risk for the ill-conditioning in  $Z$  to deteriorate the rate of convergence of the conjugate gradient iteration. The second benefit is that the projection matrix in (15.116) can also be used to compute the normal step and the Lagrange multipliers, and thus the extra cost of each of these computations is only one back solve involving the factors of this projection matrix (Byrd, Nocedal, & Waltz, 2006).

Algorithm 15.11 assumes that an initial feasible point  $x_0$  is provided (see step 1). The factorization of the system (15.116) allows us to compute such a point by solving

$$\begin{bmatrix} D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ x_0 \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix},$$

which is in fact the minimum norm solution in the norm weighted by  $D$ .

### Hessian Options

KNITRO contains some options for the first and second derivatives. The user can supply these elements, which generally results in the greatest level of efficiency and robustness of the algorithms implemented in KNITRO. However, if these elements are not possible to be supplied, then KNITRO has other options on the second derivative: the dense quasi-Newton BFGS, the dense quasi-Newton SR1, the finite-difference Hessian-vector product, and the limited-memory BFGS Hessian. The dense quasi-Newton BFGS option uses the gradient information to compute a symmetric, positive definite approximation to the Hessian matrix. Typically, this method requires more iterations to converge than the exact Hessian version. The dense quasi-Newton SR1 approach builds an approximate Hessian by using the gradient information, but in this case, the SR1 Hessian approximation is not restricted to be positive definite. Therefore, the SR1 approximation may be a better approach than BFGS if there is a lot of negative curvature in the problem. In the case of large-scale applications, the Hessian of the Lagrangian cannot be computed or it is too large to store. In KNITRO, the Hessian-vector product can be obtained by the finite differences of the gradients of the Lagrangian. Each Hessian-vector product requires one additional gradient evaluation.

In the following, let us present the performances of KNITRO/ACTIVE for solving some applications from the LACOP collection. KNITRO/ACTIVE implements an active-set method based on the sequential linear-quadratic programming (SLQP) and the projected conjugate gradient iteration, known as KNITRO/ACTIVE Option 3.

### Numerical Study—KNITRO/ACTIVE: Solving Applications from the LACOP Collection

Table 15.6 contains the performances of KNITRO/ACTIVE for solving 12 nonlinear optimization applications from the LACOP collection presented in Appendix C.

In Table 15.7 we can see the performances of KNITRO/ACTIVE for solving 6 large-scale nonlinear optimization applications from the LACOP collection.

Table 15.8 shows the performances of KNITRO/ACTIVE (option 3) for solving the applications HANG with a large number of variables.

All SNOPT, NLPQLP, and KNITRO/ACTIVE are reliable algorithms for solving small or large-scale problems. SNOPT evolved from MINOS and performs better than MINOS. Our numerical experiments show that for solving 12 small-scale applications from the LACOP collection (Table 15.1) SNOPT needs 0.31 seconds, while KNITRO/ACTIVE (option 3) (Table 15.6) needs

**Table 15.6** Performances of KNITRO/ACTIVE for solving 12 applications from the LACOP collection. Option 3. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	#nh	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	18	80	19	19	18	0.019	-47.761090
ALKI	10	3	8	22	33	58	23	31	0.022	-1768.8069
PREC	8	0	6	45	116	117	46	68	0.028	3.95116344
PPSE	9	6	0	6	5	11	7	7	0.016	5055.01174
MSP3	13	0	15	7	6	9	8	7	0.015	97.5910347
MSP5	16	0	21	21	34	54	22	32	0.032	174.786994
POOL	34	20	0	15	20	33	16	22	0.025	2569.800
TRAFO	6	0	2	30	88	100	31	60	0.026	135.07596
LATHE	10	1	14	83	127	242	84	126	0.056	-4429.3999
DES	150	50	0	24	378	79	25	41	0.369	1055.18231
CSTC	303	200	0	6	16	7	7	6	0.084	3.4800747
DIFF	396	324	0	1	0	2	1	0	0.090	0.00

**Table 15.7** Performances of KNITRO/ACTIVE for solving 6 applications from the LACOP collection. Option 3. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	#nh	<i>cpu</i>	<i>vfo</i>
HANG	1002	501	0	10	226	24	11	13	0.343	5.0685777
	2002	1001	0	10	226	24	11	13	0.982	5.0685101
FLOW	1182	754	0	10	9	11	11	10	3.456	0.15e-11
FLOWO	1556	1005	0	17	16	18	18	17	7.017	0.79e-31
POL	4004	3000	0	193	255	363	194	257	46.793	9.8214896
	6004	4500	0	219	310	464	220	308	79.314	10.545559
	8004	6000	0	74	105	164	75	105	81.193	10.621358
	10004	7500	0	150	216	337	151	214	134.66	10.291229
CAT	3003	2000	0	117	261	429	118	233	11.138	-0.048055
	6003	4000	0	155	333	551	156	299	33.039	-0.048055
	9003	6000	0	112	190	266	113	183	37.491	-0.048045
CONT	2505	2000	0	14	10	15	15	14	1.893	1.0132389
	5005	4000	0	14	11	15	15	14	4.947	1.0059224
	7505	6000	0	12	12	13	13	12	8.573	1.0045614
	10005	8000	0	16	17	17	17	16	15.245	1.0040718

**Table 15.8** Performances of KNITRO/ACTIVE for solving the HANG application from the LACOP collection. Option 3. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	#nh	<i>cpu</i>	<i>vfo</i>
HANG	4002	2001	0	10	226	24	11	13	2.964	5.0684889
	8002	4001	0	10	226	24	11	13	8.288	5.068482

In Tables 15.6, 15.7, and 15.8, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #it = the number of iterations for solving the application, #itcg = the number of conjugate gradient iterations, #nf = the number of evaluations of the functions, #ng = the number of evaluations of the gradients, #nh = the number of evaluations of the Hessians,  $cpu$  = the CPU computing time to get a solution (seconds), and  $vfo$  = the value of the objective function at the optimal solution.

0.782 seconds. For solving 15 large-scale optimization problems from the LACOP collection, SNOPT (Table 15.2) needs 64.91 seconds, while KNITRO/ACTIVE (option 3) (Table 15.7) needs 466.084 seconds. Both SNOPT and KNITRO/ACTIVE include mechanisms which ensure that the subproblems are always feasible and which guard against the rank-deficient Jacobian of the constraints. SNOPT uses the penalty (or elastic) mode which is used if the sequential quadratic problem is infeasible or if the Lagrange multiplier estimates become very large in norm. Both the BFGS and the limited-memory BFGS approximations are implemented in SNOPT and in KNITRO/ACTIVE. KNITRO/ACTIVE also implements the SR1 approximation that may be more efficient than the BFGS approximation. Many other ingredients and practical details concerning the redundant constraints, discontinuities, inaccurate gradient estimation, and scaling are implemented in these algorithms.

## Notes and References

The sequential quadratic programming (SQP) is one of the most effective methods for solving nonlinear optimization problems of different structures and complexities. It relies on a profound theoretical foundation and provides powerful algorithmic tools for the solution of large-scale relevant problems. This approach can be used both in the line-search and in the trust-region framework. SQP is a very important component in many algorithms for nonlinear optimization. The motivation for which this approach is suitable for solving nonlinear optimization problems is based on the fact that, locally, any optimization problem can be very well approximated by a quadratic model.

SQP needs to solve at each iteration a quadratic approximation of the original problem, trying to estimate which constraints need to be kept (active) and which can be ignored. Therefore, a practical consequence is that the algorithm partly walks along the boundary of the feasible region given by the constraints. The iterates are feasible with respect to all the linear constraints and to a local linearization of the nonlinear constraints, feasibility which is preserved along the iterations. The complementarity condition is satisfied by default. Once the active-set of the constraints has been correctly determined and the optimality conditions are in a given tolerance, the solver terminates with the optimal solution. The number of iterations might be high, but each iteration is relatively cheap.

SQP can be used both in the line-search and in the trust-region frameworks. The algorithms NLPQLP, KNITRO, SNOPT, and CONOPT (see Chap. 16) described in this book use SQP as a crucial component in different computational structures. If the penalty and the augmented Lagrangian methods (SPENBAR and MINOS) are effective when most of the constraints are linear, the SQP methods (NLPQLP, KNITRO, SNOPT, and CONOPT) show their strength when the problems have significant nonlinearities in constraints. A competing approach for handling nonlinear optimization problems is represented by the interior-point methods, implemented for example in KNITRO and IPOPT. The content of this chapter is based on the theoretical developments from Nocedal and

Wright (2006), Schittkowski (1986, 2002, 2005, 2009, 2010), and Gill, Murray, and Saunders (2002, 2005, 2006).

There are two types of SQP methods. In the inequality quadratic programming approach, at each iteration a general inequality-constrained quadratic programming subproblem is solved in order to compute a step and to generate an estimate of the optimal active-set. On the other hand, the equality quadratic programming methods decouple these computations and operate in two phases: in the first one, an estimate of the optimal active-set is determined, and in the second phase an equality-constrained quadratic programming subproblem is solved to find the step.

The main advantages of SQP are as follows: the iterates are feasible with respect to the linear constraints for most of the iterations; they are very efficient for solving highly constrained problems; they give better results on pathological problems; in general; they require fewer functions evaluations; i.e., they are most efficient for problems with expensive functions evaluations; they allow warm starts, permit infeasibility detection, and can capitalize on a good initial point

There is a vast literature on SQP. These methods were proposed by Wilson (1963) in his Ph.D. thesis and developed *inter alia* by Han (1976, 1977), Powell (1977, 1978a, 1978b), Byrd, Schnabel, and Shultz (1987b), Byrd, Tapia, and Zhang (1990), etc. For literature surveys see the papers by Boggs and Tolle (1989, 1995), Herskovits (1995), and Gould, Orban, and Toint (2005a).

The SLQP approach was proposed by Fletcher and Sainz de la Maza (1989) and was further developed by Byrd, Gould, Nocedal, and Waltz (2004a). The second-order corrections were suggested by Coleman and Conn (1982a), Mayne and Polak (1982), and Gabay (1982). The filter SQP was developed by Fletcher and Leyffer (2002) and Fletcher, Leyffer, and Toint (1999).

The most important software implementing sequential linear/quadratic solvers are as follows.

The KNITRO/ACTIVE is based on the works described in Byrd, Gould, Nocedal, and Waltz (2002, 2004a), Byrd, Nocedal, and Waltz (2003, 2006), Nocedal and Wright (2006), Waltz (2004). KNITRO includes both sequential linear-quadratic programming and interior-point methods. The active-set sequential linear-quadratic programming method described in this chapter to determine an estimate of the active-set at each iteration solves a linear programming subproblem that approximates the  $l_1$  exact penalty subproblem. The linear programming subproblem has an additional infinity-norm trust-region constraint. The constraints in this linear programming subproblem that are satisfied as equalities are marked as active, and they are used to set up an equality-constrained quadratic programming subproblem whose objective is a quadratic approximation of the Lagrangian of the problem at the current iterate. This quadratic programming subproblem includes an  $l_2$ -norm trust-region constraint. The equality quadratic programming subproblem is solved by using a projected conjugate gradient algorithm. The penalty parameter is updated to ensure sufficient decrease towards feasibility.

NLPQLP is an extension of the sequential quadratic programming solver NLPQL (Schittkowski, 1985) that implements a nonmonotone line-search to ensure global convergence. Our description of NLPQLP is based on the papers of Schittkowski (1986, 2002, 2005, 2009, 2010). It uses a quasi-Newton approximation to the Hessian of the Lagrangian, which is updated with the BFGS formula. To calculate the stepsize that minimizes an augmented Lagrangian merit function, a nonmonotone line-search is used. NLPQLP illustrates the sequential quadratic programming paradigm, also referred to as active-set methods, because it provides an estimate of the active-set at every iteration. In NLPQLP, the quadratic programming subproblems are modified to avoid the redundant computational effort of evaluating the gradients corresponding to the inactive constraints. The derivatives, if unavailable, can be estimated by using finite differences. If we compare NLPQLP (Table 15.4) versus

KNITRO/ACTIVE (Table 15.6) we can see that, for solving 8 nonlinear optimization applications from the LACOP collection, NLPQLP needs 308 iterations, while KNITRO/ACTIVE needs only 226.

NPSOL (Gill, Murray, Saunders, & Wright, 1986) solves general nonlinear optimization problems by using a sequential quadratic programming algorithm with a line-search on the augmented Lagrangian. The Hessian of the quadratic programming subproblem is a quasi-Newton approximation of the Hessian of the Lagrangian. The quadratic programming subproblem is solved by using a dense BFGS update. If the gradients of the functions defining the problem are not available, NPSOL can estimate them by using finite differences. All the matrices in NPSOL are dense and therefore it is not efficient for solving large-scale problems.

SNOPT is a *pure* sequential quadratic programming algorithm. The description of SNOPT follows the paper of Gill, Murray, and Saunders (2005). SNOPT solves the quadratic programming subproblems by using SQOPT (Gill, Murray, & Saunders, 2006), which is a reduced-Hessian active-set method. SNOPT uses a positive semi-definite quasi-Newton Hessian approximation  $B_k$ . If the number of the nonlinear variables is moderate, then  $B_k$  is stored as a dense matrix. Otherwise, the Hessian of the Lagrangian is updated by using the limited-memory quasi-Newton BFGS updates. The quadratic programming solver SQOPT works with a sequence of the reduced Hessian systems of the form  $Z^T B_k Z d = -Z^T g$ , where  $Z$  is a basis for the null space of  $W$ , a rectangular matrix with  $n_z$  columns. SQOPT can deal with the reduced Hessian systems in various ways, depending on the size of  $n_z$ . If the constraints are currently active in the quadratic subproblem and  $n_z$  are not excessively large, then SQOPT uses the dense Cholesky factorization  $Z^T B_k Z = R^T R$ . Otherwise, SQOPT can maintain a dense quasi-Newton approximation to avoid the cost of forming and of factorizing the reduced Hessian. SNOPT includes an option for using a projected conjugate gradient method instead of the factorization of the reduced Hessian. If a quadratic subproblem is found to be infeasible or unbounded, then SNOPT tries to solve an elastic subproblem that corresponds to a smooth reformulation of the  $l_1$  exact penalty function. The solution of the quadratic subproblem solved at the major iteration is used to obtain a search direction along which an augmented Lagrangian merit function is minimized. SNOPT uses a lot of procedures from MINOS. When the gradients are not available, SNOPT uses finite differences to estimate them. SNOPT is embedded in the GAMS technology (Andrei 2017c).

SQPlab (Gilbert, 2009) is a laboratory for testing different options of the sequential quadratic programming methods (Bonnans, Gilbert, Lemaréchal, & Sagastizábal, 2006). It implements a line-search sequential quadratic programming method that can use either the exact Hessian or a BFGS approximation of the Lagrangian.

CONOPT (Drud, 1996, 2011), described in Chap. 16 of this book, implements three active-set methods. The first one is a gradient projection method that projects the gradient of the minimizing function onto a linearization of the constraints. The second variant is a sequential linear programming method and the third one is a sequential quadratic programming method. CONOPT is a line-search method and includes algorithmic switches that automatically detect which method is most suitable.

Other software implementing SQP are filterSQP (Fletcher, & Leyffer, 2002) presented in Chap. 18 of this book, and RSQP (Bartlett, & Biegler, 2003), which maintains a quasi-Newton approximation to the reduced Hessian.



# Primal Methods: The Generalized Reduced Gradient with Sequential Linearization

16

By primal method, we understand a search method that works directly on the original problem by searching the optimal solution taking a path through the feasible region. Every iteration in these methods is feasible and along the iterations, the values of the minimizing function constantly decrease. The most important primal methods are the *feasible direction method* of Zoutendijk (1960), the *gradient projection method* of Rosen (1960, 1961), the *reduced gradient method* of Wolfe (1967), the *convex simplex method* of Zangwill (1967), and the *generalized reduced gradient method* of Abadie and Carpentier (1969). The last four methods can be embedded into the class of the *active set methods*. The idea of the active set methods is to partition the inequality constraints into two groups: those that can be treated as active (satisfied with equality in the current point) and those that have to be treated as inactive. The constraints treated as inactive are essentially ignored. Of course, the fundamental component of an active set method is the algorithm for solving an optimization problem with equality constraints only.

The primal methods have three main advantages. Firstly, they generate feasible points. Thus, if the optimization process terminates before reaching the solution, then the terminating point is feasible and possibly near the optimal solution of the problem. The second attractive advantage is that, if the optimization process is convergent, then the limit point of this sequence has to be at least a local constrained minimum. Finally, the third advantage is that the primal methods do not use the special structure of the problem such as convexity, for example. Therefore, these methods can be used for solving general constrained nonlinear optimization problems.

*The purpose of this chapter is to give a short description of the primal methods, followed by a more detailed presentation of the modern generalized reduced gradient method, which is representative for the primal methods.*

## 16.1 Feasible Direction Methods

For solving the inequality constrained problem

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_j(x) \leq 0, \quad j = 1, \dots, m, \end{aligned} \tag{16.1}$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c_j: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $j = 1, \dots, m$ , are continuously differentiable functions, a feasible direction method takes steps through the feasible region of the known form

$$x_{k+1} = x_k + \alpha_k d_k, \quad (16.2)$$

where  $d_k$  is the search direction, and  $\alpha_k$  is the stepsize. The crucial aspects of this method are that the scalar  $\alpha_k$  is chosen in such a way so as to minimize the function  $f$  under the constraint that the point  $x_{k+1}$  given by (16.2), and the segment of the line connecting  $x_k$  and  $x_{k+1}$  are all feasible. Therefore, the problem of minimizing  $f$  with respect to  $\alpha$  is nontrivial if the ray  $x_{k+1} = x_k + \alpha d_k$ ,  $\alpha > 0$ , is entirely contained in the feasible region. This is the main motivation of using the feasible directions for solving (16.1). A vector  $d_k$  is a feasible direction in the point  $x_k$  if there is an  $\bar{\alpha} > 0$  such that  $x_{k+1} = x_k + \alpha d_k$  is feasible for all  $\alpha$  with  $0 \leq \alpha \leq \bar{\alpha}$ . The concept of feasible direction is a natural extension of the concept of descent direction from the unconstrained optimization. The following two methods put in a concrete form the feasible direction methods.

**The Frank-Wolfe Method** For the problem (16.1), in which all the constraints are linear  $c_j^T x \leq b_j$ ,  $j = 1, \dots, m$ , one of the earliest feasible direction methods is the one of Frank and Wolfe (1956). In this method, given the current point  $x_k$ , the search direction is computed as  $d_k = x_k^* - x_k$ , where  $x_k^*$  is the solution of the linear programming problem

$$\begin{aligned} & \min \nabla f(x_k)^T x \\ & \text{subject to} \\ & c_j^T x \leq b_j, \quad j = 1, \dots, m. \end{aligned} \quad (16.3)$$

If function  $f$  is continuous differentiable and the polyhedron defined by the linear constraints  $c_j^T x \leq b_j$ ,  $j = 1, \dots, m$ , is bounded, or  $f(x) \rightarrow +\infty$  when  $\|x\| \rightarrow \infty$ , then, for any feasible initial point the Frank-Wolfe method converges to a local minimizer of the problem  $\min \{f(x) : c_j^T x \leq b_j, j = 1, \dots, m\}$ .

**The Zoutendijk Method** Referring to the problem (16.1) with nonlinear constraints, this method solves a sequence of linear programming subproblems as follows (Zoutendijk, 1960). Given a feasible point  $x_k$ , let  $I_k$  be the set of indices representing the active constraints, that is the constraints  $c_j(x_k) = 0$ ,  $j \in I_k$ . Then, the search direction  $d_k$  is chosen as solution of the following problem:

$$\begin{aligned} & \min \nabla f(x_k)^T d \\ & \text{subject to} \\ & c_j(x_k) + \nabla c_j(x_k)^T d \leq 0, \quad j \in I_k, \\ & \sum_{i=1}^n |d_i| = 1. \end{aligned} \quad (16.4)$$

The last equation in (16.4) is a normalizing equation that ensures a bounded solution. The other constraints ensure that the vectors of the form  $x_k + \alpha d_k$  will be feasible for sufficiently small  $\alpha > 0$ . Moreover, solving the problem (16.4) determines that  $d$  is chosen to line up as closely as possible to the negative gradient of  $f$  in the current point. The main difficulties with this simplified Zoutendijk method (16.4) are that for general problems, there may not exist any feasible directions, and in this form the simplified Zoutendijk method is not globally convergent.

Topkis and Veinot (1967) improved the simplified Zoutendijk method and suggested another procedure for the search direction computation as solution of the following linear programming problem:

$$\begin{aligned} & \min \delta \\ \text{subject to} \quad & \nabla f(x_k)^T d - \delta \leq 0, \\ & c_j(x_k) + \nabla c_j(x_k)^T d - \theta_j \delta \leq 0, \quad j = 1, \dots, m, \\ & -1 \leq d_i \leq +1, \quad i = 1, \dots, n, \end{aligned} \tag{16.5}$$

where the unknowns are  $d \in \mathbb{R}^n$  and  $\delta \in \mathbb{R}$ , while  $\theta_j, j = 1, \dots, m$ , are positive parameters. (e.g.,  $\theta_j = 1, j = 1, \dots, m$ ). Obviously, if the minimum of (16.5) corresponds to a value  $\delta^* < 0$ , then  $\nabla f(x_k)^T d < 0$ , that is  $d$  is a descent direction. Besides,  $c_j(x_k) + \nabla c_j(x_k)^T d \leq \theta_j \delta^* < 0$ , proving that all the constraints are satisfied for small values in the direction  $d$ . Details on this method are given in (Andrei, 1999a, b, pp 597–608).

## 16.2 Active Set Methods

These methods make a partition of the inequality constraints of the problem in two groups: those which are active at the current point, namely, they are satisfied as equalities, and those that are inactive. The inactive constraints are essentially ignored. Consider the problem

$$\begin{aligned} & \min f(x) \\ \text{subject to} \quad & c(x) \leq 0, \end{aligned} \tag{16.6}$$

where all the functions  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are continuously differentiable. For this problem, the necessary optimality conditions are

$$\nabla f(x) + \mu^T \nabla c(x) = 0, \tag{16.7a}$$

$$c(x) \leq 0, \tag{16.7b}$$

$$\mu^T c(x) = 0, \tag{16.7c}$$

$$\mu \geq 0, \tag{16.7d}$$

(see Definition 11.14) where  $\mu \in \mathbb{R}^m$  is the vector of the Lagrange multipliers. Denote  $A$  as the set of the indices of the active constraints, that is,  $A$  is the set of  $j$  such that  $c_j(x^*) = 0$ . In terms of active constraints, these optimality conditions can be expressed in a more simple form as

$$\nabla f(x) + \sum_{j \in A} \mu_j \nabla c_j(x) = 0, \tag{16.8a}$$

$$c_j(x) = 0, \quad j \in A, \tag{16.8b}$$

$$c_j(x) < 0, \quad j \notin A, \tag{16.8c}$$

$$\mu_j \geq 0, \quad j \in A, \quad (16.8d)$$

$$\mu_j = 0, \quad j \notin A, \quad (16.8e)$$

Observe that (16.8a) and (16.8b) correspond to the necessary optimality conditions of the equality constrained problem (see Theorem 11.11). (16.8c) guarantees that the inactive constraints are satisfied. The sign requirement of the Lagrange multipliers given by (16.8d) and (16.8e) guarantees that every constraint that is active should be active.

Obviously, if the active set of constraints were known, then the original problem could be replaced by the corresponding problem having only the equality constraints. Now, suppose that an active set of constraints was guessed and the corresponding equality constrained problem solved. Then, if the *other constraints* were satisfied and the Lagrange multipliers turned out to be nonnegative, then the solution of the equality constrained problem would be exactly the solution to the original problem.

At each iteration of an algorithm, the active set methods define a set of constraints denoted by  $W$ , termed *working set*, treated as the active set. The working set is chosen to be a subset of the constraints that are actually active at the current point. In other words, the current point is feasible for the working set. The algorithm tries to move on the surface defined by the working set of the constraints, the so-called *working surface*, in order to get an improved point, i.e., a point at which the minimizing function is reduced. At this new point, the working set may be changed and all the process is repeated. There are several methods for moving on the working surface. The most important ones (the gradient projection, the reduced gradient, the convex simplex, and the generalized reduced gradient methods) are discussed in the following sections. The asymptotic convergence properties of the active set methods depend on the procedure for moving on the working surface, since near the solution, the working set of constraints is generally equal to the correct active set.

Now, let us discuss the strategy of choosing the working set and of moving on the working surface corresponding to the active set methods. Suppose that, for a given working set  $W$  the problem with equality constraints is

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_j(x) = 0, \quad j \in W. \end{aligned} \quad (16.9)$$

Let  $\bar{x}$  be the solution of (16.9), that is  $\bar{x}$  satisfies  $c_j(\bar{x}) < 0$ ,  $j \notin W$ . Of course, this point satisfies the necessary optimality conditions

$$\nabla f(\bar{x}) + \sum_{j \in W} \mu_j \nabla c_j(\bar{x}) = 0. \quad (16.10)$$

If  $\mu_j \geq 0$  for all  $j \in W$ , then the point  $\bar{x}$  is a local solution to the original problem (16.6). On the other hand, if there is an index  $j \in W$  such that  $\mu_j < 0$ , then the objective function can be decreased by relaxing the constraint  $j$ . This follows from the sensitivity interpretation of the Lagrange multipliers (see Theorem 11.18): a small decrease in the constraint value from 0 to  $-b$  will lead to a change in the value of the minimizing function of  $\mu_j b$ , which is clearly negative. Thus, dropping the constraint  $j$  from the working set, an improved solution can be obtained. Therefore, the Lagrange multiplier indicates which constraint should be dropped from the working set. Now, since all the points determined by the algorithm must be feasible, it is necessary to monitor the values of the other constraints to ensure that they are not violated. Obviously, moving on the working surface, a new constraint boundary is attained. Therefore, this constraint is added to the working set, thus obtaining a new working set with which the process can be repeated.

In conclusion, the active set strategy proceeds by systematically dropping and adding constraints, that is starting with a given working set and minimizing the function  $f$  over the corresponding working surface. If new constraint boundaries are attained, they may be added to the working set, but no constraints are dropped from the working set. Hence, a point is obtained that minimizes  $f$  with respect to the current working set of constraints. The corresponding Lagrange multipliers are determined and if they are all nonnegative, it follows that the solution is optimal. Otherwise, one or more constraints with negative Lagrange multipliers are dropped from the working set. With this new working set, the optimization process is restarted until an optimal solution is obtained.

If for every working set  $W$  the problem  $\min\{f(x) : c_j(x) = 0, j \in W\}$  is well defined with a unique nondegenerate solution (that is for all  $j \in W, \mu_j \neq 0$ ), then the sequence of points generated by the above described active set strategy converges to the solution of the inequality constrained problem (16.6).

The difficulty with the active set method is that several problems with incorrect active sets must be solved. Besides, the solutions of these intermediate problems must be the *exact global minimum points* in order to determine the correct sign of the Lagrange multipliers and to ensure that along the iterations the current working set is not encountered again. The convergence of these methods cannot be guaranteed. They are subject to jamming, where the working set changes an infinite number of times. However, numerical experiments proved that the phenomenon of zigzagging (jamming) is very rare and the algorithms based on the active set strategy work very well in practice. The *combinatorial difficulty of the inequality constrained optimization solved by the active set methods* is presented in Sect. 11.3.

### 16.3 The Gradient Projection Method

In this method, the search direction is chosen as the projection of the negative gradient onto the working surface. Consider the problem

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & a_j^T x \leq b_j, \quad j = 1, \dots, m, \end{aligned} \tag{16.11}$$

with linear inequality constraints, where  $x \in \mathbb{R}^n$ . The constraints from (16.11) define the matrix  $A \in \mathbb{R}^{m \times n}$ , where the rows of  $A$  are defined by the constraints  $a_j^T x \leq b_j, j = 1, \dots, m$ . Assume that the constraints from (16.11) define a nonempty region  $X$ . If it exists, a feasible solution to this problem can be obtained by applying the simplex algorithm. At a given feasible point  $x$  there will be a certain number  $q$  of active constraints  $a_j^T x = b_j$  and some inactive constraints  $a_j^T x < b_j$ . The set of active constraints is  $W(x)$ . Now, at the feasible point  $x$  we are interested in finding a feasible direction  $d$  which is descent, that is it satisfies  $\nabla f(x)^T d < 0$  so that the movement in the direction  $d$  will determine a decrease of the values of the function  $f$ . Initially, consider the directions  $d$  satisfying  $a_i^T d = 0, i \in W(x)$ , so that all the working constraints remain active. Therefore, the search direction  $d$  is in the tangent subspace  $T$  defined by the working set of constraints. The search direction used in this method is exactly the projection of the negative gradient onto this subspace. To compute this projection, consider  $A_q \in \mathbb{R}^{q \times n}$  as the matrix composed by the rows corresponding to the working constraints. Assume that  $x$  is a regular point, i.e., the gradient vectors of the active constraints in  $x$  are linear independent (see Definition 11.13). In this case,  $q < n$  and  $\text{rank}(A_q) = q < n$ . The tangent subspace  $T$  in which  $d$  must lie is the subspace of the vectors which satisfy  $A_q d = 0$ . Therefore, the

range subspace  $N$  of  $A_q^T$ , i.e., the set of all the vectors of the form  $A_q^T \mu$  for  $\mu \in \mathbb{R}^q$ , is orthogonal to  $T$ . Any vector can be written as a sum of vectors from each of these two complementary subspaces. In particular,  $-\nabla f(x_k)$  can be written as

$$-\nabla f(x_k) = d_k + A_q^T \mu_k,$$

where  $d_k \in T$  and  $\mu_k \in \mathbb{R}^q$ . Therefore, having in view that  $A_q d_k = 0$ , it follows that

$$-A_q \nabla f(x_k) - (A_q A_q^T) \mu_k = 0,$$

which determines

$$\mu_k = - (A_q A_q^T)^{-1} A_q \nabla f(x_k).$$

Therefore,

$$d_k = - \left( I - A_q^T (A_q A_q^T)^{-1} A_q \right) \nabla f(x_k) \equiv -P_k \nabla f(x_k). \quad (16.12)$$

The matrix  $P_k$  from (16.12) is called the *projection matrix* corresponding to the subspace  $T$ . Observe that, since  $A_q$  is of full rank, it follows that  $A_q A_q^{-1}$  is nonsingular. From (16.12) we can write

$$d_k = -\nabla f(x_k) + A_q^T (A_q A_q^T)^{-1} A_q \nabla f(x_k).$$

Since  $\nabla f(x_k) + d_k$  is orthogonal to  $d_k$ , if  $d_k \neq 0$ , it follows that

$$\nabla f(x_k)^T d_k = (\nabla f(x_k)^T + d_k^T - d_k^T) d_k = -\|d_k\|^2 < 0,$$

Therefore, if  $d_k$  is computed as in (16.12) and is nonzero, then it is a feasible descent direction on the working set.

The problem we face now is how to compute the stepsize along the direction  $d_k$ . Obviously, when  $\alpha$  is increased from zero, the point  $x_k + \alpha d_k$  will remain feasible and the corresponding value of  $f$  will decrease. The maximum stepsize can be computed as

$$\alpha_{\max} = \max \{\alpha : \alpha > 0, x_k + \alpha d_k \in X\}.$$

The stepsize is computed as the minimum of  $f(x_k + \alpha d_k)$ , where  $\alpha$  is on the segment  $[0, \alpha_{\max}]$ . When the projected negative gradient is zero, it follows that

$$\nabla f(x_k) + \mu_k^T A_q = 0, \quad (16.13)$$

that is the point  $x_k$  satisfies the necessary conditions for a minimum on the working surface. If the components of  $\mu_k$  corresponding to the active inequalities are all nonnegative, then this fact together with (16.13) implies that the KKT conditions for the problem (16.11) are satisfied at  $x_k$  and the optimization process terminates. In this case,  $\mu_k$  is the Lagrange multiplier vector for the problem (16.11). If, on the other hand, at least one component of  $\mu_k$  corresponding to the active inequalities is negative, by relaxing the corresponding inequality it is possible to move in a new direction to get an improved point. Suppose that the component  $\mu_k^j$  of  $\mu_k$  is negative and corresponds to the inequality  $a_j^T x \leq b_j$ . The new search direction is determined by relaxing the  $j$ -th constraint and by projecting the

negative gradient onto the subspace defined by the remaining  $q - 1$  active constraints. Let  $A_{\bar{q}}$  be the matrix  $A_q$  with row  $a_j$  deleted. Then, for some  $\bar{\mu}_k$  we have

$$-\nabla f(x_k) = A_{\bar{q}}^T \mu_k, \quad (16.14)$$

$$-\nabla f(x_k) = \bar{d}_k + A_{\bar{q}}^T \bar{\mu}_k, \quad (16.15)$$

where  $\bar{d}_k$  is the projection of  $-\nabla f(x_k)$  using  $A_{\bar{q}}$ . Obviously,  $\bar{d}_k \neq 0$ , otherwise (16.15) would be a special case of (16.14) with  $\mu_k^j = 0$ , which is impossible, since the rows of  $A_q$  are linearly independent. But we know that  $\nabla f(x_k)^T \bar{d}_k < 0$ . By multiplying the transpose of (16.14) by  $\bar{d}_k$  and by using  $A_{\bar{q}} \bar{d}_k = 0$  we get

$$0 > \nabla f(x_k)^T \bar{d}_k = -\mu_k^j a_j^T \bar{d}_k.$$

Since  $\mu_k^j < 0$  it follows that  $a_j^T \bar{d}_k < 0$ . Therefore, since  $a_i^T \bar{d}_k = 0$  for  $i \in W(x_k)$ ,  $i \neq j$  and  $a_j^T \bar{d}_k < 0$ , it follows that  $j$  can be dropped from  $W(x_k)$ . Thus, a new working set of active constraints is obtained, with which a new iteration is started.

The gradient projection method can be extended for solving nonlinear problems of the form

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c(x) = 0, \end{aligned} \quad (16.16)$$

where all the functions are continuously differentiable. The idea is that at a feasible point  $x_k$  the active constraints are determined and the negative gradient is projected onto the subspace tangent to the surface determined by the active constraints in  $x_k$ . If this vector is nonzero, it determines the searching direction for the next step. However, since the surface is curved, this vector is not a feasible direction in general. In this case, additional computations must be executed to get a new feasible point. Firstly, a move is made along the projected negative gradient to a point  $y$ . Then another move is made from  $y$  in the direction perpendicular to the tangent plane at the point  $x_k$  to get a feasible point  $x_{k+1}$  on the working surface. In this case, the projection matrix at  $x_k$  is computed as

$$P_k = I - \nabla c(x_k)^T \left[ \nabla c(x_k) \nabla c(x_k)^T \right]^{-1} \nabla c(x_k).$$

The search direction  $d_k$  is computed as  $d_k = -P_k \nabla f(x_k)$ . Observe that, when moving from  $x_k$  to  $x_{k+1}$ , the Jacobian  $\nabla c$  will change and the new projection matrix cannot be found from the old one. Therefore, it must be recomputed at each iteration of the algorithm.

The computation of the projection matrix might not be so delicate, but the most important and the finest feature of the method is the problem of returning to the feasible region from points outside this region. This is a common problem in nonlinear optimization, including the interior-point methods for linear and nonlinear optimization, i.e., from any point near  $x_k$ , the difficulty is to move back to the constraint surface of (16.16) in a direction orthogonal to the tangent plane at  $x_k$ . Thus, from a point  $y$  a new point of the form  $y + \nabla c(x_k)^T v = y^*$  is determined such that  $c(y^*) = 0$ . To find a suitable first approximation to  $v$  and hence to  $y^*$ , the equation at  $x_k$  is linearized, thus obtaining

$$c\left(y + \nabla c(x_k)^T v\right) \cong c(y) + \nabla c(x_k) \nabla c(x_k)^T v.$$

Obviously, this approximate is accurate when  $\|v\|$  and  $\|y - x_k\|$  are small. Therefore, this motivates the first approximate

$$y_1 = y - \nabla c(x_k)^T \left[ \nabla c(x_k) \nabla c(x_k)^T \right]^{-1} c(y).$$

Thus, by successively repeating this process, we obtain the sequence  $\{y_j\}$  generated as

$$y_{j+1} = y_j - \nabla c(x_k)^T \left[ \nabla c(x_k) \nabla c(x_k)^T \right]^{-1} c(y_j),$$

which, initialized close enough to  $x_k$ , will converge to a solution  $y^*$ .

Let  $x^*$  be a local solution of the problem  $\min\{f(x) : c(x) = 0\}$  and suppose that  $\lambda_{\max}$  and  $\lambda_{\min}$  are the largest and the smallest eigenvalues of the Hessian  $\nabla^2 L(x^*)$  of the Lagrange function  $L(x) = f(x) + \mu^T c(x)$  restricted to the tangent subspace  $T(x^*)$ . If  $\{x_k\}$  is a sequence generated by the gradient projection method that converges to  $x^*$ , then the sequence  $\{f(x_k)\}$  converges to  $f(x^*)$  linearly with a ratio no greater than  $((\lambda_{\max} - \lambda_{\min})/(\lambda_{\max} + \lambda_{\min}))^2$ . This result on the rate of the convergence of the gradient projection method is proved by Luenberger, and Ye, (2016, pp. 375–377). Details on this method are given in (Andrei, 1999a, b, pp. 611–624).

## 16.4 The Reduced Gradient Method

The reduced gradient method elaborated by Wolfe (1967) is closely related to the simplex method for linear programming. The idea of the method is to partition the variables of the problem into two groups as the *basic* and the *nonbasic* variables. Using specific rules, the basic variables are changed into nonbasic variables and vice-versa until a criterion for stopping the iterations has been satisfied. Consider the problem

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & Ax = b, \quad x \geq 0, \end{aligned} \tag{16.17}$$

where  $x \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{m \times n}$ ,  $m < n$  and  $f$  is twice continuously differentiable. Suppose that a *nondegeneracy assumption* is satisfied, that is every collection of the  $m$  columns from  $A$  is linear independent and every basic solution has exactly  $m$  strictly positive variables. Under this assumption, any feasible solution of the problem (16.17) will have at most  $n - m$  variables assigned to zero. A given feasible vector  $x$  for (16.17) is partitioned as  $x = [y, z]^T$ , where  $y \in \mathbb{R}^m$  and  $z \in \mathbb{R}^{n-m}$ . This partition is formed in such a way so that all the components of  $y$  are strictly positive. With this, the problem (16.17) can be written as

$$\min f(x) \tag{16.18a}$$

subject to

$$By + Cz = b, \tag{16.18b}$$

$$y \geq 0, \quad z \geq 0, \tag{16.18c}$$

where  $B$  and  $C$  collect the columns from  $A$  corresponding to the variables from  $y$  and from  $z$ , respectively. Since  $B$  is nonsingular, if  $z$  is specified, then from (16.18b)  $y$  can be uniquely determined. Therefore,  $z$  represents the independent variables, while  $y$  is the vector of the dependent

variables (basic variables) computed as  $y = B^{-1}(b - Cz)$ . Observe that a small change  $\Delta z$  of the independent variables which leaves  $z + \Delta z$  nonnegative will yield by (16.18b) another feasible solution. Since  $y$  was originally taken to be strictly positive,  $y + \Delta y$  will be also positive for small  $\Delta y$ . Therefore, we move from one feasible solution to another one by choosing a  $\Delta z$  and then by moving  $z$  on the line  $z + \alpha\Delta z$ ,  $\alpha \geq 0$ . Similarly,  $y$  will move along the corresponding line  $y + \alpha\Delta y$ . If in this moving a certain variable becomes zero, a new inequality constraint becomes active. Thus, if an independent variable becomes zero, a new  $\Delta z$  must be chosen. On the other hand, if a dependent variable becomes zero, then the partition must be modified. The basic variable is declared independent, and one of the strictly positive independent variables is made dependent.

The reduced gradient method represents the problem in terms of independent variables. Obviously, since  $y$  is determined from  $z$  as  $y = B^{-1}(b - Cz)$ , it follows that the minimizing function can be considered as a function depending only on  $z$ . The gradient with respect to the independent variables  $z$  is obtained by computing the gradient of  $f(B^{-1}b - B^{-1}Cz, z)$  as

$$r^T = \nabla_z f(y, z) - \nabla_y f(y, z)B^{-1}C, \quad (16.19)$$

which is called the *reduced gradient*. A point  $(y, z)$  satisfies the first-order necessary optimality conditions if and only if

$$\begin{aligned} r_i &= 0 \quad \text{for all } z_i > 0, \\ r_i &\geq 0 \quad \text{for all } z_i = 0. \end{aligned}$$

In the reduced gradient method, the vector  $z$  of the independent variables is moved in the direction of the reduced gradient on the working surface. Thus, at each iteration a search direction

$$\Delta z_i = \begin{cases} -r_i, & i \notin W(z), \\ 0, & i \in W(z) \end{cases} \quad (16.20)$$

is determined, and  $\Delta y = -B^{-1}C\Delta z$  is computed. With this, compute the stepsizes as follows:

$$\begin{aligned} \alpha_1 &= \max \{\alpha : y + \alpha\Delta y \geq 0\}, \\ \alpha_2 &= \max \{\alpha : z + \alpha\Delta z \geq 0\}, \\ \alpha_3 &= \min \{f(x + \alpha\Delta x) : 0 \leq \alpha \leq \alpha_1, 0 \leq \alpha \leq \alpha_2\}, \end{aligned}$$

with which a new iteration is computed as  $x = x + \alpha_3\Delta x$ .

The working set corresponding to the inequality constraints is augmented whenever a new variable is zeroed. If it is a dependent variable, then a new partition of the vector  $x$  is formed, that is  $B$  and  $C$  are updated. If a point is found where  $r_i = 0$  for all  $i \notin W(z)$ , but  $r_i < 0$  for some  $j \in W(z)$ , then  $j$  is deleted from  $W(z)$ .

Even if the reduced gradient method borrows many ideas from the simplex algorithm of linear programming, it is subject to jamming. However, the global convergence can be ensured by a trivial modification of the search direction, i.e., instead of (16.20), the search direction is computed as

$$\Delta z_i = \begin{cases} -r_i, & r_i \leq 0, \\ -x_i r_i, & r_i > 0. \end{cases}$$

Details on the reduced gradient method are given in Luenberger and Ye (2016, pp. 378–381) and in Andrei (1999a, b, pp. 624–629).

## 16.5 The Convex Simplex Method

This method elaborated by Zangwill (1967) is a modification of the reduced gradient method. The major difference between this method and the reduced gradient method is that, instead of moving all or several independent variables in the direction of the negative reduced gradient, only one independent variable is changed at a time. The selection of that independent variable to change is made as in the simplex algorithm.

For the problem (16.17) at a given feasible point, consider the partition of the vector  $x$  as  $x = [y, z]^T$ , where  $y$  is the vector of the dependent (basic) variables and  $z$  is the vector of the independent variables (nonbasic). Assume that the bounds on  $x$  are  $x \geq 0$ , as in (16.17). The reduced gradient is computed as in (16.19). Then, the component  $z_i$  to be changed is determined according to the following procedure:

1. Let  $r_{i_1} = \min_i \{r_i\}$ ,
2. Let  $r_{i_2} z_{i_2} = \max_i \{r_i z_i\}$ ,
  - If  $r_{i_1} = r_{i_2} z_{i_2} = 0$ , then terminate. Otherwise
  - If  $r_{i_1} \leq -|r_{i_2} z_{i_2}|$ , increase  $z_{i_1}$ ,
  - If  $r_{i_1} \geq -|r_{i_2} z_{i_2}|$ , decrease  $z_{i_2}$ .

Once a particular component of  $z$  has been selected for changing according to the above procedure, then the corresponding  $y$  is recomputed as  $y = B^{-1}(b - Cz)$ , so that the constraints are satisfied. The components of  $z$  are changed until either a local minimum with respect to a component has been attained or the boundary of one nonnegativity constraint has been reached. The details of this method are identical to those of the reduced gradient method.

**Remark 16.1** The optimization methods presented so far have a lot in common. The concept of feasible direction is directly obtained from the optimality conditions of these methods. The methods based on this concept are susceptible to jamming. Some methods (the gradient projection, the reduced gradient, and the convex simplex) are based on the idea of partitioning the vector of variables into two groups, the dependent (basic) variables and the independent (nonbasic) variables, in such a way that the Jacobian corresponding to the dependent variables should be nonsingular. For solving large-scale problems, the dependent variables are selected in such a way that the inverse of the corresponding Jacobian is easy to be computed. By means of this partition, the constraints can be used to express the minimizing objective as a function depending only on the independent variables. Thus, the concept of the reduced gradient of the minimizing function may be introduced. This is the key element to spread the variables in and out of the basis by using similar rules as in the simplex algorithm from linear programming adapted for the nonlinear case. The rate of convergence of these methods is determined by the eigenvalues of the Hessian of the Lagrangian restricted to the subspace tangent to the active constraints. The limitations of all these methods are that, for determining the search direction they use only the information given by the reduced gradient, and they do not use the approximation of the minimizing problem by linear or by quadratic programming in the current point. This is the reason why these methods are considered as simple or naive approaches for solving nonlinear optimization problems, and we gave them a short presentation. In the next section, we present the *generalized reduced gradient method* and an improvement of it based on sequential linear or sequential quadratic programming.

## 16.6 The Generalized Reduced Gradient Method (GRG)

One of the most elaborate methods for solving general large-scale problems of nonlinear optimization is the *generalized reduced gradient method* (GRG). Its efficiency comes from the fact that it includes a multitude of fundamental concepts, both from the simplex algorithm of linear programming and from the optimization techniques and methods based on the Newton method. Actually, the GRG method is an extension of Wolfe's reduced gradient method due to Abadie and Carpentier (1969) and the improvements by Abadie (1978, 1979), Abadie and Guigou (1970), Abadie and Haggag (1979), Lasdon, Fox and Ratner (1974), Lasdon and Waren (1978, 1980, 1982), Lasdon, Waren, Jain and Ratner (1978), Gabriele and Ragsdell (1980), Drud (1976, 1983, 1984, 1985, 1995, 1996), Andrei (1985, 1987), who consider both the objective function and the constraints of the problem as sufficiently smooth functions.

At the beginning, we present the general form of the GRG method for solving nonlinear optimization problems with equality constraints and simple bounds on variables. After that, in the next section, a particular implementation of GRG with sequential linear or sequential quadratic programming, known as CONOPT, is described.

Consider the general nonlinear optimization problem

$$\begin{aligned} & \min f(X) \\ & \text{subject to} \end{aligned} \tag{16.21a}$$

$$c_i(X) = 0, \quad i = 1, \dots, m, \tag{16.21b}$$

$$l_j \leq X_j \leq u_j, \quad j = 1, \dots, n, \tag{16.21c}$$

where  $X \in \mathbb{R}^n$ ,  $l_j$  and  $u_j$  are the lower and the upper bounds on the variables  $X_j$ ,  $j = 1, \dots, n$ . Assume that  $l_j < u_j$ ,  $j = 1, \dots, n$ . Suppose that  $m < n$ , since in most cases,  $m \geq n$  involves an infeasible problem or one with a single feasible point. Moreover, suppose that  $f$  and  $c_i$ ,  $i = 1, \dots, m$ , are twice continuously differentiable. Denote  $V$  as the set of feasible points (manifold) which satisfy the constraints  $c_i(X) = 0$ ,  $i = 1, \dots, m$ , and  $P$  as the parallelepiped  $l_j \leq X_j \leq u_j$ ,  $j = 1, \dots, n$ .

The main idea of the GRG method is to use the equality constraints  $c_i(X) = 0$ ,  $i = 1, \dots, m$ , in order to express  $m$  variables, called *basic variables*, as functions of the rest of  $n - m$  variables, called *nonbasic variables*. Let  $\bar{X}$  be a feasible point and consider  $y \in \mathbb{R}^m$  as the subvector of the basic variables and  $x \in \mathbb{R}^{n-m}$  as the subvector of the nonbasic variables. Therefore,  $X$  and  $\bar{X}$  are partitioned as  $X = [x \ y]^T$  and  $\bar{X} = [\bar{x} \ \bar{y}]^T$ , respectively. With this, the equality constraints (16.21b) can be rewritten as

$$c(y, x) = 0, \tag{16.22}$$

where  $c = [c_1 \ \dots \ c_m]^T$ .

Suppose that the  $m \times m$ -dimensional Jacobian matrix  $\partial c / \partial y$  evaluated in  $\bar{X}$ , the *basic matrix*, is nonsingular. Then, from the implicit function theory (see Theorem A2.6), it follows that in a neighborhood of  $\bar{x}$  the nonlinear algebraic system (16.22) can be solved with respect to the basic variable  $y$  in order to obtain the solution  $y(x)$ . Therefore, the objective function (16.21a) can be expressed as a function which depends only on the nonbasic variables

$$F(x) = f(y(x), x). \tag{16.23}$$

Hence, in a neighborhood of  $\bar{x}$  a *reduced problem* is obtained which contains only the simple bound on the nonbasic variables  $x$

$$\begin{aligned} & \min F(x) \\ & \text{subject to} \\ & l_N \leq x \leq u_N, \end{aligned} \tag{16.24}$$

where obviously  $l_N$  and  $u_N$  are  $n - m$ -dimensional vectors with the lower and upper bounds of the nonbasic variables. The GRG method solves the original problem (16.21) by solving a sequence of subproblems (16.24) which are nonlinear optimization problems with simple bounds. The algorithms for solving the simple bounded nonlinear optimization are presented in Chap. 12.

Obviously, for the reduced problem (16.24) to have a solution, the nonbasic variables  $x$  should be free around the current point  $\bar{x}$ . Of course, the presence of the simple bounds limits the evolution of the nonbasic variables, but the algorithms which maintain the simple bound satisfied are relatively simple to design. The evolution of the basic variables is more complicated. Indeed, if some components of  $\bar{y}$  are fixed on their lower or upper bounds, then a small modification of  $x$  from  $\bar{x}$  may determine the violation of the simple bounds on  $y$ . In order to avoid such situations the following nondegeneracy hypothesis is introduced.

"For any feasible point  $X$  which satisfies the constraints of (16.21) there is a partition of it in  $m$  basic variables  $y$  and  $n - m$  nonbasic variables  $x$ , such that

$$l_B < y < u_B, \tag{16.25}$$

where  $l_B, u_B \in \mathbb{R}^m$  are the lower and the upper bounds of the basic variables and the Jacobian matrix  $B = \partial c/\partial y$  evaluated in  $X$  is nonsingular."

For the problem (16.21) let  $\bar{X}$  be a feasible point with  $y$  as the basic variables and  $x$  as the nonbasic variables. To evaluate the objective  $F(x)$  of (16.24), we have to know the basic variables  $y(x)$ . Except for the linear functions or for some particular nonlinear functions,  $y(x)$  cannot be explicitly determined. However, for a given  $x$  this can be computed by solving the algebraic nonlinear system (16.22).

### The Generalized Reduced Gradient

Therefore, for solving (16.21) its constraints are used in order to express the set of dependent (basic) variables  $y$  as functions of the independent (nonbasic) variables  $x$ . It is not possible to establish this dependence for the general nonlinear functions. However, using the method of the *variations of the constraints*, it is possible to find  $y$  as functions of  $x$ , thus obtaining a reduced problem and introducing the reduced gradient as an optimality criterion.

The differential of  $f$  is

$$df(X) = \nabla_y f(X)^T dy + \nabla_x f(X)^T dx, \tag{16.26}$$

where  $\nabla_y f(X)$  and  $\nabla_x f(X)$  are the gradients of  $f$  subject to the basic variables and the nonbasic variables, respectively. Therefore,

$$\frac{df(X)}{dx} = \nabla_x f(X)^T + \nabla_y f(X)^T \frac{dy}{dx}. \tag{16.27}$$

To eliminate the matrix  $dy/dx$  in (16.27), observe that

$$dc_i(X) = \nabla_y c_i(X)^T dy + \nabla_x c_i(X)^T dx = 0, \tag{16.28}$$

$i = 1, \dots, m$ , or in the matriceal form

$$\frac{dc(X)}{dx} = \frac{\partial c}{\partial x} + \frac{\partial c}{\partial y} \frac{dy}{dx} = 0, \quad (16.29)$$

where  $\partial c/\partial y$  and  $\partial c/\partial x$  are the Jacobians of  $c$  subject to the basic and the nonbasic variables, respectively. From the nondegeneracy hypothesis, it follows that the matrix  $\partial c/\partial y$  is nonsingular. Therefore,

$$\frac{dy}{dx} = - \left[ \frac{\partial c}{\partial y} \right]^{-1} \frac{\partial c}{\partial x}. \quad (16.30)$$

Using this in (16.27), we get

$$df = \left[ \nabla_x f(X)^T - \nabla_y f(X)^T \left[ \frac{\partial c}{\partial y} \right]^{-1} \left[ \frac{\partial c}{\partial x} \right] \right] dx \triangleq r dx, \quad (16.31)$$

where

$$r = \nabla_x f(X)^T - \nabla_y f(X)^T \left[ \frac{\partial c}{\partial y} \right]^{-1} \left[ \frac{\partial c}{\partial x} \right] \quad (16.32)$$

is the *generalized reduced gradient* of function  $f$ .

Define

$$u = - \nabla_y f(X)^T \left[ \frac{\partial c}{\partial y} \right]^{-1}. \quad (16.33)$$

With this, the generalized reduced gradient is

$$r = \nabla_x f(X)^T + u \left[ \frac{\partial c}{\partial x} \right]. \quad (16.34)$$

**Remark 16.2** If the functions of the problem  $f$  and  $c_i$ ,  $i = 1, \dots, m$ , are all linear, (16.34) are exactly the *reduced cost factors* from linear programming, and  $u$  is the vector of the *simplex multipliers*.  $\blacklozenge$

It is interesting to see the relation between the generalized reduced gradient and the KKT optimality conditions.

**Teorema 16.1** *If  $\bar{X}$  is an optimal solution of (16.21) and the gradients of all the active constraints in  $\bar{X}$  are linear independent, then in  $\bar{X}$  the KKT conditions holds.*

**Proof** Let  $u$  be the vector of the Lagrange multipliers associated to the equality constraints (16.21b) and  $\lambda$  and  $\mu$  the corresponding multipliers associated to the lower and the upper bounds from (16.21c), respectively. The Lagrange function of the problem (16.21) is

$$L = f(x, y) + u^T c(x, y) + \lambda_x^T (x - l_x) + \lambda_y^T (y - l_y) + \mu_x^T (u_x - x) + \mu_y^T (u_y - y),$$

where  $\lambda_y$  and  $\mu_y$  are the subvectors from  $\lambda$  and  $\mu$  corresponding to the basic variables, respectively, and similarly for  $\lambda_x$ ,  $\mu_x$  for the nonbasic variables.

The KKT conditions written in terms of the  $x$  and  $y$  variables are as follows:

$$\frac{\partial f}{\partial y} + u^T \frac{\partial c}{\partial y} + \lambda_y - \mu_y = 0, \quad (16.35)$$

$$\frac{\partial f}{\partial x} + u^T \frac{\partial c}{\partial x} + \lambda_x - \mu_x = 0, \quad (16.36)$$

$$\lambda \geq 0, \quad \mu \geq 0, \quad (16.37)$$

$$\begin{aligned} \lambda_x^T (x - l_x) &= \lambda_y^T (y - l_y) = 0, \\ \mu_x^T (u_x - x) &= \mu_y^T (u_y - y) = 0. \end{aligned} \quad (16.38)$$

If  $\bar{X}$  is an optimal solution of (16.21), then there exist the vectors  $\bar{u}$ ,  $\bar{\lambda}$ , and  $\bar{\mu}$ , which together with  $\bar{X}$  verify the relations (16.35, 16.36, 16.37, and 16.38). Since  $\bar{y}$  lies between its bounds, from (16.38) it follows that  $\lambda_y = \mu_y = 0$ , that is from (16.35) we have

$$u = -\frac{\partial f}{\partial y} \left[ \frac{\partial c}{\partial y} \right]^{-1}.$$

Therefore,  $u$  defined by (16.33) is exactly the vector of the Lagrange multipliers associated to the equalities constraints (16.21).

The condition (16.36) can be written as

$$\frac{\partial f}{\partial x} + u^T \frac{\partial c}{\partial x} = \mu_x - \lambda_x. \quad (16.39)$$

Observe that the left-hand side of (16.39) is  $r$ , the generalized reduced gradient of the problem (16.21). With this, from (16.38) it follows that in the minimizing point the generalized reduced gradient verifies

$$\begin{cases} r_i = -\lambda_{x_i} \leq 0, & \text{if } x_i = l_{x_i}, \\ r_i = \mu_{x_i} \geq 0, & \text{if } x_i = u_{x_i}, \\ r_i = 0, & \text{if } l_{x_i} < x_i < u_{x_i}. \end{cases} \quad (16.40)$$

In order to see the connection between (16.39) and the reduced problem (16.24), we note that, if  $x_i$  is strictly between its bounds, then  $\lambda_{x_i} = \mu_{x_i} = 0$  and therefore from (16.39), we have

$$\frac{\partial F}{\partial x_i} \equiv r_i = 0. \quad (16.41)$$

If  $x_i = l_{x_i}$ , then  $\mu_{x_i} = 0$ , and therefore

$$\frac{\partial F}{\partial x_i} \equiv r_i = -\lambda_{x_i} \leq 0. \quad (16.42)$$

On the other hand, if  $x_i = u_{x_i}$ , then  $\lambda_{x_i} = 0$ , that is

$$\frac{\partial F}{\partial x_i} \equiv r_i = \mu_{x_i} \geq 0. \quad (16.43)$$

But (16.41, 16.42, and 16.43) are exactly the optimality conditions associated to the reduced problem (16.24) expressed in terms of the generalized reduced gradient. Therefore, the KKT optimality conditions for the problem (16.21) can be viewed as the optimality conditions of the reduced problem (16.24), where  $u$  from (16.34) is the vector of the multipliers associated to the constraints from (16.21)  $\blacklozenge$

### Computation of the Search Direction and of the Stepsize

The KKT conditions for the optimality of  $X^0 = [x_0 \ y_0]^T$  are given by the existence of the vectors  $u \in \mathbb{R}^m$  and  $r \in \mathbb{R}^{n-m}$  such that

$$\frac{\partial f}{\partial x_0} + u^T \frac{\partial c}{\partial x_0} = r, \quad (16.44)$$

$$\frac{\partial f}{\partial y_0} + u^T \frac{\partial c}{\partial y_0} = 0, \quad (16.45)$$

$$r_i \begin{cases} \leq 0, & \text{if } x_{0i} = l_{x_i}, \\ \geq 0, & \text{if } x_{0i} = u_{x_i}, \\ = 0, & \text{if } l_{x_i} < x_{0i} < u_{x_i}. \end{cases} \quad (16.46)$$

From (16.44) and (16.45) it follows that

$$u = -\frac{\partial f}{\partial y_0} \left[ \frac{\partial c}{\partial y_0} \right]^{-1},$$

$$r = \frac{\partial f}{\partial x_0} - \frac{\partial f}{\partial y_0} \left[ \frac{\partial c}{\partial y_0} \right]^{-1} \frac{\partial c}{\partial x_0}.$$

If  $r$  is computed in such a way as to satisfy (16.46), then  $X^0 = [x_0 \ y_0]^T$  is a stationary point (or even a minimizer if the problem has some convexity properties). On the other hand, if  $r$  does not satisfy (16.46), then the vector of the nonbasic variables  $x$  is modified as

$$x = x_0 + \alpha d, \quad (16.47)$$

where  $d$  is the search direction and  $\alpha$  is the stepsize.

The search direction  $d = [d_1 \ \dots \ d_{n-m}]^T$  for modifying the nonbasic variables is defined in the terms of the generalized reduced gradient as

$$d_i = 0 \begin{cases} \text{if } x_{0i} = l_{x_i} & \text{and } r_i < 0, \\ \text{if } x_{0i} = u_{x_i} & \text{and } r_i > 0, \end{cases} \quad (16.48a)$$

$$d_i = -r_i, \quad \text{if } l_{x_i} < x_{0i} < u_{x_i}. \quad (16.48b)$$

On the manifold  $V$ , in a neighborhood of  $[x_0 \ y_0]^T$  a curve  $\Gamma$  can be defined as

$$x = x_0 + \alpha d, \quad \alpha \geq 0,$$

$$y = y(\alpha), \quad y(0) = y_0,$$

$$c(x_0 + \alpha d, \ y(\alpha)) = 0.$$

The tangent  $L$  at  $\Gamma$  in the point  $[x_0 \ y_0]^T$  is the semiline

$$\begin{aligned} x &= x_0 + \alpha d, \quad \alpha \geq 0, \\ y &= y_0 + \alpha h, \\ \frac{\partial c}{\partial x} d + \frac{\partial c}{\partial y} h &= 0. \end{aligned}$$

Notice that the stepsize  $\alpha$  has to be limited so that

$$\begin{aligned} l_x &\leq x_0 + \alpha d \leq u_x, \\ l_y &\leq y_0 + \alpha h \leq u_y, \end{aligned}$$

that is,  $\alpha$  is defined in a nonempty interval, possibly infinite,  $0 \leq \alpha \leq \bar{\alpha}$ . With this, a step can be taken along the line  $L$  to arrive in the point  $\tilde{X}^1 = [x_1 \ \tilde{y}_1]^T$ , where

$$\begin{aligned} x_1 &= x_0 + \alpha_1 d, \\ \tilde{y}_1 &= y_0 + \alpha_1 h, \end{aligned}$$

is definite by the stepsize  $\alpha_1$  computed as solution of the problem

$$\min_{0 \leq \alpha \leq \bar{\alpha}} f(x_0 + \alpha d, \ y_0 + \alpha h). \quad (16.49)$$

In general, the point  $\tilde{X}^1$  does not belong to the variety  $V$ . However, from  $\tilde{X}^1$  a new point  $X^1 = [x_1 \ y_1]^T \in V$  belonging to the curve  $\Gamma$  can be computed. This new point can be obtained by solving the nonlinear algebraic system

$$c(x_1, \ y) = 0. \quad (16.50)$$

with respect to the basic variables  $y$ .

Obviously, any method for solving the nonlinear algebraic system can be selected for solving (16.50), particularly the Newton method. For example, starting from  $\tilde{y}_1$ , an iteration of the Newton method is defined by

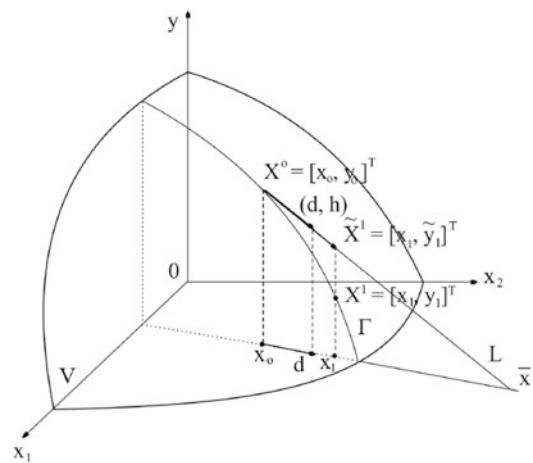
$$\begin{aligned} \Delta y_t &= - \left[ \frac{\partial c}{\partial y} \right]^{-1} c(x_1, \ \tilde{y}_t), \\ \tilde{y}_{t+1} &= \tilde{y}_t + \Delta y_t, \quad t = 1, \ 2, \ \dots . \end{aligned} \quad (16.51)$$

### Illustration of a Particular Case with $m = 1$ (A Constraint), $n = 2$ , $l = 0$ and $u = +\infty$

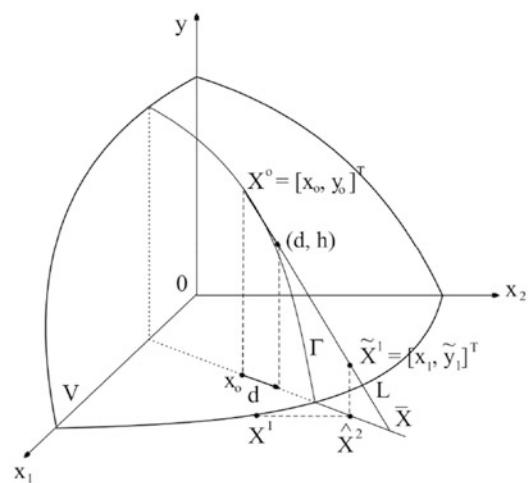
Figure 16.1 shows the points  $\tilde{X}_1$  and  $X_1$ . Suppose that the method selected for solving (16.50) converges to the point  $X^1 \in V \cap P$ . Two cases have to be considered now:

- A. If  $y_1$  obtained as solution of (16.50) satisfies  $l_y < y_1 < u_y$ , then, from  $X^1$  we can continue the iterations in the same way as from  $X^0$ . This case is illustrated in Fig. 16.1.
- B. On the other hand, if  $(y_1)_r = l_{y_r}$  or  $(y_1)_r = u_{y_r}$  for a certain index  $r$ , then the variable  $y_r$  has to leave the set of the basic variables and a nonbasic variable must enter the basis. This operation of changing the basis is executed in the same way as in the simplex algorithm from linear programming. This situation is illustrated in Figs. 16.2 and 16.3.

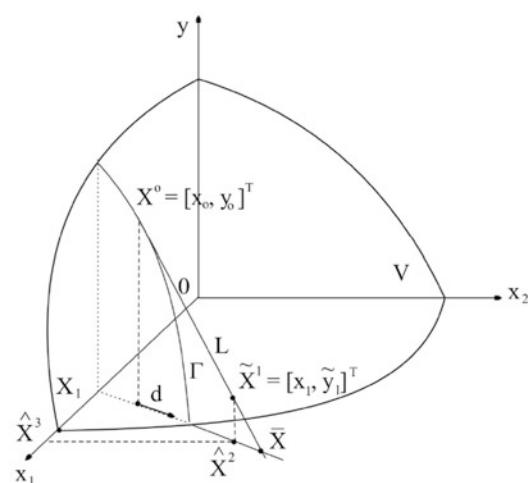
**Fig. 16.1** Computation of  $X^1 = [x_1 \ y_1]^T \in V$ . The point  $\tilde{X}^1 \in L$  corresponds to  $\alpha = \alpha_1$ , while point  $\bar{X}$  corresponds to  $\alpha = \bar{\alpha}$



**Fig. 16.2** The iterations for solving the system (16.50) starting from the point  $\tilde{X}^1$  lead to the point  $\hat{X}^2$ ,  $\hat{X}^2 \notin V$



**Fig. 16.3** The iterations for solving the system (16.50) starting from the point  $\tilde{X}^1$  lead to the points  $\hat{X}^2$ ,  $\hat{X}^3$ , and then to  $X^1$



Obviously, for different reasons, the method for solving the system (16.50) in combination with the algorithm for solving the optimization problem (16.21) can fail. The following situations may happen:

1. The method is not convergent. For example, after a certain number of iterations, let us say 20, the point  $[x_1 \ \tilde{y}_{20}]^T$  is not on  $\Gamma$  or in a neighborhood of  $\Gamma$ . Or, after a few iterations, let us say 5,  $\|c(x_1 \ \tilde{y}_5)\| \geq \|c(x_1 \ \tilde{y}_1)\|$ .
2. For a certain iteration,  $\tilde{y}_t$  determines  $f(x_1 \ \tilde{y}_t) > f(x_0 \ y_0)$ .
3. At a certain iteration,  $\tilde{y}_{t+1}$  leads to a point  $[x_1 \ \tilde{y}_{t+1}]^T$  that is not in the parallelepiped  $P$ .

In the situations (1) and (2) above, the remedy is to reduce  $\alpha_1$  (e.g., set  $\alpha_1/2$  or  $\alpha_1/10$ ) and from the new point on  $L$  continue the iterations for solving the nonlinear system (16.50). In situation (3), let  $[x_1 \ \tilde{y}_t]^T \in P$ . Then the segment of the line connecting the point  $[x_1 \ \tilde{y}_t]^T$  with  $[x_1 \ \tilde{y}_{t+1}]^T$  crosses the frontier of  $P$  in a point  $[x_1 \ \hat{y}_{t+1}]^T$  such that for a certain index  $r$ ,  $(\hat{y}_{t+1})_r = l_{y_r}$  or  $(\hat{y}_{t+1})_r = u_{y_r}$ . In this situation, a changing of the basis is executed and with this new partition of  $X$ , continue the iterations for solving the nonlinear system (16.50).

Figure 16.2 illustrates the situation in which, starting from the point  $\tilde{X}^1 \in L$  a new point  $\tilde{X}^2 = [x_1 \ \hat{y}_{t+1}]^T$  is obtained. Assume that a changing of the basis was executed, in which the variable  $y$  is replaced by  $x_2$ . Then, by solving (16.50) a new point  $X^1 \in V$  is obtained, which is considered as a starting point for the next iteration.

Another possible situation which can appear is illustrated in Fig. 16.3. In this case, starting from the point  $\tilde{X}^1 = [x_{11} \ x_{12} \ \tilde{y}_1]^T$ , the method for solving the system (16.50) determines the point  $\tilde{X}^2 = [x_{11} \ x_{12} \ 0]^T$  from which, by a changing of the basis, the point  $\tilde{X}^3 = [x_{11} \ 0 \ 0]^T$  is obtained and then the point  $X^1 \in V$  is determined.

### Algorithm 16.1 (Generalized reduced gradient)

1.	Determine a feasible initial point $X^0$ and set $k = 0$
2.	Compute the Jacobian matrix $\partial c / \partial X^k$ and $X^k$ is partitioned as $X^k = [x_k \ y_k]^T$ such that $x_k \in \mathbb{R}^{n-m}$ , $y_k \in \mathbb{R}^m$ , $l_y < y_k < u_y$ and $\text{rank}[\partial c / \partial y_k] = m$
3.	Compute the inverse of the matrix $B = [\partial c / \partial y_k]$
4.	Compute the multipliers $u = -\left[\frac{\partial f}{\partial y_k}\right]^{-1}$ and the generalized reduced gradient $r = \frac{\partial f}{\partial x_k} + u \frac{\partial c}{\partial x_k}$
5.	Compute the search direction from the space of the nonbasic variables $d = [d_1 \ \dots \ d_{n-m}]^T$ , where $d_i = \begin{cases} 0, & \text{if } (x_k)_i = l_{x_i} \quad \text{and } r_i < 0, \\ 0, & \text{if } (x_k)_i = u_{x_i} \quad \text{and } r_i > 0, \\ -r_i, & \text{if } l_{x_i} < (x_k)_i < u_{x_i}, \end{cases}$ for $i = 1, \dots, n-m$
6.	If $d = 0$ , stop; otherwise, compute the search direction in the space of the basic variables $h = -\left[\frac{\partial c}{\partial y_k}\right]^{-1} \left[\frac{\partial c}{\partial x_k}\right] d$
7.	Determine the maximum value $\bar{\alpha}$ of $\alpha$ such that $l_x \leq x_k + \alpha d \leq u_x,$ $l_y \leq y_k + \alpha h \leq u_y.$
8.	Compute $\alpha_1$ as solution of the problem: $\min_{0 \leq \alpha \leq \bar{\alpha}} f(x_k + \alpha d, y_k + \alpha h)$ .
9.	Compute the point $\tilde{X}^1 = [x_1 \ \tilde{y}_1]$ where $x_1 = x_k + \alpha_1 d$ , $\tilde{y}_1 = y_k + \alpha_1 h$

- |     |  |
|-----|--|
| 10. | <p>Starting from <math>\tilde{y}_1</math> compute a new feasible point <math>X^{k+1}</math> by solving the nonlinear algebraic system <math>c(x_1, y) = 0</math> with respect to the variables <math>y</math>. Let <math>\tilde{y}_j</math> be the iterations generated by a method for solving this system (e.g., the Newton method), <math>j = 1, 2, \dots</math>. Consider <math>p</math> as a fixed arbitrary integer.</p> <ul style="list-style-type: none"> <li>(a) If <math>\ c(x_1, \tilde{y}_j)\  \leq \epsilon</math> is not achieved for any <math>j \geq p</math>, then set <math>\alpha_1 = \alpha_1/2</math> and continue with step 9</li> <li>(b) If <math>f(x_1, \tilde{y}_j) &lt; f(x_1, \tilde{y}_j)</math> for a certain <math>j</math>, that is the function <math>f</math> is not reduced along the iterations, then set <math>\alpha_1 = \alpha_1/2</math> and continue with step 9</li> <li>(c) Let <math>y_1</math> be the solution of the system <math>c(x_1, y) = 0</math>. If <math>l_y &lt; y_1 &lt; u_y</math>, then set <math>X^{k+1} = [x_1 \ y_1]^T</math>, <math>k = k + 1</math> and continue with step 2; otherwise, execute step 11</li> </ul> |
| 11. | If $(y_1)_i = l_{y_i}$ or $(y_1)_i = u_{y_i}$ for a certain index $i$ , then make a change of variables in which the variable $y_i$ leaves the basis and a nonbasic variable, not fixed at its bounds, enters the basis  |
| 12. | Update the partition of $X$ and the inverse of the basis $B^{-1}$ and continue with step 4. ◆  |

Some comments are as follows:

1. The criterion for stopping the iterations from step 6 of the algorithm can be implemented in a more convenient way as  $|d_i| < \epsilon_i$ ,  $i = 1, \dots, n - m$ , where  $\epsilon_i > 0$  are tolerances which depend on the size of the components  $x_i$  of the vector  $x$ .
2. If after two or three reductions of  $\alpha_1$  the iterations of the Newton method implemented in step 10 of the algorithm for solving the system  $c(x_1, y) = 0$  with respect to the basic variables  $y$  fails, then a change of variables is executed.
3. One of the most important operations is the computation of the inverse of the basic matrix  $B = [\partial c / \partial y_k]$  in step 3 of Algorithm 16.1. The advanced procedures for the inverse of the basis computation make a partition of the vector  $X^k$  in step 2 of Algorithm 16.1 in such a way that the basic matrix  $B$  is nonsingular and as sparse as possible.

## 16.7 GRG with Sequential Linear or Sequential Quadratic Programming (CONOPT)

CONOPT is a very interesting and profitable combination of the generalized reduced gradient with sequential linear-programming and with sequential quadratic-programming. All these algorithms are embedded into the generalized reduced gradient (GRG) scheme as described in (Drud, 1976, 1983, 1985, 1994, 1995, 1996, 2005, 2011). CONOPT implements some algorithmic novelties regarding selection and factorization of the basis, line-search and the Newton iterations, sequential linear or quadratic programming, computation of the generalized reduced gradient of the minimizing function, advanced procedures for the inverse representation of a nonsingular matrix, etc.

CONOPT is a particularization of the generalized reduced gradient Algorithm 16.1, which considers the general nonlinear optimization problems of the following form

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c(x) = b, \\ & l \leq x \leq u, \end{aligned} \tag{16.52}$$

where  $x \in \mathbb{R}^n$ ,  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are twice continuously differentiable functions,  $b \in \mathbb{R}^m$  and  $l, u \in \mathbb{R}^n$  are simple bounds on variables. Suppose that  $n \geq m$ . Assume that  $l_i \leq u_i$ ,  $i = 1, \dots, n$ .

Some of the bound values may be minus or plus infinity. If the problem has inequality constraints, then these are transformed into equalities by using the *slack variables*. Moreover, in CONOPT it is assumed that  $x$  includes *slack* and *artificial variables* for all the constraints. This implies that the Jacobian of the constraints includes a unit matrix and therefore, it has full row rank.

CONOPT works as follows. The set of  $m$  equations  $c(x) = b$  defining the (nonlinear) constraints of the problem can be used to eliminate  $m$  variables. For the simplicity of the presentation, consider that the vector  $x$  is partitioned in the sub-vectors  $x_b \in \mathbb{R}^m$  (basic) and  $x_n \in \mathbb{R}^{n-m}$  (nonbasic). Therefore, the system of the nonlinear constraints  $c(x) = b$  can be rewritten as  $x_b = c_1(x_n)$ . It is obvious that it is not possible to find an analytic expression for the function  $c_1$ , except for some very particular cases, like the linear constraints. However, from the theory of the implicit functions (see Theorem A2.6), we have the following result. If  $(x_b^0, x_n^0)$  satisfies the system  $c(x_b^0, x_n^0) = 0$  and the Jacobian  $\partial c / \partial x_b$  has rank  $m$ , i.e., it is nonsingular, then in a neighborhood of  $(x_b^0, x_n^0)$  it is possible to transform  $c(x_b, x_n) = 0$  into  $x_b = c_1(x_n)$ . The function  $c_1$  is differentiable and its Jacobian is found by implicit differentiation as

$$\frac{\partial c}{\partial x} dx = \frac{\partial c}{\partial x_b} dx_b + \frac{\partial c}{\partial x_n} dx_n = 0. \quad (16.53)$$

Hence,

$$\frac{\partial c_1}{\partial x_n} = \frac{\partial x_b}{\partial x_n} = - \left( \frac{\partial c}{\partial x_b} \right)^{-1} \frac{\partial c}{\partial x_n}. \quad (16.54)$$

In practice, the function  $x_b = c_1(x_n)$  is computed by solving the nonlinear system  $c(x_b, x_n) = 0$  with respect to  $x_b, x_n$  being known.

The equation  $x_b = c_1(x_n)$  is introduced in the objective function to get

$$f(x_b, x_n) = f(c_1(x_n), x_n) \triangleq F(x_n),$$

where  $F(x_n)$  is differentiable with the derivative

$$\frac{\partial F}{\partial x_n} = \frac{\partial f}{\partial x_b} \frac{\partial x_b}{\partial x_n} + \frac{\partial f}{\partial x_n} = \frac{\partial f}{\partial x_n} - \frac{\partial f}{\partial x_b} \left( \frac{\partial c}{\partial x_b} \right)^{-1} \frac{\partial c}{\partial x_n}. \quad (16.55)$$

With these developments, (16.52) can be reformulated as

$$\begin{aligned} & \min F(x_n) \\ & \text{subject to} \\ & x_b = c_1(x_n), \\ & l_b < x_b < u_b, \quad l_n \leq x_n \leq u_n. \end{aligned} \quad (16.56)$$

Observe that the transformations that have been used are very similar to those from linear programming.  $x_b$  contains the basic variables. They are used to compensate for the changes in the nonbasic variables  $x_n$  so that the constraints  $c(x) = b$  are satisfied. The derivative  $\partial F / \partial x_n$  from (16.55) is similar to the vector of the *reduced costs* from linear programming. It shows the influence on the objective function of changes in the nonbasic variables, taking into consideration the corresponding changes in the basic variables. Usually,  $\partial F / \partial x_n$  is called the *reduced gradient*. The matrix  $\partial c / \partial x_b$  is similar to the *basis matrix* from linear programming. However, the major difference between the nonlinear problem and the linear problem is that in the optimal solution, the nonlinear problem can have more than  $m$  variables strictly between the lower and the upper bounds. Thus, an optimization

method like the simplex algorithm that works only with basic solutions cannot be used in the nonlinear case. The main steps of GRG in the implementation of CONOPT are as follows:

**Algorithm 16.2 (CONOPT—Drud)**

1.	Find an initial feasible solution $x^0$
2.	Compute the Jacobian matrix $\partial c/\partial x$ . Partition $x$ into $x_b \in \mathbb{R}^m$ and $x_n \in \mathbb{R}^{n-m}$ , so that $l_b < x_b < u_b$ and the submatrix $\partial c/\partial x_b$ is nonsingular
3.	Compute the inverse $(\partial c/\partial x_b)^{-1}$
4.	Compute the Lagrange multipliers as solution of the system $\left( \frac{\partial c}{\partial x_b} \right) u = -\frac{\partial f}{\partial x_b}$
5.	Compute the reduced gradient $r = \frac{\partial f}{\partial x_n} + u \left( \frac{\partial c}{\partial x_n} \right)$
6.	Compute the projection $h$ of the reduced gradient on space defined by the simple bounds of the nonbasic variables: $h_i = \begin{cases} 0, & \text{if } r_i < 0 \text{ and } x_{ni}^0 = l_{ni}, \\ 0, & \text{if } r_i > 0 \text{ and } x_{ni}^0 = u_{ni}, \\ r_i, & \text{otherwise} \end{cases}$
7.	If $h = 0$ , then stop
8.	One-dimensional search. Choose $\alpha > 0$ to minimize $F(x_n^0 + \alpha h)$ according to the following steps for different values of $\alpha$ : (i) $x_{ni} = \begin{cases} l_{ni}, & \text{if } x_{ni}^0 + \alpha h_i < l_{ni}, \\ u_{ni}, & \text{if } x_{ni}^0 + \alpha h_i > u_{ni}, \\ x_{ni}^0 + \alpha h_i, & \text{otherwise} \end{cases}$ (ii) Compute $x_b = c_1(x_n)$ , i.e., solve the system $c(x_b, x_n) = b$ subject to $x_b$ (iii) Compute $F(x_n^0 + \alpha h) = f(x_b, x_n)$
9.	Store the best solution in $x^0$ and go to step 2. ◆

The GRG algorithm is based on the same principle as the simplex algorithm for linear programming: elimination of the basic variables. Locally, the problem is therefore reduced to an optimization problem in nonbasic variables, which are lower and upper bounded. The main aspect of CONOPT is that it uses the sequential linearization. *Sequential linearization methods are defined as methods based on solving a sequence of subproblems with linearized constraints and an augmented Lagrangian as objective function.*

Every step of the GRG algorithm contains more details from the linear programming and the nonlinear optimization. *The optimizing steps are specialized in several versions according to whether the model appears to be almost linear or not.* For almost linear models, some of the linear algebra work involving Jacobian and basis matrices can be avoided or can be done by using cheap linear programming updating techniques. The second-order information is not needed. The line-search can be improved by observing that, like in linear programming, the optimal step will almost always be determined by the first variable that reaches a bound. In this case, the algorithm uses the *sequential linear-quadratic programming*. If the model is fairly nonlinear, other aspects can be optimized: the set of basic variables will often remain constant over several iterations and other parts of the sparse matrix algebra will take advantage of this. In this case, an improved search direction can be computed by using the specialized *sequential quadratic programming*.

### Some Details on the Algorithm

Firstly, in step 1 of Algorithm 16.2, as described by Drud (1994), CONOPT identifies equations that involve only one variable that is not fixed at its bounds. It solves the equations for these variables and removes the corresponding constraints. This procedure is applied repeatedly by removing the constraints. Secondly, CONOPT tries to guess a set of basic variables that are away from the bounds and uses this basis in some cheap Newton iterations. If the process does not converge quickly, then the constraints with large residuals or the constraints that deviate from the linear model are taken out of the process. In this way, CONOPT usually gets a point where the most constraints are satisfied. Finally, a procedure is used in which the sum of the slacks in the remaining violated constraints is minimized. This means that once a constraint has become satisfied, it remains satisfied for the rest of the optimization. Thus, by using this approach CONOPT finds a good initial basis that will lead to very few infeasible slacks and, therefore, involves a few iterations. In step 2 of the algorithm, the Jacobian matrix is computed at the points where the nonlinear constraints are satisfied. Moreover, this evaluation of the Jacobian is made only at the beginning of each line-search. This reduces the computing effort. Selection of the variables involves many tactical decisions and has crucial influence on the Jacobian matrix factorization in step 3. Step 4, where the Lagrange multipliers are determined, involves the solving of a linear system which uses the factorization of the basic matrix from step 3. For each step in the line-search, in step 8 CONOPT adjusts the basic variables to get a feasible solution. This involves repeated evaluations of the constraints residuals and solving with the basis factors. In addition, if one or more of the basic variables approach a bound, extra work is needed to find the exact stepsize at which the critical variable reaches the bound. To achieve feasibility, CONOPT uses a Newton process. If it makes a large step and the Newton procedure fails to converge, then it reduces the step and tries again. Since the derivatives are evaluated at a feasible point from which the search starts, the Newton process will always converge for small steps, ignoring degeneracy problems, taking only a few steps. In the following, let us describe some details on the steps of CONOPT.

### Selection and Factorization of the Basis

This is a very important step in the economy of the algorithm. During the optimization, the values of the estimated basic variables are calculated and checked against their bounds. This process works when the values of the basic variables are within the bounds for small changes in the nonbasic variables. Therefore, the basic variables should preferably be away from their bounds and the basis matrix should be in such a way that small changes in the numerical values corresponding to small movements do not change the solution of the system  $c(x_b, x_n) = 0$  too much. In other words, the basis matrix should be well conditioned. The basis must be factored, and the factors must be updated when the matrix changes.

Intensive numerical tests referring to the selection and to the factorization of the basis have given the following results.

- The condition of the basis is more important than its sparsity. A well-conditioned basis will generally lead to fewer iterations because the Hessian of the reduced objective function also becomes better conditioned.
- The set of the basic variables should not vary too much. In other words, from one iteration to another one, the second-order information in the reduced Hessian approximation can be preserved if the basis differs only by one variable before and after a change of the basis.

- At degenerate points, the basis must be very carefully controlled to avoid cycling. Nonlinear models seem to behave like linear models: degeneracy becomes more accentuated as the models grow in size.
- For poorly scaled problems, the factorization procedure by Hellerman and Rarick (1971, 1972) is slow as many proposed pivots appear to be too small relative to other elements in the column, thus resulting in many expensive column swaps.

Initially, in CONOPT the basis is LU factorized using a Markowitz (1957) pivot selection procedure, similar to the one suggested by Suhl and Suhl (1990, 1991), which is an improvement of the Hellerman and Rarick, where only updated spike columns are separately stored. The remaining columns of the basis factors are represented by the original columns in the Jacobian, as described in (Kalan, 1971). The main difference between the implementation given by Suhl and Suhl (1990) and the one given by Drud (1994) is that in CONOPT, a block decomposition of the bump is performed and the Markowitz criterion is applied to each individual block.

The Jacobian is not changed in the degenerate iterations. Therefore, a cheap simplex-like update of the basis factors similar to the one suggested by Reid (1982) is applied. This will reduce the length of the spikes during factorization.

The Jacobian will change during the nondegenerate iterations. To prevent unnecessary calculations of the Jacobian and expensive full factorizations after small steps, an estimation of the quality of the Jacobian is performed in CONOPT. This estimation is based on the speed of convergence of the Newton iterations in the line-search and on the change in the objective relative to the change predicted by a linear model (see Drud (1994)).

CONOPT keeps the old set of basic variables after the iterations in which no basic variable reached a bound. If the step was limited by a superbasic variable reaching a bound, CONOPT estimates the quality of the Jacobian as mentioned above. If the Jacobian is good, then CONOPT keeps the factorization of the basis. If it is poor or if the step was determined by nonlinearities, then the Jacobian is recomputed. The subsequent factorization uses the same pivot sequence and sparsity pattern determined during the last full factorization.

When the line-search behaves poorly because the Newton iterations do not converge even for small steps, CONOPT tries to find a better conditioned basis. The procedure is to replace a basic variable with a better superbasic. All the superbasic columns are updated with the active eta-vectors, and the largest possible pivot is selected. To avoid cycling, the basis is changed only if the pivot is greater than one. The heuristic is based on the assumption that a large determinant indicates a well-conditioned basis, a reasonable assumption for well-scaled problems.

### **Line-Search and Newton Iteration**

The selection of the search direction in CONOPT involves two parts:

- The selection of the set of the superbasic variables to be changed in the coming iteration
- The determination of a direction for the superbasic variables, given by the reduced gradient

Concerning the first part, the variables between their bounds are always superbatics. A variable at a bound is only made superbasic if its reduced gradient has the right sign, it is greater than an absolute tolerance and it is large relative to other reduced gradients. In the second part, two modes are identified to determine a direction for the superbatics: the *linear mode* and the *nonlinear mode*.

In the *linear mode*, it is expected that a linear approximation to the problem is a good one and that the stepsize in the coming line-search will be determined by bounds. The superbasic variables are

moved in the direction of the negative reduced gradient, i.e., in the steepest descent direction. CONOPT immediately tries to go as far as possible until a basic or a superbasic variable has reached a bound. This point is used as the initial point in the Newton iterations. The linear mode has two advantages. It helps save the overhead of the quasi-Newton procedure in areas where it is not needed and it can accelerate the line-search significantly by going directly to the bound.

*In the nonlinear mode*, the line-search is expected to be limited by the nonlinearity of the problem, i.e., it is expected that no variable will reach a bound. The search direction for the superbasic variables is determined from a quasi-Newton method that also determines the initial stepsize suggested to the line-search. For generating search directions in the nonlinear mode, a quasi-Newton method based on BFGS is used. With the growing size of the problems, some kind of conjugate gradient method is used to handle problems with a very large number of superbasics.

Firstly, CONOPT uses the linear mode and switches from the linear mode to the nonlinear mode after a number of consecutive line-searches in which no variable has reached a bound. After that, it switches back from the nonlinear to the linear mode after a number of consecutive line-searches in which a variable has reached a bound.

### The Newton Iterations

In CONOPT, the line-search has the following three components: (i) the Newton algorithm that restores the feasibility for the given values of the superbasic variables and for the given estimate of the basic variables, (ii) the computation of the initial estimates for the basic variables and the management of the bounds on the superbasic and basic variables, and (iii) the computation of the stepsize and the implementation of the termination criteria.

For each step in the one-dimensional search, the nonlinear system  $c(x_b, x_n) = b$  must be solved subject to  $x_b$ , for  $x_n$  fixed. Since this system must be solved several times, this is the most expensive component of CONOPT. It is important not to solve this system more accurately than necessary and not to spend too much time trying to solve it. Finally, the output of a line-search procedure consists of the value of the objective function and of the values of the basic variables. At all the intermediate points, only the objective function value is used to determine a new stepsize. The values of the basic variables are used to determine new initial values for the next Newton iteration. Therefore, the main criterion for stopping the iteration is based on the error  $df$ , where  $df$  is the change that would occur in the objective function if the residuals of the system  $c(x_b, x_n) = b$  were all reduced to zero. This can be approximated by  $df = \Delta x_j = e_j^T \Delta x_b = e_j^T J_b^{-1} z = u^T z$ , where  $u$  is the vector of the Lagrange multipliers (computed in step 4 of CONOPT) and  $z = b - c(x_b, x_n)$  is the current vector of the residuals. The convergence rate of the Newton algorithm is monitored as follows. If  $\|z\|_1 \leq 0.001n$  or  $\|z\|_1 (\|z\|_1 / \|z_{old}\|_1)^p < 0.001n$ , then the Newton iterates are convergent and the line-search continues ahead. Otherwise, the Newton algorithm terminates. ( $z_{old}$  is the residual from the previous iteration and  $p$  is an integer parameter which is modified along the iterations.) If the residuals are small, then the approximation of  $df$  is supposed to be good enough and therefore the test of the rapid convergence on  $df$  can be applied: if  $|df| \leq rtoj$ , then the Newton iterations are successfully stopped. Otherwise, if  $|df| (|df| / |df_{old}|)^p > rtoj$ , then the Newton iterations are stopped with failure and the initial stepsize must be decreased. This will reduce the number of the Newton iterations as well as the number of evaluations of the functions. The basic variables are computed as  $x_b = x_b + \Delta x_b$ , where  $\Delta x_b$  is the solution of the linear system  $J_b \Delta x_b = z$  which uses the same inverse as in the computation of the reduced gradient (see step 5). If this new  $x_b$  overflows a bound by more than a predefined threshold, then the Newton algorithm is stopped. No solution is expected to exist within the bounds and therefore the algorithm returns to the stepsize procedure in which a new and smaller step is chosen.

Otherwise, the corresponding basic variable is fixed at its bound. Some details including the Newton algorithm used in CONOPT are given in (Drud, 1985).

### The Linear Mode—Sequential Linear Programming

When the problem appears linear, CONOPT will often take many small steps, each determined by a new variable reaching a bound. Although the line-search is fast in the linear mode, each step requires one or more evaluations of the nonlinear constraint and the overall cost may become high relative to the progress. In order to avoid the many nonlinear constraints evaluations, CONOPT may replace the steepest descent direction with a *sequential linear programming* (SLP) technique, to find a search direction that anticipates the bounds on all the variables and therefore gives a larger expected change in the objective function in each line-search. The search direction and the last basis from the SLP procedure are used in an ordinary GRG-type line-search in which the solution is made feasible at each step. The SLP procedure is only used to generate good directions. CONOPT uses a test which determines whether SLP is used or not. This test is based on the progress of the algorithm to the solution. In other words, in CONOPT the feasibility is conserved at each iteration.

When the optimization is in the linear mode, then a steepest descent algorithm is used to determine the search direction. As an alternative, CONOPT implements the steepest edge algorithm (Reid, 1975; Goldfarb, 1976; Goldfarb, & Reid, 1977; Andrei, 1999a). The idea, taken from linear programming, is to scale the nonbasic variables according to the Euclidian norm of the updated column, the so-called *edge length*. A unit step for a nonbasic variable will give rise to changes in the basic variables proportional to the edge length. A unit step for a nonbasic variable with a large edge length will therefore give large changes in the basic variables. This step has two adverse effects relative to a unit step for a nonbasic variable with a small edge length: a basic variable is more likely to reach a bound after a very short step length and the large change in basic variables is more likely to give rise to larger nonlinear terms. The steepest edge algorithm has been very successful in linear programming (Goldfarb, & Reid, 1977). Also, this technique is profitable in nonlinear optimization, leading to fewer iterations for most nonlinear problems (Drud, 2005). However, the cost for maintaining the edge lengths can be more expensive in the nonlinear case. Therefore, the steepest edge procedure is mainly useful during the linear mode iterations.

### The Nonlinear Mode—Sequential Quadratic Programming

As we have already seen in the GRG algorithm, when the progress toward the solution is given by nonlinearities (and not by bounds), then CONOPT executes many small steps which determine small variations of the superbasic variables and also small variations of the reduced gradient. CONOPT can use the second-order information to make good progress and to determine if the bounds should be active or not. The second-order information is used in the *sequential quadratic programming* (SQP) procedure that, like in the SLP procedure, finds a good search direction and a good basis while maintaining the feasibility of the steps. Therefore, CONOPT is an algorithm which generates feasible iterations. Like in the SLP case, CONOPT uses a test which determines if it should use the SQP procedure or not. The SQP procedure can be inhibited by assigning a value to a special parameter. If the matrix of the second-order derivatives becomes too dense, then CONOPT skips over the SQP procedure. A dense matrix of the second derivatives will need more memory and will make the SQP iterations so slow that the possible saving in the number of iterations is wasted in the computation and manipulation of the Hessian. As in the linear mode where the sequential linear programming is used, in the nonlinear mode the sequential quadratic programming generates a scaled search direction and the expected step length in the following line-search is therefore 1. However, the step length may be less than 1 for several reasons: the line-search is ill-behaved, a basic variable reaches a bound before

predicted by the linear constraints, the objective is much more nonlinear along the search direction than expected, and the optimal step is not 1. For the superbasic variables, the search direction is determined by using the quasi-Newton BFGS updating.

In (Drud, 2005, 2011), plenty of aspects on CONOPT are presented and discussed, namely, hints on good model formulation, algebraic information on initial point determination, preprocessing, scaling, finding a feasible solution, linear and nonlinear modes, SLP procedure, SQP procedure, etc. A detailed comparison between sequential linearization methods (e.g., MINOS) versus CONOPT is presented in (Drud, 1994) with respect to finding the first feasible solution, maintaining the feasibility of iterations, computing the Jacobian matrix, the choice and factorization of the basis, the Lagrange multipliers computation, the line-search and the Newton iterations, etc. CONOPT represents a very advanced combination of three algorithms which integrate sequential linear programming and sequential quadratic programming, both of them being embedded in the generalized reduced gradient technology. The algorithm is improved with many computational procedures based on numerical observations obtained in the process of solving large classes of different nonlinear optimization problems of different dimensions and complexities.

### Numerical Study—CONOPT: Solving Applications from the LACOP Collection

Table 16.1 shows the performances of CONOPT in case of small-scale nonlinear optimization applications included in the LACOP collection (see Appendix C). Table 16.2 presents the performances of CONOPT for solving the application PENICI from the LACOP collection, while Table 16.3 shows the performances of CONOPT for large-scale nonlinear optimization applications.

**Table 16.1** Performances of CONOPT for solving 12 applications from the LACOP collection. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	# <i>nj</i>	# <i>njn</i>	# <i>nhd</i>	# <i>nhs</i>	# <i>it</i>	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	25	10	10	45	27	0.043	-47.7610909
ALKI	10	3	8	38	14	5	7	14	0.062	-1768.80696
PREC	8	0	6	18	14	8	8	27	0.037	3.9511634
PPSE	9	6	0	37	32	7	10	13	0.040	5055.011803
MSP3	13	0	15	50	31	3	12	36	0.038	97.5875096
MSP5	16	0	21	89	71	5	19	21	0.040	174.7869944
POOL	34	20	0	115	78	0	63	12	0.037	2569.80
TRAFO	6	0	2	19	18	4	15	16	0.038	135.0759628
LATHE	10	1	14	53	37	8	10	36	0.049	-4430.08793
DES	150	50	0	301	300	150	11175	37	0.169	1055.182314
CSTC	303	200	0	1304	1303	303	202	28	0.211	3.4800747
DIFF	396	324	0	1621	0	0	0	5	0.036	0.0

In Table 16.1, we have the following: *n* = the number of variables, *me* = the number of equality constraints, *mc* = the number of inequality constraints, #*nj* = the number of nonzeros in the Jacobian matrix, #*njn* = the number of nonlinear nonzeros in the Jacobian matrix, #*nhd* = the number of nonzero diagonal elements of the Hessian to the Lagrangian, #*nhs* = the number of nonzero sub-diagonal elements of the Hessian to the Lagrangian, #*it* = the number of iterations, *cpu* = the CPU computing time for solving the problem (seconds), *vfo* = the value of the objective function at the solution.

**Table 16.2** Performances of CONOPT for solving the PENICI application from the LACOP collection

	<i>n</i>	<i>me</i>	<i>mc</i>	# <i>nj</i>	# <i>njn</i>	# <i>nhd</i>	# <i>nhs</i>	# <i>it</i>	<i>cpu</i>	<i>vfo</i>
PENICI	707	602	0	3908	3305	303	1010	2243	20.07	113.990

**Table 16.3** Performances of CONOPT for solving 6 applications from the LACOP collection. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	# <i>nj</i>	# <i>njn</i>	# <i>nhs</i>	# <i>it</i>	<i>cpu</i>	<i>vfo</i>
HANG	1002	501	0	3504	1503	501	22	0.27	5.0685777
	2002	1001	0	7004	3003	1001	19	0.379	5.0685101
FLOW	1163	735	0	4979	312	0	5	0.058	0.0
FLOWO	1556	1005	0	6714	400	0	7	0.078	0.0
POL	4004	3000	0	18002	8000	0	33	0.232	14.139480
	6004	4500	0	27002	12000	0	38	0.525	14.142097
	8004	6000	0	36002	16000	0	41	0.760	14.143172
	10004	7500	0	45002	20000	0	45	1.055	14.141714
CAT	3003	2000	0	12003	12000	2002	82	0.763	-0.048055
	6003	4000	0	24003	24000	4002	81	2.136	-0.048055
	9003	6000	0	36003	36000	6002	32	1.242	-0.048053
CONT	2505	2000	0	10005	4	0	21	0.109	1.013238
	5005	4000	0	20005	4	0	30	0.270	1.005922
	7505	6000	0	30005	4	0	31	0.480	1.004561
	10005	8000	0	40005	4	0	33	0.767	1.004071

**Table 16.4** Comparison between KNITRO/ACTIVE and CONOPT

	# <i>itt</i>	<i>cput</i>
KNITRO/ACTIVE	1123	466.084
CONOPT	520	9.124

It is important to make comparisons. The numerical experiments presented in Table 15.7 (KNITRO/ACTIVE) (option 3) and Table 16.3 (CONOPT) for solving all the 15 large-scale nonlinear optimization applications are used in Table 16.4, which shows the total number of iterations (#*itt*) and the total CPU computing time (*cput*), (seconds) needed by KNITRO/ACTIVE and CONOPT, respectively, to get a solution of these applications.

It is worth noting the behavior of CONOPT when the procedure of the search direction computed by the sequential quadratic programming is inhibited. In other words, the search direction is determined by means of the reduced gradient. Table 16.5 contains the performances of CONOPT with the sequential quadratic programming inhibited.

A comparison between CONOPT with sequential quadratic programming (Table 16.3) versus CONOPT without sequential quadratic programming (Table 16.5) is given in Table 16.6, where #*itt* is the total number of iterations and *cput* is the total CPU computing time (seconds) for solving all the 15 large-scale nonlinear optimization applications.

Observe that CONOPT equipped with sequential quadratic programming for the line-search determination is much better than its variant without sequential quadratic programming. A simple analysis of Tables 16.3 and 16.5 shows that the nonlinear optimization applications in the linear mode (POL and CONT) are indifferent subject to the second-order information used by the SQP procedure.

Table 16.7 includes the performances of CONOPT with SQP for solving the application HANG with a large number of variables.

Table 16.8 shows the performances of CONOPT with SQP inhibited for solving the application HANG with a large number of variables.

**Table 16.5** Performances of CONOPT for solving 6 applications from the LACOP collection with SQP inhibited. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	<i>cpu</i>	<i>vfo</i>
HANG	1002	501	0	81	0.269	5.6857779
	2002	1001	0	151	0.838	5.0685101
FLOW	1163	735	0	5	0.058	0.0
FLOWO	1556	1005	0	7	0.066	0.0
POL	4004	3000	0	33	0.250	14.139480
	6004	4500	0	38	0.507	14.142096
	8004	6000	0	41	0.759	14.143172
	10004	7500	0	45	1.060	14.141714
CAT	3003	2000	0	283	4.965	-0.048055
	6003	4000	0	157	3.108	-0.048055
	9003	6000	0	75	3.441	-0.048053
CONT	2505	2000	0	21	0.113	1.013238
	5005	4000	0	30	0.275	1.005922
	7505	6000	0	31	0.465	1.004561
	10005	8000	0	33	0.787	1.004071

**Table 16.6** CONOPT with SQP versus CONOPT without SQP

	#itt	cput
CONOPT with SQP	520	9.124
CONOPT without SQP	1031	16.961

**Table 16.7** Performances of CONOPT with SQP for solving the HANG application from the LACOP collection

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	<i>cpu</i>	<i>vfo</i>
HANG	4002	2001	0	28	1.366	5.0684889
	8002	4001	0	31	4.766	5.0684827

**Table 16.8** Performances of CONOPT without SQP for solving the HANG application from the LACOP collection

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	<i>cpu</i>	<i>vfo</i>
HANG	4002	2001	0	769	6.311	5.0684889
	8002	4001	0	1658	24.901	5.0684828

**Table 16.9** Comparison: MINOS, KNITRO/ACTIVE, SNOPT and CONOPT. CPU computing time (seconds)

MINOS	KNITRO/ACTIVE	SNOPT	CONOPT
143.80	466.084	64.91	9.124

As a conclusion to these numerical studies, let us compare MINOS (Table 14.12), KNITRO/ACTIVE (Table 15.7), SNOPT (Table 15.2), and CONOPT with SQP (Table 16.3) subject to the total CPU computing time. Table 16.9 presents the performances of these algorithms for solving all the 15 large-scale nonlinear optimization applications with respect to the total CPU computing time.

CONOPT is one of the most elaborated algorithms for solving large-scale nonlinear optimization problems and applications. It is based on the generalized reduced gradient implemented in the sparse matrix technology by using sequential linear programming and sequential quadratic programming algorithms. The best performances are obtained with its variant which implements the sequential quadratic programming. CONOPT has one of the most advanced techniques for solving linear algebraic systems of equations based on permuting the rows and columns of the matrix to the “bump and spike” structure in order to minimize the number of the newly created nonzeros during the factorization and to maintain the numerical stability.

### Notes and References

CONOPT is a line-search algorithm described in a number of papers by Arne Drud (1976, 1983, 1985, 1994, 1995, 1996, 2005, 2011)—ARKI Consulting and Development A/S. Each of them gives different computational details on the numerical linear algebra used in CONOPT. The algorithm implements three active-set methods. The first one is a gradient projection method in the frame of the generalized reduced gradient method that projects the gradient of the objective function onto a linearization of the constraints. The second is a sequential linear programming algorithm and the third is a sequential quadratic programming algorithm. CONOPT includes algorithmic switches that automatically detect which method is the best. The search direction is computed in two modes. In the linear mode, the progress toward solution is given by bounds. In the nonlinear mode, this progress is given by nonlinearities. CONOPT is embedded in the GAMS technology (Andrei, 2017c).

For solving constrained nonlinear optimization problems, some approaches based on the gradient have been designed. Among them, the most well known are the projected gradient method proposed by J.B. Rosen (1960, 1961), the feasible direction method originally suggested and developed by Guus Zoutendijk (1960), the reduced gradient method originally proposed by Philip Wolfe (1967) for problems with linear constraints, the convex simplex method for problems with linear constraints proposed by Willard Zangwill (1967), and the generalized reduced gradient method by Abadie and Carpentier (1969). All these methods borrow some ideas from the simplex algorithm from linear programming by changing the basis in specific ways and by taking into account the nonlinear character of the objective and of the constraints.

Thus, the projected gradient method determines an initial basis by using the simplex algorithm. At every iteration, this matrix is changed by eliminating and by introducing only one row from the matrix of constraints. The search direction is obtained by projecting the negative gradient on the null subspace of the basis. The convex simplex method changes the basis by modifying only one variable. The search direction is computed by projection in three distinct cases. On the other hand, the reduced gradient method changes the basis by simultaneously modifying more variables. In this method, the search direction is obtained as the reduced gradient of the minimizing function at the subspace of the nonbasic variables. The generalized reduced gradient method solves the original problem by solving a sequence of nonlinear optimization subproblems with simple bounds specified in the subspace of the nonbasic variables defined by the nonlinear constraints. The weakness of all these methods is that they use only the information given by the projection of the negative gradient onto the null space of the basis or the reduced gradient making no attempt to use the approximation of the minimizing problem by sequential linear or by sequential quadratic programming in the current point. The most advanced is the generalized reduced gradient method with linear or sequential quadratic programming. The main components included only in this generalized reduced gradient method are an advanced inverse of the basis representation under the product form of the inverse, advanced Newton methods for solving nonlinear algebraic systems of equations, and the use of the sequential linear programming or the sequential quadratic programming according to the situations in which the model appears to be almost linear or not. This is one of the modern approaches for solving large-scale nonlinear optimization problems.



## Interior-Point Methods

17

One of the most powerful methods for solving nonlinear optimization problems known as the *interior-point method* is to be presented in this chapter. It is related to the barrier functions. The terms “interior-point methods” and “barrier methods” have the same significance and may be used interchangeably. The idea is to keep the current points in the interior of the feasible region. A method for remaining in the interior of the feasible region is to add a component to the objective function, which penalizes close approaches to the boundary. This method was first suggested by Frisch (1955) and developed both in theoretical and computational details by Fiacco and McCormick (1964, 1966, and 1968).

Interior-point methods have two implementations. The first one uses the line-search technique and the second one uses filters. The interior-point methods based on line-search are also classified into two methods. The first one is a direct extension of the interior-point methods from linear programming. They use line-searches to enforce convergence. The second one uses a quadratic model to define the step and incorporates a trust-region constraint to provide stability. For these methods, we follow the main developments from Nocedal and Wright (2006). The interior-point method with filter uses a filter for selecting the stepsize in the direction of the calculated step (Fletcher, & Leyffer, 2002).

In this chapter, we consider the interior-point method for general nonlinear optimization problems in the line-search and in the trust-region paradigms, respectively. At the same time, a variant of the interior-point algorithm is presented, which illustrates a *methodology for the theoretical development of interior-point methods* and of their convergence analysis.

Let us consider the problem

$$\min f(x) \quad (17.1a)$$

subject to

$$c_E(x) = 0, \quad (17.1b)$$

$$c_I(x) - s = 0, \quad (17.1c)$$

$$s \geq 0, \quad (17.1d)$$

where  $x \in \mathbb{R}^n$ ,  $c_E : \mathbb{R}^n \rightarrow \mathbb{R}^l$ ,  $c_I : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , are twice continuously differentiable functions, and  $s \in \mathbb{R}^m$  are slack variables introduced to transform the inequality constraints  $c_I(x) \geq 0$  into equalities. Here,  $E \triangleq \{1, \dots, l\}$  and  $I \triangleq \{1, \dots, m\}$ .

As described in Nocedal and Wright (2006), in the following we present two derivations of interior-point methods, which emphasize the main idea and the terminology for solving general

nonlinear optimization problems (17.1). The first derivation is based on *continuation* or *homotopy methods*. The second one is based on *barrier methods*.

*Homotopy methods.* By *homotopy method*, we understand a continuous transformation of a mathematical object into another one, in the sense that a simple (an easy) problem can be continuously deformed into the given (hard) problem (Henri Poincaré). The solutions to the deformed problems are related and can be tracked as the deformation proceeds. The function describing the deformation is called a homotopy map. The basic idea of the homotopy approach is to replace the original problem with a parameterized problem. At one end of the parameter range, there is a problem which is easy to solve, while the original problem is at the other end.

The KKT optimality conditions for the problem (17.1) are as follows:

$$\nabla f(x) - \nabla c_E(x)^T y - \nabla c_I(x)^T z = 0, \quad (17.2a)$$

$$Sz - \mu e = 0, \quad (17.2b)$$

$$c_E(x) = 0, \quad (17.2c)$$

$$c_I(x) - s = 0, \quad (17.2d)$$

with  $\mu = 0$ , together with

$$s \geq 0, z \geq 0. \quad (17.2e)$$

In (17.2),  $\nabla c_E(x)$  and  $\nabla c_I(x)$  are the Jacobian matrices of the equality and inequality constraints, respectively. The vectors  $y$  and  $z$  are the Lagrange multipliers associated to these constraints. Let us define  $S$  and  $Z$  as diagonal matrices whose diagonal elements are the components of the vectors  $s$  and  $z$ , respectively. Observe that (17.2) with  $\mu$  strictly positive are the *perturbed KKT conditions*. The Eq. (17.2b) with  $\mu = 0$  and the conditions (17.2e) introduce a combinatorial aspect of determining the optimal active-set of (17.1). This is the main difficulty of this approach based on the KKT conditions. The homotopy method consists in approximately solving the KKT conditions (17.2) for a sequence of positive parameters  $\{\mu_k\}$  that converges to zero, while maintaining  $s, z > 0$ . In the limit, we hope to obtain a point that satisfies the KKT conditions (17.2). Moreover, asking for the iterates to decrease a merit function or to be acceptable to a filter, we hope that the iterations converge to a minimizer of (17.1) and not simply to a KKT point. The justification of this method is as follows. In a neighborhood of  $(x^*, s^*, y^*, z^*)$  which satisfies the linear independence of the constraint qualification (Remark 11.2), the strict complementarity condition (see Definition 11.23), and the second-order sufficient conditions (see Theorem 11.16), then, for all sufficiently small positive values of  $\mu$ , the system (17.2) has a locally unique solution denoted by  $(x(\mu), s(\mu), y(\mu), z(\mu))$ . All these points describe the so-called *primal-dual central path*. When  $\mu \rightarrow 0$ , this converges to  $(x^*, s^*, y^*, z^*)$ , solution of the KKT conditions given by (17.2).

*Barrier methods.* In this approach of the derivation of the interior-point methods, a barrier problem is associated to (17.1) as

$$\min_{x,y} f(x) - \mu \sum_{i=1}^m \log(s_i), \quad (17.3a)$$

subject to

$$c_E(x) = 0, \quad (17.3b)$$

$$c_I(x) - s = 0, \quad (17.3c)$$

where  $\mu$  is a positive parameter. Note that the inequality  $s \geq 0$  is not included in (17.3) because the minimization of the barrier term  $-\mu \sum_{i=1}^m \log(s_i)$  from (17.3a) determines the components of  $s$  not to become too close to zero. Observe that the reformulation by using (17.3) avoids the combinatorial aspect of the nonlinear programs, but, as we can see, its solution does not coincide with the solution of (17.1) for  $\mu > 0$ . In this context, the barrier approach consists in finding an approximate solution of the barrier problem (17.3) for a positive sequence  $\{\mu_k\}$  that converges to zero.

It is worth comparing these two approaches based on the KKT conditions and on the barrier problem. For this, let us write the KKT conditions for (17.3) as

$$\nabla f(x) - \nabla c_E(x)^T y - \nabla c_I(x)^T z = 0, \quad (17.4a)$$

$$-\mu S^{-1} e + z = 0, \quad (17.4b)$$

$$c_E(x) = 0, \quad (17.4c)$$

$$c_I(x) - s = 0. \quad (17.4d)$$

Looking at (17.2) and (17.4), we observe that they differ only in the second equations (17.2b) and (17.4b). Notice that (17.4b) becomes very nonlinear near the solution as  $s \rightarrow 0$ . To eliminate this difficulty, it is advantageous for the Newton method to transform the rational equation (17.4b) into a quadratic one. This is done by multiplying (17.4b) by  $S$ , thus obtaining the quadratic term  $Sz$ . This is always possible because all the diagonal elements of  $S$  are positive. With this simple transformation, the KKT conditions for the barrier problem coincide with the perturbed KKT conditions given by (17.2). We emphasize that the presence of the quadratic term  $Sz$  in (17.4), the simplest possible nonlinearity introduced in the KKT system, makes the barrier method with all its defects be nontrivial for solving the general nonlinear optimization problems (17.1).

Both the homotopy and the barrier interpretation of nonlinear optimization problems are useful. The homotopy gives rise to the definition of the primal-dual directions, whereas the barrier view is crucial in the design of globally convergent algorithms.

We mention that the interior-point derives from the fact that the early barrier methods introduced by Fiacco and McCormick (1968) did not use slack variables and assumed that the initial point is feasible with the inequality constraints. These methods used the barrier function  $f(x) - \sum_{i \in I} \log(c_i(x))$  to prevent the iterates from leaving the feasible region defined by inequalities (see the logarithmic penalty-barrier function (14.53) used in the penalty-barrier algorithm SPENBAR).

## 17.1 Prototype of the Interior-Point Algorithm

Let us consider the Lagrange function associated to the problem (17.1) as

$$L(x, s, y, z) = f(x) - y^T c_E(x) - z^T (c_I(x) - s). \quad (17.5)$$

Now, applying the Newton method to the nonlinear system (17.2), in the variables  $x, s, y, z$ , the following linear algebraic system is obtained:

$$\begin{bmatrix} \nabla_{xx}^2 L & 0 & -J_E^T(x) & -J_I^T(x) \\ 0 & Z & 0 & S \\ J_E(x) & 0 & 0 & 0 \\ J_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - J_E^T(x)y - J_I^T(x)z \\ Sz - \mu e \\ c_E(x) \\ c_I(x) - s \end{bmatrix}, \quad (17.6)$$

where  $J_E(x) = \nabla c_E(x) \in \mathbb{R}^{I \times n}$  and  $J_I(x) = \nabla c_I(x) \in \mathbb{R}^{m \times n}$ , are the Jacobian matrices of the equality and inequality constraints, respectively.

The system (17.6) is called the *primal-dual system*. If it is solved, we get a solution  $d = (d_x, d_s, d_y, d_z)$  with which a new iteration  $(x^+, s^+, y^+, z^+)$  can be computed as

$$x^+ = x + \alpha_s^{\max} d_x, \quad s^+ = s + \alpha_s^{\max} d_s, \quad (17.7a)$$

$$y^+ = y + \alpha_z^{\max} d_y, \quad z^+ = z + \alpha_z^{\max} d_z, \quad (17.7b)$$

where

$$\alpha_s^{\max} = \max \{ \alpha \in (0, 1] : s + \alpha d_s \geq (1 - \tau)s \}, \quad (17.8a)$$

$$\alpha_z^{\max} = \max \{ \alpha \in (0, 1] : z + \alpha d_z \geq (1 - \tau)z \}, \quad (17.8b)$$

with  $\tau \in (0, 1)$ . The relations (17.8) for determining the stepsize, called the *fraction to the boundary rule*, are designed to prevent the variables  $s$  and  $z$  from approaching their bounds of zero too quickly. A typical value of  $\tau$  is 0.995.

To establish the prototype of the interior-point algorithm an *error function* based on the perturbed KKT system (17.2) is introduced

$$E(x, s, y, z, \mu) = \max \left\{ \|\nabla f(x) - J_E(x)^T y - J_I(x)^T z\|, \|Sz - \mu e\|, \|c_E(x)\|, \|c_I(x) - s\| \right\}, \quad (17.9)$$

where  $\|\cdot\|$  is an arbitrary vector norm.

This simple procedure described above is the basis of all the modern interior-point methods. However, in order to get efficient algorithms, various modifications are needed. For example, a major factor is the procedure for solving the primal-dual linear system (17.6). Another important procedure is to choose the sequence of the barrier parameters  $\{\mu_k\}$ . Of course, the determination of the initial point for starting the algorithm is crucial, as is the procedure for the approximation of the Hessian of the Lagrangian  $\nabla_{xx}^2 L$ .

### Algorithm 17.1 Prototype of the interior-point algorithm

- |    |   |
|----|---|
| 1. | Choose the initial elements $x_0$ and $s_0 > 0$ . Compute the initial values of the multipliers $y_0$ and $z_0 > 0$ . Choose an initial value for the barrier parameter $\mu_0 > 0$ , as well as the parameters $\sigma, \tau \in (0, 1)$ . Set $k = 0$ |
| 2. | If a stopping test for the nonlinear optimization problem (17.1) is satisfied, stop; otherwise, go to step 3  |
| 3. | If $E(x_k, s_k, y_k, z_k, \mu_k) > \mu_k$ , then go to step 4; otherwise, go to step 8  |
| 4. | Solve the primal-dual system (17.6) with respect to the searching direction $d = (d_x, d_s, d_y, d_z)$  |
| 5. | Using (17.8), compute $\alpha_s^{\max}$ and $\alpha_z^{\max}$   |
| 6. | Using (17.7), compute a new estimation of the solution $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$   |
| 7. | Set $\mu_{k+1} = \mu_k$ , $k = k + 1$ and go to step 3  |
| 8. | Choose $\mu_k \in (0, \sigma \mu_k)$ and go to step 2   |

◆

**Theorem 17.1** Let  $f(x)$ ,  $c_E(x)$ , and  $c_I(x)$  be continuously differentiable functions. Suppose that the Algorithm 17.1 generates a sequence of iterates  $\{x_k\}$  and that  $\{\mu_k\} \rightarrow 0$ . Then all the limit points  $\hat{x}$  of the sequence  $\{x_k\}$  are feasible. Moreover, if any limit point  $\hat{x}$  of the sequence  $\{x_k\}$  satisfies the linear independence constraint qualification (LICQ), then the first-order optimality conditions of the problem (17.1) hold at  $\hat{x}$ .

**Proof** To simplify the proof, as in (Nocedal, & Wright, 2006), suppose that the problem contains only the inequality constraints  $c_l(x)$  and denote them as  $c(x)$ . Let  $\hat{x}$  be a limit point of the sequence  $\{x_k\}$  and consider a subsequence  $\{x_{k_l}\}$  convergent to  $\hat{x}$ , i.e.,  $\{x_{k_l}\} \rightarrow \hat{x}$ . If  $\mu_k \rightarrow 0$ , then the error function  $E$  given by (17.9) converges to zero, that is  $(c_{k_l} - s_{k_l}) \rightarrow 0$ . By the continuity of  $c$ , this implies that  $\hat{c} \triangleq c(\hat{x}) \geq 0$ , i.e.,  $\hat{x}$  is feasible and  $s_{k_l} \rightarrow \hat{s} = \hat{c}$ . Now, suppose that the linear independence constraint qualification holds at  $\hat{x}$ . Let  $A = \{i : \hat{c}_i = 0\}$  be the set of the active constraints at  $\hat{x}$ . For any  $i \notin A$ , it follows that  $\hat{c}_i > 0$  and  $\hat{s}_i > 0$ . Therefore, by the complementarity condition (17.2b) it follows that  $[z_{k_l}]_i \rightarrow 0$ . Since  $\nabla f(x_{k_l}) - J(x_{k_l})^T z_{k_l} \rightarrow 0$ , we obtain

$$\nabla f(x_{k_l}) - \sum_{i \in A} [z_{k_l}]_i \nabla c_i(x_{k_l}) \rightarrow 0. \quad (17.10)$$

But, by the constraint qualification hypothesis LICQ, the vectors  $\{\nabla \hat{c}_i : i \in A\}$  are linear independent. Hence, from (17.10) and from the continuity of  $\nabla f(\cdot)$  and  $\nabla c_i(\cdot)$ ,  $i \in A$ , it follows that the positive sequence  $\{z_{k_l}\}$  converges to a value  $\hat{z} \geq 0$ . Taking the limit in (17.10) we get

$$\nabla f(\hat{x}) = \sum_{i \in A} \hat{z}_i \nabla c_i(\hat{x}).$$

Moreover, we have  $\hat{c}^T \hat{z} = 0$ , thus completing the proof.  $\diamond$

Observe that the theoretical basis of the interior-point methods is quite accessible. All that is required is the continuity of the functions of the problem and of their gradients, as well as the LICQ hypothesis. This is the reason why these methods have very good behavior in solving large-scale nonlinear optimization problems.

As we have already said, the interior-point methods are classified into two classes. *The first one is based on the Algorithm 17.1 which is completed by adding a linear search, a strict control of the reduction of the slack variables  $s$  and of the multipliers  $z$ , as well as the modification of the primal-dual system (17.6) when negative curvature is encountered. The second class computes the step by minimizing a quadratic model of the barrier problem (17.3) with respect to a trust-region constraint.* Both these approaches have many points in common. In the following lines, we describe some aspects of the algorithmic development. Many other details may be found in (Nocedal, & Wright, 2006), (Sun, & Yuan, 2006), (Forsgren, Gill & Wright, 2002), or (Wright, 1991).

## 17.2 Aspects of the Algorithmic Developments

This section presents some modifications and extensions of the Algorithm 17.1 that enable it to solve large-scale nonconvex problems starting from any initial point.

### Solving the Primal-Dual System

The most important issue is the linear algebra used for solving a large, sparse linear system at each iteration in order to find the search direction. Actually, the sparse linear algebra module is by far the most difficult component of Algorithm 17.1 (step 4). The system (17.6) can be rewritten in symmetric form as

$$\begin{bmatrix} \nabla_{xx}^2 L & 0 & J_E^T(x) & J_I^T(x) \\ 0 & \Sigma & 0 & -I \\ J_E(x) & 0 & 0 & 0 \\ J_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ -d_y \\ -d_z \end{bmatrix} = -\begin{bmatrix} \nabla f(x) - J_E^T(x)y - J_I^T(x)z \\ z - \mu S^{-1}e \\ c_E(x) \\ c_I(x) - s \end{bmatrix}, \quad (17.11)$$

where  $\Sigma = S^{-1}Z$ . Clearly, this symmetric form is suitable for using symmetric linear system solvers which reduce the computational effort at each iteration. For solving the system (17.11), we can use the *direct methods* (the factorization of the coefficient matrix, the Schur complement method, or the null space method), as well as the *iterative methods* (the conjugate gradient method).

Observe that the system (17.11) can be further reduced by the elimination of the variable  $d_s = -\Sigma^{-1}d_z + \Sigma^{-1}(-z + \mu S^{-1}e) = -\Sigma^{-1}d_z - \Sigma^{-1}z + \mu Z^{-1}e$ , thus obtaining

$$\begin{bmatrix} \nabla_{xx}^2 L & J_E^T(x) & J_I^T(x) \\ J_E(x) & 0 & 0 \\ J_I(x) & 0 & -\Sigma^{-1} \end{bmatrix} \begin{bmatrix} d_x \\ -d_y \\ -d_z \end{bmatrix} = -\begin{bmatrix} \nabla f(x) - J_E^T(x)y - J_I^T(x)z \\ c_E(x) \\ c_I(x) - s + \Sigma^{-1}z - \mu Z^{-1}e \end{bmatrix}. \quad (17.12)$$

This system can be factorized by the *symmetric indefinite factorization* method (Bunch, & Parlett, 1971), (Bunch, & Kauffman, 1977), (Cheng, 1998), (Cheng, & Higham, 1998). Let us denote the matrix from (17.12) by  $M$ , then this factorization computes  $P^T M P = L B L^T$ , where  $L$  is a lower triangular matrix,  $B$  is a block-diagonal matrix with  $1 \times 1$  or  $2 \times 2$  blocks and  $P$  is a permutation matrix for conserving the sparsity and for ensuring the numerical stability.

The system (17.12) can be further reduced by the elimination of the variable  $d_z = -\Sigma J_I(x)d_x - \Sigma c_I(x) + \Sigma s - z + \mu Z^{-1}e$ , thus obtaining

$$\begin{bmatrix} \nabla_{xx}^2 L + J_I^T(x)\Sigma J_I(x) & J_E^T(x) \\ J_E(x) & 0 \end{bmatrix} \begin{bmatrix} d_x \\ -d_y \end{bmatrix} = -\begin{bmatrix} \nabla f(x) - J_E^T(x)y - J_I^T(x)z + J_I^T(x)(\Sigma c_I(x) - \Sigma s + z - \mu Z^{-1}e) \\ c_E(x) \end{bmatrix}. \quad (17.13)$$

If the number of the inequality constraints is large, then the dimension of the system (17.13) is smaller than the dimension of the system (17.12). Moreover, the presence of the matrix  $J_I^T(x)\Sigma J_I(x)$  in (17.13) dramatically modifies the structure of the sparse matrix of the Hessian  $\nabla_{xx}^2 L$ , which can be rather unwanted for large-scale problems. A particular, favorable case is when the matrix  $J_I^T(x)\Sigma J_I(x)$  is diagonal, case which appears when all the constraints are simple bounds.

We emphasize that the symmetric primal-dual system (17.11) or its reduced forms (17.12) or (17.13) are ill-conditioned. The ill-conditioning is given by the presence of the matrix  $\Sigma = S^{-1}Z$ . When  $\mu \rightarrow 0$ , some elements of  $\Sigma$  tend to  $+\infty$ , while others tend to zero. However, because of the special form in which this ill-conditioning arises, the direction computed by a *stable factorization method* is usually accurate enough. Errors can appear only when the slacks  $s$  or the multipliers  $z$  become very close to zero or when the Hessian  $\nabla_{xx}^2 L$  or the Jacobian matrix  $J_E(x)$  are almost rank deficient. This is the reason why the direct factorization methods are considered the most efficient and reliable for computing the steps in the interior-point methods.

Cholesky factorization followed by triangular substitutions is a remarkable stable method to compute solutions of linear systems with symmetric positive definite coefficient matrix. Usually, Cholesky factorization of a sparse matrix produces *fill-in* that is some lower triangular locations in the

Cholesky factor contain nonzero elements even though the same locations in the original matrix are zero (Ng, & Peyton, 1993). The problem of finding ordering (permutations) that reduces the fill-in is too complicated, in fact being NP-complete (Rose, & Tarjan, 1975). Possible the best-known class of ordering heuristics is the *minimum degree* (George, & Liu, 1989), (Mészáros, 1995). Other ordering includes the *minimum local fill* and *nested-dissection orderings* (Rothberg, & Hendrickson, 1996).

Iterative techniques for solving the linear systems (17.11), (17.12), or (17.13) can also be used for the search direction computation. In this case, preconditioners that cluster the eigenvalues of  $\Sigma$  must be used (Nocedal, & Wright, 2006). For example, by introducing in (17.11) the change of variables  $\tilde{d}_s = S^{-1}d_s$  and multiply the second equation from (17.11) by  $S$ , then the term in  $\Sigma$  is transformed in  $S\Sigma S$ . Now, as  $\mu \rightarrow 0$ , since  $\Sigma = S^{-1}Z$ , it follows that all the elements of  $S\Sigma S$  cluster around  $\mu I$ . Another scaling like  $\tilde{d}_s = \Sigma^{1/2}d_s$  provides a perfect preconditioner. Indeed,  $\tilde{d}_s = \sqrt{\mu}S^{-1}d_s$  transforms  $\Sigma$  to  $S\Sigma S/\mu$ , which converges to  $I$  when  $\mu \rightarrow 0$ . To be efficient and robust, besides preconditioning that removes the ill-conditioning determined by the barrier approach, the iterative methods must be protected from possible ill-conditioning caused by the Hessian  $\nabla_{xx}^2 L$  or the Jacobian matrices  $J_E(x)$  and  $J_I(x)$ .

### Inertia and Singularity

It is known that the inertia of a matrix is a triplet of integers, which identifies the number of positive, negative, and zero eigenvalues of the matrix. For the primal-dual system (17.11), the step  $p$  is a descent direction if the matrix

$$\begin{bmatrix} \nabla_{xx}^2 L & 0 \\ 0 & \Sigma \end{bmatrix} \quad (17.14)$$

is positive definite on the null space of the matrix

$$\begin{bmatrix} J_E(x) & 0 \\ J_I(x) & -I \end{bmatrix}. \quad (17.15)$$

The positive definiteness condition holds if the inertia of the primal-dual matrix (17.11) is given by  $(n + m, l + m, 0)$ , where  $n$  is the number of variables,  $m$  is the number of inequality constraints, and  $l$  is the number of equality constraints of the problem (17.1). *In other words, if this matrix has exactly  $n + m$  positive eigenvalues,  $l + m$  negative eigenvalues and no zero eigenvalues, then the positive definiteness condition holds.*

If the primal-dual matrix (17.11) does not satisfy this inertia condition, then this can be modified as follows. Observe that the diagonal matrix  $\Sigma$  is positive definite by construction, but  $\nabla_{xx}^2 L$  can be indefinite. Therefore,  $\nabla_{xx}^2 L$  can be replaced by the matrix  $\nabla_{xx}^2 L + \delta I$ , where  $\delta > 0$  is a sufficiently large parameter to ensure that the inertia of the modified matrix is exactly  $(n + m, l + m, 0)$ .

Another problem that we face is when the matrix of the primal-dual system (17.11) is singular, caused by the rank deficiency of the matrix  $J_E$ . Observe that the matrix  $[J_I \ -I]$  is always of full rank. To overcome this singularity of the Jacobian  $J_E$ , in the matrix of the primal-dual system (17.11) a regularization parameter  $\gamma > 0$  is introduced. Hence, taking into consideration the correction of inertia and the protection to the singularity of the Jacobian of the equality constraints, the matrix of the primal-dual system (17.11) can be modified as

$$\begin{bmatrix} \nabla_{xx}^2 L + \delta I & 0 & J_E^T(x) & J_I^T(x) \\ 0 & \Sigma & 0 & -I \\ J_E(x) & 0 & -\gamma I & 0 \\ J_I(x) & -I & 0 & 0 \end{bmatrix}. \quad (17.16)$$

Suppose that we have the current value of the barrier parameter  $\mu$ , the constants  $\eta > 0$  and  $\beta < 1$  and the value of the perturbation parameter  $\delta_{old}$  from the previous iteration. Then, a procedure for selecting the parameters  $\gamma$  and  $\delta$  is as follows.

### Algorithm 17.2 Inertia correction and regularization

1. In the matrix (17.16), set  $\delta = \gamma = 0$ . Factorize the matrix (17.16)
2. If the matrix (17.16) is nonsingular and its inertia is  $(n+m, l+m, 0)$ , then compute the primal-dual step; stop
3. If the matrix (17.16) has zero eigenvalues, then set  $\gamma = 10^{-8}\eta\mu^\beta$
4. If  $\delta_{old} = 0$ , then set  $\delta = 10^{-4}$ ; otherwise, set  $\delta = \delta_{old}/2$
5. Factorize the modified matrix (17.16)
6. If the inertia of this matrix is  $(n+m, l+m, 0)$ , then set  $\delta_{old} = \delta$ . Compute the primal-dual step by solving the system (17.11) with modified matrix; stop. Otherwise, set  $\delta = 10\delta$  and go to step 5 ◆

The Algorithm 17.2 is a variant established by Nocedal and Wright (2006) of a more elaborated algorithm given by Wächter and Biegler (2006). The constants used in it are arbitrary. The purpose of the algorithm is to drastically avoid modifications of the Hessian  $\nabla_{xx}^2 L$  while minimizing the number of factorizations. Severe modifications of the matrix of the primal-dual system (17.11) deteriorate the performance of the algorithm because the second-order information given by  $\nabla_{xx}^2 L$  is modified and, in this case, the algorithm behaves like a steepest descent. The algorithm for inertia correction and regularization is used at each iteration of the interior-point algorithm.

### Update of the Barrier Parameter

It is important to update the barrier parameter in a proper way. The sequence of  $\{\mu_k\}$  must converge to zero, so in the limit, we can recover the solution of the problem (17.1). But, if  $\mu_k$  is decreased too slowly, then a large number of iterations is required by the algorithm. On the other hand, if  $\mu_k$  is decreased too quickly, then some slack variables  $s$  or multipliers  $z$  may approach zero prematurely, thus slowing the progress of the algorithm to the solution of the problem. In the following, let us present some strategies for the barrier parameter selection, which prove to be efficient in practice.

- The *Fiacco-McCormick strategy* fixes the values of the barrier parameter until the perturbed KKT conditions (17.2) have been satisfied with some accuracy. Then the barrier parameter is decreased as

$$\mu_{k+1} = \sigma_k \mu_k, \quad \sigma_k \in (0, 1). \quad (17.17)$$

The parameter  $\sigma_k$  can be selected as a constant value, for example  $\sigma_k = 0.2$ . However, it is preferable to have a strategy for the selection of  $\sigma_k$ . One idea is to choose smaller and smaller values for  $\sigma_k$  when the solution is being approached. If close to the solution, consider that  $\sigma_k \rightarrow 0$  and the parameter  $\tau$  in (17.8) converges to 1. Then the interior-point algorithm has a superlinear convergence

rate. The Fiacco-McCormic strategy works well on many problems, but it is sensitive to the initial point selection and to the scaling of the problem.

- (b) The *adaptive strategies* for updating the barrier parameter are more robust in the case of nonlinear problems. These strategies are based on complementarity exactly as in the linear programming, and they modify the barrier parameter at each iteration  $k$  as

$$\mu_{k+1} = \sigma_k \frac{s_k^T z_k}{m}. \quad (17.18)$$

A strategy for selecting  $\sigma_k$  in (17.18) is based on the interior-point methods in linear programming. Firstly, with  $\mu = 0$  in (17.11) we solve this system to get the *predictor direction*  $(\Delta x^{af}, \Delta s^{af}, \Delta y^{af}, \Delta z^{af})$ . With this, as in (17.8), determine  $\alpha_s^{af}$  and  $\alpha_z^{af}$  as the longest step that can be taken along the affine scaling direction before violating the nonnegativity conditions  $(s, z) \geq 0$ . These values are used to compute  $\mu_{af}$  as the value of the complementarity along the affine scaling step, i.e.,

$$\mu_{af} = \frac{(s_k + \alpha_s^{af} \Delta s^{af})^T (z_k + \alpha_z^{af} \Delta z^{af})}{m}. \quad (17.19)$$

With this, compute  $\sigma_k$  as

$$\sigma_k = \left( \frac{\mu_{af}}{s_k^T z_k / m} \right)^3. \quad (17.20)$$

### Merit Functions and Filters for Step Acceptance

Let us assume that we have the primal-dual direction. The next step is to determine a stepsize along this direction. For this, the merit functions or filters can be used. Since the interior-point methods can be considered as methods for solving the barrier problem (17.3), then it is quite natural to define the merit function  $\Phi$  or the filter in terms of barrier functions. For example, we may use an exact merit function of the form

$$\Phi_\sigma(x, s) = f(x) - \mu \sum_{i=1}^m \log(s_i) + \sigma(\|c_E(x)\| + \|c_I(x) - s\|), \quad (17.21)$$

where the norm in (17.21) can be  $l_1$  or  $l_2$  and  $\sigma > 0$  is a penalty parameter updated, as described in Sect. 15.3.

An interior-point algorithm with line-search is as follows. After the primal-dual step  $d$  has been computed and the maximum step lengths  $\alpha_s^{\max}$  and  $\alpha_z^{\max}$  given by (17.8) have been determined, execute a line-search by backtracking. This line-search determines the stepsizes  $\alpha_s \in (0, \alpha_s^{\max}]$  and  $\alpha_z \in (0, \alpha_z^{\max}]$ , which provide a sufficient decrease of the merit function or ensure acceptability by the filter. The new estimate of the solution is computed as

$$x^+ = x + \alpha_s d_x, s^+ = s + \alpha_s d_s, \quad (17.22a)$$

$$y^+ = y + \alpha_z d_y, z^+ = z + \alpha_z d_z. \quad (17.22b)$$

In case we use the filter, the pairs of the filter are formed by using the values of the barrier function  $f(x) - \mu \sum_{i=1}^m \log(s_i)$  on one side and the constraint violations  $\|(c_E(x), c_I(x) - s)\|$  on the other side. A step is accepted only if it is not dominated by any element in the filter (see Chap. 18).

### Quasi-Newton Approximations of the Hessian

In the interior-point algorithm, the Hessian matrix  $\nabla_{xx}^2 L$  which appears in (17.11) (or in its reduced variants (17.12) or (17.13)) can be approximated by means of the quasi-Newton methods BFGS or SR1. For large-scale problems, the *limited-memory* BFGS method is recommended (Nocedal, 1980), (Liu, & Nocedal, 1989). We emphasize that it is the approximation of the Hessian to the Lagrange function (17.5) which is computed, not the Hessian of the barrier function (17.3), which is highly ill-conditioned and changes rapidly.

If we apply the BFGS update formula, then we must have the pairs of vectors  $(\Delta x, \Delta l)$ , where  $\Delta x$  and  $\Delta l$  are the variations of the solution and of the gradient of the Lagrange function. After computing a step from  $(x_k, s_k, y_k, z_k)$  to  $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$ , these variations are defined as

$$\Delta x_k = x_{k+1} - x_k, \quad \Delta l_k = \nabla_x L(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1}) - \nabla_x L(x_k, s_k, y_k, z_k).$$

The BFGS update of  $B_{k+1}$  is defined as

$$B_{k+1} = B_k - \frac{B_k \Delta x_k (\Delta x_k)^T B_k}{(\Delta x_k)^T B_k \Delta x_k} + \frac{\Delta l_k (\Delta l_k)^T}{(\Delta l_k)^T \Delta x_k}. \quad (17.23)$$

Observe that the BFGS update (17.23) generates dense  $n \times n$  matrices. Therefore, for large-scale problems it is preferable to use the *compact representation* of the *limited-memory BFGS update*. This representation is based on the following theorem (Nocedal, & Wright, 2006).

**Theorem 17.2** *Let  $B_0$  be a symmetric and positive definite matrix and assume that a number of  $k$  pairs of vectors  $(\Delta x_i, \Delta l_i)$ ,  $i = 0, \dots, k-1$ , satisfy the condition  $(\Delta x_i)^T \Delta l_i > 0$ ,  $i = 0, \dots, k-1$ . Let  $B_k$  be the matrix obtained by applying  $k$  BFGS updates with these vector pairs to  $B_0$ , using (17.23). Then*

$$B_k = B_0 - [B_0 M_k \quad N_k] \begin{bmatrix} M_k^T B_0 M_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} M_k^T B_0 \\ N_k^T \end{bmatrix}, \quad (17.24)$$

where the  $n \times k$  matrices  $M_k$  and  $N_k$  are defined as

$$M_k = [\Delta x_0, \dots, \Delta x_{k-1}] \in \mathbb{R}^{n \times k}, \quad (17.25a)$$

$$N_k = [\Delta l_0, \dots, \Delta l_{k-1}] \in \mathbb{R}^{n \times k}, \quad (17.25b)$$

and the  $k \times k$  matrices  $L_k$  and  $D_k$  have the following form

$$(L_k)_{ij} = \begin{cases} (\Delta x_{i-1})^T \Delta l_{j-1}, & \text{if } i > j, \\ 0, & \text{otherwise,} \end{cases} \quad (17.26a)$$

$$D_k = \text{diag}[(\Delta x_0)^T \Delta l_0, \dots, (\Delta x_{k-1})^T \Delta l_{k-1}]. \quad (17.26b)$$

This result is proved by induction. The conditions  $(\Delta x_i)^T \Delta l_i > 0$ ,  $i = 0, \dots, k - 1$ , ensure that the inverse of the matrix from (17.24) exists, thus this expression being well defined.

Like in L-BFGS, in this case, we keep the  $r$  most recent pairs  $(\Delta x_i, \Delta l_i)$  and, at each iteration, this set of vectors is updated by eliminating the oldest one and by adding a newly generated pair. During the first  $r$  iterations, the updating procedure described in Theorem 17.2 can be used without any modification. For the iterations  $k > r$ , the updating procedure must be modified to reflect the changing of the set of the vector pairs  $(\Delta x_i, \Delta l_i)$ ,  $i = k - r, k - r + 1, \dots, k - 1$ . Now, let us define the  $n \times r$  matrices  $M_k$  and  $N_k$ , by

$$M_k = [\Delta x_{k-r}, \dots, \Delta x_{k-1}] \in \mathbb{R}^{n \times r}, \quad (17.27a)$$

$$N_k = [\Delta l_{k-r}, \dots, \Delta l_{k-1}] \in \mathbb{R}^{n \times r}. \quad (17.27b)$$

Then, the matrix  $B_k$  is obtained by applying  $r$  updates to the basic matrix  $B_0^{(k)} = \delta_k I$ , where  $\delta_k = 1/\gamma_k$  and  $\gamma_k = (\Delta x_{k-1})^T \Delta l_{k-1} / (\Delta l_{k-1})^T \Delta l_{k-1}$ , to get

$$B_k = \delta_k I - [\delta_k M_k \quad N_k] \begin{bmatrix} \delta_k M_k^T M_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} \delta_k M_k^T \\ N_k^T \end{bmatrix}, \quad (17.28)$$

where  $L_k$  and  $D_k$  are the  $r \times r$  matrices defined by

$$(L_k)_{i,j} = \begin{cases} (\Delta x_{k-r-1+i})^T \Delta l_{k-r-1+j}, & \text{if } i > j, \\ 0, & \text{otherwise,} \end{cases} \quad (17.29a)$$

$$D_k = \text{diag}[(\Delta x_{k-r})^T \Delta l_{k-r}, \dots, (\Delta x_{k-1})^T \Delta l_{k-1}]. \quad (17.29b)$$

After the new iteration  $x_{k+1}$  has been computed, the matrix  $M_{k+1}$  is obtained by deleting  $\Delta x_{k-r}$  and by adding the new  $\Delta x_k$ . The matrix  $N_{k+1}$  is updated in a similar way. The advantage of this compact representation is that the  $(2r \times 2r)$  matrix in the middle of (17.28) and which must be inverted is of small dimensions. Usually,  $r$ , the number of stored pairs  $(\Delta x_i, \Delta l_i)$ , is very small: 3, 5, or 7 (rarely 9). Therefore, the numerical effort for the inverse computation of this matrix is negligible.

Interior-point algorithms are implemented using the compact representation of the BFGS updating to the Hessian of the Lagrange function (17.5). Moreover, since  $B$  is positive definite, assuming that  $J_E$  is of full rank, then the matrix of the primal-dual system (17.11) is nonsingular. Hence, the solution of (17.11) can be simply obtained by the Sherman-Morrison-Woodbury formula.

### Feasible Interior-Point Methods

In many nonlinear optimization applications, we need all the iterates generated by the interior-point algorithm to be feasible with respect to some or to all the inequality constraints. Interior-point methods provide a natural framework for designing feasible algorithms. If the current iterate  $x$  satisfies the inequality constraint  $c_I(x) > 0$ , then it is easy to adapt the primal-dual iteration given by (17.11) so that the feasibility is conserved. After computing the step  $d$ , we let  $x^+ = x + d_x$ , redefine the slacks as  $s^+ = c_I(x^+)$  and test whether the new point  $(x^+, s^+)$  is acceptable for the merit function. If so, we define this point as the new iterate. Otherwise, we reject step  $d$  and compute a new one, a shorter trial step. Clearly, in a line-search algorithm backtracking is used, while in a trust-region method a new step is computed by reducing the trust-region radius.

### 17.3 Line-Search Interior-Point Algorithm

To describe the line-search interior-point algorithm, let us denote by  $D\Phi_\sigma(x, s, d)$  the directional derivative of the merit function  $\Phi_\sigma$  in the direction  $d$ . The algorithm of this method is very close to the prototype Algorithm 17.1, which contains two imbricate loops, the first being responsible for testing the KKT optimality conditions and the second one for solving the primal-dual system (17.11). If in the line-search a quasi-Newton method is used, then it is necessary to choose an initial symmetric and positive definite matrix  $B_0$  which is updated along the iteration by quasi-Newton techniques.

**Algorithm 17.3** *Line-search interior-point algorithm*

1.	Choose $x_0$ and $s_0 > 0$ . Compute the initial values of the multipliers $y_0$ and $z_0 > 0$ . Choose an initial value of the barrier parameter $\mu_0 > 0$ , as well as the parameters $\sigma, \tau \in (0, 1)$ and the tolerances $\varepsilon_\mu$ and $\varepsilon_{TOL}$ . Select an $n \times n$ matrix, symmetric and positive definite $B_0$ . Set $k = 0$
2.	If $E(x_k, s_k, y_k, z_k, 0) \geq \varepsilon_{TOL}$ , then continue with step 3; otherwise, stop. The current point is the optimal solution of the problem
3.	If $E(x_k, s_k, y_k, z_k, \mu) \geq \varepsilon_\mu$ , then go to step 4; otherwise, continue with step 10
4.	Solve the primal-dual system (17.11) for the primal-dual direction $d = (d_x, d_s, d_y, d_z)$
5.	Using (17.8), compute $\alpha_s^{\max}$ and $\alpha_z^{\max}$ . Set $d_w = [d_x, d_s]$
6.	Compute the stepsizes $\alpha_s$ and $\alpha_z$ satisfying $\alpha_s \in (0, \alpha_s^{\max}]$ and $\alpha_z \in (0, \alpha_z^{\max}]$ , as well as the condition $\Phi_\sigma(x_k + \alpha_s d_x, s_k + \alpha_s d_s) \leq \Phi_\sigma(x_k, s_k) + \eta \alpha_s D\Phi_\sigma(x_k, s_k, d_w)$
7.	Using (17.22), compute a new estimation of the solution $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$
8.	If a quasi-Newton approach is used, update $B_k$
9.	Set $k = k + 1$ . Continue with step 3
10.	Set $\mu = \sigma\mu$ and update $\varepsilon_\mu$ . Continue with step 2

◆

In step 4 of this algorithm, the matrix of the system (17.11) is very likely to be modified as in (17.16). If the merit function can cause the Maratos effect, then a second-order correction or a nonmonotone strategy can be implemented. The barrier tolerance  $\varepsilon_\mu$  can be selected as  $\varepsilon_\mu = \mu$ , as in the prototype Algorithm 17.1.

To be reliable and to ensure the convergence, the algorithm must be equipped with some protection mechanisms which substantially modify the line-search in step 6. These mechanisms refer to a careful control of the decrease in the barrier parameter  $\mu$  and in the (inner) convergence tolerance  $\varepsilon_\mu$ , and they let the parameter  $\tau$  in (17.8) converge to 1 rapidly enough.

Another specific difficulty in interior-point methods is the lack of coordination between the step computation and the satisfaction of the bounds on variables. An approach is to monitor the stepsizes  $\alpha_s$  and  $\alpha_z$  in (17.7). If they are smaller than a given threshold, then the primal-dual step is replaced by a step that guarantees progress in feasibility and improvement in optimality.

An alternative to using the merit function is to use a filter to perform the line-search. In a filter method, when the stepsizes are very small, the feasibility restoration phase can be started (Fletcher, & Leyffer, 2002).

## 17.4 A Variant of the Line-Search Interior-Point Algorithm

In the following, we present a *variant* of a line-search interior-point algorithm for solving general nonlinear optimization problems that include simple bounds on variables. We also point out some very important details which emphasize the mathematical technology based on the interior-point theory (Andrei, 1998b, 1998c).

Let us consider the problem

$$\min f(x) \quad (17.30a)$$

subject to

$$c_E(x) = 0, \quad (17.30b)$$

$$c_I(x) \geq 0, \quad (17.30c)$$

$$l \leq x \leq u, \quad (17.30d)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_E: \mathbb{R}^n \rightarrow \mathbb{R}^{me}$ ,  $c_I: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are supposed to be twice continuously differentiable functions. The bounds  $l \in \mathbb{R}^n$  and  $u \in \mathbb{R}^n$  may have any values with  $l \leq u$ . The idea of the algorithm for solving (17.30) is to use the first-order KKT optimality conditions in an iterative computational scheme of the Newton type, in which the inequality constraints are penalized in a barrier term. The updating of the barrier parameter is given by a merit function. The stepsize is based on the Wolfe line-search combined with the interior-point centrality and with the condition of limitation of the convergence of the complementarity conditions to zero compared to the optimality conditions. We mention that this mathematical technology is also found in (Kortanek, Potra, & Ye, 1991), (Goldfarb, Liu, & Wang, 1991), (El-Bakry, Tapia, Tsuchiya, & Zhang, 1996), (Vanderbei, & Shanno, 1997, 1999) and (Gay, Overton, & Wright, 1997).

### KKT Optimality Conditions

Considering the slack variables  $s \in \mathbb{R}^m$  associated to the functional inequality constraints (17.30c) and  $w, v \in \mathbb{R}^n$  associated to the simple bounds (17.30d), the problem (17.30) is reformulated as

$$\min f(x) \quad (17.31a)$$

subject to

$$c_E(x) = 0, \quad (17.31b)$$

$$c_I(x) - s = 0, \quad (17.31c)$$

$$x - w = l, \quad (17.31d)$$

$$x + v = u, \quad (17.31e)$$

$$s \geq 0, w \geq 0, v \geq 0. \quad (17.31f)$$

Now, the inequality constraints (17.31f) are transferred into the objective function by a barrier term, thus obtaining the problem

$$\min f(x) - \mu \sum_{i=1}^m \log(s_i) - \mu \sum_{j=1}^n \log(w_j) - \mu \sum_{j=1}^n \log(v_j) \quad (17.32a)$$

subject to

$$c_E(x) = 0, \quad (17.32b)$$

$$c_I(x) - s = 0, \quad (17.32c)$$

$$x - w - l = 0, \quad (17.32d)$$

$$x + v - u = 0. \quad (17.32e)$$

The Lagrange function of (17.32) is

$$\begin{aligned} L(x, s, w, v, y, z, p, q, \mu) = & f(x) - \mu \sum_{i=1}^m \log(s_i) - \mu \sum_{j=1}^n \log(w_j) - \mu \sum_{j=1}^n \log(v_j) \\ & - y^T c_E(x) - z^T (c_I(x) - s) - p^T (x - w - l) - q^T (x + v - u), \end{aligned} \quad (17.33)$$

where  $y$ ,  $z$ ,  $p$ , and  $q$  are the Lagrange multipliers. Hence, the first-order optimality conditions are as follows:

$$\nabla f(x) - J_E(x)^T y - J_I(x)^T z - p + q = 0, \quad (17.34a)$$

$$c_E(x) = 0, \quad (17.34b)$$

$$c_I(x) - s = 0, \quad (17.34c)$$

$$x - w - l = 0, \quad (17.34d)$$

$$x + v - u = 0, \quad (17.34e)$$

$$SZe - \mu e = 0, \quad (17.34f)$$

$$WPe - \mu e = 0, \quad (17.34g)$$

$$VQe - \mu e = 0, \quad (17.34h)$$

where the matrices  $S$ ,  $Z$ ,  $W$ ,  $P$ ,  $V$ , and  $Q$  are diagonal with the elements  $s_i$ ,  $z_i$ ,  $w_i$ ,  $p_i$ ,  $v_i$ , and  $q_i$ , respectively.  $e = [1, 1, \dots, 1]^T$ .  $J_E(x)$  and  $J_I(x)$  are the Jacobian matrices of the equality  $c_E(x) = 0$  and of the inequality constraints  $c_I(x) \geq 0$ , respectively.

If in (17.34) the barrier parameter  $\mu$  is set to zero, then the last three equations are exactly the complementarity slackness conditions. Usually, these last three equations are called  $\mu$  complementarity conditions. Observe that the KKT system (17.34) is a nonlinear algebraic one with  $5n + 2m + me$  equations and a similar number of unknowns, parameterized by the barrier parameter  $\mu$ . For solving this system, the Newton algorithm is fundamental, which is known to be very efficient near the solution. Supposing that the system (17.34) has a solution, then for each  $\mu > 0$  we get a solution  $(x_\mu, s_\mu, w_\mu, v_\mu, y_\mu, z_\mu, p_\mu, q_\mu)$ . The path

$$\left\{ \left( x_\mu, s_\mu, w_\mu, v_\mu, y_\mu, z_\mu, p_\mu, q_\mu \right) : \mu > 0 \right\}$$

is called the *primal-dual central path*. The interior-point algorithm which will be presented in this section is an iterative procedure. At each iteration, the algorithm attempts to move toward a point on the central path closer to the optimal point than the current point is. If the barrier parameter  $\mu$  is set to zero, then the KKT conditions (17.34) can be partitioned into two classes:

$$CO(t) = \begin{bmatrix} \nabla f(x) - J_E(x)^T y - J_I(x)^T z - p + q \\ c_E(x) \\ c_I(x) - s \\ x - w - l \\ x + v - u \end{bmatrix}, \quad (17.35)$$

$$CT(s, z, w, p, v, q) = \begin{bmatrix} SZe \\ WPe \\ VQe \end{bmatrix}. \quad (17.36)$$

It is obvious that if we could have a point  $t = [x, y, z, p, q, s, w, v]^T$  to satisfy the system

$$F(t) \triangleq \begin{bmatrix} CO(t) \\ CT(s, z, w, p, v, q) \end{bmatrix} = 0, \quad (17.37)$$

then the component  $x$  of this point would be the solution of the problem (17.30). The separation of the optimality conditions into these two classes is crucial for the elaboration of an efficient algorithm for solving (17.30) and also for proving its convergence.

For linear programming, the system (17.34) is much simpler. The only nonlinear expressions in (17.34) are simple multiplications of the slack and dual variables, and the presence of these simple nonlinearities makes the subject of linear programming nontrivial.

### The Newton System

As above, denote  $t = [x, y, z, p, q, s, w, v]^T$ . Then, the Newton method applied to the system (17.37) consists in the determination of the direction  $\Delta t = [\Delta x, \Delta y, \Delta z, \Delta p, \Delta q, \Delta s, \Delta w, \Delta v]^T$  as solution of the following perturbed system:

$$F'(t)\Delta t = -F(t) + \mu\hat{e}, \quad (17.38)$$

where  $\hat{e} \in \mathbb{R}^{5n+2m+me}$  is a vector with zero components, except for the last  $2n + m$  ones, which are all equal to one.  $F'(t)$  is the Jacobian of the function  $F(t)$  computed at the current point  $t$ . The new point is computed as

$$t^+ = t + \alpha\Delta t, \quad (17.39)$$

where  $\alpha$  is the stepsize.

Two problems are critical with this algorithm: the choice of  $\alpha$  at each iteration and possibly, the modification of the system (17.38) in order to find a local solution to it. For linear or convex quadratic programming, the modification of (17.38) is not necessary, and the stepsize at each iteration is determined by standard ratio tests (Lustig, Marsten, & Shanno, 1990, 1991, 1992, 1994), (Vanderbei,

1990, 1994, 2001), (Ye, 1997), (Wright, 1997). In the case of general nonlinear optimization problems, it is well known that for a poor initial estimate of the solution, the Newton method may diverge and so the linear system (17.38) needs to be modified once again. Besides, the technique for choosing the stepsize  $\alpha$  is more complex. In order to achieve the convergence to a solution of (17.34), El-Bakry, Tapia, Tsuchiya, and Zhang (1996) introduced a merit function and showed that for a proper choice of the barrier parameter  $\mu$  there is a stepsize  $\alpha$  such that the algorithm (17.39) is convergent to a solution of (17.30) provided that the Jacobian  $F'(t)$  from (17.38) remains nonsingular at each iteration (Shanno, Breitfeld, & Simantiraki, 1996), (Shanno, & Simantiraki, 1997). A modification of the merit function introduced by El-Bakry, Tapia, Tsuchiya, and Zhang (1996) was considered by Vanderbei and Shanno (1997). They used the  $l_2$  norm of the constraints multiplied by a parameter  $\beta > 0$  and proved that there is a value of  $\beta$  such that the direction given by the Newton system is a descent direction for their merit function. The corresponding algorithm was implemented as a nonlinear version of LOQO by Vanderbei (1995).

### The Newton Direction Determination

The Newton system for direction determination involves the computation of the Jacobian matrix of the function  $F(t)$  defined in (17.37). From (17.34) we get

$$\begin{bmatrix} K(x, y, z) & -J_E(x)^T & -J_I(x)^T & -I & I & 0 & 0 & 0 \\ J_E(x) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ J_I(x) & 0 & 0 & 0 & 0 & -I & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & -I & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 & I \\ 0 & 0 & S & 0 & 0 & Z & 0 & 0 \\ 0 & 0 & 0 & W & 0 & 0 & P & 0 \\ 0 & 0 & 0 & 0 & V & 0 & 0 & Q \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta p \\ \Delta q \\ \Delta s \\ \Delta w \\ \Delta v \end{bmatrix} = \begin{bmatrix} \chi \\ -c_E(x) \\ -c_I(x) + s \\ l + w - x \\ u - v - x \\ -SZe + \mu e \\ -WP + \mu e \\ -VQ + \mu e \end{bmatrix}. \quad (17.40)$$

This system is not symmetric, but it is easy to be written in a symmetric form by multiplying the second and the third equations by  $-1$  and the last three equations by  $S^{-1}$ ,  $W^{-1}$ , and  $V^{-1}$ , respectively. After completing this very simple algebraic operation and after eliminating the variables  $\Delta s$ ,  $\Delta w$ , and  $\Delta v$  as

$$\Delta s = \mu Z^{-1}e - Se - Z^{-1}S\Delta z, \quad (17.41a)$$

$$\Delta w = \mu P^{-1}e - We - P^{-1}W\Delta p, \quad (17.41b)$$

$$\Delta v = \mu Q^{-1}e - Ve - Q^{-1}V\Delta q, \quad (17.41c)$$

we get the following *reduced Newton system*:

$$\begin{bmatrix} K(x, y, z) & -J_E(x)^T & -J_I(x)^T & -I & I & 0 \\ -J_E(x) & 0 & 0 & 0 & 0 & 0 \\ -J_I(x) & 0 & -Z^{-1}S & 0 & 0 & 0 \\ -I & 0 & 0 & -P^{-1}W & 0 & 0 \\ I & 0 & 0 & 0 & -Q^{-1}V & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta p \\ \Delta q \end{bmatrix} = \begin{bmatrix} \chi \\ c_E(x) \\ c_I(x) - \mu Z^{-1}e \\ x - l - \mu P^{-1}e \\ u - x - \mu Q^{-1}e \end{bmatrix}, \quad (17.42)$$

where

$$K(x, y, z) = \nabla^2 f(x) - \sum_{i=1}^{me} y_i \nabla^2 c_{Ei}(x) - \sum_{i=1}^m z_i \nabla^2 c_{Ii}(x), \quad (17.43)$$

$$\chi = -\nabla f(x) + J_E(x)^T y + J_I(x)^T z + p - q. \quad (17.44)$$

As above,  $J_E(x) = \nabla c_E(x)$  and  $J_I(x) = \nabla c_I(x)$ , are the Jacobian matrices of the equality constraints  $c_E(x) = 0$  and of the inequality constraints  $c_I(x) \geq 0$ , respectively.

Observe that the matrix of the reduced Newton system (17.42) is symmetric of order  $(3n + m + me)$ . By solving this system and by using (17.41), we get the search direction  $\Delta t = [\Delta x, \Delta y, \Delta z, \Delta p, \Delta q, \Delta s, \Delta w, \Delta v]^T$ . The existence of a solution for (17.42) implies the existence of the inverse of the diagonal matrices  $Z$ ,  $P$ , and  $Q$ , which assumes a certain condition of initialization of the algorithm.

The standard Newton method assumptions are as follows (Dennis, & Schnabel, 1983), (Andrei, 1998c).

- (i) There is a solution for the problem (17.30), and the associated dual variables satisfy the KKT conditions (17.34).
- (ii) The Hessian matrices  $\nabla^2 f(x)$ ,  $\nabla^2 c_{Ei}(x)$ ,  $i = 1, \dots, me$ ,  $\nabla^2 c_{Ii}(x)$ ,  $i = 1, \dots, m$ , exist and are locally Lipschitz continuous at  $x^*$ .
- (iii) The set of vectors  $\{\nabla c_{E1}(x^*), \dots, \nabla c_{Eme}(x^*)\} \cup \{\nabla c_{Ii}(x^*), i \in A(x^*)\}$  is linearly independent, where  $A(x^*)$  is the set of the active inequality constraints at  $x^*$ .
- (iv) For every vector  $d \neq 0$  satisfying  $\nabla c_{Ei}(x^*)^T d = 0$ ,  $i = 1, \dots, me$ , and  $\nabla c_{Ii}(x^*)^T d = 0$ ,  $i \in A(x^*)$ , we have  $d^T K(x^*, y^*, z^*) d > 0$ .
- (v) For  $i = 1, \dots, m$ ,  $z_i c_{Ii}(x^*) > 0$  and for  $j = 1, \dots, n$ ,  $p_j^* (x_j^* - l_j) > 0$  and  $q_j^* (u_j - x_j^*) > 0$ .

**Proposition 17.1** Suppose that the conditions (i)-(v) hold and  $s^* = c_I(x^*)$ ,  $w^* = x^* - l$  and  $v^* = u - x^*$ . Then the Jacobian matrix  $F'(t^*)$  of function  $F(t)$  given by (17.40) is nonsingular.

**Proof** Consider the reduced problem in which only the inequality constraints active in the minimizer point are used. Then, from the theory of the equality constraints optimization it follows that the matricial block given by the first five rows and columns of the matrix (17.40) is nonsingular. Hence, the nonsingularity of the matrix from (17.40) follows from the strict complementarity condition (v) and from the nonsingularity of the matricial block from (17.40). ◆

Generally, in current implementations the reduced Newton system (17.42) is not used in the form in which it appears. Usually, the reduction is continued by the elimination from (17.42) of the variables  $\Delta z$ ,  $\Delta p$  and  $\Delta q$  as

$$\Delta z = \mu S^{-1} e - S^{-1} Z c_I(x) - S^{-1} Z J_I(x) \Delta x, \quad (17.45a)$$

$$\Delta p = \mu W^{-1} e - W^{-1} P(x - l) - W^{-1} P \Delta x, \quad (17.45b)$$

$$\Delta q = \mu V^{-1} e - V^{-1} Q(u - x) + V^{-1} Q \Delta x, \quad (17.45c)$$

Hence, the system (17.42) is further reduced as

$$\begin{bmatrix} \bar{K} & -J_E(x)^T \\ J_E(x) & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r + \mu \bar{r} \\ c_E(x) \end{bmatrix}, \quad (17.46)$$

where

$$\bar{K} = K(x, y, z) + J_I(x)^T S^{-1} Z J_I(x) + W^{-1} P + V^{-1} Q, \quad (17.47a)$$

$$r = \chi - J_I(x)^T S^{-1} Z c_I(x) - W^{-1} P(x - l) + V^{-1} Q(u - x), \quad (17.47b)$$

$$\bar{r} = J_I(x)^T S^{-1} e + W^{-1} e - V^{-1} e. \quad (17.47c)$$

Now, by solving (17.46) we get the variables  $\Delta x$  and  $\Delta y$  with which the rest of the variables are immediately obtained from (17.45) and (17.41). Notice that, as in linear programming, we must solve a symmetric undefined algebraic system, but in this case the Jacobian and the Hessian matrices of the functions defining the problem are involved, which is much more complicated.

### The Merit Function

By solving the system (17.46) and by using (17.45) and (17.41), we get a direction  $\Delta t$  pointing to the central path. We must now determine the stepsize along this direction. This is done by a merit function associated to the optimality conditions (17.34). The idea of the merit function is to enable a progress toward a local minimizer of the problem by conserving the feasibility of the solution. In our algorithm, the merit function used for the line-search is the squared  $l_2$  norm of the KKT conditions (17.34), that is

$$\Phi(t) = \|F(t)\|_2^2 = F(t)^T F(t). \quad (17.48)$$

Denote  $\Phi_k = \Phi(t_k)$  the value of the merit function at the iterate  $t_k$  and  $\Phi_k(\alpha) = \Phi(t_k + \alpha \Delta t_k)$ , which illustrates the dependences of the merit function on the stepsize  $\alpha$ . Clearly,  $\Phi_k(0) = \Phi(t_k) = \Phi_k$ . Therefore,

$$\Phi_k(\alpha) = F(t_k + \alpha \Delta t_k)^T F(t_k + \alpha \Delta t_k). \quad (17.49)$$

The following proposition shows that the search direction given by the perturbed Newton system (17.38)

$$\Delta t_k = [F'(t_k)]^{-1} [-F(t_k) + \mu \hat{e}] \quad (17.50)$$

is a descent one for the merit function  $\Phi(t)$ .

**Proposition 17.2** *The direction  $\Delta t_k$  given by (17.50), solution of the perturbed Newton system (17.38), is a descent direction for the merit function (17.48).*

**Proof** Considering the derivative of  $\Phi_k(\alpha)$  at  $\alpha = 0$ , we get

$$\begin{aligned} \Phi'(0) &= 2F(t_k)^T F'(t_k) \Delta t_k = 2F(t_k)^T F'(t_k) [F'(t_k)]^{-1} [-F(t_k) + \mu \hat{e}] \\ &= 2F(t_k)^T [-F(t_k) + \mu \hat{e}] = -2\|F(t_k)\|_2^2 + 2\mu F(t_k)^T \hat{e}, \end{aligned}$$

that is  $\Phi'(0) < 0$  if and only if  $2\mu F(t_k)^T \hat{e} \leq 2\|F(t_k)\|_2^2$ . This determines the following estimation of the barrier parameter  $\mu$  to be used in the logarithmic barrier function (17.32a):

$$\mu \leq \frac{\|F(t_k)\|_2^2}{F(t_k)^T \hat{e}} = \frac{\|F(t_k)\|_2^2}{s_k^T z_k + w_k^T p_k + v_k^T q_k}. \quad (17.51)$$

Therefore, choosing at every iteration the value of the barrier parameter  $\mu$  as in (17.51), the direction  $\Delta t_k$  given by (17.50) is indeed a descent direction for the merit function  $\Phi(t_k)$ .  $\blacklozenge$

### The Stepsize Determination

As we have learned from linear programming, (see (Wright, 1997)), for the nonlinear optimization problem (17.30) the distance from centrality is also given by

$$\xi_k = \frac{\min_i \{s_i z_i, w_i p_i, v_i q_i\}}{\frac{s_k^T z_k + w_k^T p_k + v_k^T q_k}{2n+m}}. \quad (17.52)$$

Clearly,  $0 < \xi_k \leq 1$  and  $\xi_k = 1$  if and only if  $s_i z_i$ ,  $w_i p_i$  and  $v_i q_i$  are equal to a constant for all  $i$ . To specify a value for  $\alpha$ , the following function is firstly introduced:

$$\Theta^I(\alpha) = \frac{\min_i \{s_i(\alpha) z_i(\alpha), w_i(\alpha) p_i(\alpha), v_i(\alpha) q_i(\alpha)\}}{\frac{s(\alpha)^T z(\alpha) + w(\alpha)^T p(\alpha) + v(\alpha)^T q(\alpha)}{2n+m}} - \gamma \tau_1, \quad (17.53)$$

where  $\tau_1$  is the initial distance from centrality

$$\tau_1 = \frac{\min_i \{s_i^0 z_i^0, w_i^0 p_i^0, v_i^0 q_i^0\}}{\frac{s^{0T} z^0 + w^{0T} p^0 + v^{0T} q^0}{2n+m}}, \quad (17.54)$$

and  $\gamma \in (0, 1)$  is a constant by which we can modify the distance from centrality. Observe that for  $t = t_0$  and  $\gamma = 1$ , it follows that  $\Theta^I(0) = 0$ . Besides,  $\Theta^I(\alpha)$  is a piecewise quadratic function. To choose the stepsize  $\alpha_k$  at each iteration,  $\alpha_k$  must satisfy  $\Theta^I(\alpha) \geq 0$  for all  $\alpha \in [0, \alpha_k]$ , i.e.,

$$\frac{\min_i \{s_i(\alpha) z_i(\alpha), w_i(\alpha) p_i(\alpha), v_i(\alpha) q_i(\alpha)\}}{\frac{s(\alpha)^T z(\alpha) + w(\alpha)^T p(\alpha) + v(\alpha)^T q(\alpha)}{2n+m}} \geq \gamma_k \tau_1, \quad (17.55)$$

where the variables of the problem are considered at the iteration  $k$  and  $\gamma_k \in (0, 1)$ . Since  $\Theta^I(\alpha)$  is a piecewise quadratic function, from (17.55) it follows that  $\alpha_k$  can be easily computed.

Considering the merit function (17.48), the Wolfe conditions for its minimization are

$$\Phi(t_{k+1}) \leq \Phi(t_k) + \beta \alpha_k \nabla \Phi(t_k)^T \Delta t_k, \quad (17.56)$$

$$\nabla \Phi(t_{k+1})^T \Delta t_k \geq \delta \nabla \Phi(t_k)^T \Delta t_k, \quad (17.57)$$

where  $0 < \beta < \delta < 1$  are parameters responsible for the reduction of both the merit function and of the rate of decreasing this function along the direction  $\Delta t_k$ . Having in view that  $\Phi(t_{k+1}) = \Phi_k(0)$ , the first Wolfe condition (17.56) is equivalent to

$$\Phi_k(\alpha) \leq \Phi_k(0) + \beta \alpha_k \Phi'_k(0). \quad (17.58)$$

**Proposition 17.3** *For the merit function (17.48), taking*

$$\mu_k = \sigma_k \frac{s_k^T z_k + w_k^T p_k + v_k^T q_k}{2n+m} \quad (17.59)$$

where  $\sigma_k \in (0, 1)$ , we have

$$\Phi'_k(0) = -2 \left[ \Phi_k(0) - \frac{\sigma_k}{2n+m} (s_k^T z_k + w_k^T p_k + v_k^T q_k)^2 \right]. \quad (17.60)$$

**Proof** By direct computation we get

$$\begin{aligned} \Phi'_k(0) &= -2\Phi_k(0) + 2\mu F(t_k)^T \hat{e} = -2\Phi_k(0) + 2\mu [s_k^T z_k + w_k^T p_k + v_k^T q_k] \\ &= -2\Phi_k(0) + 2\sigma_k \frac{s_k^T z_k + w_k^T p_k + v_k^T q_k}{2n+m} [s_k^T z_k + w_k^T p_k + v_k^T q_k] \\ &= -2 \left[ \Phi_k(0) - \frac{\sigma_k}{2n+m} (s_k^T z_k + w_k^T p_k + v_k^T q_k)^2 \right]. \end{aligned} \quad \blacklozenge$$

**Proposition 17.4** At every iteration we have

$$\frac{(s_k^T z_k + w_k^T p_k + v_k^T q_k)^2}{2n+m} \leq \Phi_k(0). \quad (17.61)$$

**Proof** By simple algebraic manipulation, we obtain

$$\begin{aligned} \frac{(s_k^T z_k + w_k^T p_k + v_k^T q_k)^2}{2n+m} &\leq \|SZe\|_2^2 + \|WPe\|_2^2 + \|VQe\|_2^2 \\ &\leq \|SZe\|_2^2 + \|WPe\|_2^2 + \|VQe\|_2^2 + \|CO(t_k)\|_2^2 = \Phi_k(0). \end{aligned} \quad \blacklozenge$$

It is easy to show that the estimation of the barrier parameter  $\mu_k$  given by (17.59) with  $\sigma_k \in (0, 1)$  is smaller than the estimation recommended by (17.51). Therefore,  $\mu_k$  given by (17.59) ensures the descent character of the search direction  $\Delta t_k$ . Moreover, the following proposition gives an estimation of the reduction of the values of the merit function.

**Proposition 17.5** The direction  $\Delta t_k$ , solution of the perturbed Newton system (17.38) with  $\mu$  given by (17.59), is a descent direction for the merit function  $\Phi(t)$  at every  $t_k$ . Moreover, if the first Wolfe condition (17.56) is satisfied, then

$$\Phi_k(\alpha_k) \leq [1 - 2\alpha_k\beta(1 - \sigma_k)]\Phi_k(0). \quad (17.62)$$

**Proof** As we know,

$$\Phi'_k(0) = -2\|F(t_k)\|_2^2 + 2\mu_k (s_k^T z_k + w_k^T p_k + v_k^T q_k).$$

Taking  $\mu_k$  as in (17.59), it follows that

$$\begin{aligned} \Phi'_k(0) &= -2\Phi_k(0) + 2\sigma_k \frac{(s_k^T z_k + w_k^T p_k + v_k^T q_k)^2}{2n+m} \\ &\leq -2\Phi_k(0) + 2\sigma_k \Phi_k(0) = -2\Phi_k(0)(1 - \sigma_k) < 0, \end{aligned}$$

proving the descent character of the direction given by the perturbed Newton system. Moreover, taking into consideration the above propositions (17.3 and 17.4), from (17.58) we have

$$\begin{aligned}
\Phi_k(\alpha) &\leq \Phi_k(0) + \beta\alpha_k \Phi'_k(0) \\
&= \Phi_k(0) + \beta\alpha_k \left[ -2 \left( \Phi_k(0) - \sigma_k \frac{(s_k^T z_k + w_k^T p_k + v_k^T q_k)^2}{2n+m} \right) \right] \\
&\leq \Phi_k(0) - 2\beta\alpha_k \Phi_k(0) + 2\beta\alpha_k \sigma_k \Phi_k(0) \\
&= [1 - 2\alpha_k \beta (1 - \sigma_k)] \Phi_k(0),
\end{aligned}$$

which proves the proposition.  $\diamond$

This proposition shows that the sequence  $\{\Phi_k\}$  is monotonous and nonincreasing. Therefore, for all  $k$ ,  $\Phi_k \leq \Phi_0$ . Moreover, if the sequence of the stepsize  $\{\alpha_k\}$  is bounded away from zero and the parameter  $\sigma_k$  is bounded away from one at every iteration, then the merit function is linearly convergent to zero. Hence, the above inequality (17.62) is equivalent to

$$\frac{\|F(t_{k+1})\|_2}{\|F(t_k)\|_2} \leq [1 - 2\alpha_k \beta (1 - \sigma_k)]^{1/2}. \quad (17.63)$$

Some numerical examples illustrate that a problem that may lead to the nonconvergence of the algorithm is the case in which the sequence  $\{\|CT(s_k, z_k, w_k, p_k, v_k, q_k)\|\}$  converges to zero faster than the sequence  $\{\Phi(t_k)\}$ . In such a case, the sequence of the stepsizes  $\{\alpha_k\}$  is decreasing to zero, thus determining the nonconvergence of the algorithm. To avoid this situation, let us introduce the following function:

$$\Theta''(\alpha) = s(\alpha)^T z(\alpha) + w(\alpha)^T p(\alpha) + v(\alpha)^T q(\alpha) - \gamma \tau_2 \|CO(t(\alpha))\|_2, \quad (17.64)$$

where

$$\tau_2 = \frac{s_0^T z_0 + w_0^T p_0 + v_0^T q_0}{\|CO(t_0)\|_2}, \quad (17.65)$$

and  $\gamma \in (0, 1)$  is a constant, the same as in (17.53). Observe that for  $t = t_0$  and  $\gamma = 1$ ,  $\Theta''(0) = 0$ . In general,  $\Theta''(\alpha)$  is a nonlinear function. For choosing the stepsize  $\alpha_k$  at every iteration it is necessary for  $\alpha_k$  to satisfy

$$\Theta''(\alpha_k) \geq 0. \quad (17.66)$$

**Proposition 17.6** *Let  $\{t_k\}$  be a sequence generated as solution of (17.38). Then,*

$$\min \{1, 0.5\tau_2\} \Phi(t_k) \leq (s_k^T z_k + w_k^T p_k + v_k^T q_k)^2 \leq (2n+m)\Phi(t_k). \quad (17.67)$$

**Proof** The second inequality follows from proposition 17.4. Hence, we prove only the first one. Since  $\Theta^i(\alpha_k) \geq 0$ , for  $i = 1, 2$ , from (17.64) with  $\gamma_k \geq 1/2$  we have

$$(s_k^T z_k + w_k^T p_k + v_k^T q_k) \geq (1/2)\tau_2 \|CO(t_k)\|_2.$$

Therefore,

$$\begin{aligned}
& (s_k^T z_k + w_k^T p_k + v_k^T q_k) \\
& \geq \frac{1}{2} [\|SZe\|_2 + \|WPe\|_2 + \|VQe\|_2 + 0.5\tau_2 \|CO(t_k)\|_2] \\
& \geq \frac{1}{2} \min \{1, 0.5\tau_2\} \|F(t_k)\|_2,
\end{aligned}$$

which completes the proof.  $\diamond$

This proposition shows that the complementarity conditions are bounded and  $\gamma_k$  must be selected as a decreasing sequence with  $1/2 \leq \gamma_k \leq \gamma_{k-1}$ .

Having in view all these developments, at each iteration, the stepsize  $\alpha_k$  is computed as a solution of the following system of algebraic inequalities:

$$l \leq x_k + \alpha_k \Delta x_k \leq u, \quad (17.68a)$$

$$s(\alpha_k), z(\alpha_k), w(\alpha_k), p(\alpha_k), v(\alpha_k), q(\alpha_k) > 0, \quad (17.68b)$$

$$\Theta^I(\alpha) \geq 0, \alpha \in [0, \alpha_k], \quad (17.68c)$$

$$\Theta^{II}(\alpha_k) \geq 0, \quad (17.68d)$$

$$\Phi_k(\alpha_k) \leq \Phi_k(0) + \alpha_k \beta \Phi'_k(0), \quad (17.68e)$$

$$\nabla \Phi_k(\alpha_k)^T \Delta t_k \geq \delta \nabla \Phi_k(0)^T \Delta t_k, \quad (17.68f)$$

where  $0 < \beta < \delta < 1$ .

To determine  $\alpha_k$  satisfying (17.68), a strategy of *reducing the interval* can be used. The first two relations (17.68a) and (17.68b) are easy to implement. As in linear programming, the corresponding ratio test is performed, thus obtaining a value  $\alpha_m$  which maintains the positivity of the variables as well as the simple bounds on variables. Then, a value  $\alpha_k \in (0, \alpha_m]$  which satisfies the conditions (17.68c), (17.68d), (17.68e), and (17.68f) is selected. However, the selection of  $\alpha_k$  from the interval  $(0, \alpha_m]$  is not simple.

### Primal-Dual Interior-Point Algorithm

The interior-point algorithm for solving (17.30) based on the above developments has three main parts. They refer to the computations of the following: the search direction, the barrier parameter, and the stepsize.

#### Algorithm 17.4 Primal-dual interior-point algorithm—PDIP

- |    |  |
|----|--|
| 1. | <i>Initialization.</i> Choose an initial point $t_0 = [x_0, y_0, z_0, p_0, q_0, s_0, w_0, v_0]$ , such that $l \leq x_0 \leq u, (s_0, z_0) > 0, (w_0, p_0) > 0, (v_0, q_0) > 0$ . Choose the parameters $\epsilon > 0, \beta \in (0, 1/2], \gamma_{k-1} = 1$ and $\rho \in (0, 1)$ . Set $k = 0$ |
| 2. | <i>Convergence test.</i> Compute the value of the merit function $\Phi(t_k) = F(t_k)^T F(t_k)$ . If $\Phi(t_k) \leq \epsilon$ , stop; otherwise, go to step 3  |
| 3. | <i>Evaluation of the barrier parameter.</i> Choose $\sigma_k \in (0, 1)$ . Using (17.59), compute the barrier parameter $\mu_k$  |
| 4. | <i>Computing the search direction.</i> Determine $\Delta t_k$ by solving the reduced Newton system (17.46) and by using (17.45) and (17.41)  |

5. *Stepsize determination.* Use the Armijo technique:
- Choose  $1/2 \leq \gamma_k \leq \gamma_{k-1}$ .
  - Compute  $\alpha_m$  as the longest value of  $\alpha_k$  which satisfies (17.68a) and (17.68b).
  - Determine  $\alpha_k^l \in (0, \alpha_m]$  as the smallest positive root, such that  $\Theta^l(\alpha) \geq 0$  for all  $\alpha \in (0, \alpha_k^l]$ .
  - Determine  $\alpha_k^H \in (0, \alpha_k^l]$ , such that  $\Theta^H(\alpha_k^H) \geq 0$ .
  - Set  $\bar{\alpha}_k = \min\{\alpha_k^l, \alpha_k^H\}$ .
  - Set  $\alpha_k = \rho^j \bar{\alpha}_k$ , such that  $\alpha_k$  satisfies (17.68e), where  $j = 0, 1, \dots$  is the smallest integer with this property
6. *Updating the variables.* Set  $t_{k+1} = t_k + \alpha_k \Delta t_k$ ,  $k = k + 1$  and go to step 2

Some comments are as follows (details can be found in (Andrei, 1998c)).

- (*Search direction*) In proposition 17.2, we showed that the step direction  $\Delta t$  given by (17.50) is descent for the merit function provided that the value of the barrier parameter is selected as in (17.51).

**Proposition 17.7** Suppose that  $f$ ,  $c_E$ , and  $c_I$  are twice continuously differentiable, the derivative of  $CO(t)$  defined by (17.35) is Lipschitz continuous and the set of gradients  $\{\nabla c_{E1}(x_k), \dots, \nabla c_{Eme}(x_k)\} \cup \{\nabla c_{Ii}(x_k), i \in A(x^*)\}$ , is linearly independent for  $k$  sufficiently large, where  $A(x^*)$  is the set of active inequality constraints at  $x^*$ . Then, the sequence  $\{\Delta t_k\}$  generated by the PDIP algorithm is bounded.

**Proof** By permuting the rows and columns,  $F'(t_k)$  can be rearranged as

$$F'(t_k) = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix},$$

where

$$A = \begin{bmatrix} K(x_k, y_k, z_k) & -J_E(x_k)^T & -J_I(x_k)^T \\ -J_E(x_k) & 0 & 0 \\ -J_I(x_k) & 0 & 0 \end{bmatrix},$$

the matrices  $B$  and  $C$  being very easily identified from  $F'(t)$ . By assumptions of the proposition, the matrix  $A$  is invertible and  $\|A^{-1}\|$  is uniformly bounded. But

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(C - B^TA^{-1}B)^{-1}B^TA^{-1} & -A^{-1}B(C - B^TA^{-1}B)^{-1} \\ -(C - B^TA^{-1}B)^{-1}B^TA^{-1} & (C - B^TA^{-1}B)^{-1} \end{bmatrix},$$

which is bounded, since every matrix involved is bounded. Hence,  $[F'(t_k)]^{-1}$  is uniformly bounded, proving the proposition.  $\blacklozenge$

- (*Stepsize*) A crucial point of the algorithm is the stepsize computation. The following proposition shows that the sequence  $\{\bar{\alpha}_k\}$  generated in step 5e of the PDIP algorithm is bounded away from zero.

**Proposition 17.8** Suppose that the functions  $f(x)$ ,  $c_E(x)$ , and  $c_I(x)$  are twice continuously differentiable and the derivative of  $CO(t)$  is Lipschitz continuous with constant  $L$ . If the sequence  $\{\sigma_k\}$  is bounded away from zero, then the sequence  $\{\bar{\alpha}_k\}$  generated by the algorithm is also bounded away from zero.

**Proof** Since  $\bar{\alpha}_k = \min \{\alpha_k^I, \alpha_k^{II}\}$ , it suffices to show that the sequences  $\{\alpha_k^I\}$  and  $\{\alpha_k^{II}\}$  are bounded away from zero. Let us suppress the subscript  $k$ . As in (El-Bakry, Tapia, Tsuchiya, and Zhang, 1996), let us define the vectors  $a(\alpha) = [s(\alpha), w(\alpha), v(\alpha)]^\top \in \mathbb{R}^{m+2n}$  and  $b(\alpha) = [z(\alpha), p(\alpha), q(\alpha)]^\top \in \mathbb{R}^{m+2n}$ . Then, the function  $\Theta^I(\alpha)$  from (17.53) can be written as

$$\Theta^I(\alpha) = \frac{\min_i \{a_i(\alpha)b_i(\alpha)\}}{\frac{a(\alpha)^\top b(\alpha)}{2n+m}} - \gamma\tau_1.$$

From the definition of  $\alpha^I$  (step 5c of the PDIP algorithm), we can see that  $\alpha$  is the largest number in  $[0, \alpha_m]$  such that

$$a_i(\alpha)b_i(\alpha) - \gamma\tau_1 a(\alpha)^\top b(\alpha)/(2n+m) \geq 0$$

for every  $i = 1, \dots, 2n+m$  and  $\alpha \in [0, \alpha^I]$ . Now, define

$$\eta_i = \left| \Delta a_i \Delta b_i - \gamma\tau_1 \frac{\Delta a^\top \Delta b}{2n+m} \right|.$$

From proposition 17.7, it follows that  $\Delta t$  is bounded. Then, there is a positive constant  $M$  such that  $\eta_i \leq M$ . Straightforward computation shows that for  $\alpha \in [0, \alpha^I]$ , we have

$$\begin{aligned} & a_i(\alpha)b_i(\alpha) - \gamma\tau_1 \frac{a(\alpha)^\top b(\alpha)}{2n+m} \\ &= \left[ a_i b_i - \gamma\tau_1 \frac{a^\top b}{2n+m} \right] + \alpha \left[ a_i \Delta b_i + b_i \Delta a_i - \gamma\tau_1 \frac{a^\top \Delta b + b^\top \Delta a}{2n+m} \right] \\ &\quad + \alpha^2 \left[ \Delta a_i \Delta b_i - \gamma\tau_1 \frac{\Delta a^\top \Delta b}{2n+m} \right] \geq \alpha(1 - \gamma\tau_1)\mu - \alpha^2 \left| \Delta a_i \Delta b_i - \gamma\tau_1 \frac{\Delta a^\top \Delta b}{2n+m} \right| \\ &= \alpha(1 - \gamma\tau_1)\mu - \eta_i \alpha^2 \geq \alpha(1 - \gamma\tau_1)\mu - M\alpha^2 \geq 0. \end{aligned}$$

Hence,

$$\alpha^I \geq (1 - \gamma\tau_1)\mu/M.$$

But, as we know,  $\mu = \sigma \frac{a^\top b}{2n+m}$  and for  $\sigma$  bounded away from zero it follows that  $\mu$  is bounded below. Hence,  $\alpha^I$  is bounded away from zero.

Let us now show that the sequence  $\{\alpha^{II}\}$  generated by step 5 of the PDIP algorithm is bounded away from zero. According to the mean-value theorem for vector-valued functions (Dennis and Schnabel, 1983), we have

$$\begin{aligned}
CO(t + \alpha\Delta t) &= CO(t) + \alpha \left[ \int_0^1 \nabla CO(t + \xi\alpha\Delta t) d\xi \right] \Delta t \\
&= CO(t) + \alpha \nabla CO(t) \Delta t + \alpha \left[ \int_0^1 (\nabla CO(t + \xi\alpha\Delta t) - \nabla CO(t)) d\xi \right] \Delta t \\
&= (1 - \alpha)CO(t) + \alpha \left[ \int_0^1 (\nabla CO(t + \xi\alpha\Delta t) - \nabla CO(t)) d\xi \right] \Delta t,
\end{aligned}$$

where the last equality is from (17.38). Having in view that the derivative of  $CO(t)$  is Lipschitz continuous with constant  $L$ , we obtain

$$\|CO(t + \alpha\Delta t)\| \leq \|CO(t)\| |1 - \alpha| + L\alpha^2 \|\Delta t\|^2.$$

Using this inequality, we have

$$\begin{aligned}
\Theta''(\alpha) &= a(\alpha)^T b(\alpha) - \gamma\tau_2 \|CO(t + \alpha\Delta t)\| \\
&= a^T b + \alpha [a^T \Delta b + b^T \Delta a] + \alpha^2 \Delta a^T \Delta b - \gamma\tau_2 \|CO(t + \alpha\Delta t)\| \\
&\geq |1 - \alpha| a^T b + \alpha \sigma a^T b + \alpha^2 \Delta a^T \Delta b - \gamma\tau_2 (\|CO(t)\| |1 - \alpha| + L\alpha^2 \|\Delta t\|^2) \\
&= |1 - \alpha| (a^T b - \gamma\tau_2 \|CO(t)\|) + \alpha \sigma a^T b + \alpha^2 (\Delta a^T \Delta b - \gamma\tau_2 L \|\Delta t\|^2) \\
&\geq \alpha [\sigma a^T b - \alpha |\Delta a^T \Delta b - \gamma\tau_2 L \|\Delta t\|^2].
\end{aligned}$$

From proposition 17.7, there is a constant  $N > 0$  such that

$$|\Delta a^T \Delta b - \gamma\tau_2 L \|\Delta t\|^2| \leq N.$$

Hence  $\Theta''(\alpha) \geq \alpha[\sigma a^T b - \alpha N]$ . From the condition (17.66), it follows that  $\alpha'' \geq \sigma a^T b / N$ . Since the sequence  $\{\sigma_k\}$  is bounded away from zero, then the sequence  $\{\alpha_k''\}$  is also bounded away from zero.  $\blacklozenge$

3. (Convergence) The convergence of the PDIP algorithm is proved as follows.

**Theorem 17.3** Suppose that the functions  $f(x)$ ,  $c_E(x)$ , and  $c_I(x)$  are twice continuously differentiable, and the derivative of  $CO(t)$  is Lipschitz continuous. Let  $\{t_k\}$  be the sequence generated by the PDIP algorithm, where  $\{\sigma_k\} \subset (0, 1)$  is bounded away from zero and one. Then the sequence  $\{F(t_k)\}$  converges to zero and for any limit point  $t^* = [x^*, y^*, z^*, p^*, q^*, s^*, w^*, v^*]^T$ ,  $x^*$  is a KKT point of the problem (17.30).

**Proof** Firstly, note that the sequence  $\{\|F(t_k)\|\}$  is monotonously decreasing, hence it is convergent. By contradiction, suppose that the sequence  $\{\|F(t_k)\|\}$  is not convergent to zero. Then, from proposition 17.5, we have

$$\Phi_k(\alpha_k)/\Phi_k(0) \leq 1 - 2\alpha_k\beta(1 - \sigma_k).$$

Therefore, from proposition 17.8 it follows that the sequence  $\{\Phi_k\}$  linearly converges to zero. This leads to a contradiction. On the other hand, from proposition 17.2 we have  $\nabla\Phi(t_k)\Delta t_k = -2F(t_k)^T F(t_k) + 2\mu_k F(t_k)^T \hat{e}$ . Since the sequence  $\{\alpha_k\}$  is bounded away from zero, it follows that the backtracking line-search used in step 5 of the PDIP algorithm produces

$$\frac{\nabla\Phi(t_k)\Delta t_k}{\|\Delta t_k\|} = \frac{-2[F(t_k)^T F(t_k) - \mu_k F(t_k)^T \hat{e}]}{\|\Delta t_k\|} \rightarrow 0.$$

Therefore, from proposition 17.7,  $\Delta t_k$  is bounded away from zero, i.e.,

$$\Phi(t_k) - \mu_k [s_k^T z_k + w_k^T p_k + v_k^T q_k] \rightarrow 0.$$

However,

$$\Phi(t_k) - \mu_k [s_k^T z_k + w_k^T p_k + v_k^T q_k] \geq (1 - \sigma_k)\Phi(t_k).$$

Therefore, it must hold that  $\Phi(t_k) \rightarrow 0$  because the sequence  $\{\sigma_k\}$  is bounded away from one. Again, this leads to a contradiction. Hence, the sequence  $\{\|F(t_k)\|\}$  must be convergent to zero. Since the KKT conditions for the problem (17.30) are satisfied by  $t^*$ , it follows that  $x^*$  is a KKT point for (17.30).  $\blacklozenge$

**Example 17.1** Let us consider the following nonlinear optimization problem (Hock, & Schittkowski, 1981), (Andrei, 1998c)

$$\begin{aligned} & \min x_1 x_4 (x_1 + x_2 + x_3) + x_3 \\ & \text{subject to} \\ & x_1 x_2 x_3 x_4 - 25 \geq 0, \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 - 40 = 0, \\ & 1 \leq x_i \leq 5, i = 1, 2, 3, 4. \end{aligned}$$

The evolution of some elements corresponding to a simple variant of the PDIP algorithm is presented in Table 17.1.

**Table 17.1** Evolution of some elements of PDIP. Example 17.1

$k$	$f(x_k)$	$\Phi(t_k)$	$\ CO(t_k)\ _2$	$\ CT(\cdot)\ _2$
0	16.0000	534.076	465.076	69.0000
1	16.9494	11.2666	8.97539	2.291261
2	17.1584	0.253076	8.97539	0.135262
3	16.9951	0.020296	1.41485	0.61477e-2
4	17.0168	0.45552e-3	0.18875e-3	0.26677e-3
5	17.0158	0.11076e-4	0.52654e-6	0.10549e-4
6	17.0144	0.42452e-6	0.56172e-9	0.42387e-6
7	17.0141	0.16245e-7	0.65172e-9	0.16244e-7
8	17.0140	0.24373e-10	0.1999e-14	0.2437e-10

**Table 17.2** Evolution of parameters of PDIP, Example 17.1

$k$	$\gamma_k$	$\sigma_k$	$\alpha_{\max}$	$\alpha_k$	$\mu_k$
0	0.750000	0.1	1.090858	1.039720	0.233333
1	0.625000	0.2	0.992283	0.926926	0.90664e-1
2	0.562500	0.2	1.170237	1.0	0.22643e-1
3	0.531250	0.2	1.210422	1.0	0.48802e-2
4	0.515625	0.2	1.165723	1.0	0.10580e-2
5	0.507812	0.2	1.220644	1.0	0.21612e-3
6	0.503906	0.19530	1.238018	1.0	0.42383e-4
7	0.501953	0.03823	1.039720	1.0	0.16244e-5

In this variant of PDIP, the condition (17.68d) was not implemented. Although this condition was not implemented, yet, in the last part of the optimization process we have  $\|CO(t_k)\|_2 < \|CT(\cdot)\|_2$ . This behavior ensures the convergence of the algorithm. Table 17.2 shows the evolution of the algorithm parameters.

The parameter  $\gamma_k$  is updated as  $\gamma_{k+1} = 0.5 + (\gamma_k - 0.5)/2$ . Some other updating formula may be imagined so that  $\gamma_k \in [0.5, \gamma_{k-1}]$ . The parameter  $\sigma_k$  is computed as

$$\sigma_k = \begin{cases} \eta_1, & \text{if } \sigma_k \leq \eta_2(s_k^T z_k + w_k^T p_k + v_k^T q_k), \\ \eta_2(s_k^T z_k + w_k^T p_k + v_k^T q_k), & \text{if } \sigma_k > \eta_2(s_k^T z_k + w_k^T p_k + v_k^T q_k), \end{cases}$$

where  $\eta_1 = 0.1$  and  $\eta_2 = 100$ . Observe that the stepsize  $\alpha_k \rightarrow 1$ , exactly as in the “pure” Newton method.

The solution of the problem is  $x^* = [1, 5, 5, 1]$ . The objective value is 17.0140173. The SPENBAR (Andrei, 1996a, b, c) gives the same solution involving 8 major iterations, 143 minor iterations, and 591 evaluations of the functions defining the problem.

**Example 17.2** Consider the problem (Andrei, 2015a, pp.777)

$$\begin{aligned} & \min 24.55x_1 + 26.75x_2 + 39x_3 + 40.5x_4 \\ & \text{subject to} \\ & 2.3x_1 + 5.6x_2 + 11.1x_3 + 1.3x_4 - 5 \geq 0, \\ & 12x_1 + 11.9x_2 + 41.8x_3 + 52.1x_4 - 21 \\ & \quad - 1.645(0.28x_1^2 + 0.19x_2^2 + 20.5x_3^2 + 0.62x_4^2)^{1/2} \geq 0, \\ & x_1 + x_2 + x_3 + x_4 - 1 = 0, \\ & 0 \leq x_i \leq 10, i = 1, 2, 3, 4. \end{aligned}$$

The PDIP algorithm gives the results from Tables 17.3 and 17.4. Again, observe that the algorithm has the same behavior along the iterations. Even if the condition  $\Theta''(\alpha_k) \geq 0$  was not implemented in PDIP, we can see that after very few iterations,  $\|CO(t_k)\|_2 < \|CT(\cdot)\|_2$ . This ensures the convergence of the algorithm. Again, note that the stepsize converges to 1, like in the pure Newton method.

The solution to this problem is  $x^* = [0.6355, 0.286E - 7, 0.3127, 0.05177]$ . MINOS gives the same solution in 5 major iterations, 18 minor iterations, and 23 evaluations of the function defining

**Table 17.3** Evolution of some elements of PDIP. Example 17.2

$k$	$f(x_k)$	$\Phi(t_k)$	$\ CO(t_k)\ _2$	$\ CT(\cdot)\ _2$
0	130.8000	7422.758	7009.758	413.0000
1	38.03691	26.02877	17.60484	8.423930
2	30.49207	0.278046	0.40382e-3	0.2776424
3	29.96742	0.85973e-2	0.37688e-5	0.85935e-2
4	29.91532	0.51103e-3	0.86534e-4	0.42450e-3
5	29.89686	0.26626e-4	0.90809e-5	0.17545e-4
6	29.89512	0.59983e-6	0.49498e-8	0.59488e-6
7	29.89452	0.26198e-7	0.23005e-8	0.23898e-7
8	29.89439	0.18476e-9	0.12652e-9	0.58238e-10
9	29.89438	0.34199e-12	0.34150e-12	0.49357e-15

**Table 17.4** Evolution of parameters of PDIP. Example 17.2

$k$	$\gamma_k$	$\sigma_k$	$\alpha_{\max}$	$\alpha_k$	$\mu_k$
0	0.750000	0.1	0.9581337	0.9498877	0.51
1	0.625000	0.2	1.098776	1.0	0.1158525
2	0.562500	0.2	1.032449	1.0	0.0282029
3	0.531250	0.2	1.058755	1.0	0.56835e-2
4	0.515625	0.2	1.006443	0.9938785	0.11798e-2
5	0.507812	0.2	1.106218	1.0	0.24329e-3
6	0.503963	0.2	1.244491	1.0	0.48777e-4
7	0.501953	0.048885	1.051240	1.0	0.23898e-5
8	0.500976	0.0024132	1.002405	1.0	0.58237e-8

**Table 17.5** Performances of PDIP for solving 4 applications from the LACOP collection

	$n$	$me$	$mc$	#it	#nf	$KKT$	$CT$	$cpu$	$vfo$
ELCH	10	3	0	18	19	0.185290e-12	0.147536e-11	0.01	-47.761090
ALKI	10	3	8	21	22	0.572627e-08	0.257441e-34	0.01	-1768.8069
MSP3	13	0	15	26	27	0.383793e-08	0.133552e-11	0.02	97.587510
POOL	34	20	0	19	20	0.896223e-10	0.896147e-10	0.1	2785.8000

In this table, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #it = the number of iterations to obtain a solution, #nf = the number of evaluations of the functions defining the problem (including the gradients and Hessians),  $KKT$  = the norm of the KKT conditions,  $CT$  = the norm of the centrality conditions (see (17.36)),  $cpu$  = the CPU computing time for obtaining a solution (seconds),  $vfo$  = the value of the objective function at the optimal point

the problem. On the other hand, SPENBAR needs 10 major iterations, 455 minor iterations, and 2083 evaluations of the functions of the problem, giving the same solution. NLPQLP gives the same solution, needing 14 iterations, and 17 evaluations of the functions.

Table 17.5 shows the performances of PDIP for solving some applications from the LACOP collection.

## 17.5 Trust-Region Interior-Point Algorithm

There are two main differences between the trust-region interior-point and the line-search interior-point algorithms. The first one is that the trust-region interior-point is not a real primal-dual method. Unlike the line-search interior-point method in which both the primal and the dual variables are simultaneously computed, the trust-region interior-point algorithm firstly computes a step in the variables  $(x, s)$  and then the estimates of the multipliers (dual variables). The second difference is that the trust-region interior-point algorithm uses a scaling of the variables, which discourages the movement toward the boundary of the feasible region. This makes the trust-region interior-point algorithms have better convergence properties than those of the line-search interior-point.

In the following, the trust-region algorithm for finding approximate solutions of a fixed barrier problem is presented. Then a trust-region interior-point algorithm is described, where the barrier parameter is driven to zero.

### An Algorithm for the Barrier Problem

Let us consider the barrier problem (17.3). This is an equality constrained optimization problem that can be solved by using a sequential quadratic programming method with trust-region. However, a direct application of this method is not possible because it generates steps that tend to violate the positivity of the slack variables and therefore they frequently cut short the trust-region constraint. To overcome this difficulty, Nocedal and Wright (2006) suggest designing another sequential quadratic programming method.

At iterate  $(x, s)$ , for a given value of the barrier parameter  $\mu$ , the estimates of the Lagrange multipliers (dual variables)  $(y, z)$  are firstly computed and then the step  $p = (p_x, p_s)$  is determined, which approximately solves the following subproblem:

$$\min_{p_x, p_s} \nabla f(x)^T p_x + \frac{1}{2} p_x^T \nabla_{xx}^2 L p_x - \mu e^T S^{-1} p_s + \frac{1}{2} p_s^T \Sigma p_s \quad (17.69a)$$

subject to

$$J_E(x)p_x + c_E(x) = r_E, \quad (17.69b)$$

$$J_I(x)p_x - p_s + (c_I(x) - s) = r_I, \quad (17.69c)$$

$$\|(p_x, S^{-1}p_s)\|_2 \leq \Delta, \quad (17.69d)$$

$$p_s \geq -\tau s. \quad (17.69e)$$

In this problem,  $\Sigma = S^{-1}Z$  and the scalar parameter  $\tau \in (0, 1)$  is chosen close to 1, (for example  $\tau = 0.995$ ). The inequality (17.69e) has the same role as the fraction to the boundary rule given by (17.8). The constraints (17.69b) and (17.69c) are linear. Clearly, we would like to set  $r = (r_E, r_I) = 0$ , but this can cause the constraints (17.69b) (17.69c), and (17.69d) to be incompatible or to give a step  $p$  that makes little progress toward feasibility. Therefore, the parameter  $r$  is selected by an auxiliary computation like in the sequential quadratic programming with trust-region.

The choice of the objective function (17.69a) is motivated by the fact that the first-order optimality conditions of (17.69a), (17.69b), and (17.69c) are exactly those given by (17.2) in which (17.2b) is scaled by  $S^{-1}$ .

The trust-region constraint (17.69d) guarantees that the problem (17.69) has a finite solution even when  $\nabla_{xx}^2 L(x, s, y, z)$  is not positive definite, and therefore this Hessian never needs to be modified. Besides, the trust-region formulation ensures that adequate progress is made at every iteration.

The scaling  $S^{-1}$  used in (17.69d) is motivated by the crucial fact that the shape of the trust-region must take into account the requirement that the slacks should not approach zero prematurely. The scaling  $S^{-1}$  restricts those components  $i$  of  $p_s$  for which  $s_i$  are close to their lower bound of zero.

In the following, we present the sequential quadratic programming algorithm with trust-region for the barrier problem. This means that at every iteration the subproblem (17.69) is solved. The condition of stopping the iterations is defined by the error function (17.9) which uses the perturbed KKT system (17.2) as well as by the sufficient reduction of the merit function  $\Phi_\sigma$  defined by (17.21). The algorithm is as follows.

**Algorithm 17.5** *Trust-region algorithm for barrier problems*

- |    |  |
|----|--|
| 1. | Choose the initial point $x_0$ and $s_0 > 0$ . Compute the initial values of the multipliers $y_0$ and $z_0 > 0$ . Select an initial value of the barrier parameter $\mu > 0$ . Consider a tolerance $\varepsilon_\mu$ and set $k = 0$   |
| 2. | If $E(x_k, s_k, y_k, z_k, \mu) \geq \varepsilon_\mu$ , then continue with step 3; otherwise, stop; the current point is the solution of the problem  |
| 3. | Compute $p = (p_x, p_s)$ as an approximate solution of the subproblem (17.69)  |
| 4. | If $p$ provides sufficient decrease in the merit function $\Phi_\sigma$ , then set $x_{k+1} = x_k + p_x$ , $s_{k+1} = s_k + p_s$ , compute the new multiplier estimates $y_{k+1}, z_{k+1} > 0$ and set $\Delta_{k+1} \geq \Delta_k$ . Continue with step 5. Otherwise, define $x_{k+1} = x_k$ , $s_{k+1} = s_k$ and set $\Delta_{k+1} < \Delta_k$ . Continue with step 3 |
| 5. | Set $k = k + 1$ and go to step 2   |

◆

Now, let us discuss some aspects regarding the determination of an approximate solution of the subproblem (17.69) as well as the estimates  $(y_{k+1}, z_{k+1})$  of the Lagrange multipliers.

### Solving the Subproblem (17.69)

This problem is approximately solved. At the very beginning, a change of variable is made, which transforms the trust-region constraint (17.69d) into a ball. By defining

$$\tilde{p} = \begin{bmatrix} p_x \\ \tilde{p}_s \end{bmatrix} = \begin{bmatrix} p_x \\ S^{-1}p_s \end{bmatrix}, \quad (17.70)$$

the subproblem (17.69) becomes

$$\min_{p_x, \tilde{p}_s} \nabla f(x)^T p_x + \frac{1}{2} p_x^T \nabla_{xx}^2 L p_x - \mu e^T \tilde{p}_s + \frac{1}{2} \tilde{p}_s^T S \Sigma S \tilde{p}_s \quad (17.71a)$$

subject to

$$J_E(x)p_x + c_E(x) = r_E, \quad (17.71b)$$

$$J_I(x)p_x - S\tilde{p}_s + (c_I(x) - s) = r_I, \quad (17.71c)$$

$$\|(p_x, \tilde{p}_s)\|_2 \leq \Delta, \quad (17.71d)$$

$$\tilde{p}_s \geq -\tau e. \quad (17.71e)$$

To compute the vectors  $r_E$  and  $r_I$  consider the following subproblem in the variables  $v = (v_x, v_s)$ :

$$\min_v \|J_E(x)v_x + c_E(x)\|_2^2 + \|J_I(x)v_x - Sv_s + (c_I(x) - s)\|_2^2 \quad (17.72a)$$

subject to

$$\|(v_x, v_s)\|_2 \leq 0.8\Delta, \quad (17.72b)$$

$$v_s \geq -\frac{\tau}{2}e. \quad (17.72c)$$

If the constraint (17.72c) is ignored, then the problem (17.72) has a standard form of a trust-region problem and therefore an approximate solution of it can be obtained by the known techniques. If the solution of this problem violates the bounds (17.72c), then a backtracking can be used to satisfy them.

Having a solution  $(v_x, v_s)$  for (17.72), the vectors  $r_E$  and  $r_I$  are computed as

$$r_E = J_E(x)v_x + c_E(x), \quad (17.73a)$$

$$r_I = J_I(x)v_x - Sv_s + (c_I(x) - s). \quad (17.73b)$$

With these developments, we are ready to compute an approximate solution  $\tilde{d}$  for (17.71) as follows. From (17.73), we can note that  $v$  is a particular solution of the linear constraints (17.71b) and (17.71c). Therefore, the subproblem (17.71a), (17.71b), and (17.71c), which is a quadratic programming problem with equality constraints, can be solved by using, for example, the projected conjugate gradient algorithm. During the solving process, the satisfaction of the trust-region constraint (17.71d) is monitored. If the boundary of this region is reached or if a negative curvature is detected or if an approximate solution is obtained, then the algorithm is stopped. If the solution obtained by the projected conjugate gradient algorithm does not satisfy the bounds (17.71e), then a backtracking is used until all these are satisfied. After the step  $(p_x, \tilde{p}_s)$  has been computed, the vector  $p$  is recovered from (17.70).

It should be mentioned that at each iteration, the projected conjugate gradient algorithm requires solving a linear system in order to perform the projection operation. For the quadratic programming (17.71a), (17.71b), and (17.71c), this projection matrix is

$$\begin{bmatrix} I & \hat{A}^T \\ \hat{A} & 0 \end{bmatrix}, \text{ where } \hat{A} = \begin{bmatrix} J_E(x) & 0 \\ J_I(x) & -S \end{bmatrix}. \quad (17.74)$$

Although this trust-region approach requires the solution of an augmented linear system, the matrix (17.74) is simpler than the primal-dual matrix from (17.11). Moreover, the advantage is that the Hessian  $\nabla_{xx}^2 L$  needs never to be factored because the conjugate gradient approach requires only products of this matrix with vectors. Besides, the matrix  $S\Sigma S$  from (17.71a) has a much tighter distribution of eigenvalues than  $\Sigma$ . Therefore, the conjugate gradient method will run much better in the presence of ill-conditioning, being much more suitable for solving the quadratic problem (17.71a), (17.71b), and (17.71c) (Nocedal, & Wright, 2006).

### Lagrange Multipliers Estimates and Step Acceptance

As we have already seen in step 4 of Algorithm 17.5, the estimates of the Lagrange multipliers  $y_{k+1}$  and  $z_{k+1} > 0$  are computed. This is done as follows. At the iterate  $(x, s)$ , the  $(y, z)$  are chosen as the least-square multipliers corresponding to the subproblem (17.71a), (17.71b), and (17.71c), i.e.,

$$\begin{bmatrix} y \\ z \end{bmatrix} = \left( \widehat{A} \widehat{A}^T \right)^{-1} \widehat{A} \begin{bmatrix} \nabla f(x) \\ -\mu e \end{bmatrix}, \quad (17.75)$$

where  $\widehat{A}$  is the matrix from (17.74). It is quite possible that the estimates  $z$  obtained from (17.75) may not always be positive. To enforce the positivity of this multiplier, the following computational scheme can be used:

$$z_i = \min \{10^{-3}, \mu/s_i\}, \quad i = 1, \dots, m. \quad (17.76)$$

Observe that if all the components of  $z$  are defined as in (17.76), then the matrix  $\Sigma$  reduces to  $\mu S^{-2}$ . The quantities  $\mu/s_i$  are called the *i-th primal multiplier estimate* of multipliers.

As in the standard trust-region method, step  $p$  is accepted if

$$\text{ared}(p) \geq \eta \text{pred}(p), \quad (17.77)$$

where

$$\text{ared}(p) = \Phi_\sigma(x, s) - \Phi_\sigma(x + p_x, s + p_s), \quad (17.78)$$

$$\text{pred}(p) = q_\sigma(0) - q_\sigma(p), \quad (17.79)$$

$$q_\sigma(p) = \nabla f(x)^T p_x + \frac{1}{2} p_x^T \nabla_{xx}^2 L p_x - \mu e^T S^{-1} p_s + \frac{1}{2} p_s^T \Sigma p_s + \sigma m(p), \quad (17.80)$$

$$m(p) = \left\| \begin{bmatrix} J_E(x)p_x + c_E(x) \\ J_I(x)p_x - p_s + c_I(x) - s \end{bmatrix} \right\|_2. \quad (17.81)$$

In (17.77),  $\eta$  is a constant in  $(0, 1)$ , for example  $\eta = 10^{-8}$ .

To determine a corresponding value for the penalty parameter  $\sigma$ , we impose that this is large enough so that

$$\text{pred}(p) \geq \rho \sigma (m(0) - m(p)), \quad (17.82)$$

where  $\rho$  is a parameter in  $(0, 1)$ . Observe that this condition is exactly as (15.38). Therefore, the value of  $\sigma$  can be computed by the procedure described in (15.39) from the sequential quadratic programming.

### Description of the Trust-Region Interior-Point Algorithm

For updating the barrier parameter, the Fiacco-McCormick strategy is used. The condition of stopping the algorithm uses the error function  $E$  defined by (17.9). If the quasi-Newton approach is used, then the Hessian  $\nabla_{xx}^2 L$  is replaced by a symmetric approximation (BFGS or limited memory BFGS).

#### Algorithm 17.6 Trust-region interior-point algorithm

- |    |  |
|----|--|
| 1. | Choose the value for the parameters $\eta > 0$ , $\tau \in (0, 1)$ , $\sigma \in (0, 1)$ , $\rho \in (0, 1)$ , and $\xi \in (0, 1)$ . Choose the tolerances $\epsilon_\mu$ and $\epsilon_{TOL}$ small enough. If the quasi-Newton approach is used, choose a symmetric $n \times n$ matrix $B_0$ . Choose the initial values of the parameters $\mu > 0$ and $\Delta_0$ as well as the initial points $x_0, s_0 > 0$ . Set $k = 0$ |
| 2. | If $E(x_k, s_k, y_k, z_k, 0) > \epsilon_{TOL}$ , then go to step 3; otherwise, stop  |
| 3. | If $E(x_k, s_k, y_k, z_k, \mu) > \epsilon_\mu$ , then go to step 4; otherwise, go to step 13   |

4.	Using (17.75) and (17.76), compute the Lagrange multipliers
5.	Compute $\nabla_{xx}^2 L(x_k, s_k, y_k, z_k)$ or update a quasi-Newton approximation $B_k$ and define $\Sigma_k = S_k^{-1} Z_k$
6.	Compute the normal step $v_k = (v_x, v_s)$
7.	Compute $\tilde{p}_k$ by applying the projected conjugate gradient algorithm to the problem (17.71)
8.	Using (17.70), compute the total step $p_k$
9.	Update $v_k$ to satisfy ((17.82))
10.	Compute $\text{pred}(p_k)$ and $\text{ared}(p_k)$ by (17.79) and (17.78), respectively
11.	If $\text{ared}(p_k) \geq \eta \text{pred}(p_k)$ , then set $x_{k+1} = x_k + p_x$ , $s_{k+1} = s_k + p_s$ , and choose $\Delta_{k+1} \geq \Delta_k$ . Otherwise, set $x_{k+1} = x_k$ , $s_{k+1} = s_k$ and choose $\Delta_{k+1} < \Delta_k$
12.	Set $k = k + 1$ and go to step 3
13.	Set $\mu = \sigma\mu$ , update $\varepsilon_\mu$ and go to step 2

◆

The tolerance on the barrier parameter can be defined as  $\varepsilon_\mu = \mu$ , or an adaptive strategy can be used for its updating. It is quite clear that the merit function can lead to the Maratos effect. In this case, the second-order corrections or a nonmonotone strategy can be used. Algorithm 17.6 is implemented in KNITRO/INTERIOR (Byrd, Hribar, and Nocedal, 1999), which uses the exact Hessian or a quasi-Newton approximation of this matrix and follows to be presented in the next section.

## 17.6 Interior-Point Sequential Linear-Quadratic Programming (KNITRO/INTERIOR)

In Chap. 15, the KNITRO/ACTIVE algorithm based on the active-set sequential linear-quadratic programming method has been presented. This chapter describes the KNITRO/INTERIOR algorithm, together with its numerical performances for solving large-scale general continuous nonlinear optimization problems. KNITRO/INTERIOR provides two procedures for computing the steps within the interior-point approach. In the version INTERIOR-CG, each step is computed using a projected conjugate gradient iteration. It factors a projection matrix and uses the conjugate gradient method to approximately minimize a quadratic model of the barrier problem. In the version INTERIOR-DIRECT, the algorithm attempts to compute a new iterate by solving the primal-dual KKT system using direct linear algebra. In case this step cannot be guaranteed to be of good quality or if a negative curvature is detected, then the new iterate is computed by the INTERIOR-CG algorithm. The description of the KNITRO/INTERIOR-CG algorithm is given in (Byrd, Hribar, & Nocedal, 1999) and its global convergence theory is presented in (Byrd, Gilbert, & Nocedal, 2000). The method implemented in the KNITRO/INTERIOR-DIRECT algorithm is described in (Waltz, Morales, Nocedal, and Orban, 2003).

Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad (17.83a)$$

subject to

$$c_E(x) = 0, \quad (17.83b)$$

$$c_I(x) \geq 0, \quad (17.83c)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c_E: \mathbb{R}^n \rightarrow \mathbb{R}^I$ , and  $c_I: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are twice continuously differentiable functions.

The interior-point or the barrier methods implemented in KNITRO associate the following barrier problem to (17.83):

$$\min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} f(x) - \mu \sum_{i=1}^m \log(s_i) \quad (17.84a)$$

subject to

$$c_E(x) = 0, \quad (17.84b)$$

$$c_I(x) - s = 0, \quad (17.84c)$$

where  $s \in \mathbb{R}^m$  is a vector of slack variables and  $\mu > 0$  is the barrier parameter. The interior-point methods consist in finding the approximate solutions of the barrier problem (17.84) for a sequence of positive barrier parameters  $\{\mu_k\}$  that converges to zero.

The KKT optimality conditions for (17.84) are as follows:

$$\nabla f(x) - J_E^T(x)y - J_I^T(x)z = 0, \quad (17.85a)$$

$$-\mu e + Sz = 0, \quad (17.85b)$$

$$c_E(x) = 0, \quad (17.85c)$$

$$c_I(x) - s = 0, \quad (17.85d)$$

where  $e = [1, \dots, 1]^T$ ,  $S = \text{diag}(s_1, \dots, s_m)$ ,  $J_E(x)$ , and  $J_I(x)$  are the Jacobian matrices corresponding to the equality and inequality constraints vectors, respectively. The vectors  $y$  and  $z$  are the Lagrange multipliers (dual variables) associated to the equality and inequality constraints. We also have  $s, z > 0$ .

In the line-search approach, the Newton method is applied to (17.85), backtracking if necessary, so that the variables  $s$  and  $z$  remain positive and a merit function is sufficiently reduced. In the trust-region approach, a quadratic programming problem is associated to (17.84) and the step of the algorithm is an approximate solution of this quadratic programming sub-problem. These two approaches are implemented in the KNITRO/INTERIOR-DIRECT and KNITRO/INTERIOR-CG algorithms, respectively.

A very important component in the interior-point methods is the procedure for choosing the sequence of the barrier parameters  $\{\mu_k\}$ . KNITRO contains several options. In the *Fiacco-McCormic monotone strategy*, the barrier parameter  $\mu$  is held fixed for a series of iterations until the KKT (17.85) are satisfied to some accuracy. In the *adaptive strategy*, the barrier parameter is updated at every iteration using different rules: the rule implemented in LOQO based on the deviation of the minimum complementarity pair from the average (Vanderbei, & Shanno, 1999), a probing strategy that uses Mehrotra's predictor step to select a target value for  $\mu$ , a quality-function approach and some other rules described in (Nocedal, Wächter, & Waltz, 2005).

To control the quality of the steps, both interior-point algorithms implemented in KNITRO make use of a nondifferentiable merit function

$$\Phi_\sigma(x, s) = f(x) - \mu \sum_{i=1}^m \log(s_i) + \sigma \|c_E(x)\|_2 + \sigma \|c_I(x) - s\|_2, \quad (17.86)$$

where  $\sigma > 0$ . A step is acceptable only if it provides a sufficient decrease of the merit function (17.86). Let us now present the interior-point algorithms implemented in KNITRO.

### KNITRO/INTERIOR-DIRECT Algorithm

In this algorithm, the search direction is determined by the direct solving of the Newton system associated to the nonlinear system given by the KKT optimality conditions (17.85). To obtain global

convergence in the presence of the nonconvexity of the Hessian or of the Jacobian singularities, this step may be replaced under certain circumstances by a safeguarding trust-region step. KNITRO/INTERIOR-DIRECT is described in (Byrd, Nocedal, and Waltz, 2006). (See also (Waltz, Morales, Nocedal, and Orban, 2003).)

By applying the Newton method to the system (17.85) in the variables  $x, s, y, z$ , we get

$$\begin{bmatrix} \nabla_{xx}^2 L & 0 & -J_E^T(x) & -J_I^T(x) \\ 0 & Z & 0 & S \\ J_E(x) & 0 & 0 & 0 \\ J_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = -\begin{bmatrix} \nabla f(x) - J_E^T(x)y - J_I^T(x)z \\ Sz - \mu e \\ c_E(x) \\ c_I(x) - s \end{bmatrix}, \quad (17.87)$$

where, as usual,  $L$  represents the Lagrange function

$$L(x, s, y, z) = f(x) - y^T c_E(x) - z^T (c_I(x) - s). \quad (17.88)$$

If the inertia of the matrix in (17.87) is

$$(m + n, l + m, 0), \quad (17.89)$$

then the step  $d$  determined as a solution of (17.87) can be guaranteed to be a descent direction for the merit function (17.86). In this case, compute the scalars

$$\alpha_s^{\max} = \max \{ \alpha \in (0, 1] : s + \alpha d_s \geq (1 - \tau)s \}, \quad (17.90a)$$

$$\alpha_z^{\max} = \max \{ \alpha \in (0, 1] : z + \alpha d_z \geq (1 - \tau)z \}, \quad (17.90b)$$

where  $\tau = 0.995$ . If  $\min \{ \alpha_s^{\max}, \alpha_z^{\max} \}$  is not too small, then perform a backtracking line-search that computes the stepsizes

$$\alpha_s \in (0, \alpha_s^{\max}], \alpha_z \in (0, \alpha_z^{\max}], \quad (17.91)$$

which achieve a sufficient decrease of the merit function (17.86).

The new iterate is computed as

$$x^+ = x + \alpha_s d_x, s^+ = s + \alpha_s d_s, \quad (17.92a)$$

$$y^+ = y + \alpha_z d_y, z^+ = z + \alpha_z d_z. \quad (17.92b)$$

On the other hand, if the inertia of the matrix in (17.87) is not as in (17.89) or if the stepsize  $\alpha_s$  or  $\alpha_z$  is less than a given threshold, then the step  $d$  solution of (17.87) is rejected. In this case, the search direction  $d$  is not a descent one for the merit function (17.86). The algorithm reverts to the trust-region method implemented in KNITRO/INTERIOR-CG, algorithm guaranteed to provide a successful step even in the presence of a negative curvature or singularity and which is described in the next section.

It is worth mentioning that this strategy implemented in KNITRO/INTERIOR-DIRECT is different from the line-search interior-point algorithms IPOPT (Wächter, & Biegler, 2005a, 2005b, 2006) and LOQO (Vanderbei, & Shanno, 1999). Whenever the inertia condition is not satisfied, then IPOPT and LOQO modify the Hessian  $\nabla_{xx}^2 L$ . Instead, KNITRO/INTERIOR-DIRECT implements the above strategy because it permits to compute a step by using a null space approach without modifying the Hessian  $\nabla_{xx}^2 L$ . Moreover, using the trust-region step guarantees progress in case the line-search approach fails (Wächter, & Biegler, 2000), (Byrd, Marazzi, & Nocedal, 2004b).

It is known that the step lengths  $\alpha_s$  or  $\alpha_z$  in (17.92) converge to zero when the line-search iterations converge to nonstationary points. In this case, the algorithm monitors these step lengths. If one of them is smaller than a given threshold, then the iterate given by (17.87) and (17.92) is discarded and replaced with the trust-region step.

Therefore, the algorithm monitors two criteria: the *inertia* of the matrix in (17.87) and the *step lengths* (17.91). If these two criteria are satisfied, then we continue the iterations by using the Newton method for solving (17.87) by the direct method.

For the initialization of the algorithm, it is necessary to compute the initial values of the multipliers  $y_0$  and  $z_0$ . The multipliers are computed as the least-squares solution of the system given by (17.85a) and (17.85b). If the line-search is discarded, then one or more KNITRO/INTERIOR-CG iterations are executed until one of them provides sufficient reduction in the merit function. The KNITRO/INTERIOR-DIRECT algorithm presented below uses  $D\Phi_\sigma(x, s, d)$ , which is the directional derivative of the merit function  $\Phi_\sigma$  along a direction  $d$ .

### Algorithm 17.7 KNITRO/INTERIOR-DIRECT—Byrd, Hribar, and Waltz

1. Choose the initial point  $x_0, s_0 > 0$ , and the parameters  $\eta > 0$  and  $0 < \alpha_{\min} < 1$ . Compute the initial values of the multipliers  $y_0$  and  $z_0 > 0$ . Choose a value of the trust-region radius  $\Delta_0 > 0$  and a value for the barrier parameter  $\mu > 0$ . Set  $k = 0$
2. If a test for stopping the iterations is satisfied, then stop; otherwise, go to step 3
3. If the perturbed KKT optimality conditions (17.85) are approximately satisfied, then go to step 9; otherwise, go to step 4
4. Factorize the matrix of the primal-dual system (17.87) and compute  $neig$  as the number of the negative eigenvalues of this matrix
5. Set  $LineSearch = False$
6. If  $neig \leq l + m$ , then:
  - Solve the system (17.87) to obtain the direction  $d = (d_x, d_s, d_y, d_z)$
  - Define  $w = (x_k, s_k)$  and  $d_w = (d_x, d_s)$
  - Compute  $\alpha_s^{\max}$  and  $\alpha_z^{\max}$  as in (17.90)
  - If  $\min\{\alpha_s^{\max}, \alpha_z^{\max}\} > \alpha_{\min}$ , then:
    - Update the penalty parameter  $\sigma_k$  (see the merit function below)
    - Compute a stepsize  $\alpha_s = \bar{\alpha}\alpha_s^{\max}$ ,  $\bar{\alpha} \in (0, 1]$ , such that  $\Phi_\sigma(w + \alpha_s d_w) \leq \Phi_\sigma(w) + \eta\alpha_s D\Phi_\sigma(w, d_w)$
    - If  $\alpha_s > \alpha_{\min}$ , then:
      - Set  $\alpha_z = \bar{\alpha}\alpha_z^{\max}$
      - Compute  $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$  as in (17.92)
      - Set  $LineSearch = True$
7. If  $LineSearch = False$ , then compute  $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$  using the algorithm KNITRO/INTERIOR-CG described in the next section
8. Compute  $\Delta_{k+1}$ . Set  $k = k + 1$  and go to step 3
9. Choose a new smaller value for the barrier parameter  $\mu$  and go to step 2

◆

Observe that at every iteration the algorithm computes and maintains a trust-region radius  $\Delta_k$  in case it needs to revert to the trust-region KNITRO/INTERIOR-CG algorithm in step 7.

### KNITRO/INTERIOR-CG Algorithm

The second algorithm implemented in KNITRO computes the search directions by using a quadratic model and trust-regions. This strategy permits great freedom in the choice of the Hessian and provides a mechanism for coping with the Jacobian and Hessian singularities. However, the iterations of this algorithm are more complex. The KNITRO/INTERIOR-CG algorithm is described in (Byrd, Hribar, and Nocedal, 1999) and analyzed in (Byrd, Gilbert, and Nocedal, 2000).

To describe this algorithm, observe that the barrier problem (17.84) is an equality constrained optimization problem which can be solved by using a sequential quadratic programming (SQP) method with trust-regions. However, a straightforward application of the SQP method to the barrier problem leads to inefficient steps that tend to violate the positivity of the slack variables and frequently cuts short the trust-region constraint. To overcome this difficulty, Byrd, Nocedal, and Waltz (2006) suggested the following SQP method associated to the barrier problem (17.84).

At the current iterate  $(x_k, s_k)$  and for a given value of the barrier parameter  $\mu$ , firstly compute the Lagrange multiplier estimates  $(y_k, z_k)$  and then the step  $d = (d_x, d_s)$  as solution of the following subproblem:

$$\begin{aligned} \min_{d_x, d_s} \quad & \nabla f(x_k)^T d_x + \frac{1}{2} d_x^T \nabla_{xx}^2 L(x_k, s_k, y_k, z_k) d_x \\ & - \mu e^T S_k^{-1} d_s + \frac{1}{2} d_s^T \Sigma_k d_s \end{aligned} \quad (17.93a)$$

subject to

$$c_E(x_k) + J_E(x_k) d_x = r_E, \quad (17.93b)$$

$$c_I(x_k) + J_I(x_k) d_x - d_s - s_k = r_I, \quad (17.93c)$$

$$\|d_x, S_k^{-1} d_s\|_2 \leq \Delta_k, \quad (17.93d)$$

$$d_s \geq -\tau s, \quad (17.93e)$$

where  $\Sigma_k = S_k^{-1} Z_k$  and  $\tau = 0.995$ . Ideally, the residues  $r_E$  and  $r_I$  should be zero, i.e.,  $r = (r_E, r_I) = 0$ , but since this value can determine the constraints to be incompatible or produce a poor step, we would rather choose  $r$  as the smallest vector such that the constraints (17.93b), (17.93c), and (17.93d) should be consistent.

The choice of the objective function (17.93a) is motivated by the fact that the first-order optimality conditions of (17.93a), (17.93b), and (17.93c) are given by (17.85) with the second block of equations scaled by  $S^{-1}$ . Therefore, the steps computed by using (17.93) are related to those of the line-search algorithm described in the previous section. The trust-region constraint (17.93d) guarantees that (17.93) has a finite solution even if the Hessian of the Lagrangian  $\nabla_{xx}^2 L(x_k, s_k, y_k, z_k)$  is not positive definite. Hence, the Hessian matrix never needs to be modified in this algorithm. The scaling  $S_k^{-1}$  used in the trust-region constraint is crucial in the economy of the algorithm.

### Step Computation

The presence of the nonlinear constraints (17.93d) and of the bounds (17.93e) makes the highly accurate solving of (17.93) difficult. However, useful inexact solutions can be computed at a moderate cost. KNITRO uses the null space approach, in which step  $d$  is computed as a sum of a *normal step*  $v$  that attempts to satisfy the linear constraints (17.93b) and (17.93c) with  $r = 0$  and also, possibly, the trust-region, and of a *tangential step* that lies on the tangent space of the constraints and that tries to achieve optimality.

The normal step  $v = (v_x, v_s)$ , is the solution of the following subproblem:

$$\min_v \|J_E v_x + c_E\|_2^2 + \|J_I v_x - v_s + c_I - s\|_2^2 \quad (17.94a)$$

subject to

$$\|(v_x, S_k^{-1} v_s)\|_2 \leq 0.8\Delta, \quad (17.94b)$$

in which the arguments of the functions appearing in (17.94) have been omitted. An inexact solution to the subproblem (17.94) is computed by using a dogleg approach, which minimizes (17.94a) along a piecewise linear path composed of a steepest descent step in the norm used in (17.94b) and a minimum-norm Newton step with respect to the same norm. The scaling  $S^{-1}v_s$  in the norm from (17.94b) tends to limit the extent to which the bounds on the slack variables are violated.

Once the normal step  $v$  has been computed, the vectors  $r_E$  and  $r_I$  from (17.93b) and (17.93c), respectively, are computed as residuals, namely,

$$r_E = J_E v_x + c_E, \quad r_I = J_I v_x - v_s + (c_I - s).$$

Once the normal step  $v = (v_x, v_s)$  is computed, the subproblem (17.93) can be written as

$$\min_{d_x, d_s} \nabla f^T d_x - \mu e^T S^{-1} d_s + \frac{1}{2} (d_x^T \nabla_{xx}^2 L d_x + d_s^T \Sigma d_s) \quad (17.95a)$$

subject to

$$J_E d_x = J_E v_x, \quad (17.95b)$$

$$J_I d_x - d_s = J_I v_x - v_s, \quad (17.95c)$$

$$\|(d_x, S^{-1}d_s)\|_2 \leq \Delta, \quad (17.95d)$$

which is called *tangential subproblem*. Now, to find an approximate solution  $d$  of the subproblem (17.95), firstly introduce the scaling

$$\tilde{d}_s = S^{-1}d_s, \quad (17.96)$$

which transforms (17.95d) into a sphere. Then the projected conjugate gradient (CG) method to the transformed quadratic program is applied, where all the iterates are in the linear manifold defined by (17.95b) and (17.95c). While solving by using CG, the strategy of Steihaug is used, with the monitoring of the satisfaction of the trust-region constraint (17.95d). The iterations are stopped if the boundary of this region is reached or if a negative curvature is detected. Finally, if necessary, step  $d$  is truncated to satisfy (17.93e). The KNITRO/INTERIOR-CG algorithm is as follows.

#### Algorithm 17.8 KNITRO/INTERIOR-CG—Byrd, Hribar, and Waltz

1.	Choose $x_0, s_0 > 0$ and $\Delta_0 > 0$ . Choose a value for $\eta > 0$ . Set $k = 0$
2.	If a test for stopping the algorithm is satisfied, stop; otherwise, go to step 3
3.	If the perturbed KKK system (17.85) is approximately satisfied, go to step 11; otherwise, go to step 4
4.	Compute the normal step $v_k = (v_x, v_s)$
5.	Compute the Lagrange multipliers $y_k$ and $z_k > 0$
6.	Compute the total step $d_k$ by applying the projected conjugate gradient algorithm to the subproblem (17.95a), (17.95b), and (17.95c)
7.	Update the penalty parameter $\sigma_k$ (see the merit function below)
8.	Compute $ared_k(d_k)$ and $pred_k(d_k)$ by using the relations below
9.	If $ared_k(d_k) \geq \eta pred_k(d_k)$ , then set $x_{k+1} = x_k + d_x, s_{k+1} = s_k + d_s$ and update $\Delta_{k+1}$ ; otherwise, set $x_{k+1} = x_k, s_{k+1} = s_k$ and choose $\Delta_{k+1} < \Delta_k$
10.	Set $k = k + 1$ and go to step 3
11.	Choose a smaller value for the barrier parameter $\mu$ and go to step 2



In Algorithm 17.8,

$$\text{ared}(d) = \Phi_\sigma(x, s) - \Phi_\sigma(x + d_x, s + d_s) \quad (17.97)$$

is the actual reduction in the merit function.  $\text{pred}(d)$  is immediately defined.

The Lagrange multiplier estimates  $(y_k, z_k)$  are computed by the least squares approximation to the Eqs. (17.85a) and (17.85b) evaluated at the point  $x_k$  and truncated to ensure the positivity of  $z_k$ .

Observe that the interior-point Algorithm 17.8 is asymptotically equivalent to the standard line-search interior-point algorithms, but it is significantly different in two respects. Firstly, it is not a fully primal-dual method because the multipliers are computed as a function of the primal variables  $(x, s)$  as opposed to the formulation (17.87) in which the primal and the dual variables are computed simultaneously from their previous values. Secondly, the trust-region method uses a scaling of the variables that avoids moving toward the boundary of the feasible region. Therefore, the algorithm generates steps quite different from those produced by a line-search method.

### Merit Function

The role of the merit function (17.86) is to determine whether a step can be accepted. Byrd, Hribar, and Nocedal (1999) show that the efficiency of the algorithm depends on the choice of the penalty parameter  $\sigma$ . In both interior-point algorithms implemented in KNITRO/INTERIOR, at every iteration  $\sigma$  is chosen so that the decrease in the quadratic model of the merit function produced by a step  $d$  is proportional with  $\sigma$  multiplied by the decrease in the linearized constraints.

More exactly, suppose that either the KNITRO/INTERIOR-DIRECT algorithm or KNITRO/INTERIOR-CG has generated a step  $d$ . Then define the following linear/quadratic model of the merit function  $\Phi_\sigma$ :

$$Q_\sigma(d) = \nabla f^T d_x - \mu e^T S^{-1} d_s + \frac{\xi}{2} (d_x^T \nabla_{xx}^2 L d_x + d_s^T \Sigma d_s) + \sigma(m(0) - m(d)), \quad (17.98)$$

where

$$m(d) = \left\| \begin{bmatrix} J_E d_x + c_E \\ J_I d_x - d_s + c_I - s \end{bmatrix} \right\|_2 \quad (17.99)$$

represents the first-order violation of the constraints and  $\xi$  is a parameter that follows to be discussed. Define the predicted decrease in the merit function as

$$\text{pred}(d) = Q_\sigma(0) - Q_\sigma(d). \quad (17.100)$$

In all the cases, the penalty parameter  $\sigma$  is chosen large enough such that

$$\text{pred}(d) \geq \rho \sigma(m(0) - m(d)), \quad (17.101)$$

where  $0 < \rho < 1$  (for example  $\rho = 0.1$ ).

If the value of  $\sigma$  from the previous iteration satisfies (17.101), then  $\sigma$  is left unchanged. Otherwise,  $\sigma$  is increased so that it satisfies the inequality (17.101) with some accuracy.

For the trust-region method as implemented in KNITRO/INTERIOR-CG, set  $\xi = 1$  in (17.98). On the other hand, in KNITRO/INTERIOR-DIRECT, parameter  $\xi$  is defined as

$$\xi = \begin{cases} 1, & \text{if } d_x^T \nabla_{xx}^2 L d_x + d_s^T \Sigma d_s > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (17.102)$$

This choice of  $\xi$  guarantees that the directional derivative of  $\Phi_\sigma$  in the direction  $d$  is negative.

### Computational Aspects

KNITRO/INTERIOR contains many algorithmic options and features that are listed and discussed in the documentation of the package (Waltz, 2004). In the following, we shall present some of these options implemented in KNITRO.

*Initial point strategy.* It is well known that any interior-point algorithm is very sensitive to the choice of the initial point. In KNITRO, several different strategies are implemented. One strategy is as follows. At an initial point  $x_0$  specified by the user, an affine scaling step  $d^A = (d_x^A, d_s^A, d_y^A, d_z^A)$  is computed by solving the system (17.87) with  $\mu = 0$ . Then,  $s_1 = \max\{1, |s_0 + d_s^A|\}$  and  $z_1 = \max\{1, |z_0 + d_z^A|\}$  are defined, where the operators max and absolute are component-wise applied. The primal variables  $x$  and the multipliers  $y$  associated to the equality constraints are not altered, i.e.,  $(x_1, y_1) = (x_0, y_0)$ . Finally, the initial value of the barrier parameter is computed as  $\mu_1 = s_1^T z_1 / m$ .

*Hessian options.* There are some options in KNITRO for using the second derivatives. One possibility is that the user can supply the first and the second derivatives, which generally results in the greatest level of robustness and efficiency for all the three algorithms implemented in KNITRO. In some particular applications, the Hessian  $\nabla_{xx}^2 L$  cannot be computed or is too large to store. In these cases, KNITRO/INTERIOR-CG and KNITRO/ACTIVE allow the user to provide products Hessian-vectors at every iteration. Another variant implemented in all the three algorithms in KNITRO is to approximate  $\nabla_{xx}^2 L$  by quasi-Newton updates: BFGS, memory-less BFGS, or SR1. For example, BFGS is implemented as

$$B_{k+1} = B_k + \frac{q_k q_k^T}{s_k^T q_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}, \quad (17.103)$$

where  $q_k = \nabla_x L(x^+, s^+, y^+, z^+) - \nabla_x L(x, s^+, y^+, z^+)$  and  $s_k = x_{k+1} - x$ .

*Feasible iterations.* In some applications, it is desirable for all the iterates generated by the optimization algorithm to be feasible with respect to some or to all of the inequality constraints. KNITRO uses the following procedure to generate feasible iterates. If the current iteration  $x$  satisfies the constraints  $c_I(x) > 0$ , then, after computing step  $d$ , set  $x^+ = x + d_x$ . The slack variables are redefined as  $s^+ = c_I(x^+)$  and tested whether the point  $(x^+, s^+)$  is acceptable for the merit function  $\Phi_\sigma$ . If the case, this point is defined as the new iterate, otherwise this point is rejected and a new shorter one is computed (in a line-search method by backtracking and in a trust-region method by reducing the trust-region radius). The justification of this strategy is as follows. If at a trial point, we have  $c_I(x^+) \leq 0$ , for some inequality constraint, then the value of the merit function is  $+\infty$ , and this trial point is rejected. This strategy also rejects the steps  $x + d_x$  that are too close to the boundary of the feasible region because such steps increase the barrier term  $-\mu \sum_{i=1}^m \log(s_i)$  in the merit function.

*Crossover technique.* The situation is that the interior-point methods provide only an approximate estimate of the solution and of the optimal active-set. However, in many practical applications, it is useful to know accurately which constraints are active. Moreover, it is often important to know the highly accurate estimates of the Lagrange multipliers. They are important in the sensitivity analysis of the solution to some variations of certain parameters in the problem. Increasing the accuracy of the

solution or of the active-set can be done by switching from the interior-point method to an active-set iteration, a process which is called *crossover*. This technique was introduced for the first time in linear programming by Megiddo (1989). In linear programming, the crossover technique involves two stages: identifying the active constraints and moving from a nonbasic optimal solution to a nearby basic one. However, in nonlinear programming, this form of crossover cannot be used because we cannot expect the set of active constraints to correspond to a basic solution. In nonlinear optimization, the crossover was introduced for the first time by Byrd, Nocedal, and Waltz (2006). It seeks to identify a set of active constraints (with linearly independent constraint gradients) and computes a solution at which these constraints are approximately satisfied, solution which determines a stationary point of the Lagrangian by using only these constraints.

The crossover procedure internally commutes to the KNITRO/ACTIVE algorithm *after* the KNITRO/INTERIOR-DIRECT or the KNITRO/INTERIOR-CG algorithms have solved the problem to the required tolerance. Firstly, solve the equality quadratic programming (15.110) to generate a new solution estimate. If this step does not solve the problem immediately, then the full KNITRO/ACTIVE algorithm is started with an initial LP trust-region radius  $\Delta_k^{LP}$  computed on the basis of the active set estimate. The idea is to choose  $\Delta_k^{LP}$  small enough to exclude all the inactive constraints, but large enough to include the active ones. The crossover algorithm as described in (Byrd, Nocedal, and Waltz, 2006) is as follows.

**Algorithm 17.9 KNITRO crossover algorithm**

1. The interior-point algorithms DIRECT or CG terminate with a solution  $(x_k, s_k, y_k, z_k)$  which satisfies the stopping tolerance  $\varepsilon_{TOL}$
2. Estimate the set of the active constraints  $A$
3. Using this active-set estimate, generate a step  $d^Q$  by solving the equality quadratic programming problem (15.110). Perform a line-search to determine the stepsize  $\alpha^Q$ . If  $x_k + \alpha^Q d^Q$  satisfies the stopping tolerances, then the algorithm stops with this point and with the corresponding multipliers; otherwise, go to step 4
4. Determine the initial LP trust-region radius  $\Delta_0^{LP}$  and the penalty parameter  $\sigma_0$  for the KNITRO/ACTIVE (Algorithm 15.9) as
$$\Delta_0^{LP} = \min \left\{ \frac{c_i(x_k, s_k)}{\|\nabla c_i(x_k, s_k)\|} : i \notin A \right\},$$

$$\sigma_0 = 10 \| (y_k, z_k) \|_\infty$$
5. Start the KNITRO/ACTIVE algorithm using the initial point  $(x_k, s_k, y_k, z_k)$ ,  $\Delta_0^{LP}$  and  $\sigma_0$  ◆

In step 3 of Algorithm 17.9, the active set is estimated by using a tolerance test rather than by solving the linear programming problem (15.105). This is because, in some difficult problems, the cost of solving the linear programming subproblem can be nontrivial and, as motivated by Byrd, Nocedal and Waltz (2006), the cost of the crossover procedure shall be a small part of the overall solution time. Therefore, it is not necessary to solve (15.105) to identify the optimal active set. If strict complementarity holds at the solution, the initial estimate of the active-set based on the simple tolerance test will be correct and the crossover will succeed in one iteration without solving (15.105).

The formula for computing the initial trust-region radius  $\Delta_0^{LP}$  from step 4 of Algorithm 17.9, guarantees that if the active set estimate is correct, then this initial trust-region radius will be small enough to exclude all the inactive constraints. As motivated by the theory of  $l_1$  exact penalty functions, the penalty parameter is initialized to be a little larger than the Lagrange multiplier of the largest magnitude at the interior-point solution.

*Practical hints.* KNITRO implements four algorithms. An algorithm is selected according to the value of the parameter *option*. For *option 0*, KNITRO will automatically choose the best algorithm based on the problem characteristics. For *option 1*, KNITRO will use the INTERIOR-DIRECT algorithm. For *option 2*, KNITRO will use the INTERIOR-CG algorithm. For *option 3*, KNITRO will use the ACTIVE algorithm (see Chap. 15).

### Numerical Study—KNITRO: Solving Applications from the LACOP Collection

In the following, Tables 17.6, 17.7, and 17.8 present the performances of KNITRO with option 0 (KNITRO will automatically try to choose the best algorithm based on the problem characteristics), option 1 (KNITRO will use the INTERIOR-DIRECT algorithm), and KNITRO with option 2 (KNITRO will use the INTERIOR-CG algorithm) for solving 12 nonlinear optimization applications from the LACOP collection described in Appendix C.

**Table 17.6** Performances of KNITRO for solving 12 applications from the LACOP collection. Option 0. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	#nh	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	6	0	7	7	6	0.143	-47.761090
ALKI	10	3	8	10	0	13	11	10	0.011	-1768.8069
PREC	8	0	6	10	9	11	11	10	0.081	3.95116321
PPSE	9	6	0	8	0	9	9	8	0.009	5055.01179
MSP3	13	0	15	26	7	36	27	26	0.044	97.5875095
MSP5	16	0	21	23	10	31	24	23	0.045	174.786942
POOL	34	20	0	9	0	10	10	9	0.026	2569.800
TRAFO	6	0	2	16	18	19	17	16	0.009	135.07592
LATHE	10	1	14	23	5	50	24	23	0.020	-4430.0879
DES	150	50	0	39	350	99	40	39	0.479	1055.1823
CSTC	303	200	0	4	0	5	5	4	0.035	3.4800745
DIFF	396	324	0	2	0	3	1	0	0.016	0

**Table 17.7** Performances of KNITRO/INTERIOR-DIRECT for solving 12 applications from the LACOP collection. Option 1. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	#nh	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	6	0	7	7	0	0.130	-47.761090
ALKI	10	3	8	10	0	13	11	10	0.009	-1768.8069
PREC	8	0	6	10	9	11	11	10	0.012	3.95116321
PPSE	9	6	0	8	0	9	9	8	0.008	5055.01179
MSP3	13	0	15	26	7	36	27	26	0.012	97.5875095
MSP5	16	0	21	23	10	31	24	23	0.014	174.786942
POOL	34	20	0	9	0	10	10	9	0.010	2569.800
TRAFO	6	0	2	16	18	19	17	16	0.009	135.07592
LATHE	10	1	14	23	5	50	24	23	0.017	-4430.0879
DES	150	50	0	39	350	99	40	39	0.470	1055.1823
CSTC	303	200	0	4	0	5	5	4	0.011	3.4800745
DIFF	396	324	0	2	0	3	1	0	0.020	0

**Table 17.8** Performances of KNITRO/INTERIOR-CG for solving 12 applications from the LACOP collection. Option 2. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	#nh	<i>cpu</i>	<i>vfo</i>
ELCH	10	3	0	6	27	7	7	6	0.008	-47.761090
ALKI	10	3	8	20	79	21	21	20	0.011	-1768.8065
PREC	8	0	6	22	124	24	23	22	0.057	4.2867607
PPSE	9	6	0	5	13	6	6	5	0.008	5055.01300
MSP3	13	0	15	36	237	37	37	36	0.019	97.5875197
MSP5	16	0	21	84	325	146	85	84	0.035	174.795082
POOL	34	20	0	17	59	18	18	17	0.011	2569.800
TRAFO	6	0	2	37	103	52	38	37	0.017	135.076250
LATHE	10	1	14	27	118	33	28	27	0.016	-4430.0806
DES	150	50	0	28	262	79	29	28	0.237	1055.1823
CSTC	303	200	0	4	10	5	5	4	0.019	3.4800745
DIFF	396	324	0	1	0	2	1	0	0.024	0

In the tables above, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #it = the number of iterations to obtain a solution, #itcg = the number of conjugate gradient iterations, #nf = the number of evaluations of the functions defining the problem, #ng = the number of evaluations of the gradients of the functions defining the problem, #nh = the number of evaluations of the Hessian,  $cpu$  = the CPU computing time for obtaining a solution of the problem (seconds), and  $vfo$  = the value of the objective function at the optimal point

**Table 17.9** Performances of the KNITRO algorithms. Small-scale nonlinear optimization applications

Option	#itt	#nft	#ngt	#nht	<i>cput</i>
0	176	293	186	174	0.918
1 (DIRECT)	176	293	186	168	0.722
2 (CG)	287	430	298	286	0.462
3 (ACTIVE)	278	731	289	418	0.782

The performances of KNITRO/ACTIVE (option 3) for solving small-scale nonlinear optimization applications are presented in Table 15.6. Table 17.9 presents the total number of iterations (#itt), the total number of evaluations of the functions defining the problems (#nft), the total number of evaluations of the gradients of the functions (#ngt), the total number of evaluations of the Hessian (#nht), and the total CPU time (*cput*) in seconds for solving 12 small-scale nonlinear optimization applications considered in this numerical study with option 0 (Table 17.6), option 1 (Table 17.7), option 2 (Table 17.8), and option 3 (Table 15.6) respectively.

Tables 17.10, 17.11, and 17.12 show the numerical performances of KNITRO with option 0, option 1, and with option 2, respectively, for solving 6 large-scale nonlinear optimization applications of different dimensions from the LACOP collection.

Performances of KNITRO/ACTIVE (option 3) for solving 15 large-scale nonlinear optimization applications are presented in Table 15.7. Table 17.13 contains a comparison of these four variants of KNITRO subject to the total number of iterations (#itt), the total number of conjugate gradient iterations (#itcg), the total number of evaluations of the function defining the problem (#nft), the total number of evaluations of the gradients of the functions defining the problem (#ngt), the total CPU computing time to obtain a solution (*cput*) for solving 15 large-scale nonlinear optimization applications considered in this numerical study, with option 0 (Table 17.10), option 1 (Table 17.11), option 2 (Table 17.12), and option 3 (Table 15.7), respectively.

**Table 17.10** Performances of KNITRO for solving 6 applications from the LACOP collection. Option 0. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	<i>cpu</i>	<i>vfo</i>
HANG	2002	1001	0	21	73	44	22	0.349	5.0685101
	4002	2001	0	81	328	291	82	3.039	5.0684889
FLOW	1182	754	0	5	0	6	6	0.160	0.311e-11
FLOWO	1556	1005	0	10	19	11	11	1.031	0.107e-6
POL	4004	3000	0	27	35	29	28	1.203	14.009791
	6004	4500	0	26	35	27	27	1.836	13.990035
	8004	6000	0	28	36	29	29	2.568	14.009035
	10004	7500	0	30	39	31	31	3.454	14.007359
CAT	3003	2000	0	8	0	9	9	0.301	-0.048052
	6003	4000	0	8	0	9	9	0.674	-0.048048
	9003	6000	0	7	0	8	8	0.993	-0.048045
CONT	2505	2000	0	9	0	10	10	0.109	1.0132439
	5005	4000	0	10	0	11	11	0.403	1.0059324
	7505	6000	0	11	0	12	12	0.743	1.0045764
	10005	8000	0	11	0	12	12	0.947	1.0040918

**Table 17.11** Performances of KNITRO/INTERIOR-DIRECT for solving 6 applications from the LACOP collection. Option 1. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	<i>cpu</i>	<i>vfo</i>
HANG	2002	1001	0	21	73	44	22	0.425	5.068510
	4002	2001	0	81	328	291	82	3.068	5.0684889
FLOW	1182	754	0	5	0	6	6	0.160	3.85e-11
FLOWO	1556	1005	0	10	19	11	11	1.088	1.071e-5
POL	4004	3000	0	27	35	29	28	1.291	14.009791
	6004	4500	0	26	35	27	27	1.839	13.990035
	8004	6000	0	28	36	29	29	2.586	14.009035
	10004	7500	0	30	39	31	31	3.481	14.007359
CAT	3003	2000	0	8	0	9	9	0.353	-0.048052
	6003	4000	0	8	0	9	9	0.650	-0.048048
	9003	6000	0	7	0	8	8	1.055	-0.048045
CONT	2505	2000	0	9	0	10	10	0.190	1.0132439
	5005	4000	0	10	0	11	11	0.408	1.0059324
	7505	6000	0	11	0	12	12	0.734	1.0045764
	10005	8000	0	11	0	12	12	0.968	1.0040918

### Application Penici

Application L16 is described in the LACOP collection. Table 17.14 presents the performances of the algorithms implemented in KNITRO for solving this application.

There are a number of packages implementing the interior-point methods for nonlinear optimization, both in the line-search and in the trust-region framework. The line-search implementation includes LOQO (Vanderbei, & Shanno, 1999), KNITRO/INTERIOR-DIRECT (Waltz, Morales, Nocedal, & Orban, 2003), IPOPT (Wächter, & Biegler, 2006), BARNLP (Betts, Eldersveld, Frank, & Lewis, 2000), and MOSEK (Andresen, & Andresen, 2000). The trust-region algorithm was implemented in KNITRO/INTERIOR-CG (Byrd, Hribar, & Nocedal, 1999). All these interior-point packages are strong competitors of the leading active-set and of the augmented Lagrangian

**Table 17.12** Performances of KNITRO/INTERIOR-CG for solving 6 applications from the LACOP collection. Option 2. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#itcg	#nf	#ng	cpu	vfo
HANG	2002	1001	0	6	141	7	7	0.211	5.0685100
	4002	2001	0	6	142	7	7	0.538	5.0684889
FLOW	1182	754	0	29	174	30	30	1.080	1.831e-11
FLOWO	1556	1005	0	48	731	49	49	4.104	1.8483e-5
POL	4004	3000	0	44	179	46	45	1.546	14.202608
	6004	4500	0	43	353	45	44	2.866	14.243107
	8004	6000	0	19	197	23	20	1.938	14.265885
	10004	7500	0	19	153	23	20	2.208	14.260745
CAT	3003	2000	0	10	46	11	11	0.379	-0.048047
	6003	4000	0	6	18	7	7	0.571	-0.047980
	9003	6000	0	7	25	8	8	1.069	-0.047954
CONT	2505	2000	0	13	92	14	14	0.357	1.0134407
	5005	4000	0	13	98	14	14	0.758	1.006334
	7505	6000	0	14	95	15	15	1.211	1.005174
	10005	8000	0	13	98	14	14	1.532	1.004447

**Table 17.13** Performances of the KNITRO algorithms. Large-scale nonlinear optimization applications

Option	#itt	#itcgt	#nft	#ngt	cpu
0	292	565	539	307	17.81
1 (DIRECT)	292	565	539	307	18.296
2 (CG)	290	2542	313	305	20.367
3 (ACTIVE)	1123	2197	2711	1138	466.084

**Table 17.14** Performances of KNITRO. Application PENICI ( $n = 707$ ,  $l = 602$ ,  $m = 0$ )

Option	#it	#itcg	#nf	#ng	cpu
0	397	2922	433	398	7.374
1 (DIRECT)	397	2922	433	398	7.357
2 (CG)	307	3994	559	308	4.786
3 (ACTIVE)	Too many iterations				

packages, like MINOS (Murtagh, & Saunders, 1987), SNOPT (Gill, Murray, & Saunders, 2002), LANCELOT (Conn, Gould, & Toint, 1992b), and KNITRO/ACTIVE (Byrd, Gould, Nocedal, & Waltz, 2004). For solving nonlinear optimization problems and applications, the interior-point and the active-set methods are the most suitable, while the augmented Lagrangian methods seem to be less efficient and robust. KNITRO implements a crossover technique from the interior-point to the active-set modes (Byrd, Nocedal, & Waltz, 2006). KNITRO/INTERIOR-CG implements a trust-region algorithm using the projected conjugate gradient iteration for computing the step, which allows the method to work when only the Hessian-vector products are available, not the Hessian itself.

In the interior-point methods, the primal-dual linear system to be solved at every iteration has the same block structure which can be speculated to get efficient solutions. Both the direct solutions using the sparse matrix technology and the conjugate gradient methods are available for solving the primal-

**Table 17.15** Comparisons between KNITRO and CONOPT. Small-scale nonlinear optimization applications

	CONOPT	KNITRO 0	KNITRO 1	KNITRO 2	KNITRO 3
#itt	272	176	176	287	278
cput	0.80	0.918	0.722	0.462	0.782

**Table 17.16** Comparisons between KNITRO and CONOPT. Large-scale nonlinear optimization applications

	CONOPT	KNITRO 0	KNITRO 1	KNITRO 2	KNITRO 3
#itt	520	292	292	290	1123
cput	9.124	17.81	18.296	20.368	466.084

dual system. However, the interior-point methods, unlike the active-set methods, consider all the constraints at each iteration, even if they are irrelevant to the solution, thus enlarging the cost of the primal-dual iteration. The main drawbacks of the interior-point methods is their sensitivity to the choice of the initial point, the scaling of the problem and the update strategy for the barrier parameter  $\mu$ . The interior-point methods may have difficulties and their convergence can be slow if the iterates approach the boundary of the feasible region prematurely. To get a robust code, some algebraic techniques must be implemented, such as second-order corrections, iterative refinement, resetting the parameters, procedure for computing the approximate solutions of the normal and tangential subproblems, etc.

A comparison of codes KNITRO and CONOPT for solving 12 small-scale applications from the LACOP collection is shown in Table 17.15. Table 17.16 presents a comparison of KNITRO versus CONOPT for solving 15 large-scale applications from the LACOP collection.

### Notes and References

Interior-point methods generate iterations that avoid the boundary of the feasibility region defined by the inequality constraints. Along the optimization process, the iterates are allowed to get closer and closer to the boundary of the feasibility region and converge to the optimal solution which might lie on the boundary. From the practical point of view, interior-point methods typically require only tens of iterations. Each iteration consists in solving a large linear system of equations which takes into consideration all the variables and constraints so that each iteration is quite expensive. In other words, interior-point methods approximately solve a sequence of perturbed KKT systems by driving a barrier parameter to zero. These methods can be regarded as perturbed Newton methods applied to the KKT system in which the primal/dual variables are kept positive. All the characteristics of the optimization process given by optimality, feasibility, and complementarity are simultaneously reduced. The main advantages of the interior-point methods are as follows: they can better exploit the second derivatives and their structure; they are efficient for loosely constrained optimization problems and are easily implemented in computing programs.

This chapter is based on the papers (Nocedal, & Wright, 2006) and (Andrei, 1998c). The interior-point methods were first developed for linear programming by Ilya Dikin (1936–2008) (1967, 1974) and then by Karmarkar (1984). The extension to the quadratic and nonlinear programming was quite natural. There is a vast literature on nonlinear interior-point methods. For a comprehensive list of references, we recommend the papers by Forsgren, Gill, and Wright (2002) and Gould, Orban, and Toint (2005a). The book by Conn, Gould, and Toint (2000) also gives a thorough presentation of several interior-point algorithms. The papers by Ulbrich, Ulbrich and Vicente (2004) and Wächter

and Biegler (2005a, 2005b) present an analysis of interior-point algorithms that use filter globalization. The primal barrier methods for nonlinear optimization were originally proposed by Frisch (1955) and were further analyzed by Fiacco and McCormick (1968). In this chapter, the prototype of interior-point algorithms was presented first. After that, a variant of the line-search interior-point algorithm was deeply discussed, where a methodology of the interior-point algorithms development and analysis was given. The conclusion is that for having an efficient and robust interior-point algorithm, some methods based on sequential linearization combined with line-search or trust-region globalization strategies have to be introduced.

A history of the barrier function methods was given by Nash (1998). Shanno (2012) presented interesting comments on “who invented the interior-point algorithm.” His conclusion is that the history of the interior-point method goes back to Fiacco and McCormick. It seems that the terms *interior-point* and *primal central path* appear in the book by Fiacco and McCormick. The most important packages implementing the interior-point methods are as follows.

KNITRO/INTERIOR implements two methods: trust-region and line-search interior-point/barrier methods. The idea of the algorithm is to solve a sequence of barrier subproblems for a decreasing sequence of barrier parameters. For accepting the step and for ensuring the global convergence it uses a merit or a penalty function. The barrier subproblems are solved by a sequence of linearized primal-dual equations. For solving the primal-dual system, KNITRO implements two options: the direct factorization of the system and the preconditioned conjugate gradient method. The preconditioned conjugate gradient method solves the indefinite primal-dual system by projecting constraints onto the null space of the equality constraints. KNITRO uses the crossover technique to obtain an active set from the solution of the interior-point method.

LOQO (Vanderbei, & Shanno, 1999) uses an infeasible primal-dual interior-point method for solving general nonlinear optimization problems and applications. The inequality constraints are added to the minimizing function by using a log-barrier function. To get a solution to the system of nonlinear equations that is obtained by applying the first-order necessary optimality conditions to the barrier function, Newton’s method is used. The solution of this system gives a search direction. A filter or a merit function is used to accept the next iterate obtained by performing a line-search along the search direction. The filter allows a point that improves either the feasibility or the barrier objective as compared to the current iterate. The merit function is the barrier function and an additional  $l_2$  norm of the violation of constraints. The exact Hessian of the Lagrangian is used in Newton’s method. When the problem is nonconvex, then the Hessian is perturbed by adding the matrix  $\delta I$  to it, where  $I$  is the identity matrix and  $\delta > 0$  is a scalar chosen in such a way that the perturbed Hessian is positive definite.

Ipfilter (Ulbrich, Ulbrich, & Vicente, 2004) is an interior-point filter method. Firstly, a step is generated by solving the KKT system. Then, a specially designed filter is used for selecting the stepsize in the direction of the calculated step. The first component of the filter is the sum of the constraint violation and centrality, while the second component is the norm of the gradient of the Lagrangian.

IPOPT (Wächter, & Biegler, 2005a, 2005b) is a line-search filter interior-point method. The outer loop approximately minimizes a sequence of nonlinear equality constrained barrier subproblems for a decreasing sequence of barrier parameters. The inner loop uses a line-search filter sequential quadratic programming method to approximately solve each barrier subproblem. IPOPT is described in Chap. 19 of this monograph.



## Filter Methods

# 18

This chapter focuses on the filter methods developed by Fletcher and Leyffer (2002) as a new technique for the globalization of nonlinear optimization algorithms. These methods are motivated by the aim of avoiding the need to choose penalty parameters in penalty functions or in augmented Lagrangian functions and their variants. Let us consider the nonlinear optimization problems with inequality constraints

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_i(x) \leq 0, i = 1, \dots, m, \end{aligned} \tag{18.1}$$

where the objective function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  and the functions  $c_i: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , defining the constraints are supposed to be twice continuously differentiable. The methods for solving this problem are based on the Newton method. Given an estimate  $x_k$  of the solution  $x^*$  of (18.1), a linear or a quadratic approximation of (18.1) is solved, thus obtaining a new estimate  $x_{k+1}$  which is expected to be better than the previous one. Near a solution, this approach is guaranteed to be convergent. However, far away from the solution the sequence  $\{x_k\}$  generated by the above procedure may not converge. In this situation, away from the solution, the idea is to use the Newton method again, but considering the penalty or the merit functions. The penalty functions or the merit functions are a combination of the objective function and a measure of the constraints violation, such as  $h(x) = \|c(x)^+\|_\infty$ , where  $c(x) = [c_1(x), \dots, c_m(x)]^\top$  and  $c_i^+ = \max\{0, c_i\}$ . A well-known example is the  $l_1$  exact penalty function  $p(x, \sigma) = f(x) + \sigma h(x)$ , where  $\sigma > 0$  is the penalty parameter. If  $\sigma$  is sufficiently large, then this penalty function can be minimized in an iterative procedure to ensure progress to the solution.

This approach based on the penalty or the merit functions is interesting, but not without difficulties. Unfortunately, a suitable value of the penalty parameter depends on the solution of the problem (18.1), that is  $\sigma > \|\lambda^*\|_D$ , where  $\lambda^*$  is the vector of the Lagrange multipliers and  $\|\cdot\|_D$  is the dual norm. Hence, the determination of a suitable penalty parameter is difficult. Moreover, if the penalty parameter is too large, then any monotonic method would be forced to follow the nonlinear constraint manifold very closely, resulting in much shortened Newton steps and therefore in slow convergence. All these deficiencies are very well known and for their avoidance; some computational schemes have been imagined like methods based on the augmented Lagrange function (modified or not) or interior point methods. In this section, we present the methods for solving nonlinear

optimization problems known as filter methods (Fletcher, & Leyffer, 1998, 1999, 2002), (Fletcher, Leyffer, & Toint, 2002b, 2006).

The idea of a filter method is not of combining the objective and the constraints violation into a single function. Instead, (18.1) is viewed as a *biobjective optimization problem* that minimizes  $f(x)$  and  $h(x)$ . However, as we can immediately see, the second objective is more important because we must ensure that  $h(x^*) = 0$ . Fletcher and Leyffer (2002) introduced the concept of *domination* taken from the multi-objective optimization. A point  $x_k$  dominates a point  $x_l$  if and only if  $f(x_k) \leq f(x_l)$  and  $h(x_k) \leq h(x_l)$ . Therefore, a filter is defined as a list of pairs  $(h(x_l), f(x_l))$  such that no pair dominates another pair.

This concept can be implemented in different ways. The filter methods use the *sequential linear programming* or the *sequential quadratic programming* in the context of the trust-region methods. At iteration  $k = 0$ , the filter is initialized as  $F_k = \{(u, -\infty)\}$ , where  $u$  is an upper bound on the acceptable constraint violation. Solving the linear programming problem or the quadratic programming problem, a solution  $s$  is obtained. If this point is acceptable by the filter, then we set  $x_{k+1} = x_k + s$  and possibly increase the trust-region radius and update the filter (adding the previous point and removing any dominated entries). On the other hand, if the step is dominated by the current filter, then we reject it, set  $x_{k+1} = x_k$ , reduce the trust-region radius, and solve the linear or the quadratic programming problem in a new context. For the process to be convergent, this general description of the filter method needs some refinements:

1. *Filter envelope.* To avoid the convergence to an infeasible limit point where  $h(x^*) > 0$ , an envelope around the current filter is added (Chin, 2002), (Chin, & Fletcher, 2003). A new iterate is acceptable if for all  $l \in F_k$ ,  $h_{k+1} \leq \beta h_l$  or  $f_{k+1} \leq f_l - \gamma h_{k+1}$ , where  $f_k = f(x_k)$ ,  $h_k = h(x_k)$  and  $0 < \beta, \gamma < 1$  are constants.
2. *Sufficient reduction.* The filter alone cannot ensure convergence to the stationary points. For example, if the sequence  $\{x_k\}$  satisfies the condition  $h_{k+1} \leq \beta h_k$ , then the iterates could converge to an arbitrary feasible point. Therefore, if the constraint violation becomes small, a sufficient reduction condition can be imposed as follows: denote the predicted reduction by  $\Delta q_k$  as the value of the objective function of the linear or of the quadratic problem and introduce the following *switching condition*: if  $\Delta q_k > 0$ , then check  $f_k - f_{k+1} \geq \sigma \Delta q_k$ , where  $\sigma \in (0, 1)$  is a constant.
3. *Feasibility restoration.* By reducing the trust-region radius of the linear or the quadratic problems, these may become inconsistent. That is, the current point is too far from the feasibility region to get significant progress toward the solution. In this case, a sequential linear or quadratic programming problem is invoked, that minimizes the constraint violation  $h(x)$ . This restoration phase is left once an acceptable point has been obtained.

The filter algorithm contains an *inner* and an *outer* iteration. During the inner iteration, the trust-region radius is reduced until either an acceptable point is found or the restoration phase is initiated. The aim of the restoration phase is to find an acceptable iterate  $x_{k+1}$  such that the linear (or quadratic) programming problem is compatible for some smaller values of the trust-region radius. The iterate and the filter are updated in the outer iteration, which also initializes the trust-region radius to a given lower bound. The filter is updated by adding  $(h_k, f_k)$  to  $F_k$ , which corresponds to an *h-type iteration* after we move to  $x_{k+1}$ . Fletcher and Leyffer (2002) motivate the switching condition as follows. When close to a feasible point, it is expected that the linear or the quadratic model of the problem predicts a decrease in the objective function, that is  $\Delta q_k > 0$ . However, far away from a feasible point the predicted reduction is usually negative, that is  $\Delta q_k < 0$ . The iterations that satisfy the switching condition are called *f-type iterations*. All the other steps are called *h-type iterations*. Observe that if  $h_k = 0$  at a non-stationary point, then  $\Delta q_k > 0$ , therefore implying that only f-typed steps can be

accepted. In other words, no points for which  $h_k = 0$  are added to the filter. This ensures that the restoration phase generates only filter-acceptable points.

## 18.1 Sequential Linear Programming Filter Algorithm

Suppose that the constraints from (18.1) include some linear constraints, for example, simple bounds on variables that define a non-empty bounded region  $X$ . In the current point  $x$ , for the problem (18.1) we can associate the following linear programming sub-problem modified with the trust-region radius  $\rho > 0$ , as

$$\begin{aligned} & \min_{d \in \mathbb{R}^n} g^T d \\ & \text{subject to:} \\ & c_i + a_i^T d \leq 0, i = 1, \dots, m, \\ & \|d\|_\infty \leq \rho, \end{aligned} \tag{18.2}$$

where  $g = \nabla f(x)$ ,  $c_i = c_i(x)$  and  $a_i = \nabla c_i(x)$ . Observe that the  $l_\infty$  norm is used to define the trust-region because it is very easy to be implemented by adding simple bounds to the linear programming subproblem. If it exists, let  $d$  be the solution of (18.2). Denote

$$\Delta l = -g^T d \tag{18.3}$$

as the *predicted reduction* in  $f(x)$  and with

$$\Delta f = f(x) - f(x + d) \tag{18.4}$$

as the *actual reduction* in  $f(x)$ . The measure of the constraint infeasibility which is used in this text is given by

$$h(c) = \|c^+\|_\infty, \tag{18.5}$$

where  $c_i^+ = \max \{0, c_i\}$ ,  $i = 1, \dots, m$ .

A filter is defined by pairs of values  $(h, f)$  obtained by evaluating  $h(c(x))$  and  $f(x)$  for different values of  $x$ . A pair  $(h_i, f_i)$  dominates another pair  $(h_j, f_j)$  if and only if both  $h_i \leq h_j$  and  $f_i \leq f_j$  are satisfied, thus indicating that the former point is at least as good as the latter in respect to both measures. The filter is defined to be a list of pairs  $(h_i, f_i)$  such that no pair dominates any other. A point  $x$  is acceptable for inclusion in the filter if its  $(h, f)$  pair is not dominated by any entry in the filter. This is the condition that

$$\text{either } h < h_i \text{ or } f < f_i \tag{18.6}$$

for all  $i \in F$ , where  $F$  denotes the current set of filter entries. We may include a point in the filter by which we mean that its pair  $(h, f)$  is added to the list of pairs in the filter and any pairs in the filter that are dominated by the new pair are deleted from the filter.

Observe that this definition of the filter is not suitable for proving the convergence because it allows points to accumulate in the neighborhood of a filter entry that has  $h_i > 0$ . However, this may be corrected by defining a small envelope around the current filter in which the points are accepted. Therefore, the condition for a point to be *acceptable* to the filter is that its pair  $(h, f)$  satisfies

$$\text{either } h \leq \beta h_i \text{ or } f \leq f_i - \gamma h_i \quad (18.7)$$

for all  $i \in F$ , where  $\beta$  and  $\gamma$  are parameters such that  $1 > \beta > \gamma > 0$ , with  $\beta$  close to one and  $\gamma$  close to zero. Observe that the first inequality from (18.7) is exactly the reduction in  $h$ . The dependence on  $h_i$  of the second inequality is a small artifact to enable the convergence of the algorithm. Since  $\gamma$  is small, the extra term has little practical impact.

It is also convenient to introduce an upper bound

$$h(c(x)) \leq \beta u \quad (18.8)$$

( $u > 0$ ) on the constraint infeasibility, which is readily implemented by initializing the filter with the entry  $(u, -\infty)$ .

As we know, a common feature in a trust-region algorithm for the unconstrained minimization is the use of a sufficient reduction criterion

$$\Delta f \geq \sigma \Delta l, \quad (18.9)$$

where  $\Delta l$  is positive and  $\sigma \in [0, 1]$  is a given parameter. However, in the trust-region algorithm,  $\Delta l$  may be negative or even zero, case in which this test is no longer appropriate. Fletcher and Leyffer (2002) suggest using (18.9) with  $\sigma \geq \gamma$  only when  $\Delta l$  is sufficiently positive, which can be achieved by testing the inequality

$$\Delta l \geq \delta h^2, \quad (18.10)$$

where  $h$  refers to  $h(c(x))$  evaluated at the current point and  $\delta > 0$  is a given parameter close to zero. Again, the dependence of the right-hand side of (18.10) on  $h$  is an artifact to enable the convergence to be proved.

Let us denote the linear programming subproblem (18.2) by  $LP(x, \rho)$ , then the sequential linear programming filter algorithm is as follows.

### Algorithm 18.1 Sequential linear programming filter—filterSD

1.	Choose an initial point $x \in X$ and initialize the filter with $(u, -\infty)$ . Set $k = 1$
2.	Consider the restoration phase to find a point $x_k \in X$ acceptable for inclusion in the filter such that $LP(x_k, \tilde{\rho})$ is compatible for a $\tilde{\rho} \geq \rho^0$ , and initialize $\rho = \tilde{\rho}$
3.	Solve the subproblem $LP(x_k, \rho)$
4.	If $LP(x_k, \rho)$ is infeasible, then place $(h_k, f_k)$ in the filter ( <i>h-type iteration</i> ). Set $k = k + 1$ and go to step 2
5.	If $LP(x_k, \rho)$ has the solution $d$ , then if $d = 0$ , stop; the current point is a KKT point for the problem
6.	If $d \neq 0$ , then evaluate $f(x_k + d)$ and $c(x_k + d)$
7.	If $x_k + d$ is not acceptable for inclusion in the filter, then set $\rho = \rho/2$ and go to step 3
8.	If $x_k + d$ is acceptable for inclusion in the filter and $\Delta f < \sigma \Delta l$ and $\Delta l \geq \delta(h_k)^2$ , then set $\rho = \rho/2$ and go to step 3
9.	If $x_k + d$ is acceptable for inclusion in the filter and $\Delta f < \sigma \Delta l$ and $\Delta l \geq \delta(h_k)^2$ are not satisfied, then set $\rho_k = \rho$ , $d_k = d$ , $\Delta l_k = \Delta l$ and $\Delta f_k = \Delta f$
10.	If $\Delta l_k < \delta(h_k)^2$ , then the pair $(h_k, f_k)$ is included in the filter ( <i>h-type iteration</i> )
11.	Set $x_{k+1} = x_k + d_k$ , $k = k + 1$ , initialize $\rho \geq \rho^0$ and go to step 3

◆

Some important details as well as the convergence of the filter algorithm with sequential linear programming are given in (Fletcher, Leyffer, & Toint, 1999). Mainly, the convergence theorem is based on the necessary optimality condition by Fritz John and on the following standard hypothesis:

the set  $X$  defined by the linear constraints of the problem (18.1) is non-empty and bounded and the functions  $f(x)$  and  $c(x)$  of the problem are twice continuously differentiable on  $X$ . When the algorithm is applied, one of four different possible outcomes can occur:

- (i) The restoration phase iterates infinitely and fails to find a point  $x$  which is acceptable to the filter and for which  $LP(x, \rho)$  is compatible for some  $\rho \geq \rho^0$ .
- (ii) A KKT point is found ( $d = 0$  solves  $LP(x_k, \rho)$  for some  $k$ ).
- (iii) All the iterations are  $f$ -type iterations for  $k$  sufficiently large.
- (iv) There exists an infinite subsequence of  $h$ -type iterations.

The following theorem proves the convergence of the sequential linear programming filter algorithm, as specified in (Fletcher, Leyffer, & Toint, 1999).

**Theorem 18.1** *If the standard hypothesis holds, then for the sequential linear programming filter algorithm either (i) or (ii) occurs or the sequence of the iterates generated by the algorithm has an accumulation point that satisfies the Fritz John necessary optimality conditions.* ◆

More discussions about the sequential linear programming filter algorithm (filterSD) can be found in (Fletcher, & Leyffer, 1998, 2002). In the following, let us present some numerical results with this algorithm.

**Example 18.1** To illustrate the running of the filterSD algorithm, let us consider the nonlinear optimization problem (Duran, & Grossmann, 1986):

$$\begin{aligned} \min & (10x_1 - 18 \ln(x_2 + 1) - 7x_3 + 5x_4 + 6x_5 + 8x_6 - 19.2 \ln(x_1 - x_2 + 1) + 10) \\ \text{subject to} & \\ & 0.8 \ln(x_2 + 1) + 0.96 \ln(x_1 - x_2 + 1) - 0.8x_3 \geq 0, \\ & \ln(x_2 + 1) + 1.2 \ln(x_1 - x_2 + 1) - x_3 - 2x_6 \geq -2, \\ & x_2 - x_1 \leq 0, \\ & x_2 - 2x_4 \leq 0, \\ & x_1 - x_2 - 2x_5 \leq 0, \\ & x_4 + x_5 \leq 1, \end{aligned}$$

where

$$0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2, 0 \leq x_3 \leq 1, 0 \leq x_4 \leq 1, 0 \leq x_5 \leq 1, 0 \leq x_6 \leq 1.$$

Let  $x_0 = [0, 0, 0, 0, 0, 0]^T$  be the initial point. The filterSD algorithm gives the following results. The objective function value in the initial point is  $f(x_0) = 10$ . The value of the constraints at the initial point are  $c_i(x_0) = 0$ ,  $i = 1, \dots, 6$ . The optimal solution is  $x^*$  and the value of the constraints in the optimal point  $c(x^*)$  are as follows:

$$x^* = \begin{bmatrix} 1.146514 \\ 0.546592 \\ 1 \\ 0.273296 \\ 0.299960 \\ 0 \end{bmatrix} \quad c(x^*) = \begin{bmatrix} 0 \\ 0 \\ -0.5999216 \\ 0 \\ 0 \\ 0.57325729 \end{bmatrix}.$$

The value of the objective function in the optimal point is equal to  $f(x^*) = 0.7592843922$ . This solution was obtained in 5 iterations, 29 evaluations of the functions of the problem, and 23 evaluations of the gradients of the functions. Along the iterations, the value of the trust-region radius was  $\rho = 1000$ .  $\diamond$

### Numerical Study—filterSD: Solving Applications from the LACOP Collection

Table 18.1 presents the performances of an implementation of the filterSD algorithm given by Fletcher for solving 8 nonlinear optimization applications from the LACOP collection described in Appendix C.

Table 18.2 presents a comparison of filterSD (Table 18.1) versus NLPQLP (Table 15.4) and KNITRO/ACTIVE (option 3) (Table 15.6) for solving some nonlinear optimization applications from

**Table 18.1** Performances of filterSD for solving 8 applications from the LACOP collection

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#nf	#ng	<i>vfo</i>
ALKI	10	3	8	10	89	66	-1768.8069
ELCH	10	3	0	4	293	238	-47.761090
PREC	8	0	6	12	383	301	3.95116351
PPSE	9	6	0	5	27	24	5055.01180
MSP3	13	0	15	9	45	42	97.5482075
MSP5	16	0	21	21	195	165	174.786994
POOL	34	20	0	3	7	7	2785.80000
LATHE	10	1	14	12	43	43	-4434.0019

In this table we have:  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #it = the number of iterations to get a solution, #nf = the number of the evaluations of the functions defining the problem, #ng = the number of the evaluations of the gradients of the functions,  $vfo$  = the value of the objective function in the optimal point

**Table 18.2** Comparison of filterSD versus NLPQLP and KNITRO

	filterSD		NLPQLP		KNITRO (3)	
	#it	#nf	#it	#nf	#it	#nf
ELCH	4	293	21	30	18	19
ALKI	10	89	53	69	22	58
PREC	12	383	19	20	45	117
LATHE	12	43	19	19	83	242
PPSE	5	27	9	10	6	11
MSP3	9	45	146	264	7	9
POOL	3	7	22	22	15	33
TOTAL	55	887	289	434	196	489

the LACOP collection subject to the number of iterations ( $\#it$ ) and to the number of evaluations of the functions defining the problem ( $\#nf$ ).

In filterSD, for solving the linear programming subproblems with simple bounds on variables, a recursive form of an active-set method is used at each iteration. To solve degeneracy, the Wolfe method is implemented. A limited memory reduced gradient sweep method is used for minimization in the null space, so the KKT point is usually a local minimizer. Observe that filterSD has very good numerical performances for solving nonlinear optimization problems (see Table 18.2). The main advantage of this algorithm is that it avoids using second derivatives and also avoids storing an approximate reduced Hessian matrix by taking a limited memory spectral gradient approach based on the Ritz values (Fletcher, 2011).

## 18.2 Sequential Quadratic Programming Filter Algorithm

In this section, we give some details on the filter algorithm with sequential quadratic programming for solving the nonlinear optimization problem (18.1). The filter method views (18.1) as a biobjective nonlinear optimization problem: the first is the minimization of the objective function  $f$  and the second is the satisfaction of the constraints. These two conflicting aims can be formalized as

$$\min f(x) \quad (18.11)$$

and

$$\min h(c(x)), \quad (18.12)$$

where

$$h(c(x)) \triangleq \|c(x)^+\|_1 \triangleq \sum_{i=1}^m c_i(x)^+ \quad (18.13)$$

is the  $l_1$  norm of the constraint violation and  $c_i(x)^+ = \max\{0, c_i(x)\}$ ,  $i = 1, \dots, m$ . Observe that the problem of satisfiability has been written as a minimization problem. Hence, (18.11) and (18.12) represent a biobjective nonlinear optimization problem. Here, the  $l_1$  norm is used because it has some convenient features that can be exploited in defining the algorithm.

The fundamental idea of the algorithm is to use the filter as a criterion for accepting or rejecting a step in the sequential quadratic programming method. In other words, given the current point  $x_k$ , the solution of the quadratic programming subproblem

$$\begin{aligned} & \min_d \frac{1}{2} d^T W_k d + d^T g_k \\ & \text{subject to} \\ & A_k^T d + c_k \leq 0, \\ & \|d\|_\infty \leq \rho, \end{aligned} \quad (18.14)$$

generates a step  $d_k$ . Here,  $g(x) = \nabla f(x)$ ,  $W_k$  is (an approximation to) the Hessian of the Lagrangian  $L(x, \lambda) = f(x) + \lambda^T c(x)$ ,  $A_k$  is the Jacobian of the constraints and  $c_k$  is the value of the constraints at the point  $x_k$ . Set  $x_{k+1} = x_k + d_k$ . The new point  $x_{k+1}$  is accepted by the filter if the corresponding pair  $(h_{k+1}, f_{k+1})$  is not dominated by any other point in the filter. Otherwise, the step is rejected and the trust-region radius  $\rho$  is reduced. The use of the  $l_\infty$  norm in (18.14) ensures that (18.14) remains tractable as quadratic programming.

The above strategy based on solving a sequence of quadratic programming subproblems must be completed in order to cope with the situations in which the subproblems are infeasible. In a sequential quadratic programming trust-region method, reducing the trust-region radius will give rise to an infeasible quadratic programming subproblem if the current point is infeasible in the nonlinear optimization problem. Usually, a trust-region algorithm arrives at this situation after rejecting a number of consecutive steps. Thus, it is not sufficient to simply increase the trust-region radius to regain feasibility. An infeasible quadratic programming subproblem also occurs when the linearized constraints are themselves inconsistent. To overcome this situation, Fletcher and Leyffer (2002) chose the strategy to minimize  $h(c(x))$ . This is referred to as the *restoration phase*, which tries to get close to the feasible region of the nonlinear optimization problem. A possible outcome is that the restoration phase finds a nonzero minimum of  $h(c(x))$  which is taken as an indication that the nonlinear optimization problem is infeasible. Otherwise, we can assume that the algorithm finds a point at which the quadratic programming subproblem is feasible and therefore the above method can be continued from this point.

Having in view all these developments, the basic sequential quadratic programming filter algorithm starts the iterations with an initial guess  $x_0$  and an initial estimate of the Lagrange multipliers  $\lambda_0$ . On the subsequent iterations the Lagrange multipliers are updated whenever the sequential quadratic programming subproblem generates a new point that is accepted by the filter.

**Algorithm 18.2** Basic sequential quadratic programming filter—filterSQP

- |    |  |
|----|--|
| 1. | Choose an initial point $x_0$ , a value for the parameter $\rho$ and set $k = 1$   |
| 2. | Test a criterion for stopping the iterations   |
| 3. | Solve the quadratic programming subproblem (18.14)   |
| 4. | If (18.14) is infeasible, find a new point $x_{k+1}$ using the restoration phase; otherwise, determine the step $d_k$ and provisionally set $x_{k+1} = x_k + d_k$ and continue with step 5   |
| 5. | If the pair $(h_{k+1}, f_{k+1})$ is acceptable to the filter, then:<br>accept $x_{k+1}$ and add the pair $(h_{k+1}, f_{k+1})$ to the filter,<br>remove the points dominated by the pair $(h_{k+1}, f_{k+1})$ ,<br>possibly increase the trust-region radius,<br>otherwise,<br>set $x_{k+1} = x_k$ and reduce the trust-region radius |
| 6. | Set $k = k + 1$ and go to step 2   |

◆

Algorithm 18.2 is the basic filter algorithm with sequential quadratic programming. It is very easy to implement, but some algorithmic extensions are required to exclude the situations in which it might fail or might converge slowly. Fletcher and Leyffer (1998, 2002) present some extensions and techniques for its protection against certain difficulties in solving nonlinear optimization problems. Some of these extensions are as follows.

*Second-order correction step.* One of the most important property of any sequential quadratic programming algorithm is that it usually has second-order convergence near the solution. However, the use of a non-differentiable penalty function can preclude the acceptance of the unit step arbitrarily close to the solution, thus preventing second-order convergence. This is known as the Maratos effect. This difficulty can be avoided by computing a correction to the step that eliminates second-order contributions of the nonlinear constraints. This is the second-order correction step. In the filter algorithm, it is quite possible for a sequential quadratic programming unit step to increase both the objective and the constraint violation functions.

The second-order correction step is computed whenever  $x_{k+1} = x_{(k+1, 0)}$  is rejected by the filter in Algorithm 18.2. In this case, the quadratic programming subproblem that is solved is defined as

$$\begin{aligned} \min_d \frac{1}{2} d^T W_k d + d^T g_k \\ \text{subject to} \\ A_k^T d + c_{(k+1,l)} - A_k^T d_{(k,l)} \leq 0, \\ \|d\|_\infty \leq \rho, \end{aligned} \quad (18.15)$$

for  $l = 0, 1, \dots$ , where  $c_{(k+1, 0)} = c_{k+1}$  and  $d_{(k, 0)} = d_k$ . Let  $\hat{d}_k \triangleq d_{(k,l+1)}$  be the solution of (18.15). The new point  $x_{(k+1,l)} = x_k + \hat{d}_k$  is tested in the filter. If the pair  $(h_{(k+1,l)}, f_{(k+1,l)})$  is acceptable to the filter, then step  $d_{(k,l+1)}$  is accepted and the trust-region radius may be increased. Otherwise, a sequence of second-order correction steps is performed, generating a sequence of trial points  $x_{(k+1,l)}$ ,  $l = 0, 1, \dots$ , until one of the following holds:

- (i) An acceptable trial point  $x_{(k+1,j)}$  is found.
- (ii) An infeasible quadratic programming problem is detected.
- (iii) The rate of convergence of the second-order correction steps defined by

$$r \triangleq \frac{h_{(k+1,l)}}{h_{(k+1,l-1)}}$$

is considered to be too slow, or

- (iv) An almost feasible point with  $h_{(k+1,l)} < \varepsilon$  is generated, where  $\varepsilon$  is a tolerance used in solving the problem (18.1).

In case (i), the next iteration is  $x_{k+1} = x_{(k+1,j)}$  and the filter with sequential quadratic programming continues from step 2. In all the other cases, the steps are rejected and the trust-region radius is reduced. The last case (iv) ensures the finiteness of this process. The best second-order correction step is stored by using a penalty function estimate to rank the steps.

*Feasibility restoration phase.* This is the most complex part of the algorithm. One difficulty in using the trust-region approach is that the reduction of the trust-region radius may cause the quadratic programming subproblem to become infeasible. Besides, the linearizations of the nonlinear constraints may themselves be inconsistent. The strategy of Fletcher and Leyffer (2002) for dealing with this situation is to enter a restoration phase in which the purpose is to get closer to the feasible region by minimizing  $h(c(x))$ .

If an infeasible quadratic programming problem is detected, then the solver exits with a solution of the linear programming problem

$$\begin{aligned} \min_d \sum_{j \in J} \nabla c_j(x_k)^T d \\ \text{subject to:} \\ \nabla c_j(x_k)^T d + c_j(x_k) \leq 0, j \in J^\perp, \\ \|d\|_\infty \leq \rho. \end{aligned} \quad (18.16)$$

In other words, the quadratic programming solver partitions the constraints into two index sets:  $J \subset \{1, 2, \dots, m\}$  and its complement  $J^\perp = \{1, 2, \dots, m\} \setminus J$ . The set  $J$  contains infeasible constraints at the point  $x_k$ :  $c_j(x_k) + \nabla c_j(x_k)^T d > 0$ ,  $j \in J$ , whose  $l_1$  sum is minimized at the solution to the linear programming problem (18.16) with respect to the constraints in  $J^\perp$  being satisfied.

The strategy in the restoration phase is to apply a sequential quadratic programming trust-region method to the nonlinear problem

$$\begin{aligned} \min_x \sum_{j \in J} c_j(x)^+ \\ \text{subject to:} \\ c_j(x) \leq 0, j \in J^\perp, \end{aligned} \tag{18.17}$$

that is defined by this partitioning into the sets  $J$  and  $J^\perp$ . However, the difficulty is that it is quite possible for  $J$  and  $J^\perp$  to change from iteration  $k$  to iteration  $k + 1$ , thus making it difficult to enforce convergence. Therefore, the restoration phase consists of a sequence of sequential quadratic programming iterations that continues while the subproblem (18.14) is infeasible. At each iteration, first check the feasibility of the system

$$\nabla c_j(x_k)^T d + c_j(x_k) \leq 0, j = 1, \dots, m, \tag{18.18a}$$

$$\|d\|_\infty \leq \rho. \tag{18.18b}$$

If the system (18.18) has a feasible solution, then the restoration phase ends. Otherwise, the sets  $J$  and  $J^\perp$  which solve (18.16) are determined. Next, a quadratic programming subproblem is constructed by adding the second-order term  $(1/2)d^T W_k d$  into the objective function of (18.16), where

$$W(x, \lambda) = \sum_{j \in J} \nabla^2 c_j(x) + \sum_{j \in J^\perp} \lambda_j \nabla^2 c_j(x). \tag{18.19}$$

The multipliers  $\lambda_j$ ,  $j \in J^\perp$ , are those obtained from the solution of (18.16). Note that for solving (18.16) a warm start can be used, i.e., the solution of (18.16) can be used to initialize the solution of the new quadratic programming problem. The iterations in the restoration phase are continued until either a feasible quadratic programming problem is encountered (case in which the algorithm can return to the basic sequential quadratic programming algorithm) or an infeasible KKT point of (18.17) is found for some sets  $J$  and  $J^\perp$  (case in which the algorithm terminates with the indication that (18.1) is locally infeasible).

For solving (18.17), Fletcher and Leyffer (2002) suggest using the filter method by introducing the so-called *restoration filter* (or *phase I filter*) defined by

$$h_J \triangleq h(c_J(x)) \triangleq \sum_{j \in J} c_j(x)^+$$

and similarly  $h_{J^\perp}$ . A phase I filter is a list of pairs  $(h_J, h_{J^\perp})$  such that no pair dominates any other.

Fletcher and Leyffer (2002) present a computational situation which leads to the introduction of the *blocking entry* in the filter. Consider the situation in which (18.1) has a global solution  $x^{**}$  and a worse local solution  $x^*$ , as well as that  $x_0$  is a feasible point fairly close to  $x^{**}$ , but the subsequent iterates  $x_k$  are convergent to  $x^*$ . Then, if  $f(x_0) \leq f(x^*)$ , then the filter entry  $(h_0, f_0)$  prevents the sequential quadratic programming iteration from converging to  $x^*$ . In this case, we would like to backtrack to  $x_0$ , but our decision not to store  $x_0$  with the filter information precludes this. The pair

$(h_0, f_0)$  is called *blocking entry* in the filter. Some more details are found in (Fletcher, & Leyffer, 2002). The algorithm for the restoration phase is as follows.

**Algorithm 18.3 Feasibility restoration algorithm**

1. Consider the current point  $x_k$ , the trust-region radius  $\rho$  and a constraint upper bound  $u$  which measures the violation of the constraints
2. Test a criterion for stopping the iterations
3. Solve (18.16) and determine the sets  $J$  and  $J^\perp$ , which remain fixed for this iteration
4. If (18.16) is feasible, then return to a normal sequential quadratic programming and clear the restoration filter. Otherwise, add the quadratic term (18.19) to the objective function of (18.16) and solve phase I of the quadratic programming algorithm. Let  $d_k$  be the solution and set  $x_{k+1} = x_k + d_k$
5. If  $(h_J^{k+1}, h_{J^\perp}^{k+1})$  is acceptable to the phase I filter, then:
  - accept  $x_{k+1}$  and add the pair  $(h_J^{k+1}, h_{J^\perp}^{k+1})$  to the restoration filter,
  - remove the points dominated by  $(h_J^{k+1}, h_{J^\perp}^{k+1})$  from the filter,
  - possibly increase the trust-region radius  $\rho$ .
 Otherwise, solve a sequence of quadratic programming second-order correction problems to get the step  $\hat{d}_k$ . Set  $\hat{x}_{k+1} = x_k + \hat{d}_k$  and go to step 6
6. If  $(\hat{h}_J^{k+1}, \hat{h}_{J^\perp}^{k+1})$  is acceptable to the phase I restoration filter, then:
  - accept  $\hat{x}_{k+1}$  and add  $(\hat{h}_J^{k+1}, \hat{h}_{J^\perp}^{k+1})$  to the restoration filter,
  - remove the points dominated by  $(\hat{h}_J^{k+1}, \hat{h}_{J^\perp}^{k+1})$  from the restoration filter,
  - possibly increase the trust-region radius  $\rho$ .
 Otherwise, if  $J$  has changed from the iteration  $k - 1$  and  $\hat{h}_{(k,L)} < u$ , then:
  - accept the best second-order correction step, i.e., set  $x_{k+1} = \hat{x}_{k+1}$ ,
  - remove all the blocking entries from the restoration filter,
  - reduce the upper bound  $u$  to  $u = \max \{\hat{h}_{(k,L)}, u/10\}$ ,
  - otherwise:
    - reject the step, i.e., set  $x_{k+1} = x_k$ ,
    - reduce the trust-region radius  $\rho$ .
7. Set  $k = k + 1$  and go to step 2

◆

Observe that the trust-region radius is changed in the feasibility restoration algorithm. At its return to the sequential quadratic programming, the current value of  $\rho$  reflects how well a first-order Taylor series approximates the nonlinear constraints in  $x_k$ . Therefore, this value of  $\rho$  is used in the next step of the sequential quadratic programming.

*Sufficient reduction.* A new iterate  $x_{k+1}$  is acceptable to the filter if

$$h_{k+1} \leq \beta h_l \quad (18.20)$$

or if

$$f_{k+1} \leq f_l - \max \{\alpha_1 \Delta q_l, \alpha_2 h_l \mu_l\} \quad (18.21)$$

are satisfied for all the filter entries  $l$ . Here,  $\beta$ ,  $\alpha_1$  and  $\alpha_2$  are positive constants which can be taken to be:  $\beta = 0.99$ ,  $\alpha_1 = 0.25$ , and  $\alpha_2 = 0.0001$ , respectively. An estimation of the penalty parameter  $\mu_l$  is computed as the least power of ten but larger than  $\|\lambda_l\|_\infty$  and cuts this value off so as to lie in the interval  $[10^{-6}, 10^6]$ . The value of  $h_l \mu_l$  can be used as a predicted reduction in  $f$ . Both  $\Delta q_l$  and  $\mu_l$  are stored along with  $f_l$  and  $h_l$ . Observe that  $\mu_l$  measures the marginal effect of the changes in  $f$  due to

changes in  $h$ . This is the reason for using the term  $\alpha_2 h_l \mu_l$  in (18.21). A quadratic programming step predicts a reduction of  $h_l$  to zero, equivalent to a predicted reduction in  $f$  of  $h_l \mu_l$ .

*Beyond the extreme points of the filter.* Mainly, the filter is a reliable oracle for deciding whether or not to accept a step as long as the new constraint violation lies within the range of the constraint violations recorded in the filter. However, the current heuristics do not exclude the possibility of generating a sequence of filter entries in which  $\{f_k\}$  is monotonically increasing and  $\{h_k\}$  is monotonically decreasing without converging to a KKT point. But, as long as the sufficient reduction criterion in  $h$  is satisfied by the sequence, the points will be accepted by the filter. To overcome this possibility, a new additional heuristic is introduced, when  $h_{k+1}$  provides sufficient reduction from  $h_1$ , where  $h_1$  now refers to the leftmost entry in the filter. An overestimate  $\mu = 1000\mu_1$  of the penalty parameter is considered. It requires  $x_{k+1}$ , which provides a reduction in the resulting exact penalty function. The value of the exact penalty function corresponding to the leftmost entry  $f_1, h_1$  is  $f_1 + \mu h_1$ . The new point is then accepted if, in addition to the sufficient reduction from  $h_1$ , it satisfies  $f_{k+1} + \mu h_{k+1} \leq f_1 + \mu h_1$ . Fletcher and Leyffer (2002) called this the *North-West corner rule*. A similar rule is applied in the *South-East corner of the filter*.

Now, the complete sequential quadratic programming algorithm based on the filter concept can be presented as follows.

**Algorithm 18.4** *Filter sequential quadratic programming—filterSQP*

1.	Choose the initial point $x_0$ and the initial trust-region radius $\rho$ . Set $k = 0$
2.	Test a criterion for stopping the iterations
3.	Solve (18.14) for a step $d_k$ . If (18.14) is infeasible, then continue with the restoration phase (see Algorithm 18.3) and return to the normal sequential quadratic programming when $x_{k+1}$ is found, whose corresponding quadratic programming is feasible. Otherwise, set $x_{k+1} = x_k + d_k$ and go to step 4
4.	If $(h_{k+1}, f_{k+1})$ is acceptable to the filter, then: accept $x_{k+1}$ and add $(h_{k+1}, f_{k+1})$ to the filter, remove the points dominated by $(h_{k+1}, f_{k+1})$ from the filter, possibly increase the trust-region radius $\rho$ , otherwise, go to step 5
5.	If $h_{k+1} > 0$ , then solve a sequence of quadratic programming problems for the second-order correction step $\hat{d}_k$ . Set $\hat{x}_{k+1} = x_k + \hat{d}_k$ If $(\hat{h}_{k+1}, \hat{f}_{k+1})$ is acceptable to the filter, then: accept $\hat{x}_{k+1}$ and $(\hat{h}_{k+1}, \hat{f}_{k+1})$ to the filter, remove the points dominated by $(\hat{h}_{k+1}, \hat{f}_{k+1})$ from the filter, possibly increase the trust-region radius $\rho$ , otherwise, go to step 6
6.	If there is no acceptable point in the filter, then: If the algorithm is in the first iteration after the restoration phase, then: accept the best second-order correction step, i.e., set $x_{k+1} = \hat{x}_{k+1}$ , remove all the blocking entries from the filter, reduce the upper bound to: $u = \max \{\hat{h}_{k+1}, u/10\}$ , otherwise, reject step $\hat{d}_k$ , i.e., set $x_{k+1} = x_k$ and reduce the trust-region radius $\rho$
7.	Set $k = k + 1$ and go to step 2

Some more details regarding a specific implementation of this algorithm can be found in Fletcher and Leyffer (2002).

**Example 18.2** Let us consider the nonlinear optimization problem from the example 18.1 solved by means of the filter with sequential linear programming—filterSD. The filterSQP algorithm in the implementation given by Fletcher and Leyffer (1999) with initial point  $x_0 = [0, 0, 0, 0, 0, 0]^T$  gives the following results.

The value of the objective function in the initial point is  $f(x_0) = 10$ . FilterSQP determines the solution  $x^* = [1.14651, 0.54659, 1, 0.27329, 0.29996, 0]^T$  for which the value of the objective is  $f(x^*) = 0.75928439$ . This solution was obtained in 4 iterations, 5 evaluations of the functions defining the problem, 5 evaluations of the gradients of the functions, and 5 evaluations to the Hessian of the Lagrange function. The number of quadratic programming subproblems is 4. The value of the norm of constraints at the solution is  $\|c(x^*)\| = 1.3222E - 10$ . The final trust-region radius is  $\rho = 10$ . The norm of the KKT optimality conditions is equal to  $1.393E - 8$ .  $\blacklozenge$

The global convergence of the trust-region sequential quadratic programming filter algorithm, filterSQP, under reasonable conditions (the iterates  $x_k$  are in a compact set, the functions  $f(x)$  and  $c(x)$  are twice continuously differentiable, the Hessian of the Lagrange function associated to the problem is bounded), was proved by Fletcher, Gould, Leyffer, Toint, and Wächter (2002a). The idea of the proof is to decompose the step into its normal and tangential components. This allows for an approximate solution of the quadratic subproblem and incorporates the safeguarding test described above. The normal step is responsible for the feasibility of the linearized constraints from (18.14). The tangential step reduces the value of the objective function model while continuing to satisfy the constraints from (18.14).

### Notes and References

The filter method was suggested by Fletcher at a SIAM Conference in 1999. The filter method is motivated by the aim of avoiding the need to choose penalty parameters, as it occurs with the use of  $l_1$  penalty functions or the augmented Lagrangian functions. The filter methods know two implementations: with sequential linear programming (filterSD) and with sequential quadratic programming (filterSQP). The package filterSD calls the linearly constrained problem solver *glcpd* or the quadratic programming solver *qlcpd*. The package filterSQP (Fletcher, & Leyffer, 1998) implements a sequential quadratic programming solver with a “filter” to promote global convergence. A quadratic approximation to the original problem is solved within a trust-region, where trust the region-radius is changed adaptively by the algorithm. The solver runs with a dense or a sparse linear algebra package and a robust quadratic programming solver *bqpd*. *bqpd* is a null-space active set method that builds up a factorization of the reduced Hessian, which is the projection of the Hessian onto the null space of constraints that are currently regarded as being active.

The global convergence of the filter method with sequential linear programming was proved by Fletcher, Leyffer, and Toint (1999). Later on, the global convergence of the filter method with sequential quadratic programming was presented by Fletcher, Leyffer, and Toint (2002b), Ulbrich (2004), and Fletcher, Leyffer, and Shen (2009). These filter methods were independently suggested by Surry, Radcliffe, and Boyd (1995) and Lemaréchal, Nemirovskii, and Nesterov (1995). Ulbrich, Ulbrich, and Vicente (2004) considered a trust-region filter method based on the acceptance of the trial steps using the norm of the optimality conditions. On the other hand, Benson, Shanno, and Vanderbei (2002a, 2002b) proposed several heuristics based on the idea of the filter method, for which improved efficiency was reported versus the merit function approach. Even if the filter methods can solve optimization problems, it is not known yet whether the filter strategy has advantages over the merit functions.



## Interior-Point Filter Line-Search

19

In this chapter, a combination called IPOPT of the interior-point algorithm with the filter line-search for solving large-scale nonlinear optimization problems, proposed by Wächter and Biegler (2005a, b), is presented. As we know, to allow convergence from poor starting points and to enforce progress to the solution, the interior point methods both in the trust-region and in the line-search frameworks with exact penalty merit function were developed. For example, KNITRO uses the  $l_1$  exact penalty function (Byrd, Gilbert, & Nocedal, 2000; Byrd, Hribar, & Nocedal, 1999). On the other hand, Fletcher and Leyffer (2002) proposed the filter methods as an alternative to the merit functions, as a tool for global convergence guarantee in the algorithms for nonlinear optimization. The idea of the filter is that the trial points are accepted if they improve the objective function value or the constraint violation instead of a combination of these two measures defined by the merit function. Even if the filter methods include different heuristics, they have been adapted to the barrier methods in a number of ways. For example, Ulbrich, Ulbrich, and Vicente (2004) considered a trust-region filter method and accepted the trial step on the basis of the norm of the optimality conditions. Benson, Shanno, and Vanderbei (2002a) proposed several heuristics using the filter methods, for which the improved efficiency is reported compared to their previous merit function approach. The global convergence of an interior point algorithm with filter line-search was analyzed by Wächter and Biegler (2001). Here, the assumptions made for the analysis of the global convergence are less restrictive than those made for the line-search interior-point methods for nonlinear optimization developed, for example, by El-Bakry, Tapia, Tsuchiya, and Zhang (1996), Yamashita (1998) or Tits, Wächter, Bakhtiari, Urban, and Lawrence (2003).

In this context, IPOPT is an interior-point filter line-search algorithm for solving large-scale nonlinear optimization problems developed by Wächter and Biegler (2001, 2006). This algorithm combines the primal-dual interior-point algorithms with the filter line-search for solving problems of the form

$$\min f(x) \quad (19.1a)$$

subject to

$$c(x) = 0, \quad (19.1b)$$

$$x_L \leq x \leq x_U, \quad (19.1c)$$

where  $x \in \mathbb{R}^n$ ,  $x_L \in \mathbb{R}^n$  and  $x_U \in \mathbb{R}^n$  are lower and upper bounds on variables,  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $c : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , with  $m \leq n$ , are assumed to be twice continuously differentiable. Problems with general nonlinear inequality constraints can be reformulated in the above form by introducing the slack variables.

In the following, we shall present the primal-dual interior-point algorithm IPOPT, which involves the step computation, the line-search with filter, the second-order corrections, and the restoration of feasibility as well as some heuristics which improve the performances of the algorithm. Finally, numerical results for solving some nonlinear optimization applications together with comparisons are presented.

## 19.1 Basic Algorithm IPOPT

### The Primal-Dual Barrier Approach

To simplify the presentation, we begin by presenting the method for solving the problem

$$\min f(x) \quad (19.2a)$$

$$\begin{aligned} & \text{subject to} \\ & c(x) = 0, \end{aligned} \quad (19.2b)$$

$$x \geq 0. \quad (19.2c)$$

Later on, we will see the method for solving general nonlinear optimization problems with simple bounds on variables. As a barrier method, the proposed algorithm computes an approximate solution of a sequence of barrier subproblems

$$\min_{x \in \mathbb{R}^n} \varphi_\mu(x) \triangleq f(x) - \mu \sum_{i=1}^n \log(x^i) \quad (19.3a)$$

$$\begin{aligned} & \text{subject to} \\ & c(x) = 0, \end{aligned} \quad (19.3b)$$

for a decreasing sequence of barrier parameters  $\{\mu_j\}$  converging to zero, where  $x^i$  is the  $i$ -th component of vector  $x$ . This problem can be interpreted as applying a homotopy method to the primal-dual equations

$$\nabla f(x) + \nabla c(x)\lambda - z = 0, \quad (19.4a)$$

$$c(x) = 0, \quad (19.4b)$$

$$XZe - \mu e = 0, \quad (19.4c)$$

where the homotopy parameter  $\mu$  is driven to zero (Byrd, Liu, & Nocedal, 1997). In (19.4),  $\lambda \in \mathbb{R}^m$  and  $z \in \mathbb{R}^n$  are the Lagrange multipliers associated to (19.2b) and (19.2c), respectively. Observe that the equations (19.4) with  $\mu = 0$  and the constraints  $x, z \geq 0$  are exactly the KKT optimality conditions for (19.2). Clearly, if the constraint qualifications (see Remark 11.2) are satisfied, then these are the first-order optimality conditions.

The strategy for solving (19.2) is to compute an approximate solution to the barrier problem (19.3) for a fixed value of  $\mu$ , and then decrease the barrier parameter and continue the solution of the next barrier problem from the approximate solution of the previous one, etc.

From the primal-dual equations (19.4), we can see that the optimality error for the barrier problem can be defined as

$$E_\mu(x, \lambda, z) = \max \left\{ \frac{\|\nabla f(x) + \nabla c(x)\lambda - z\|_\infty}{s_d}, \quad \|c(x)\|_\infty, \quad \frac{\|XZe - \mu e\|_\infty}{s_c} \right\}, \quad (19.5)$$

where the scaling parameters  $s_d, s_c \geq 1$  are defined below.  $E_0(x, \lambda, z)$  is the value of (19.5) for  $\mu = 0$ . This measures the optimality error for the original problem (19.2). With this, we can consider that the algorithm terminates with an approximate solution  $(x^*, \lambda^*, z^*)$  if

$$E_0(x^*, \lambda^*, z^*) \leq \varepsilon_{TOL}, \quad (19.6)$$

where  $\varepsilon_{TOL} > 0$  is a tolerance for stopping the algorithm.

Even if the problem is well scaled, the multipliers  $\lambda$  and  $z$  might become very large, for example when the gradients of the active constraints are nearly linear dependent at the solution of (19.2). In this case, the algorithm faces major numerical difficulties. In order to adapt the termination criteria to handle such situations, in (19.5) the scaling factors are introduced as

$$s_d = \max \left\{ 1, \frac{\|\lambda\|_1 + \|z\|_1}{(m+n)s_{\max}} \right\}, \quad s_c = \max \left\{ 1, \frac{\|z\|_1}{ns_{\max}} \right\}$$

In this way, a component of the optimality error is scaled whenever the average value of the multipliers becomes larger than a fixed number  $s_{\max} \geq 1$ . (For example  $s_{\max} = 100$ .)

The algorithm has two types of iterations: *major iterations* denoted by  $j$ , in which the value of the barrier parameter  $\mu_j$  is modified, and *minor iterations* denoted by  $k$ , in which the barrier problem is solved for a fixed value  $\mu_j$  of the barrier parameter. In order to achieve the superlinear convergence of the algorithm, i.e., to get a local solution of (19.2) satisfying the second-order sufficient optimality conditions (see Theorem 11.13), Wächter and Biegler (2006) follow the approach proposed by Byrd, Liu, and Nocedal (1997). Therefore, at the major iteration  $j$  it is required that the approximate solution  $(x_{j+1}^*, \lambda_{j+1}^*, z_{j+1}^*)$  of the barrier problem (19.3) for a given value of the barrier parameter  $\mu_j$  should satisfy the tolerance test

$$E_{\mu_j}(x_{j+1}^*, \lambda_{j+1}^*, z_{j+1}^*) \leq \kappa_e \mu_j \quad (19.7)$$

before the algorithm continues with the solution of the next barrier problem, where  $\kappa_e > 0$  is a given constant. The barrier parameter is updated as

$$\mu_{j+1} = \max \left\{ \frac{\varepsilon_{TOL}}{10}, \quad \min \left\{ \kappa_\mu \mu_j, \quad \mu_j^{\theta_\mu} \right\} \right\}, \quad (19.8)$$

with the constants  $\kappa_\mu \in (0, 1)$  and  $\theta_\mu \in (1, 2)$ . In this way, the barrier parameter is decreased at a superlinear rate. On the other hand, given the desired tolerance  $\varepsilon_{TOL}$ , the updating given by (19.8) does not allow  $\mu$  to become smaller than necessary, thus avoiding numerical difficulties at the end of the optimization procedure.

## Solving the Barrier Problem

For solving the barrier problem (19.3) in which the barrier parameter is fixed at the value  $\mu_j$ , the algorithm uses the Newton method applied to the primal-dual equations (19.4). As we have already said, the minor iterations necessary for solving the barrier problem are denoted by  $k$ . Therefore, given an iteration  $(x_k, \lambda_k, z_k)$  with  $x_k, z_k > 0$ , then the search direction  $(d_k^x, d_k^\lambda, d_k^z)$  is obtained as solution of the linearization of (19.4) at  $(x_k, \lambda_k, z_k)$ , namely,

$$\begin{bmatrix} W_k & J_k & -I \\ J_k^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^\lambda \\ d_k^z \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) + J_k \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{bmatrix} \quad (19.9)$$

In (19.9),  $J_k = \nabla c(x_k)$  and  $W_k$  is the Hessian of the Lagrange function associated to the problem (19.2)

$$L(x, \lambda, z) = f(x) + c(x)^T \lambda - z. \quad (19.10)$$

Instead of solving the nonsymmetric linear system (19.9), the proposed method computes the solution equivalently by first solving the smaller symmetric linear system

$$\begin{bmatrix} W_k + \Sigma_k & J_k \\ J_k^T & 0 \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^\lambda \end{bmatrix} = - \begin{bmatrix} \nabla \varphi_{\mu_j}(x_k) + J_k \lambda_k \\ c(x_k) \end{bmatrix} \quad (19.11)$$

derived from (19.9) by eliminating the last block row, where  $\Sigma_k = X_k^{-1} Z_k$ . The vector  $d_k^z$  is then computed as

$$d_k^z = \mu_j X_k^{-1} e - z_k - \Sigma_k d_k^x \quad (19.12)$$

As we know, in most line-search methods, we have to ensure that the matrix  $W_k + \Sigma_k$  from (19.11) projected onto the null space of the constraint Jacobian  $J_k^T$ , must be positive definite. This is necessary to guarantee certain descent properties of the filter line-search used in the algorithm described below. Also, if  $J_k$  does not have full rank, the iteration matrix in (19.11) is singular, i.e., the system (19.11) does not have any solutions. Therefore, as protection, it might be necessary to modify the matrix of the system (19.11). Wächter and Biegler (2006) suggested the following modification of the system (19.11):

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & J_k \\ J_k^T & -\delta_c I \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^\lambda \end{bmatrix} = - \begin{bmatrix} \nabla \varphi_{\mu_j}(x_k) + J_k \lambda_k \\ c(x_k) \end{bmatrix}, \quad (19.13)$$

where  $\delta_w, \delta_c \geq 0$ . The choice of these constants is to be discussed.

Having a solution to the systems (19.13) and using (19.12), we must determine the step sizes. Since  $x$  and  $z$  are both positive at an optimal solution of the barrier problem (19.3), then this property has to be maintained for all the iterations. Therefore, in order to achieve this property, we first compute

$$\alpha_k^{\max} = \max \{ \alpha \in (0, 1) : x_k + \alpha d_k^x \geq (1 - \tau_j) x_k \}, \quad (19.14a)$$

$$\alpha_k^z = \max \{ \alpha \in (0, 1) : z_k + \alpha d_k^z \geq (1 - \tau_j) z_k \}, \quad (19.14b)$$

where the parameter  $\tau_j \in (0, 1)$  is computed as

$$\tau_j = \max \{ \tau_{\min}, 1 - \mu_j \}, \quad (19.15)$$

$\tau_{\min} \in (0, 1)$  being its minimum value. In interior point methods,  $\tau_j$  is known as the parameter which represents the *fraction-to-the-boundary* of the feasible region. With this, the next iteration is computed as

$$x_{k+1} = x_k + \alpha_k d_k^x, \quad (19.16a)$$

$$\lambda_{k+1} = \lambda_k + \alpha_k d_k^\lambda, \quad (19.16b)$$

$$z_{k+1} = z_k + \alpha_k^z d_k^z, \quad (19.16c)$$

where  $\alpha_k^z$  is computed as in (19.14b) and, in order to ensure the global convergence, the step size  $\alpha_k \in (0, \alpha_k^{\max}]$  for the remaining variables is determined by a backtracking line-search procedure exploring a decreasing sequence of trial steps:  $\alpha_{k,l} = 2^{-l} \alpha_k^{\max}$  (with  $l = 0, 1, \dots$ ) of Armijo type.

Observe the difference between the computation of  $z_{k+1}$  and the rest of the variables. We can see that the parameter which represents the fraction-to-the-boundary is used only in computing the variables  $x_{k+1}$  and  $z_{k+1}$ .

For achieving the superlinear convergence of the algorithm, Wächter and Biegler (2001) prove that the *primal-dual barrier term Hessian*  $\Sigma_k$  does not deviate too much from the *primal Hessian*  $\mu_j X_k^{-2}$ . This requirement is achieved by resetting the components  $z_{k+1}^i$  of  $z_{k+1}$  as

$$z_{k+1}^i = \max \left\{ \min \left\{ z_{k+1}^i, \frac{\kappa_\Sigma \mu_j}{x_{k+1}^i} \right\}, \frac{\mu_j}{\kappa_\Sigma x_{k+1}^i} \right\}, \quad i = 1, \dots, n, \quad (19.17)$$

where  $\kappa_\Sigma \geq 1$  is a fixed parameter. This strategy guarantees that each component  $\sigma_{k+1}^i$  of the diagonal matrix  $\Sigma_{k+1}$  is in the interval

$$\sigma_{k+1}^i \in \left[ \frac{1}{\kappa_\Sigma} \frac{\mu_j}{(x_k^i)^2}, \kappa_\Sigma \frac{\mu_j}{(x_k^i)^2} \right]. \quad (19.18)$$

In IPOPT,  $\kappa_\Sigma = 10^{10}$ .

## Line-Search Filter Method

This method interprets the solving of the barrier problem (19.3) for  $\mu_j$  fixed as a biobjective optimization problem with two goals: minimizing the objective function  $\varphi_{\mu_j}(x)$  and minimizing the constraint violation defined by  $\theta(x) \triangleq \|c(x)\|$ . As seen in Chap. 18, for solving this biobjective minimization problem, the emphasis is on minimizing the constraint violation.

In this context, in the line-search with backtracking a trial point  $x_k(\alpha_{k,l}) \triangleq x_k + \alpha_{k,l} d_k^x$  is considered as acceptable if it leads to sufficient progress toward either goal compared to the current iterate, i.e., if

$$\theta(x_k(\alpha_{k,l})) \leq (1 - \gamma_\theta) \theta(x_k), \quad (19.19a)$$

$$\text{or } \varphi_{\mu_j}(x_k(\alpha_{k,l})) \leq \varphi_{\mu_j}(x_k) - \gamma_\varphi \theta(x_k), \quad (19.19b)$$

holds for the fixed constants  $\gamma_\theta, \gamma_\varphi \in (0, 1)$ .

However, as we can see, each of the above criteria is complicated enough and therefore is replaced by requiring sufficient progress in the barrier objective function whenever for the current iterate, we have  $\theta(x_k) \leq \theta^{\min}$  for some constant  $\theta^{\min} \in (0, \infty]$  and the following *switching condition*

$$\nabla \varphi_{\mu_j}(x_k)^T d_k^x < 0, \quad \text{and} \quad \alpha_{k,l} \left[ -\nabla \varphi_{\mu_j}(x_k)^T d_k^x \right]^{s_\varphi} > \delta[\theta(x_k)]^{s_\theta}, \quad (19.20)$$

with the constants  $\delta > 0$ ,  $s_\theta > 1$ ,  $s_\varphi \geq 1$ , is satisfied. Observe that if  $\theta(x_k) \leq \theta^{\min}$  and (19.20) is true for the current stepsize  $\alpha_{k,l}$ , then the trial point has to satisfy the Armijo condition

$$\varphi_{\mu_j}(x_k(\alpha_{k,l})) \leq \varphi_{\mu_j}(x_k) + \eta_\varphi \alpha_{k,l} \nabla \varphi_{\mu_j}(x_k)^T d_k^x \quad (19.21)$$

instead of the condition (19.19) for the trial point to be acceptable. In (19.21),  $\eta_\varphi \in (0, 1/2)$  is a constant.

The algorithm maintains a filter, i.e., a set  $F_k \subseteq \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq 0\}$  computed for each iteration  $k$ . The filter  $F_k$  contains the combinations of the constraint violation values  $\theta$  and the objective function values  $\varphi$  which are prohibited for a successful trial point at iteration  $k$ , i.e. during the line-search, a trial point  $x_k(\alpha_{k,l})$  is rejected if  $(\theta(x_k(\alpha_{k,l})), \varphi_{\mu_j}(x_k(\alpha_{k,l}))) \in F_k$ . We say that the trial point is not acceptable to the current filter. At the beginning of the optimization process, the filter is initialized as

$$F_0 = \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq \theta^{\max}\}, \quad (19.22)$$

for some  $\theta^{\max}$ , so that the algorithm will never allow trial points that have a constraint violation larger than  $\theta^{\max}$  to be accepted. Later, after every iteration, the filter is augmented using the update formula

$$F_{k+1} = F_k \cup \left\{ (\theta, \varphi) \in \mathbb{R}^2 : \theta \geq (1 - \gamma_\theta) \theta(x_k) \text{ and } \varphi \geq \varphi_{\mu_j}(x_k) - \gamma_\varphi \theta(x_k) \right\}, \quad (19.23)$$

where the accepted trial step size does not satisfy the switching condition (19.20) or in which the Armijo condition (19.21) does not hold. This ensures that the iterates cannot return to the neighborhood of  $x_k$ . On the other hand, if both (19.20) and (19.21) hold for the accepted stepsize, the filter remains unchanged. This procedure ensures that the algorithm cannot cycle indefinitely, for example between two points that alternately decrease the constraint violation and the barrier objective function.

Finally, there are cases in which it is not possible to find a trial step size  $\alpha_{k,l}$  that satisfies the above criteria. In these cases, a minimum desired stepsize is approximated by using linear models of the involved functions. For this, Wächter and Biegler (2006) define the function

$$\alpha_k^{\min} \triangleq \gamma_\alpha \begin{cases} \min \left\{ \gamma_\theta, \frac{\gamma_\varphi \theta(x_k)}{-\nabla \varphi_{\mu_j}(x_k)^T d_k^x}, \frac{\delta[\theta(x_k)]^{s_\theta}}{\left[ -\nabla \varphi_{\mu_j}(x_k)^T d_k^x \right]^{s_\varphi}} \right\}, \\ \quad \text{if } \nabla \varphi_{\mu_j}(x_k)^T d_k^x < 0 \text{ and } \theta(x_k) \leq \theta^{\min}, \\ \min \left\{ \gamma_\theta, \frac{\gamma_\varphi \theta(x_k)}{-\nabla \varphi_{\mu_j}(x_k)^T d_k^x} \right\}, \\ \quad \text{if } \nabla \varphi_{\mu_j}(x_k)^T d_k^x < 0 \text{ and } \theta(x_k) > \theta^{\min}, \\ \gamma_\theta, \\ \quad \text{otherwise,} \end{cases} \quad (19.24)$$

where  $\gamma_\alpha \in (0, 1]$  is a “safety factor.” If the backtracking line-search encounters a trial step size with  $\alpha_{k,l} \leq \alpha_k^{\min}$ , then the algorithm reverts to a feasibility restoration phase. That is, the algorithm tries to find a new iterate  $x_{k+1} > 0$  which is acceptable to the current filter and for which the condition (19.19) holds by reducing the constraint violation using some kind of iterative method. Observe that the restoration phase might not be able to produce a new iterate for the filter line-search method, for example when the problem is infeasible.

## Second-Order Corrections

Many methods for nonlinear optimization use second-order corrections to improve the proposed step in case a trial point was rejected. A second-order correction (SOC) for some step  $d_k^x$  aims at reducing the infeasibility by applying an additional Newton step for the constraints at the point  $x_k + d_k^x$  using the Jacobian  $J_k$  evaluated at  $x_k$ . The details are given in (Wächter & Biegler, 2006) and are as follows.

If the first trial, the stepsize  $\alpha_{k,0}$  was rejected and if  $\theta(x_k(\alpha_{k,0})) \geq \theta(x_k)$ , then a second-order correction  $d_k^{x,soc}$  is computed, such that

$$J_k^T d_k^{x,soc} + c(x_k + \alpha_{k,0} d_k^x) = 0. \quad (19.25)$$

The new corrected search direction is obtained as

$$d_k^{x,cor} = d_k^{x,soc} + \alpha_{k,0} d_k^x. \quad (19.26)$$

The condition (19.25) does not uniquely define the second-order correction, and different choices would be possible. In order to avoid additional matrix factorization, the proposed method uses the same matrix as in (19.13) to compute the overall corrected step (19.26) as solution of the system

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & J_k \\ J_k^T & -\delta_c I \end{bmatrix} \begin{bmatrix} d_k^{x,cor} \\ d_k^x \end{bmatrix} = - \begin{bmatrix} \nabla \varphi_{\mu_j}(x_k) + J_k \lambda_k \\ c_k^{soc} \end{bmatrix}, \quad (19.27)$$

where

$$c_k^{soc} = \alpha_{k,0} c(x_k) + c(x_k + \alpha_{k,0} d_k^x) \quad (19.28)$$

is obtained from (19.13), (19.25), and (19.26).

As soon as the corrected search direction  $d_k^{x,cor}$  has been obtained, the fraction-to-the-boundary rule is applied to get

$$\alpha_k^{soc} \triangleq \max \left\{ \alpha \in (0, 1] : x_k + \alpha d_k^{x,cor} \geq (1 - \tau_j) x_k \right\} \quad (19.29)$$

and we test if the resulting trial point  $x_k^{soc} \triangleq x_k + \alpha_k^{soc} d_k^{x,cor}$  is accepted in the filter. Note that the original search direction  $d_k^x$  is still used in (19.20) and on the right-hand side of (19.21). Also,  $x_k^{soc}$  replaces  $x(\alpha_k)$  in (19.21). If this trial point passes the tests, then it is accepted as the new iterate. Otherwise, additional second-order corrections are applied until the correction step has not decreased the constraint violation by a fraction  $\kappa_{soc} \in (0, 1)$  or until a maximum number  $p^{\max}$  of second-order corrections has been performed. In this case, the original search direction  $d_k^x$  is restored and the regular backtracking line-search is restarted with a shorter step size  $\alpha_{k,1} = \alpha_{k,0}/2$ .

## The Algorithm

The following filter line-search algorithm for solving the barrier subproblem (19.3) can be presented.

### Algorithm 19.1 Line-search filter barrier algorithm—Wächter and Biegler

1.	Choose an initial point $(x_0, \lambda_0, z_0)$ with $x_0, z_0 > 0$ , an initial value for the barrier parameter $\mu_0 > 0$ , and the constants $\varepsilon_{TOL}, s_{\max} \geq 1, \kappa_e > 0, \kappa_\mu \in (0, 1), \theta_\mu \in (1, 2), \tau_{\min} \in (0, 1), \kappa_\Sigma > 1, \theta_{\max} \in (\theta(x_0), \infty], \theta_{\min} > 0, \gamma_\theta, \gamma_\varphi \in (0, 1), \delta > 0, \gamma_\alpha \in (0, 1], s_\theta > 1, s_\varphi \geq 1, \eta_\varphi \in (0, 1/2), \kappa_{soc} \in (0, 1)$ and $p^{\max} \in \{0, 1, 2, \dots\}$
2.	<i>Initialization.</i> Initialize $j = 0, k = 0$ , as well as the filter $F_0$ as in (19.22). Compute $\tau_0$ as in (19.15)
3.	<i>Test the convergence for the overall problem.</i> If $E_0(x_k, \lambda_k, z_k) \leq \varepsilon_{TOL}$ , then stop
4.	<i>Test the convergence for the barrier problem.</i> If $E_{\mu_j}(x_k, \lambda_k, z_k) \leq \kappa_e \mu_j$ , then go to step 5
5.	Compute $\mu_{j+1}$ and $\tau_{j+1}$ as in (19.8) and (19.15), respectively
6.	Re-initialize the filter $F_k = \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq \theta^{\max}\}$
7.	If $k = 0$ , then continue with step 4; otherwise, go to step 8
8.	<i>Compute the search direction.</i> Compute $(d_k^x, d_k^\lambda, d_k^z)$ as solution of the system (19.13), where the parameters $\delta_w$ and $\delta_c$ are computed from Algorithm IC for correction of inertia
9.	<i>Backtracking. Initialization of linear search.</i> Set $\alpha_{k,0} = \alpha_k^{\max}$ , where $\alpha_k^{\max}$ is computed as in (19.14a) and set $l = 0$
10.	<i>Backtracking. Compute the new trial point.</i> Set $x_k(\alpha_{k,l}) = x_k + \alpha_{k,l} d_k^x$
11.	<i>Backtracking. Test on acceptability to the filter.</i> If $(\theta(x_k(\alpha_{k,l})), \varphi_{\mu_j}(x_k(\alpha_{k,l}))) \in F_k$ , then reject the trial step and go to step 13
12.	<i>Backtracking. Test on sufficient decrease with respect to the current iterate</i> Case I: $\theta(x_k) \leq \theta^{\min}$ and (19.20) holds. If (19.21) holds, then accept the trial step $x_{k+1} = x_k(\alpha_{k,l})$ and continue with step 19; otherwise, go to step 13 Case II: $\theta(x_k) > \theta^{\min}$ or (19.20) is not satisfied. If (19.19) holds, then accept the trial step $x_{k+1} = x_k(\alpha_{k,l})$ and go to step 19; otherwise, continue with step 13
13.	<i>Backtracking. Initialize the second-order correction.</i> If $l > 0$ or $\theta(x_{k,0}) < \theta(x_k)$ , then skip the second-order correction and go to step 18; otherwise, initialize the second-order correction by setting $p = 1$ and $c_k^{soc}$ as in (19.28). Initialize $\theta_{old}^{soc} = \theta(x_k)$
14.	<i>Backtracking. Compute the second-order correction.</i> Compute $d_k^{x,cor}$ and $d_k^\lambda$ as solutions of the system (19.27). Compute $\alpha_k^{soc}$ from (19.29) and then set $x_k^{soc} = x_k + \alpha_k^{soc} d_k^{x,cor}$
15.	<i>Backtracking. Test on acceptability to the filter (in SOC).</i> If $(\theta(x_k^{soc}), \varphi_{\mu_j}(x_k^{soc})) \in F_k$ , then reject the trial step size and go to step 18
16.	<i>Backtracking. Test on sufficient reduction with respect to the current iterate in SOC.</i> Case I: $\theta(x_k) \leq \theta^{\min}$ and (19.20) holds for $\alpha_{k,0}$ . If (19.21) holds with $x_k(\alpha_{k,l})$ replaced by $x_k^{soc}$ , then the trial step $x_{k+1} = x_k^{soc}$ is accepted and go to step 19; otherwise, continue with step 17 Case II: $\theta(x_k) > \theta^{\min}$ or (19.20) is not satisfied for $\alpha_{k,0}$ . If (19.19) holds for $x_k(\alpha_{k,l})$ replaced by $x_k^{soc}$ , then the trial step $x_{k+1} = x_k^{soc}$ is accepted and go to step 19; otherwise, continue with step 17
17.	<i>Backtracking. The next second-order correction.</i> If $p = p^{\max}$ or $\theta(x_k^{soc}) > \kappa_{soc} \theta_{old}^{soc}$ , then abort second-order correction and go to step 18; otherwise, set $p = p + 1, c_k^{soc} = \alpha_k^{soc} c_k^{soc} + c(x_k^{soc})$ and $\theta_{old}^{soc} = \theta(x_k^{soc})$ . Go back to step 14
18.	<i>Backtracking. Choose a new trial step size.</i> Set $\alpha_{k,l+1} = \alpha_{k,l}/2$ and $l = l + 1$ . If the trial step size becomes too small, i.e., $\alpha_{k,l} < \alpha_k^{\min}$ , where $\alpha_k^{\min}$ is defined as in (19.24), then continue with the feasibility restoration phase, i.e., step 22; otherwise, go back to step 10
19.	<i>Accept the trial point.</i> Set $\alpha_k = \alpha_{k,l}$ (or $\alpha_k = \alpha_k^{soc}$ if the SOC point was accepted in step 16) and update the Lagrange multiplier estimates $\lambda_{k+1}$ and $z_{k+1}$ using (19.16b) and (19.16c) respectively, with $\alpha_k^z$ computed as in (19.14b). If necessary, apply (19.17) to correct $z_{k+1}$
20.	<i>Augment the filter.</i> If (19.20) or (19.21) do not hold for $\alpha_k$ , then augment the filter as in (19.23); otherwise, the filter is unchanged, i.e., $F_{k+1} = F_k$
21.	<i>Continue with the next iteration.</i> Set $k = k + 1$ and go back to step 3.

22. | *Feasibility restoration phase.* Augment the filter as in (19.23) and compute a new iterate  $x_{k+1} > 0$  by decreasing the infeasibility measure  $\theta(x)$ , so that  $x_{k+1}$  is acceptable to the augmented filter, i.e.,  $(\theta(x_{k+1}), \varphi_{\mu_j}(x_{k+1})) \notin F_{k+1}$ . Then continue with the regular iteration in step 21 ◆

At every iteration, at least one trial point will be tested before the algorithm may switch to the feasibility restoration phase in step 22. Also, the condition in step 7 ensures that eventually, at least one step is taken for each decreased value of the barrier parameter.

The algorithm is complex, with a multitude of parameters which specify the optimization conditions. Wächter and Biegler (2006) give the values of the constants used in their implementation as  $\kappa_e = 10$ ,  $\kappa_\mu = 0.2$ ,  $\theta_\mu = 1.5$ ,  $\tau_{\min} = 0.99$ ,  $\gamma_\theta = 10^{-5}$ ,  $\gamma_\varphi = 10^{-5}$ ,  $\delta = 1$ ,  $\gamma_\alpha = 0.05$ ,  $s_\theta = 1.1$ ,  $s_\varphi = 2.3$ ,  $\eta_\varphi = 10^{-4}$ ,  $\kappa_{soc} = 0.99$ ,  $p^{\max} = 4$ , as well as  $\mu_0 = 0.1$ ,  $\theta^{\max} = 10^4 \max \{1, \theta(x_0)\}$  and  $\theta^{\min} = 10^{-4} \max \{1, \theta(x_0)\}$ , where  $x_0$  is the starting point. At the same time, they present some implementation details which lead to the IPOPT algorithm. In the following, we will only present some of the most interesting details in order to complete Algorithm 19.1 for it to be as close as possible to IPOPT described in (Wächter, 2002) or (Wächter & Biegler, 2006).

## 19.2 Implementation Details

### General Lower and Upper Bounds

For simplicity, Algorithm 19.1 is dedicated to solving nonlinear optimization problems (19.2). However, this algorithm can be immediately generalized for solving general problems (19.1). In particular, for the problem (19.1) with simple bounds on variables  $x_L \leq x \leq x_U$ , the barrier problem (19.3) becomes

$$\min_{x \in \mathbb{R}^n} \varphi_{\mu_j}(x) = f(x) - \mu_j \sum_{i \in I_L} \log(x^i - x_L^i) - \mu_j \sum_{i \in I_U} \log(x_U^i - x^i) \quad (19.30a)$$

subject to  
 $c(x) = 0,$  (19.30b)

where  $I_L = \{i : x_L^i \neq -\infty\}$  and  $I_U = \{i : x_U^i \neq +\infty\}$ . At the same time, for the multipliers  $z$  the simple bounds  $z_L^i$  and  $z_U^i$  are introduced for all the finite lower and upper bounds, and the primal-dual Hessian of the barrier terms  $\Sigma_k$  is defined as the sum of the matrices  $\Sigma_k^L = \text{diag}(\sigma_{k,1}^L, \dots, \sigma_{k,n}^L)$  and  $\Sigma_k^U = \text{diag}(\sigma_{k,1}^U, \dots, \sigma_{k,n}^U)$ , where

$$\begin{aligned} \sigma_{k,i}^L &= \begin{cases} z_{L,k}^i / (x_k^i - x_L^i), & \text{if } i \in I_L, \\ 0, & \text{otherwise,} \end{cases} \\ \sigma_{k,i}^U &= \begin{cases} z_{U,k}^i / (x_U^i - x_k^i), & \text{if } i \in I_U, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Moreover, for  $i \notin I_L$ , define  $z_{L,k}^i = 0$  and for  $i \notin I_U$ , define  $z_{U,k}^i = 0$ .

## Initialization

Since Algorithm 19.1 requires that the iterates should strictly satisfy the bounds constraints (19.1c), it follows that it is often necessary to modify the user-provided initial point so that it is sufficiently away from the boundary. For this purpose, each component  $i$  of the initial point which has only one (let us say, a lower) bound is modified by

$$x_0^i = \max \{x_0^i, x_L^i + \kappa_1 \max \{1, |x_L^i|\}\},$$

where  $\kappa_1 > 0$  is a constant. Similar modifications can be made for the only upper-bounded variables. The initial value of a variable  $x^i$  bounded on two sides is projected into the interval  $[x_L^i + p_L^i, x_U^i - p_U^i]$ , where the perturbations  $p_L^i$  and  $p_U^i$  are computed as

$$\begin{aligned} p_L^i &\triangleq \min \{\kappa_1 \max \{1, |x_L^i|\}, \kappa_2 (x_U^i - x_L^i)\}, \\ p_U^i &\triangleq \min \{\kappa_1 \max \{1, |x_U^i|\}, \kappa_2 (x_U^i - x_L^i)\}, \end{aligned}$$

where  $\kappa_2 \in (0, 1/2)$  (For example:  $\kappa_1 = \kappa_2 = 10^{-2}$ .)

The dual variables corresponding to the bound constraints are initialized to one, component-wise. Finally, using the possibly modified initial point  $x_0$  and the initial bound multipliers, the multipliers  $\lambda_0$  for the equality constraints are obtained as least-square solutions for the dual infeasibility (19.4a), i.e., by solving the linear system

$$\begin{bmatrix} I & J(x_0) \\ J(x_0)^T & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda_0 \end{bmatrix} = - \begin{bmatrix} \nabla f(x_0) - z_{L,0} + z_{U,0} \\ 0 \end{bmatrix}, \quad (19.31)$$

where  $w$  are temporary variables discarded after this computation. However, if  $\lambda_0$  obtained in this way is too large, i.e.,  $\|\lambda_0\|_\infty > \lambda_{\max}$ , (with  $\lambda_{\max} = 10^3$  for example), then the least-square estimate  $\lambda_0$  is discarded and set  $\lambda_0 = 0$ . ( $J(x_0) = \nabla c(x_0)$ ).

## Handling Unbounded Solution Sets

In some cases, the optimal point set for (19.1) does not consist of isolated points, but contains an unbounded connected component. Then the objective function of the corresponding barrier problem (19.30) for a fixed value of the barrier parameter  $\mu_j$  is unbounded below over the feasible set since a log-barrier term converges to  $-\infty$  as its argument goes to infinity. Therefore, the method for solving the barrier problem might fail to converge, even though the original problem is well-posed. To prevent this behavior, linear damping terms for all the variables with exactly one finite bound are added to the barrier objective function (19.30a), which then becomes

$$\begin{aligned} \varphi_{\mu_j}(x) &= f(x) - \mu_j \sum_{i \in I_L} \log(x^i - x_L^i) - \mu_j \sum_{i \in I_U} \log(x_U^i - x^i) \\ &\quad + \kappa_d \mu_j \sum_{i \in I_L \setminus I_U} (x^i - x_L^i) + \kappa_d \mu_j \sum_{i \in I_U \setminus I_L} (x_U^i - x^i), \end{aligned}$$

where  $\kappa_d > 0$  is a constant independent of  $\mu_j$  ( $\kappa_d = 10^{-4}$  for example). In this way, the divergence of the variables that have only one bound is penalized. On the other hand, the effect of the damping term is reduced when  $\mu_j$  decreases.

## Inertia Correction

In order to be able to compute the search direction, we need to ensure that the matrix of the system (19.11) is nonsingular. In addition, the filter line-search method requires the matrix  $W_k + \Sigma_k$  projected onto the null space of the constraint Jacobian  $J_k^T$  being positive definite. But these conditions are satisfied if the matrix of the system (19.11) has the inertia equal to  $(n, m, 0)$ , i.e., if this matrix has exactly  $n$  positive,  $m$  negative, and *no* zero eigenvalues. Therefore, if the inertia of this matrix is not  $(n, m, 0)$ , then the linear system (19.13) is re-solved with different trial values for the scalars  $\delta_w$ ,  $\delta_c \geq 0$  until the inertia is as desired. Observe that the desired inertia is obtained if  $\delta_w$  is sufficiently large and the constraint Jacobian  $J_k^T$  has full rank. If  $J_k^T$  is rank-deficient, the matrix of the system (19.13) is singular as long as  $\delta_c$  is zero. Therefore, a positive value for  $\delta_c$  and a sufficiently large value of  $\delta_w$  ensures the correct eigenvalue signatures for the matrix from (19.13). However, in practice, the matrix of the system (19.13) can become so ill-conditioned that the factorization cannot be successfully performed, even with very large values of  $\delta_w$  and some positive values for  $\delta_c$ . In this case, Algorithm 19.1 switches directly to the feasibility restoration phase in step 22. These observations motivate the following heuristics for choosing the parameters  $\delta_c$  and  $\delta_w$  (Wächter & Biegler, 2005a).

### Algorithm 19.2 *Inertia correction algorithm*

- |    |  |
|----|--|
| 1. | Choose the constants $0 < \bar{\delta}_w^{\min} < \bar{\delta}_w^0 < \bar{\delta}_w^{\max}$ , $\bar{\delta}_c > 0$ , $\kappa_c \geq 0$ , $0 < \kappa_w^- < 1 < \kappa_w^+ < \bar{\kappa}_w^+$ . Initialize $\delta_w^{last} = 0$   |
| 2. | Attempt to factorize the unmodified matrix of the system (19.13) with $\delta_w = \delta_c = 0$ . If the matrix is nonsingular and its inertia is $(n, m, 0)$ , then use the resulting search direction in the line-search obtained by solving the system (19.13); otherwise, continue with step 3 |
| 3. | If the matrix from (19.13) has zero eigenvalues, then set $\delta_c = \bar{\delta}_c \mu^{\kappa_c}$ ; otherwise, set $\delta_c = 0$   |
| 4. | If $\delta_w^{last} = 0$ , then set $\delta_w = \bar{\delta}_w^0$ ; otherwise, set $\delta_w = \max\left\{\bar{\delta}_w^{\min}, \kappa_w^- \delta_w^{last}\right\}$   |
| 5. | Attempt to factorize the matrix from (19.13) modified with these values of $\delta_c$ and $\delta_w$ . If inertia is equal to $(n, m, 0)$ , then set $\delta_w^{last} = \delta_w$ and use the resulting search direction in the line-search, otherwise, continue with step 6                       |
| 6. | If $\delta_w^{last} = 0$ , then set $\delta_w = \bar{\kappa}_w^+ \delta_w$ ; otherwise, set $\delta_w = \kappa_w^+ \delta_w$   |
| 7. | If $\delta_w > \bar{\delta}_w^{\max}$ , then abort the search direction computation, skip the backtracking line-search and switch directly to the restoration phase in step 22 of Algorithm 19.1. Otherwise, continue with step 5  |

The above procedure was established by Wächter and Biegler (2006). In step 2 it first checks if the unmodified matrix has the desired inertia so that the pure Newton search direction is used whenever possible. If this is the case, then apply the Newton method for the search direction computation; otherwise, increase  $\delta_w$ . The first trial value is based on  $\delta_w^{last}$ , which stores the perturbation value from the last modification of the matrix from (19.13). In this way, the smallest perturbation is found necessary to avoid the factorization in step 5 of the algorithm for too small values of  $\delta_w$ . The reason for using a much larger factor  $\bar{\kappa}_w^+$  in step 6 of the algorithm for the first necessary correction rather than for the correction in later iterations is that we want to avoid a high number of trial factorizations when the scale of the problem and the order of the magnitude for a successful correction is not known yet. The selection of a nonzero value for  $\delta_c$  in step 3 is based on the assumption that the singularity is caused by the rank-deficient constraint Jacobian. Note that the nonzero value for  $\delta_c$  in step 3 converges to zero as  $\mu \rightarrow 0$  if  $\kappa_c > 0$ , so that the perturbation  $\delta_c$  is smaller when a solution of the problem is approached.

## Automatic Scaling of the Problem

The Newton steps for the primal-dual system (19.11) for solving the nonlinear system (19.4) are invariant to scaling the variables, the objective, and the constraints functions, i.e., to replacing  $x, f$  and  $c$  by  $\tilde{x} = D_x x$ ,  $\tilde{f}(x) = d_f f(x)$  and  $\tilde{c}(x) = D_c c(x)$  for some  $d_f > 0$  and for the positive definite diagonal matrices  $D_x = \text{diag}(d_x^1, \dots, d_x^n)$  and  $D_c = \text{diag}(d_c^1, \dots, d_c^m)$ . However, the optimization algorithm with its initialization procedures, globalization strategy, and stopping criteria usually behaves very differently for different scaling factors, particularly if the scaling factors are very large or very small.

The idea of scaling in IPOPT is that changing a variable by a given amount has a comparable effect on any function which depends on this variable, or in other words, the nonzero elements of the function gradients are of the same order of magnitude.

Another scaling procedure available in IPOPT as an option consists in applying an equilibration algorithm to the first derivative matrix

$$M_0 = \begin{bmatrix} \nabla_x c(x_0)^T \\ \nabla_x f(x_0)^T \end{bmatrix}$$

to obtain the scaling matrices  $D_x$  and  $D_{cf} = \text{diag}(D_c, d_f)$  so that the nonzero elements in  $D_{cf} M_0 D_x^{-1}$  are of order one. (The algorithm for the equilibration of the matrices is the one implemented in the subroutines MC19 and MC29 from the Harwell library (AEA Technology, 2002)). Similarly, another procedure for scaling computes the scaling factors so that the matrix

$$\begin{bmatrix} D_x^{-1} & 0 \\ 0 & D_c \end{bmatrix} \begin{bmatrix} \nabla_{xx}^2 f(x_0) & \nabla_x c(x_0) \\ \nabla_x c(x_0)^T & 0 \end{bmatrix} \begin{bmatrix} D_x^{-1} & 0 \\ 0 & D_c \end{bmatrix}$$

has nonzero entries close to one. These procedures are available to users as options in IPOPT. Even if these strategies seem to work well in certain instances, the performances of the algorithm are not conclusive with scaling.

In IPOPT, Wächter and Biegler (2006) apply an automatic scaling procedure. Given a threshold value  $g_{\max}$ , for example  $g_{\max} = 100$ , then the scaling factors are chosen as

$$d_f = \min \left\{ 1, g_{\max} / \| \nabla_x f(x_0) \|_{\infty} \right\},$$

$$d_c^j = \min \left\{ 1, g_{\max} / \| \nabla_x c^j(x_0) \|_{\infty} \right\}, \quad j = 1, \dots, m,$$

and  $D_x = I$ . Observe that this procedure will never multiply a function by a number larger than one and that all the gradient components in the scaled problem are at most of size  $g_{\max}$  at the starting point.

The scaling factors are computed only at the beginning of the optimization at  $x_0$ .

## Feasibility Restoration Phase

A very important component of the filter line-search algorithm is the feasibility restoration phase (see step 22 of Algorithm 19.1). The task of the restoration phase is to compute a new iteration acceptable to the augmented filter  $F_{k+1}$  by decreasing the infeasibility whenever the regular backtracking line-search procedure cannot make sufficient progress and when the step size becomes too small (see step 18 of Algorithm 19.1). Besides, when the inertia is corrected as in Algorithm 19.2, the method switches to the restoration phase whenever the linear system (19.13) is very ill-conditioned and

cannot be successfully factorized, despite the modifications of the matrix from (19.13). The feasibility restoration phase has another significant purpose, namely, to detect the local infeasibility. If the problem is infeasible, then the algorithm is not able to generate sufficient progress in the regular backtracking line-search procedure and reverts to the restoration phase. In Wächter and Biegler (2001), the global convergence of the filter line-search method is proved under the hypothesis that in the neighborhood of feasible points the gradients of the active constraints are linearly independent. Consequently, the algorithm does not switch to the feasibility restoration phase at feasible points. However, this assumption might be violated in practice and the restoration phase might be called at a point with a very small value of  $\theta$ . In IPOPT, two procedures for feasibility restorations are used. (In order to avoid confusion, the overbars are used to denote quantities referring to the restoration phase, and the subscript  $t$  is used for the restoration phase iteration counter).

**Minimization of the constraint violation** This is the first algorithm of the feasibility restoration phase. The purpose of this method is to return a new iterate  $x_{k+1} > 0$  with  $(\theta(x_{k+1}), \varphi_{\mu_j}(x_{k+1})) \notin F_{k+1}$  for step 22 of Algorithm 19.1 or to converge to a nonzero minimizer (or at least to a stationary point) of some norm of the constraint violation. The restoration phase algorithm applies the primal-dual interior-point filter line-search algorithm to a smooth reformulation of the optimization problem

$$\min_{\bar{x} \in \mathbb{R}^n} \|c(\bar{x})\|_1 + \frac{\xi}{2} \|D_R(\bar{x} - \bar{x}_R)\|_2^2 \quad (19.32a)$$

subject to

$$\bar{x} \geq 0. \quad (19.32b)$$

In (19.32a), a term is included in the objective function that penalizes the deviation from a reference point  $\bar{x}_R$ , where  $\xi > 0$  is a weighting parameter and the scaling matrix  $D_R$  is defined by

$$D_R = \text{diag}(\min\{1, 1/|\bar{x}_R^1|\}, \dots, \min\{1, 1/|\bar{x}_R^n|\})$$

The reference point  $\bar{x}_R$  is chosen to be the iterate  $x_k$  at which the restoration phase is called in step 22 of Algorithm 19.1. In this way, we seek to decrease the constraint violation and try to avoid a large deviation from  $\bar{x}_R$ , which determines undesired significant increase in the barrier objective function  $\varphi_{\mu_j}$ .

A smooth reformulation of the problem (19.32) is obtained by introducing the nonnegative variables  $\bar{p}, \bar{n} \in \mathbb{R}^m$  that capture the positive and negative parts of the constraints

$$\min_{\bar{x} \in \mathbb{R}^n, \bar{p}, \bar{n} \in \mathbb{R}^m} \sum_{i=1}^m (\bar{p}^i + \bar{n}^i) + \frac{\xi}{2} \|D_R(\bar{x} - \bar{x}_R)\|_2^2 \quad (19.33a)$$

subject to

$$c(\bar{x}) - \bar{p} + \bar{n} = 0, \quad (19.33b)$$

$$\bar{x}, \bar{p}, \bar{n} \geq 0. \quad (19.33c)$$

We can see that this optimization problem is of the form given in (19.2). Therefore, the filter line-search interior-point algorithm can be applied to solve a sequence of barrier problems of the following form:

$$\begin{aligned} \min_{\bar{x} \in \mathbb{R}^n, \bar{p}, \bar{n} \in \mathbb{R}^m} \quad & \rho \sum_{i=1}^m (\bar{p}^i + \bar{n}^i) + \frac{\xi}{2} \|D_R(\bar{x} - \bar{x}_R)\|_2^2 \\ & - \bar{\mu} \sum_{i=1}^n \log(\bar{x}^i) - \bar{\mu} \sum_{i=1}^m \log(\bar{p}^i) - \bar{\mu} \sum_{i=1}^m \log(\bar{n}^i) \end{aligned} \quad (19.34a)$$

subject to

$$c(\bar{x}) - \bar{p} + \bar{n} = 0. \quad (19.34b)$$

Observe that an additional scaling parameter  $\rho > 0$  has been introduced. This is to allow a relative scaling of the overall objective function (19.34a) with respect to the constraints (19.34b). Moreover, if the regularization parameter  $\xi > 0$  is chosen sufficiently small, the optimization problem (19.33) is the exact penalty formulation of the problem: “*find the feasible point that is closest to the reference point  $\bar{x}_R$  in a weighted norm*”:

$$\min_{\bar{x} \in \mathbb{R}^n} \|D_R(\bar{x} - \bar{x}_R)\|_2^2$$

subject to

$$c(\bar{x}) = 0, \quad \bar{x} \geq 0.$$

In addition to the original variables  $\bar{x}$ , the barrier problem (19.34) contains the variables  $\bar{p}$  and  $\bar{n}$ . The corresponding primal-dual equations (similar to (19.4)) include their dual variables, say  $\bar{z}_p$  and  $\bar{z}_n$ . The search directions for the line-search are obtained by linearization of these equations and can be written as

$$\begin{bmatrix} \bar{W} + \xi D_R^2 + \bar{\Sigma} & \nabla c(\bar{x}) \\ \nabla c(\bar{x})^T & -\bar{\Sigma}_p^{-1} - \bar{\Sigma}_n^{-1} \end{bmatrix} \begin{bmatrix} \bar{d}^x \\ \bar{d}^\lambda \end{bmatrix} = - \begin{bmatrix} \xi D_R^2(\bar{x} - \bar{x}_R) + \nabla c(\bar{x})\bar{\lambda} - \bar{\mu}\bar{X}^{-1}e \\ c(\bar{x}) - \bar{p} + \bar{n} + \rho\bar{Z}_p^{-1}(\bar{\mu}e - \bar{p}) + \rho\bar{Z}_n^{-1}(\bar{\mu}e - \bar{n}) \end{bmatrix}, \quad (19.35)$$

where  $\bar{W} = \sum_{i=1}^m \bar{\lambda}^i \nabla_{xx}^2 c(\bar{x})$ ,  $\bar{\Sigma} = \bar{X}^{-1}\bar{Z}$ ,  $\bar{\Sigma}_p = \bar{P}^{-1}\bar{Z}_p$  and  $\bar{\Sigma}_n = \bar{N}^{-1}\bar{Z}_n$ . Subsequently,  $\bar{d}^p$ ,  $\bar{d}^n$ ,  $\bar{d}^{z_p}$  and  $\bar{d}^{z_n}$  are obtained from

$$\begin{aligned} \bar{d}^p &= \bar{Z}_p^{-1} \left( \bar{\mu}e + \bar{P} \left( \bar{\lambda} + \bar{d}^\lambda \right) - \rho\bar{p} \right), \quad \bar{d}^{z_p} = \bar{\mu}\bar{P}^{-1}e - \bar{z}^p - \bar{\Sigma}_p\bar{d}^p, \\ \bar{d}^n &= \bar{Z}_n^{-1} \left( \bar{\mu}e + \bar{N} \left( \bar{\lambda} + \bar{d}^\lambda \right) - \rho\bar{n} \right), \quad \bar{d}^{z_n} = \bar{\mu}\bar{N}^{-1}e - \bar{z}^n - \bar{\Sigma}_n\bar{d}^n, \\ \bar{d}^z &= \bar{\mu}\bar{X}^{-1}e - \bar{z} - \bar{\Sigma}\bar{d}^x. \end{aligned}$$

Observe that the structure of the nonzero elements of the linear system (19.35) is identical to the one in (19.13). Therefore, for solving (19.35) the same symbolic factorization may be used.

The filter line-search method applied to the problem (19.34) might itself revert to a restoration phase. If this occurs, then the optimal solution of (19.34) is computed *for a fixed value of  $\bar{x}$* , namely, the current iterate  $\bar{x}_t$ , and this solution is used as the “result” of the restoration phase. In this case, since (19.34) becomes separable, this can be done by solving a quadratic equation for each  $(\bar{p}^i, \bar{n}^i)$ , i.e.,

$$\begin{aligned}\bar{n}^i &= \frac{\bar{\mu} - \rho c^i(\bar{x})}{2\rho} + \sqrt{\left(\frac{\bar{\mu} - \rho c^i(\bar{x})}{2\rho}\right)^2 + \frac{\bar{\mu} c^i(\bar{x})}{2\rho}}. \\ \bar{p}^i &= c^i(\bar{x}) + \bar{n}^i.\end{aligned}$$

**Reduction of the KKT error** This method tries to achieve reduction in the norm of the primal-dual equations by using the regular iteration steps. If in step 22 of Algorithm 19.1 the feasibility restoration phase is invoked, then the algorithm does not immediately revert to the feasibility restoration phase. Instead, the reduction in the norm of the primal-dual equations is tried. To describe this procedure, let  $F_\mu(x, \lambda, z)$  denote the left-hand side of the nonlinear system (19.4). Then, given a constant  $\kappa_F \in (0, 1)$ , for example  $\kappa_F = 0.999$ , the following algorithm reduces the error in the KKT system.

**Algorithm 19.3 KKT error reduction algorithm**

1. Initialize the feasibility restoration phase with the parameter  $t = 0$  and choose the initial point  $(\bar{x}_0, \bar{\lambda}_0, \bar{z}_0) = (x_k, \lambda_k, z_k)$
2. Compute a search direction  $(\bar{d}_t^x, \bar{d}_t^\lambda, \bar{d}_t^z)$  using the system (19.11) and (19.12)
3. Apply the fraction-to-the-boundary rule  

$$\bar{\beta}_t \triangleq \max \{ \beta \in (0, 1) : \bar{x}_t + \beta \bar{d}_t^x \geq (1 - \tau_j) \bar{x}_t, \bar{z}_t + \beta \bar{d}_t^z \geq (1 - \tau_j) \bar{z}_t \}$$
4. Test whether  

$$\|F_\mu(\bar{x}_{t+1}, \bar{\lambda}_{t+1}, \bar{z}_{t+1})\|_1 \leq \kappa_F \|F_\mu(\bar{x}_t, \bar{\lambda}_t, \bar{z}_t)\|_1,$$
where  $(\bar{x}_{t+1}, \bar{\lambda}_{t+1}, \bar{z}_{t+1}) = (\bar{x}_t, \bar{\lambda}_t, \bar{z}_t) + \bar{\beta}_t (\bar{d}_t^x, \bar{d}_t^\lambda, \bar{d}_t^z)$ .  
If this condition of reduction is not satisfied, then discard the trial point and switch to the procedure for minimizing the constraint violation (described above) by using  $\bar{x}_t$  as a reference point. Otherwise, continue with step 5
5. If  $(\theta(\bar{x}_{t+1}), \varphi_{\mu_t}(\bar{x}_{t+1})) \notin F_{k+1}$ , then continue the regular interior-point Algorithm 19.1 from the point  $(x_{t+1}, \{\lambda\}_{t+1}, z_{t+1}) = (\bar{x}_{t+1}, \bar{\lambda}_{t+1}, \bar{z}_{t+1})$ . Otherwise, set  $t = t + 1$  and go to step 2

In a neighborhood of a strict local solution satisfying the second-order sufficient optimality conditions (see Theorem 11.6) for the barrier problem, the projection of the Hessian  $W_t + \Sigma_t$  onto the null space of the constraint Jacobian  $\nabla c(\bar{x}_t)^T$  is positive definite. Therefore, no modification of the matrix of the system (19.13) as described in the inertia correction procedure is applied. As a consequence, the search directions computed by solving the system (19.11) and the relations (19.12) are the Newton steps applied to the nonlinear system (19.4). Hence, the above algorithm will accept those steps and will quickly converge toward this solution, provided it is started sufficiently close. The performance and robustness of IPOPT on larger models heavily rely on the solver for sparse symmetric indefinite linear systems.

## Numerical Study—IPOPT: Solving Applications from the LACOP Collection

The performances of IPOPT for solving the applications from the LACOP collection described in Appendix C are illustrated in Tables 19.1 and 19.2.

Appendix C includes the application L16 – *Fed-batch fermenter for penicillin production (PENICI)*. Table 19.3 shows the performance of IPOPT for solving this application.

**Table 19.1** Performances of IPOPT for solving 12 applications from the LACOP collection. Small-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#nf	#ng	#nh	cpu	vfo
ELCH	10	3	0	10	11	11	10	0.143	-47.761090
ALKI	10	3	8	13	14	14	13	0.011	-1768.8069
PREC	8	0	6	16	17	17	16	0.081	3.95116343
PPSE	9	6	0	11	12	12	11	0.009	5055.01180
MSP3	13	0	15	19	20	20	19	0.044	97.5875091
MSP5	16	0	21	28	29	29	28	0.045	174.786994
POOL	34	20	0	13	14	14	13	0.026	2569.7999
TRAFO	6	0	2	9	10	10	9	0.009	135.07596
LATHE	10	1	14	27	28	28	27	0.020	-4430.0875
DES	150	50	0	88	137	89	88	0.479	1055.18231
CSTC	303	200	0	5	6	6	5	0.035	3.4800741
DIFF	396	324	0	1	2	2	1	0.016	0

**Table 19.2** Performances of IPOPT for solving 6 applications from the LACOP collection. Large-scale nonlinear optimization applications

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#nf	#ng	#nh	cpu	vfo
HANG	2002	1001	0	6	7	7	6	0.180	5.0685100
	4002	2001	0	7	8	8	7	0.310	5.0684889
FLOW	1182	754	0	9	10	10	9	0.272	0.226e-10
FLOWO	1556	1005	0	19	20	20	19	0.597	0.689e-7
POL	4004	3000	0	66	67	67	66	3.845	14.216290
	6004	4500	0	54	55	55	54	3.741	14.198908
	8004	6000	0	67	68	68	67	6.337	14.190847
	10004	7500	0	179	180	180	179	21.750	13.981795
CAT	3003	2000	0	15	16	16	15	0.252	-0.048055
	6003	4000	0	15	16	16	15	0.639	-0.048055
	9003	6000	0	15	16	16	15	0.909	-0.048055
CONT	2505	2000	0	14	15	15	14	0.412	1.0132389
	5005	4000	0	13	14	14	13	0.706	1.0059224
	7505	6000	0	15	16	16	15	0.962	1.0045614
	10005	8000	0	15	16	16	15	1.264	1.0040718

**Table 19.3** Performances of IPOPT for solving the PENICI application

	<i>n</i>	<i>me</i>	<i>mc</i>	#it	#nf	#ng	#nh	cpu	vfo
PENICI	707	602	0	519	765	512	519	6.81	113.98986
	1407	1202	0	423	623	424	423	9.93	113.98942

In the tables above, we have  $n$  = the number of variables,  $me$  = the number of equality constraints,  $mc$  = the number of inequality constraints, #it = the number of iterations to obtain a solution, #nf = the number of evaluations of the functions defining the problem, #ng = the number of evaluations of the gradients of the functions defining the problem, #nh = the number of evaluations of the Hessian, cpu = the CPU computing time for obtaining a solution of the problem (seconds), vfo = the value of the objective function at the optimal point

**Table 19.4** Performances of the IPOPT algorithm. Small-scale nonlinear optimization applications

	#itt	#nft	#ngt	#nht	cput
IPOPT	240	300	252	240	0.918

**Table 19.5** Performances of the IPOPT algorithm. Large-scale nonlinear optimization applications

	#itt	#nft	#ngt	#nht	cput
IPOPT	509	524	524	509	42.176

Table 19.4 presents the total number of iterations (#itt), the total number of evaluations of the function defining the problem (#nft), the total number of evaluations of the gradients of the functions defining the problem (#ngt), the total number of evaluations of the Hessian (#nht), and the total CPU computing time to obtain a solution (cput) for solving 12 small-scale nonlinear optimization applications (see Table 19.1) considered in this numerical study.

Table 19.5 shows the performances of IPOPT for solving 15 large-scale nonlinear optimization applications (see Table 19.2) considered in this numerical study.

From Table 17.13, we can see that for solving 15 large-scale applications from the LACOP collection subject to the CPU computing time, KNITRO with option 0 needs 17.81 seconds, being top performer versus IPOPT which needs 42.176 seconds.

There is a great difference between KNITRO and IPOPT. In both variants KINTO/ACTIVE and KNITRO/INTERIOR, the sequential linear or sequential quadratic programming in different computational structures are used. The KNITRO/ACTIVE uses a new active-set method based on the sequential linear-quadratic programming (SLQP) and the projected conjugate gradient iteration. On the other hand, in KNITRO/INTERIOR two procedures for computing the steps are used. In the version INTERIOR-DIRECT, the algorithm attempts to compute a new iterate by solving the primal-dual KKT system by using the direct linear algebra. In the version INTERIOR-CG, each step is computed by using a projected conjugate gradient iteration. It factors a projection matrix and uses the conjugate gradient method to approximately minimize a quadratic model of the barrier problem. The stepsizes are determined by minimizing merit functions. All the algorithms included in KNITRO have a very strong theoretical justification.

IPOPT, on the other hand, combines the primal-dual interior-point algorithms with the filter line-search, where the search direction is computed as solution of the first-order optimality conditions, and the stepsize is computed by the filter technique. The algorithm depends by a multitude of parameters which have a great impact on its performance. Although both the merit functions and filters are important globalization techniques of the interior-point methods, currently it is unclear whether filter techniques are preferable to the merit functions.

## Notes and References

This Chapter has described the interior-point method with filter line-search. The representative for this method is IPOPT. The content of this chapter is based on the papers of Wächter and Biegler (2005a, b, 2006). This is a line-search filter interior-point algorithm with two loops. The outer loop approximately minimizes a sequence of nonlinearly equality constrained barrier subproblems for a decreasing sequence of barrier parameters. The inner loop uses a line-search filter sequential quadratic programming problem to approximately solve each barrier subproblem. The global

convergence of each barrier subproblem is enforced through a line-search filter method (Fletcher & Leyffer, 1999, 2002). The filter is reset after each barrier parameter update. The steps are computed by solving a primal-dual system corresponding to the KKT conditions of the barrier problem, using the subroutines from the HSL mathematical software library collection of Fortran codes for large scientific computation (MA77, MA86, MA97). The algorithm controls the inertia of the primal-dual system by adding  $\delta I$  to the Hessian of the Lagrangian, where  $\delta$  is a positive parameter, thus ensuring the descent properties. The inner iterations include the second-order correction steps and some mechanisms for switching to a feasibility restoration if the stepsize becomes too small. To approximate the Hessian of the Lagrangian, IPOPT has an option for using the limited memory BFGS updates



# Direct Methods for Constrained Optimization

20

As we have already seen in Chap. 9, the direct methods for unconstrained optimization do not use derivative information. From the multitude of these methods, only the NELMEAD by Nelder and Mead (1965), NEWUOA by Powell (2004, 2006), and DEEPS by Andrei (2021a) have been discussed. They are suitable for solving unconstrained optimization problems with a small number of variables (let us say up to 100).

This chapter is dedicated to presenting the direct methods for solving constrained optimization problems and applications. They use only the function values that define the problem along a sequence of points  $\{x_k\}$  that is expected to be convergent to a point in which the constraints are satisfied, and the objective value is less than its value in the initial point. The direct methods are suitable in situations in which the smoothness of the functions of the problem is not satisfied or when the values of the functions are obtained by simulation of some complex mathematical structures.

The concept *optimization based on simulation* is applied in the analysis, design, and control of complex physical systems by using methods of optimization. Thus, by computational simulation, some quantities necessary for optimization are generated. Every simulation needs the execution of a finite sequence of computing programs involving, for example, the generation of discretizations and solving some systems of differential equations. The results of simulation must be processed in order to obtain the final values of the minimizing function and of the constraints. Observe that these computational complications are not suitable for using optimization methods based on derivative information. Even if the automatic differentiation has been successfully applied for solving some optimization problems, it is not appropriate for the optimization based on simulation. The concept of *nonsmooth optimization* refers to the problems that involve, for example, functions of the type  $|x|$  or more general nondifferentiable functions. On the other hand, there are some optimization problems that involve some computational schemes of the type *if-then-else* which make the corresponding problems nonsmooth.

These sorts of optimization problems are treated by the direct methods. The crucial aspect of the direct methods is that they are based on the hypothesis that in the set of trying solutions there is an order relation which allows us to specify that a trying solution is better than another one and at every iteration, there is a finite number of new possible solutions determined by the searching method. This aspect emphasizes the important difference between the direct methods of optimization and the methods based on derivative information (gradient, Hessian) in which there is a continuum of trying solutions.

The purpose of this chapter is to detail two different direct methods for constrained optimization together with their numerical performances for solving real applications of optimization from the LACOP collection. The first method is COBYLA—Constrained Optimization BY Linear Approximation of Powell (1993). At every iteration of this method, the objective function and constraints are linearly approximated by interpolation in vertices of a simplex structure. The corresponding linear programming problem completed with a constraint of trust-region type is solved, thus obtaining a new approximation to the solution of the original problem. The second method is an extension of the successive quadratic penalty method which does not involve derivative information. This method approximately minimizes a sequence of merit functions, in which the penalization of the violation of the constraints is progressively enlarged (Liuzzi, Lucidi, & Sciandrone, 2010).

## 20.1 COBYLA Algorithm

The COBYLA algorithm is an extension of the Nelder-Mead direct method for solving constrained optimization problems (see Chap. 9). For solving the unconstrained optimization problem  $\min f(x)$ , where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , the Nelder-Mead method computes at a given iteration, the minimizing function values in  $n + 1$  points of a simplex  $\{x^j : j = 0, 1, \dots, n\}$  which satisfy the *nondegeneracy condition*. Let  $x^l$  be the vertex of the simplex for which  $f(x^l) = \max \{f(x^j) : j = 0, 1, \dots, n\}$ . Since  $x^l$  is the vertex of the simplex in which the minimizing function has the highest value, the algorithm replaces  $x^l$  with the point

$$x_{new}^l = -\theta x^l + \frac{(1 + \theta)}{n} \sum_{j=0, j \neq l}^n x^j, \quad (20.1)$$

where the *reflection coefficient*  $\theta$  is a constant in the open interval  $(0, 1)$ . Obviously, if  $f(x_{new}^l)$  is the smallest value of the minimizing function, then the reflection coefficient can be enlarged. Formula (20.1) defines the point  $x_{new}^l$  by extrapolation along a line connecting  $x^l$  with the centroid of the rest of the vertices of the simplex. Hence, if  $f$  is a linear function, then  $f(x_{new}^l)$  is smaller than the average of the values  $\{f(x^j) : j = 0, 1, \dots, n, j \neq l\}$ , so that the iteration is a success subject to the reduction of the function's values. On the other hand, if the iteration is not a success, then the algorithm shrinks the current simplex defined by the vertices  $\{x^j : j = 0, 1, \dots, n\}$ . This idea of the Nelder-Mead algorithm is adapted for solving the constrained optimization problems

$$\begin{aligned} & \min f(x) \\ & \text{subject to} \\ & c_i(x) \geq 0, \quad i = 1, \dots, m, \end{aligned} \quad (20.2)$$

where it is supposed that the functions  $f$  and  $c_i$ ,  $i = 1, \dots, m$ , can be evaluated in any point  $x$ . For solving this problem, Powell (1993) suggests the following algorithm based on the evaluation of the functions defining the problem in the vertices  $\{x^j : j = 0, 1, \dots, n\}$  of a nondegenerate simplex in  $\mathbb{R}^n$ . In this case, there are the linear functions  $\hat{f}$  and  $\hat{c}_i$ ,  $i = 1, \dots, m$ , that interpolate  $f$  and  $c_i$ ,  $i = 1, \dots, m$ , in the vertices of the current simplex with which (20.2) can be approximated by the following linear programming problem:

$$\begin{aligned}
& \min \hat{f}(x) \\
& \text{subject to} \\
& \quad \hat{c}_i(x) \geq 0, \quad i = 1, \dots, m.
\end{aligned} \tag{20.3}$$

Observe that the functions  $\hat{f}$  and  $\hat{c}_i$ ,  $i = 1, \dots, m$ , are uniquely determined. The modification of the variables is under a trust-region constraint which closes the feasible domain of the linear program (20.3). The radius of the trust-region determines the stepsizes. For comparing two solutions, the algorithm uses the following merit function:

$$\Phi(x) = f(x) + \mu [\max \{-c_i(x) : i = 1, \dots, m\}]_+, \tag{20.4}$$

where  $\mu$  is a parameter and  $v_+ = \max \{0, v\}$ . Observe that if  $x \in \mathbb{R}^n$  is feasible, then  $\Phi(x) = f(x)$ . In this context, the point  $x \in \mathbb{R}^n$  is better than the point  $y \in \mathbb{R}^n$  if and only if  $\Phi(x) < \Phi(y)$ . Observe that the merit function (20.4) includes the constraints of the problem penalized by  $\mu$ . With this, the COBYLA algorithm may be described as follows.

Let us suppose that the vertices  $\{x^j : j = 0, 1, \dots, n\}$ , the radius  $\rho$  of the trust-region and a value of the penalty parameter  $\mu$  from the merit function (20.4) are known. The vertices of the current simplex are ordered so that  $x^0$  is the best vertex, that is

$$\Phi(x^0) \leq \Phi(x^j), \quad j = 1, \dots, n. \tag{20.5}$$

Let  $x^*$  be the vector of the variables of the linear programming problem (20.3). Then the trust-region condition is

$$\|x^* - x^0\|_2 \leq \rho. \tag{20.6}$$

With this, let  $x^*$  be the minimum of the linear approximation  $\hat{f}(x)$  subject to the linear constraints from (20.3) and the inequality (20.6). It is quite possible for the constraints from (20.3) to be incompatible with (20.6). In this case, define  $x^*$  as solution of the minimization of the largest violation of the constraints subject to (20.6). The computation of  $x^*$  is made by continuously increasing  $\rho$  from the zero value to a current value. The sequence of points  $x^*$  generated for different values of  $\rho$  defines a piece-wise linear continuous trajectory.

Let us describe the procedure for the modification of the penalty parameter  $\mu$  which depends on  $x^*$ . Initially,  $\mu = 0$ . Observe that the reduction of the merit function  $\Phi(x^*) < \Phi(x^0)$  is not obtained if the value of  $\mu$  does not satisfy the condition  $\hat{\Phi}(x^*) < \hat{\Phi}(x^0)$ , where  $\hat{\Phi}$  is the linear approximation

$$\hat{\Phi}(x) = \hat{f}(x) + \mu [\max \{-\hat{c}_i(x) : i = 1, \dots, m\}]_+. \tag{20.7}$$

Therefore, the following procedure for modifying the penalty parameter  $\mu$  can be used. Let  $\bar{\mu}$  be the smallest nonnegative value of  $\mu$  for which  $\hat{\Phi}(x^*) < \hat{\Phi}(x^0)$ . The existence of  $\bar{\mu}$  is a direct consequence of the definition of  $x^*$ . With this, the modification of  $\mu$  is the following. If  $\mu \geq 3\bar{\mu}/2$ , then  $\mu$  is unchanged, otherwise set  $\mu = 2\bar{\mu}$ . The selection of the factors 3/2 and 2 is obtained from numerical experiments with COBYLA. By increasing  $\mu$ , it is possible for (20.5) not to be satisfied. In this case, the optimality of  $x^0$  is restored by changing two vertices of the simplex. The computation of  $x^*$  and the modification of  $\mu$  are repeated until  $x^0$  is the vertex with the smallest value of the minimizing function and the value of  $\mu$  is acceptable.

The strategy for updating the radius of the trust-region is based on the principle that the value of  $\rho$  is maintained until the iterations cease to achieve a sufficient reduction of the merit function,

otherwise the value of  $\rho$  is reduced. However, a lot of simplexes may appear along the iterations. Before updating  $\rho$ , the current simplex must be “acceptable.” An unacceptable simplex is the one for which (20.3) is an inadequate approximation of the original problem. Therefore, define an acceptable simplex as follows. For  $j = 1, \dots, n$ , let  $\sigma^j$  be the Euclidian distance from the vertex  $x^j$  to the opposite face of this vertex from the current simplex. Let  $\eta^j$  be the length of the segment connecting the vertices  $x^j$  and  $x^0$ . The simplex is called “acceptable” if and only if the inequalities

$$\sigma^j \geq \alpha\rho, \quad (20.8a)$$

$$\eta^j \leq \beta\rho, \quad (20.8b)$$

are satisfied for  $j = 1, \dots, n$ , where  $\alpha$  and  $\beta$  are two constants so that  $0 < \alpha < 1 < \beta$ . (Powell recommends  $\alpha = 0.25$  and  $\beta = 2.1$ .) The radius of the trust-region is reduced if either  $\|x^* - x^0\| \leq \rho/2$  or

$$\frac{\Phi(x^0) - \Phi(x^*)}{\widehat{\Phi}(x^0) - \widehat{\Phi}(x^*)} < 0.1. \quad (20.9)$$

In COBYLA, two values of  $\rho$  are used: the initial value  $\rho_{beg}$  and the final value  $\rho_{end}$ . (Powell recommends  $\rho_{beg} = 1.5$  and  $\rho_{end} = 10^{-8}$ .) If the conditions of reducing  $\rho$  are satisfied, then the following procedure is considered. If  $\rho \leq \rho_{end}$ , then the algorithm is stopped, the solution of the problem is given by  $x^0$ , except for the case in which  $\Phi(x^*) < \Phi(x^0)$ , when  $x^*$  is preferable as solution. On the other hand, if  $\rho > \rho_{end}$ , then the radius of the trust-region is updated as

$$\rho_{new} = \begin{cases} \rho/2, & \text{if } \rho > 3\rho_{end}, \\ \rho_{end}, & \text{if } \rho \leq 3\rho_{end}. \end{cases} \quad (20.10)$$

Numerical experiments show that whenever  $\rho$  is reduced, the parameter  $\mu$  is also reduced. To update the parameter  $\mu$  the following procedure may be used. Let us say that the  $i$ -th constraint is important in the merit function if  $i$  is in the set

$$I = \{i : c_i^{\min} < c_i^{\max}/2\} \cap \{1, \dots, m\}, \quad (20.11)$$

where  $c_i^{\min}$  and  $c_i^{\max}$  are the smallest and the largest value of  $c_i(x)$  in the vertices of the current simplex, respectively. If  $I = \emptyset$ , then set  $\mu = 0$ . Otherwise,  $\mu$  is updated as

$$\mu = \frac{\max_{j=0,1,\dots,n} f(x^j) - \min_{j=0,1,\dots,n} f(x^j)}{\min \left\{ [c_i^{\max}]_+ - c_i^{\min} : i \in I \right\}}, \quad (20.12)$$

if and only if  $\mu$  is reduced. The initial simplex is obtained by using  $\rho_{beg}$  and the initial point given by the user. Let  $x^0$  be the initial point. Then, for  $j = 1, \dots, n$ , set  $x^j = x^0 + \rho_{beg}e_j$ , where  $e_j$  is the  $j$ -th column of the identity matrix. For any value of  $j$ , if  $f(x^j) < f(x^0)$ , then  $x^0$  is replaced by  $x^j$ , so that  $x^0$  becomes the vertex in which the minimizing function is minimum.

The point  $x^*$  is not computed at every iteration because the aim is to satisfy the acceptability conditions (20.8a, 20.8b) of the current simplex. Hence, a point  $x^\Delta$  which improves the acceptability of the current simplex is computed. In COBYLA, at the current iteration,  $x^*$  is computed instead of  $x^\Delta$  if and only if at least one of the following five conditions is satisfied: (C1) the algorithm is at the first iteration, (C2) at the previous iteration the value of  $\rho$  was reduced, (C3) the previous iteration computed  $x^\Delta$ , (C4) the previous iteration computed  $x^*$  and the value of the merit function was reduced at least with one tenth from the predicted reduction (see (20.9)), (C5) the current simplex is

acceptable. When none of the above five conditions is satisfied, then  $x^\Delta$  is defined in the following way. If  $\eta^j > \beta\rho$ , for all  $j = 1, \dots, n$ , then let  $l$  be the smallest integer from  $[1, n]$  which satisfies the condition

$$\eta^l = \max \{\eta^j : j = 1, \dots, n\}. \quad (20.13)$$

Otherwise,  $l$  is obtained from the condition

$$\sigma^l = \min \{\sigma^j : j = 1, \dots, n\}. \quad (20.14)$$

Now the vertex  $x^l$  is replaced with  $x^\Delta$ , that is  $x^\Delta$  is far away from the face of the simplex which is opposite to the vertex  $x^l$ . Therefore, let  $v^l$  be a vector of unitary length, perpendicular to this face of the simplex. Then,

$$x^\Delta = x^0 \pm \gamma\rho v^l, \quad (20.15)$$

where  $\pm$  is selected to minimize the approximation  $\widehat{\Phi}(x^\Delta)$ , while  $\gamma$  is a constant from the interval  $(\alpha, 1)$  ( $\gamma = 1/2$  in COBYLA). Therefore, the next iteration is given by the simplex with the vertices  $\{x^j : j = 0, 1, \dots, n, j \neq l\}$  and  $x^\Delta$ . This is an iteration which computes  $x^\Delta$ .

When an iteration computes  $x^*$ , then we must choose one from the following three options: reduce  $\rho$ , maintain the value of  $\rho$  for another iteration which will compute  $x^*$  or maintain  $\rho$  for an iteration which will improve the acceptability of the simplex. Since (20.9) is used if and only if  $x^*$  satisfies the condition

$$\|x^* - x^0\|_2 \geq \rho/2, \quad (20.16)$$

it follows that the values  $f(x^*)$  and  $c_i(x^*)$ ,  $i = 1, \dots, m$ , are computed only when the inequality (20.16) holds. These values can be included in the linear approximation (20.3) by replacing a vertex from  $\{x^j : j = 1, \dots, n\}$  of the current simplex with  $x^*$  as follows. Compute the scalars  $\{\bar{\sigma}^j : j = 1, \dots, n\}$ , where  $\bar{\sigma}^j$  is the distance from  $x^*$  to the face of the current simplex which is opposite to  $x^j$ . With this, the set  $J = \{j : \bar{\sigma}^j \geq \sigma^j\} \cup \{j : \bar{\sigma}^j \geq \alpha\rho\}$  is determined, where  $\alpha$  is defined in (20.8). Then, the optimum vertex from the next iteration is the point

$$\bar{x}^0 = \begin{cases} x^*, & \Phi(x^*) < \Phi(x^0), \\ x^0, & \Phi(x^*) \geq \Phi(x^0). \end{cases} \quad (20.17)$$

If the set  $J$  is nonempty, then let  $l$  be the last element of  $J$ , so that  $\|x^l - \bar{x}^0\|_2 = \max \{\|x^j - \bar{x}^0\|_2 : j \in J\}$ . If one of the following conditions  $\Phi(x^*) < \Phi(x^0)$ ,  $\bar{\sigma}^l > \sigma^l$  or both of them are satisfied, then  $x^l$  is replaced with  $x^*$ , where  $l$  is obtained from the relation  $\bar{\sigma}^l/\sigma^l = \max \{\bar{\sigma}^j/\sigma^j : j = 1, \dots, n\}$ . Thus, the simplex is updated at the iterations which compute the values of the minimizing function and of the constraints in  $x^*$ , the only exception being the iterations where, besides the inequalities  $\{\bar{\sigma}^j \leq \sigma^j : j = 1, \dots, n\}$  and  $\Phi(x^*) \geq \Phi(x^0)$ , the distance  $\|x^l - \bar{x}^0\|_2$  is upper bounded by  $\delta\rho$  for any  $j \in J$ , where  $\delta$  is a constant from the interval  $(1, \beta]$ . (In COBYLA,  $\delta = 1.1$ ). With this, the description of COBYLA is complete.

To prove the convergence of the algorithm, the following points need to be established: (A1) the penalty parameter  $\mu$  from the merit function (20.4) remains finite, (A2) the number of reductions of the radius of the trust-region  $\rho$  is finite, (A3) if and only if  $\mu$  and  $\rho$  remain unchanged, then the optimum vertex  $x^0$  cannot be retained for an infinite number of iterations, (A4) if  $\mu$  and  $\rho$  remain

**Table 20.1** Performances of COBYLA.  $\rho_{beg} = 1.5d0$ ,  $\rho_{end} = 1.$   $d - 8$ .

	$n$	$m$	$vfi$	$nri$	$\#nf$	$vfo$	$nrf$
ALKI	10	34	-872.3874	0.448994	1552	-1550.38851	0.5086e-12
CAM	10	43	-47.12388	0.912991	182	-43.859947	0.2957e-14
DES	15	10	3.7046642	572.8970	1440	6.4746914	0.9222e-15
HANG	20	2	4.953514	21.98809	97484	5.0690569	0.7105e-13
MSP3	13	41	449.70	202.04113	1420391	97.587578	0.156e-13
MSP5	16	53	284.66957	85.358034	108956	175.619466	0.0
POOL	34	108	2743.3678	5886.4115	1025	2779.2255	0.9570e-10
PREC	8	22	3.6573657	0.4350181	3552	3.9511635	0.1665e-15
PPSE	9	30	4853.3335	1.184889	230	5055.0118	0.844e-15
TRAFO	6	2	137.06643	0.0	6986	135.075962	0.222e-15

In this table, we have  $n$  = the number of variables,  $m$  = the number of constraints including the simple bounds on variables,  $vfi$  = the value of the objective function in the initial point,  $nri$  = the norm of the vector given by the violated constraints in the initial point,  $\#nf$  = the number of the evaluations of the functions defining the problem,  $vfo$  = the value of the objective function in the final point,  $nrf$  = the norm of the vector given by the violated constraints in the final point

unchanged, then the number of replacements of the optimum vertex is finite. Powell (1993) proved that any sequence of iterations generated by COBYLA which do not modify  $\rho$  or  $x^0$  is finite. Moreover, consider an iteration which replaces the vertex  $x^l$  of the current simplex with  $x^\Delta$  and take the distances from the vertices (excluding  $x^0$ ) to the opposite faces in the old simplex and in the new simplex  $\{\sigma_{old}^j : j = 1, \dots, n\}$  and  $\{\sigma_{new}^j : j = 1, \dots, n\}$ , respectively. Then, besides  $\sigma_{new}^l = \gamma\rho$ , the following inequalities  $\sigma_{new}^j \geq \sigma_{old}^j$ ,  $j = 1, \dots, n, j \neq l$ , hold.

## Numerical Study—COBYLA Algorithm

Table 20.1 shows the performances of COBYLA for solving 10 applications from the LACOP collection described in Appendix C.

---

## 20.2 DFL Algorithm

The following algorithm is based on the penalization of the constraints in the objective function, in the frame of the direct methods. The DFL method, developed by Liuzzi, Lucidi, and Sciandrone (2010), consists in the approximate minimization of a sequence of merit functions, in which the penalization of the violated constraints is increased along the iterations. The minimization of the merit functions involves only the values of the functions defining the problem, without referring to the derivative information about these functions. Consider the problem

$$\begin{aligned}
& \min f(x) \\
& \text{subject to} \\
& c(x) \leq 0, \\
& l \leq x \leq u,
\end{aligned} \tag{20.18}$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $c: \mathbb{R}^n \rightarrow \mathbb{R}^m$  are known functions for which their values can be computed in any point from the domain  $X = \{x \in \mathbb{R}^n : l \leq x \leq u\}$  defined by the simple bound constraints. For solving (20.18), the objective function is augmented with terms which penalize the violation of the constraints, thus obtaining the following penalty function:

$$Q(x, \mu) = f(x) + \frac{1}{\mu} \left( \sum_{j=1}^m [c_j(x)^+]^2 + \sum_{i=1}^n [(x_i - u_i)^+]^2 + \sum_{i=1}^n [(l_i - x_i)^+]^2 \right), \tag{20.19}$$

where, as usual,  $v^+ = \max \{0, v\}$ . As it is known, if the minimum  $x_k^*$  of the penalty function  $Q(x, \mu)$  can be determined for a sequence  $\{\mu_k\}$  so that  $\mu_k \rightarrow 0$ , then the sequence  $\{x_k^*\}$  is convergent to a global minimum  $x^*$  of (20.18) (Fiacco, & McCormick, 1968). More exactly, Nocedal and Wright (2006) prove the convergence to a stationary point of (20.18) in very mild conditions of regularity, provided that, for every  $k$ , a stationary point of  $Q(x, \mu_k)$  can be determined with increased accuracy. In other words, if  $\{x_k\}$  is a sequence of points satisfying the condition

$$\|\nabla Q(x_k, \mu_k)\| \leq \tau_k, \tag{20.20}$$

where  $\{\tau_k\}$  is a sequence of scalars such that  $0 < \tau_{k+1} < \tau_k$ , for any  $k$ , and  $\tau_k \rightarrow 0$ , then if the sequence  $\{x_k\}$  (or, at least a subsequence of it) is convergent to the point  $\tilde{x}$  in which the gradients of the active constraints are linear independent,  $\tilde{x}$  is a stationary point of (20.18).

These theoretical results, very well established in the case of differentiable functions, can be extended in the case in which there is no access to the derivative information of the functions defining the problem (20.18). Observe that the simple bound constraints can be explicitly treated because their gradients are known. Therefore, the following penalty function can be introduced (Liuzzi, Lucidi, & Sciandrone, 2010).

$$P(x, \mu) = f(x) + \frac{1}{\mu} \sum_{j=1}^m [c_j(x)^+]^q, \tag{20.21}$$

where  $q > 1$  (for example  $q = 1.1$ ) and where only the nonlinear constraints are introduced in (20.21). (For the linear constraints, their gradients are known.) With this, for any fixed value of  $\mu$  the following problem is considered:

$$\begin{aligned}
& \min P(x, \mu) \\
& \text{subject to} \\
& l \leq x \leq u.
\end{aligned} \tag{20.22}$$

The direct methods are based only on the values of the functions defining the problem computed along a set of directions able at limit to give sufficient information to recover the necessary optimality condition of order one. In the context of the constrained optimization, in which the penalty parameter is updated to zero, the procedure for its updating must be in accordance with the procedure of selecting the points where the functions defining the problem are evaluated, the so-called probe points. The idea is that the penalty parameter needs to be reduced to zero more slowly than the

maximum step used in the computation procedure of the probe points, where the values of the problem's functions are evaluated.

The following proposition, proved by Liuzzi, Lucidi, and Sciandrone (2010), shows a very general result which can be used to show the convergence to the stationary points of a sequence of iterations generated by a direct search algorithm (without involving the derivative information) based on the approximate minimization of the penalty function  $P(x, \mu)$  on  $X$ . The proposition gives sufficient conditions of the procedure for the computation of the probe points, as well as of the procedure for updating the penalty parameter which guarantees the convergence to a stationary point of (20.18).

**Proposition 20.1** *Let  $\{\mu_k\}$  be a bounded sequence of penalty parameters. Let  $\{x_k\}$  be a sequence of points so that  $x_k \in X$  for any  $k$  and let  $\bar{x}$  be a limit point of the subsequence  $\{x_k\}_K$ , where  $K \subseteq \{0, 1, \dots\}$ . Suppose that  $\bar{x}$  satisfies the Mangasarian-Fromovitz (MFCO) constraint qualification (see Remark 11.2) and for any  $k$  large enough the following conditions hold:*

(i) *For any  $d^i \in D \cap D(\bar{x})$  there exist the vectors  $y_k^i$  and the scalars  $\xi_k^i > 0$  such that*

$$y_k^i + \xi_k^i d^i \in X, \quad (20.23a)$$

$$P(y_k^i + \xi_k^i d^i, \mu_k) \geq P(y_k^i, \mu_k) - o(\xi_k^i), \quad (20.23b)$$

$$\lim_{k \rightarrow \infty, k \in K} \frac{\max \{\xi_k^i, \|x_k - y_k^i\|\}}{\mu_k} = 0, \quad (20.23c)$$

where  $D(x) = \{d \in \mathbb{R}^n : d_i \geq 0 \text{ if } x_i = l_i, d_i \leq 0 \text{ if } x_i = u_i, i = 1, \dots, n\}$  is the cone of admissible directions in  $x$  subject to the simple bound constraints and  $D = \{\pm e_1, \dots, \pm e_n\}$ , where  $e_i$  is the  $i$ -th column of the unity matrix.

(ii) *For any  $k \in K$*

$$\lim_{k \rightarrow \infty, k \in K} \mu_k \|c(x_k)^+\| = 0. \quad (20.23d)$$

Then  $\bar{x}$  is a stationary point of (20.18). ◆

A measure of the stationarity of the current iteration  $x_k$  for (20.22) is given by

$$\max_{i: d^i \in D \cap D(\bar{x})} \{\xi_k^i, \|x_k - y_k^i\|\}, \quad (20.24)$$

(see (Kolda, Lewis, & Troczon, 2003)). Therefore, the limit from (20.23c) shows that the current measure of the stationarity tends to zero faster than the penalty parameter  $\mu_k$  does.

With these preparatives, let us present a variant of the direct search algorithm for solving the problem (20.18). The algorithm uses a linear search which approximately minimizes a penalty function along some search directions. At every iteration, the searching directions are parallel with the axes of coordinates. The algorithm is based on the sensitivity of the objective function along the searching directions. This is the reason why the algorithm computes different stepsizes along each searching direction. In particular, at every iteration the following quantities are computed:

- $\bar{\alpha}^i$ ,  $i = 1, \dots, n$ , as the maximum stepsize that can be taken along the searching directions without leaving the domain  $X$ ,
- $\alpha_k^i$ ,  $i = 1, \dots, n$ , as the results of the approximate minimizations of the penalty function along the directions  $d_k^i$ ,
- $\tilde{\alpha}_k^i$ ,  $i = 1, \dots, n$ , as the stepsizes from the previous iteration used as initial stepsizes at the current iteration.

**Algorithm 20.1** *DFL Algorithm*

1.	Consider the initial point $x_0 \in X$ , as well as the parameters: $\mu_0 > 0$ , $\gamma > 0$ , $\theta \in (0, 1)$ , $p > 1$ . Consider $\tilde{\alpha}_0^i > 0$ and set $d_0^i = e_i$ , for $i = 1, \dots, n$ . Consider a sequence of positive numbers $\eta_k \rightarrow 0$
2.	<i>Minimization on the cone D.</i> Set $i = 1$ and $y_k^i = x_k$
3.	Compute $\bar{\alpha}^i$ so that $y_k^i + \bar{\alpha}^i d_k^i \in X$ . Set $\tilde{\alpha}_k^i = \min \{\bar{\alpha}^i, \tilde{\alpha}_k^i\}$ If $\tilde{\alpha}_k^i > 0$ and $P(y_k^i + \tilde{\alpha}_k^i d_k^i, \mu_k) \leq P(y_k^i, \mu_k) - \gamma(\tilde{\alpha}_k^i)^2$ , then compute $\alpha_k^i$ using the procedure $PE(\bar{\alpha}^i, \tilde{\alpha}_k^i, y_k^i, d_k^i, \gamma, \alpha_k^i)$ . Set $\tilde{\alpha}_{k+1}^i = \alpha_k^i$ , $d_{k+1}^i = d_k^i$ and go to step 6
4.	Compute $\bar{\alpha}^i$ so that $y_k^i - \bar{\alpha}^i d_k^i \in X$ . Set $\tilde{\alpha}_k^i = \min \{\bar{\alpha}^i, \tilde{\alpha}_k^i\}$ If $\tilde{\alpha}_k^i > 0$ and $P(y_k^i - \tilde{\alpha}_k^i d_k^i, \mu_k) \leq P(y_k^i, \mu_k) - \gamma(\tilde{\alpha}_k^i)^2$ , then compute $\alpha_k^i$ using the procedure $PE(\bar{\alpha}^i, \tilde{\alpha}_k^i, y_k^i, -d_k^i, \gamma, \alpha_k^i)$ . Set $\tilde{\alpha}_{k+1}^i = \alpha_k^i$ , $d_{k+1}^i = -d_k^i$ and go to step 6
5.	Set $\alpha_k^i = 0$ and $\tilde{\alpha}_{k+1}^i = \theta \tilde{\alpha}_k^i$
6.	Set $y_k^{i+1} = y_k^i + \alpha_k^i d_k^i$ . If $i < n$ , then set $i = i + 1$ and go to step 3
7.	If $\max_{i=1, \dots, n} \{\tilde{\alpha}_k^i, \alpha_k^i\} \leq \mu_k^p$ and $\ c(x_k)^+\  > \eta_k$ , then set $\mu_{k+1} = \theta \mu_k$ ; otherwise, set $\mu_{k+1} = \mu_k$
8.	Determine $x_{k+1} \in X$ such that $P(x_{k+1}, \mu_k) \leq P(y_k^{i+1}, \mu_k)$ . Set $k = k + 1$ and go to step 2

The procedure  $PE(\cdot)$  used in the frame of Algorithm 20.1 (see steps 3 and 4) is a step for computing the stepsize  $\alpha_k^i$  which approximately minimizes the penalty function along the directions  $d_k^i$ :

**Algorithm 20.2** *Procedure  $PE(\bar{\alpha}, \tilde{\alpha}, y, p, \gamma, \alpha)$* 

1.	Select $\delta \in (0, 1)$
2.	Set $\alpha = \tilde{\alpha}$
3.	Determine $\tilde{\alpha} = \min \{\bar{\alpha}, (\alpha/\delta)\}$
4.	If $\alpha = \bar{\alpha}$ or $P(y + \tilde{\alpha} p, \mu_k) > P(y, \mu_k) - \gamma(\tilde{\alpha})^2$ , return
5.	Set $\alpha = \tilde{\alpha}$ and go to step 3

Liuzzi, Lucidi, and Sciandrone (2010) prove that the *DFL algorithm is well defined*. Moreover, if the sequence of positive numbers  $\{\mu_k\}$  monotone decreasing is such that  $\lim_{k \rightarrow \infty} \mu_k = \bar{\mu} > 0$ , then  $\lim_{k \rightarrow \infty} \alpha_k^i = 0$ , for  $i = 1, \dots, n$ , and  $\lim_{k \rightarrow \infty} \tilde{\alpha}_k^i = 0$ , for  $i = 1, \dots, n$ . On the other hand, if the sequence of positive numbers  $\{\mu_k\}$  monotone decreasing is such that  $\lim_{k \rightarrow \infty} \mu_k = 0$ , then  $\lim_{k \rightarrow \infty, k \in K} \alpha_k^i = 0$ , for  $i = 1, \dots, n$ , and  $\lim_{k \rightarrow \infty, k \in K} \tilde{\alpha}_k^i = 0$ , for  $i = 1, \dots, n$ , where  $K = \{k : \mu_{k+1} < \mu_k\}$ .

Observe that the DFL algorithm considers the following search directions at every iteration  $k$ :

$$D_k = \{d_k^1, -d_k^1, \dots, d_k^n, -d_k^n\} = \{\pm e_1, \dots, \pm e_n\} = D,$$

where  $e_i$  is the  $i$ -th column of the unity matrix. At every iteration, the algorithm gets information on the behavior of the penalty function both in the direction  $d_k^i$  and in the direction  $-d_k^i$ . In particular, along all the directions  $d_k^i$ ,  $i = 1, \dots, n$ , the algorithm identifies the following circumstances:

- (a) If the initial stepsize  $\tilde{\alpha}_k^i$  does not produce a reduction of the penalty function, then either  $y_k^i + \tilde{\alpha}_k^i d_k^i \notin X$ , or  $P(y_k^i + \tilde{\alpha}_k^i d_k^i, \mu_k) > P(y_k^i, \mu_k) - \gamma(\tilde{\alpha}_k^i)^2$ .
- (b) If the initial stepsize  $\tilde{\alpha}_k^i$  produces a reduction of the penalty function, then both conditions  $y_k^i + \tilde{\alpha}_k^i d_k^i \in X$  and  $P(y_k^i + \tilde{\alpha}_k^i d_k^i, \mu_k) \leq P(y_k^i, \mu_k) - \gamma(\tilde{\alpha}_k^i)^2$  are satisfied and a stepsize  $\alpha_k^i$  is generated by the linear search, so that either  $y_k^i + \frac{\alpha_k^i}{\delta} d_k^i \notin X$ , or  $P(y_k^i + \frac{\alpha_k^i}{\delta} d_k^i, \mu_k) > P(y_k^i, \mu_k) - \gamma\left(\frac{\alpha_k^i}{\delta}\right)^2$ .

A similar analysis may be obtained along the direction  $-d_k^i$  (Liuzzi, Lucidi, & Sciandrone, 2010). This analysis determines that the initial stepsize  $\tilde{\alpha}_k^i$  should satisfy either  $y_k^i + \tilde{\alpha}_k^i (-d_k^i) \notin X$ , or  $P(y_k^i + \tilde{\alpha}_k^i (-d_k^i), \mu_k) > P(y_k^i, \mu_k) - \gamma(\tilde{\alpha}_k^i)^2$ . Otherwise, compute a stepsize  $\alpha_k^i$  such that either  $y_k^i + \frac{\alpha_k^i}{\delta} (-d_k^i) \notin X$  or  $P(y_k^i + \frac{\alpha_k^i}{\delta} (-d_k^i), \mu_k) > P(y_k^i, \mu_k) - \gamma\left(\frac{\alpha_k^i}{\delta}\right)^2$ . In other words, the algorithm considers the search along the directions  $\pm d_k^i$ , extracting information from the behavior of the penalty function  $P(\cdot)$  along these directions. The following theorem shows the convergence of the DFL algorithm.

**Theorem 20.1** *Let  $\{x_k\}$  be the sequence generated by the DFL algorithm. Suppose that any limit point of the sequence  $\{x_k\}$  satisfies the Mangasarian-Fromovitz (MFCO) constraint qualification (see Remark 11.2). Then there exists a limit point  $\bar{x}$  of the sequence  $\{x_k\}$ , which is a stationary point of the problem (20.18).*  $\blacklozenge$

Let  $\{x_k\}$  and  $\{\mu_k\}$  be the sequences generated by the DFL algorithm and assume that any limit point of  $\{x_k\}$  satisfies the Mangasarian-Fromovitz constraint qualification. If  $\lim_{k \rightarrow \infty} \mu_k = \bar{\mu} > 0$ , then any limit point of  $\{x_k\}$  is a stationary point of (20.18).

In step 1 of DFL the parameters are initialized as  $\gamma = 10^{-6}$ ,  $\theta = 0.5$ ,  $p = 2$ , and  $\tilde{\alpha}_0^i = \max\{10^{-3}, \min\{1, |(x_0)_i|\}\}$ ,  $i = 1, \dots, n$ .

## Numerical Study—DFL Algorithm

Table 20.2 shows the performances of DFL for solving 9 applications from the LACOP collection presented in Appendix C.

A comparison between COBYLA and DFL (see Tables 20.1 and 20.2, respectively) shows that for minimizing the applications considered in this numerical study, COBYLA generates better approximate solutions than DFL. An explanation of this behavior is that at every iteration, COBYLA considers a linear approximation of the minimizing function and of the constraints and solves a linear programming problem, which is more appropriate than the minimization of a simple penalty function.

**Table 20.2** Performances of DFL

	<i>n</i>	<i>m</i>	<i>vfi</i>	<i>nri</i>	#iter	#nf	<i>vfo</i>	<i>nrf</i>
ALKI	10	14	-872.387	0.448994	672	1336	-931.120	0.15e-10
ELCH	10	6	-20.9602	1.449137	271	499	-40.89478	0.218e-6
CAM	10	23	-47.1238	1.71630	306622	809653	-45.54602	0.368799
LATHE	10	16	-2931.46	0.0	714	961	-4429.1529	0.0
MSP3	13	15	449.70	202.0401	1068	1774	50.0	0.09901
MSP5	16	21	284.669	85.35803	1778	2049	209.96775	0.741584
PPSE	9	12	4853.33	1.696721	6460942	>2e+7	8063.5088	0.003715
PREC	8	6	3.65736	0.435018	489	979	4.1431963	0.0
TRAFO	6	2	137.066	0.020303	20865	63112	136.27182	0.000126

In this table, we have  $n$  = the number of variables,  $m$  = the number of constraints,  $vfi$  = the value of the objective function in the initial point,  $nri$  = the norm of the vector given by the violated constraints in the initial point, #iter = the number of the iterations, #nf = the number of evaluations of the functions defining the problem,  $vfo$  = the value of the objective function in the final point,  $nrf$  = the norm of the vector given by the violated constraints in the final point

In other words, the use of successive linear programming is more advantageous than the penalty function. The weakness of the DFL algorithm is that at every iteration the penalty function is approximately minimized along the axes of coordinates.

### Notes and References

We emphasized that for derivative-free methods like the ones presented in this book, the only thing we can obtain is a point where the minimizing function value is smaller than or equal to the value in its initial point. Nothing can be said about its optimality, but having in view the derivative scarcity of information on the minimizing function, the result obtained may be of use for practical considerations.

Excellent reviews and perspectives highlighting the recent developments of the derivative-free optimization methods for both unconstrained and constrained optimization with deterministic, stochastic, or structured objectives were given by Rios and Shainidis (2013) and by Larson, Menickelly, and Wild (2019). A review of the derivative-free algorithms followed by a systematic comparison of 22 related implementations using a test set of 502 problems was given by Rios and Shainidis (2013). Their conclusion is not definitive. They emphasize that the ability of all these solvers to obtain good solutions diminishes with the increasing size of the problem. Besides, attaining the best solutions even for small problems is a challenge for most current derivative-free solvers and there is no single solver whose performance dominates that of all the others. The dimension (the number of variables) of the problems and the nonsmoothness rapidly increase the complexity of the search and decrease the performances of all solvers.

---

# Appendix A: Mathematical Review

---

## A1. Elements of Applied Numerical Linear Algebra

### Vectors

Define a column  $n$ -vector to be an array of  $n$  numbers denoted as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

The number  $x_i$ ,  $i = 1, \dots, n$ , is called the  $i$ -th *component* of the vector  $x$ . Define the set of real numbers by  $\mathbb{R}$ . The space of the real vectors of length  $n$  is denoted by  $\mathbb{R}^n$ . The vectors are always column vectors. The *transpose* of  $x$  is denoted by  $x^T$ . Therefore,  $x^T$  is a row vector. Given the vectors  $x, y \in \mathbb{R}^n$ , the *scalar product* is defined by

$$x^T y = \sum_{i=1}^n x_i y_i.$$

The vectors  $x, y \in \mathbb{R}^n$  are *orthogonal (perpendicular)* if  $x^T y = 0$ . This is denoted by writing  $x \perp y$ . If  $x$  and  $y$  are orthogonal and  $x^T x = 1$  and  $y^T y = 1$ , then we say that  $x$  and  $y$  are *orthonormal*.

A set of vectors  $v_1, \dots, v_k$  is said to be *linearly dependent* if there are the scalars  $\lambda_1, \dots, \lambda_k$ , not all zero, so that  $\sum_{i=1}^k \lambda_i v_i = 0$ . If no such set of scalars exists, then the vectors are said to be *linearly independent*. A *linear combination* of the vectors  $v_1, \dots, v_k$  is a vector of the form  $\sum_{i=1}^k \lambda_i v_i$ , where all  $\lambda_i$  are scalars.

Let  $\{x_1, \dots, x_n\}$  be a set of vectors. The *span* of this set of vectors denoted  $\text{span}\{x_1, \dots, x_n\}$  is the set of all the vectors that can be expressed as a linear combination of  $\{x_1, \dots, x_n\}$ . That is,

$$\text{span}\{x_1, \dots, x_n\} = \left\{ v : v = \sum_{i=1}^n \alpha_i x_i, \alpha_i \in \mathbb{R} \right\}.$$

If  $\{x_1, \dots, x_n\}$  is a set of  $n$  linearly independent vectors where each  $x_i \in \mathbb{R}^n$ , then  $\text{span}\{x_1, \dots, x_n\} = \mathbb{R}^n$ . In other words, any vector  $v \in \mathbb{R}^n$  can be written as a linear combination of  $x_1, \dots, x_n$ . A linearly independent set of vectors that span  $\mathbb{R}^n$  is said to be a *basis* for  $\mathbb{R}^n$ .

## Norms of Vectors

For a vector  $x \in \mathbb{R}^n$ , the following *norms* can be defined:

$$\|x\|_1 = \sum_{i=1}^n |x_i|, \quad \|x\|_2 = (x^T x)^{1/2}, \quad \|x\|_\infty = \max_{i=1, \dots, n} |x_i|.$$

The norm  $\|\cdot\|_2$  is often called the *Euclidean norm* or the  $l_2$  norm. On the other hand,  $\|\cdot\|_1$  is referred to as the  $l_1$  norm and  $\|\cdot\|_\infty$  as the  $l_\infty$  norm. All these norms measure the *length* of the vector in some sense, and they are equivalent, i.e., each one is bounded above and below by a multiple of the other. More exactly, for all  $x \in \mathbb{R}^n$  it follows that

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty \text{ and } \|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty.$$

In general, a norm is any mapping  $\|\cdot\|$  from  $\mathbb{R}^n$  to the nonnegative real numbers that satisfy the following properties:

1. For all  $x, y \in \mathbb{R}^n$ ,  $\|x + y\| \leq \|x\| + \|y\|$ , with equality if and only if one of the vectors  $x$  and  $y$  is a nonnegative scalar multiple of the other one.
2.  $\|x\| = 0 \Rightarrow x = 0$ ,
3.  $\|\alpha x\| = |\alpha|\|x\|$ , for all  $\alpha \in \mathbb{R}$  and  $x \in \mathbb{R}^n$ .

The *magnitude* of a vector  $x$  is  $\|x\|_2 = (x^T x)^{1/2}$ . The *angle* between two nonzero vectors  $x, y \in \mathbb{R}^n$  is defined to be the number  $\theta \in [0, \pi]$  so that  $\cos\theta = x^T y / \|x\| \|y\|$ .

For the Euclidian norm, the *Cauchy-Schwarz inequality* holds  $|x^T y| \leq \|x\| \|y\|$ , with equality if and only if one of these vectors is a nonnegative multiple of the other one. In particular,

$$|x^T y| = \left| \sum_i x_i y_i \right| \leq \sum_i |x_i| |y_i| \leq (\max_i |x_i|) \left( \sum_i |y_i| \right) = \|x\|_\infty \|y\|_1.$$

The *Hölder inequality*, a generalization of the Cauchy-Schwarz inequality, states that for all  $a_i > 0$ ,  $b_i > 0$ ,  $i = 1, \dots, n$ ,  $p, q > 0$  so that  $1/p + 1/q = 1$

$$\sum_{i=1}^n a_i b_i \leq \left( \sum_{i=1}^n a_i^p \right)^{1/p} \left( \sum_{i=1}^n b_i^q \right)^{1/q}.$$

## Matrices

A *matrix* is a rectangular array of numbers with  $m$  rows and  $n$  columns specified by its elements  $a_{ij}$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ . The space of the real  $m \times n$  matrices is denoted by  $\mathbb{R}^{m \times n}$ . A *submatrix* of a given matrix  $A$  is an array obtained by deleting any combination of rows and columns from  $A$ . The *leading  $j \times j$  principal submatrix* of  $A$  is denoted by  $A(1:j, 1:j)$ . The *transpose* of  $A \in \mathbb{R}^{m \times n}$  denoted by  $A^T$  is the  $n \times m$  matrix with elements  $a_{ji}$ . In other words, the  $(i, j)$ -th entry of  $A^T$  is the  $(j, i)$ -th entry of  $A$ . Therefore, if  $A \in \mathbb{R}^{m \times n}$ , then  $A^T \in \mathbb{R}^{n \times m}$ . The matrix  $A$  is *squared* if  $m = n$ . For a square matrix  $A = (a_{ij}) \in \mathbb{R}^{n \times n}$  the elements  $a_{11}, a_{22}, \dots, a_{nn}$  define the *main diagonal* of the matrix. A squared matrix is *symmetric* if  $A = A^T$ . A matrix  $A \in \mathbb{R}^{n \times n}$  is *diagonal* if  $a_{ij} = 0$  for all  $i \neq j$ . The *identity matrix* denoted by  $I$  is the square diagonal matrix whose diagonal elements are all 1.

A square matrix  $A = (a_{ij})$  is said to be *lower triangular* if  $a_{ij} = 0$  for  $i < j$ . A *unit lower triangular* matrix is a lower triangular matrix with all the diagonal elements equal to 1. The matrix  $A$  is said to be *upper triangular* if  $a_{ij} = 0$  for  $i > j$ . A matrix  $A \in \mathbb{R}^{n \times n}$  is *tridiagonal* if  $a_{ij} = 0$  for  $|i - j| > 1$ . A matrix  $A \in \mathbb{R}^{n \times n}$  is *pentadiagonal* if  $a_{ij} = 0$  for  $|i - j| > 2$ . A matrix  $A$  is *normal* if  $A^T A = A A^T$ .

## Matrix Norms

The most widely used matrix norms are defined in terms of vector norms. If  $A$  is a matrix and  $\|x\|$  is a vector norm, then the *induced matrix norm*  $\|A\|$  is defined by

$$\|A\| = \max_{\|x\|=1} \|Ax\|.$$

Every induced matrix norm satisfies  $\|Ax\| \leq \|A\| \|x\|$  for all vectors  $x$ . Also,  $\|AB\| \leq \|A\| \|B\|$  for the matrices  $A$  and  $B$ . The matrix norms corresponding to the above vector norms are

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |A_{ij}|, \quad \|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}, \quad \|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |A_{ij}|,$$

where  $\lambda_{\max}(A^T A)$  is the largest eigenvalue of  $A^T A$ . If  $A$  is a square symmetric matrix, then  $\|A\|_2 = |\lambda_{\max}(A)|$ , where  $\lambda_{\max}(A)$  is the eigenvalue of the largest magnitude of  $A$ .

The *Frobenius norm* of  $A \in \mathbb{R}^{m \times n}$  is defined as

$$\|A\|_F^2 = \sum_{i,j} |a_{ij}|^2 = \text{tr}(A^T A),$$

where for the matrix  $A^{n \times n} = (a_{ij})$ ,  $\text{tr}(A) = a_{11} + \cdots + a_{nn}$  is the trace of  $A$ . The *ellipsoid norm* is defined as  $\|x\|_A = (x^T A x)^{1/2}$ , where  $A$  is a symmetric and positive definite matrix.

## Subspaces

For a function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ , let  $R(f)$  denote the *range* of  $f$ . That is  $R(f) = \{f(x) : x \in \mathbb{R}^n\} \subseteq \mathbb{R}^m$  is the set of all the “*images*” when  $x$  varies over  $\mathbb{R}^n$ . The *range* of a matrix  $A \in \mathbb{R}^{m \times n}$  denoted  $R(A)$  is the span of the columns of  $A$ . That is,

$$R(A) = \{v \in \mathbb{R}^m : v = Ax, x \in \mathbb{R}^n\}.$$

Therefore,  $R(A)$  is the space spanned by the columns of  $A$  (*column space*). The range of  $A^T$  is the span of the columns of  $A^T$ . But, the columns of  $A^T$  are just the rows of  $A$ . Therefore,

$$R(A^T) = \{w \in \mathbb{R}^n : w = A^T y, y \in \mathbb{R}^m\}$$

is the space spanned by the rows of  $A$  (*row space*).

The dimension of  $R(A)$  is the *rank* of  $A$ , denoted  $\text{rank}(A)$ . The rank of a matrix  $A$  is equal to the maximum number of linearly independent columns in  $A$ . This number is also equal to the maximum number of linearly independent rows in  $A$ . The rank of  $A \in \mathbb{R}^{m \times n}$  can never be greater than the minimum of  $m$  and  $n$ . The  $m \times n$  matrix  $A$  is said to be of *full rank* if the rank of  $A$  is equal to the minimum of  $m$  and  $n$ .

The *null space of a matrix*  $A \in \mathbb{R}^{m \times n}$  is the set

$$N(A) = \{x : Ax = 0\} \subseteq \mathbb{R}^n.$$

In other words,  $N(A)$  is the set of all the solutions to the homogeneous system  $Ax = 0$ . For  $A \in \mathbb{R}^{m \times n}$ , the set  $N(A^T) = \{y \in \mathbb{R}^m : A^T y = 0\} \subseteq \mathbb{R}^m$  is called the *left-hand null space* of  $A$  because  $N(A^T)$  is the set of all the solutions to the left-hand homogeneous system  $y^T A = 0^T$ . Observe that the vectors in  $R(A)$  are of size  $m$ , while the vectors in  $N(A)$  are of size  $n$ . Therefore, the vectors in  $R(A^T)$  and  $N(A)$  are both in  $\mathbb{R}^n$ . The following equations are true:

1.  $\{w : w = u + v, u \in R(A^T), v \in N(A)\} = \mathbb{R}^n$ .
2.  $R(A^T) \cap N(A) = \{0\}$ .

In other words,  $R(A^T)$  and  $N(A)$  are disjoint subsets that together span the entire space of  $\mathbb{R}^n$ . The *fundamental theorem of linear algebra* states that

$$N(A) \oplus R(A^T) = \mathbb{R}^n,$$

where  $n$  is the number of columns of  $A$  and  $\oplus$  denotes the *direct sum* of two sets. (If  $S_1$  and  $S_2$  are two sets, then  $S_1 \oplus S_2 = \{u + v : u \in S_1, v \in S_2\}$ .) Often, the sets of this type are called *orthogonal complements* and we write  $R(A^T) = N(A)^\perp$ .

If  $A \in \mathbb{R}^{m \times n}$  then:

1.  $N(A) = \{0\}$  if and only if  $\text{rank}(A) = n$ .
2.  $N(A^T) = \{0\}$  if and only if  $\text{rank}(A) = m$ .

For  $A \in \mathbb{R}^{m \times n}$  the following statements are true:

1.  $R(A^T A) = R(A^T)$  and  $R(A A^T) = R(A)$ .
2.  $N(A^T A) = N(A)$  and  $N(A A^T) = N(A^T)$ .

For all the matrices  $A \in \mathbb{R}^{m \times n}$ ,  $\dim R(A) + \dim N(A) = n$ . Traditionally,  $\dim N(A)$  is known as the *nullity* of  $A$ .

## Inverse of a Matrix

A squared  $n \times n$  matrix  $A$  is *nonsingular* if for any vector  $b \in \mathbb{R}^n$  there exists  $x \in \mathbb{R}^n$  so that  $Ax = b$ . For nonsingular matrices  $A$ , there exists a unique  $n \times n$  matrix  $B$  so that  $AB = BA = I$ . The matrix  $B$  is denoted by  $A^{-1}$  and is called the *inverse* of  $A$ . For nonsingular matrices  $A$  and  $B$ , the following properties hold:

1.  $(A^{-1})^{-1} = A$ ,
2. If the product  $AB$  exists and it is nonsingular, then  $(AB)^{-1} = B^{-1}A^{-1}$ ,
3.  $(A^T)^{-1} = (A^{-1})^T$ ,
4.  $(cA)^{-1} = c^{-1}A^{-1}$ , for any nonzero scalar  $c$ .
5. If  $A$  is nonsingular and symmetric, then  $A^{-1}$  is symmetric.
6. If  $A \in \mathbb{R}^{n \times n}$  is nonsingular, then  $\text{rank}(A) = n$ .
7.  $\det(A) \neq 0$ , where  $\det(A)$  is the determinant of  $A$ .

If  $A, B \in \mathbb{R}^{n \times n}$ , then the matrix  $B$  is the *approximate inverse* of  $A$  if  $\|I - BA\| < 1$ .

*Banach lemma:* If  $A, B \in \mathbb{R}^{n \times n}$  and  $B$  is an approximate inverse of  $A$ , then  $A$  and  $B$  are both nonsingular and

$$\|A^{-1}\| \leq \frac{\|B\|}{1 - \|I - BA\|}, \quad \|B^{-1}\| \leq \frac{\|A\|}{1 - \|I - BA\|},$$

and

$$\|A^{-1} - B\| \leq \frac{\|B\|\|I - BA\|}{1 - \|I - BA\|}, \quad \|A - B^{-1}\| \leq \frac{\|A\|\|I - BA\|}{1 - \|I - BA\|}.$$

*Von Neumann lemma:* If  $\|A\| < 1$ , then  $I - A$  is nonsingular.

*Sherman-Morrison formula.* Let  $a, b \in \mathbb{R}^n$  be two vectors so that  $1 + b^T a \neq 0$ . It is straightforward to verify by direct multiplication that

$$(I + ab^T)^{-1} = I - \frac{ab^T}{1 + b^T a}.$$

Let  $A \in \mathbb{R}^{n \times n}$  be a nonsingular matrix and  $a, b \in \mathbb{R}^n$  two vectors, so that  $1 + b^T A^{-1} a \neq 0$ . Then, the inverse of the matrix  $B = A + ab^T$  is

$$\begin{aligned} B^{-1} &= (A + ab^T)^{-1} = (A(I + A^{-1}ab^T))^{-1} = (I + A^{-1}ab^T)^{-1}A^{-1} \\ &= \left( I - \frac{A^{-1}ab^T}{1 + b^T A^{-1} a} \right) A^{-1} = A^{-1} - \frac{A^{-1}ab^T A^{-1}}{1 + b^T A^{-1} a}. \end{aligned}$$

If  $1 + b^T A^{-1} a = 0$ , then  $B$  is a singular matrix. This is often called the Sherman-Morrison rank-one update formula because, when  $a \neq b \neq 0$ , then  $\text{rank}(ab^T) = 1$ .

A generalization of the Sherman-Morrison formula is as follows. If  $C, D \in \mathbb{R}^{n \times p}$  so that  $(I + D^T A^{-1} C)^{-1}$  exists, then

$$(A + CD^T)^{-1} = A^{-1} - A^{-1}C(I + D^T A^{-1}C)^{-1}D^T A^{-1},$$

known as Sherman-Morrison-Woodbury formula.

*Some results for the unconstrained optimization quasi-Newton BFGS methods.*

(1) Let

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

be the BFGS updating formula, where  $B_k \in \mathbb{R}^{n \times n}$  is invertible and  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^T s_k > 0$ . If  $H_k = B_k^{-1}$ , then the inverse of  $B_{k+1}$ , denoted by  $H_{k+1}$ , is computed by twice applying the Sherman-Morrison update formula as

$$H_{k+1} = H_k - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{y_k^T s_k} + \left( 1 + \frac{y_k^T H_k y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k}.$$

(2) Let

$$B_{k+1} = \delta_k \left[ B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \right] + \gamma_k \frac{y_k y_k^T}{y_k^T s_k}$$

be the *scaled* BFGS updating formula, where  $B_k \in \mathbb{R}^{n \times n}$  is invertible,  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^T s_k > 0$  and  $\delta_k, \gamma_k \in \mathbb{R}$  are two known nonzero scalar parameters. If  $H_k = B_k^{-1}$ , then the inverse of  $B_{k+1}$ , denoted by  $H_{k+1}$ , is computed by twice applying the Sherman-Morrison update formula as

$$H_{k+1} = \frac{1}{\delta_k} \left[ H_k - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{y_k^T s_k} + \left( \frac{\delta_k}{\gamma_k} + \frac{y_k^T H_k y_k}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k} \right].$$

(3) Consider

$$B_{k+1} = \delta_k \left[ I - \frac{s_k s_k^T}{\|s_k\|^2} \right] + \frac{y_k y_k^T}{y_k^T s_k},$$

where  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^T s_k > 0$ ,  $s_k \neq 0$  and  $\delta_k \in \mathbb{R}$  is a known nonzero scalar parameter. Then the inverse of  $B_{k+1}$ , denoted by  $H_{k+1}$ , is computed by twice applying the Sherman-Morrison update formula as

$$H_{k+1} = \frac{1}{\delta_k} I - \frac{1}{\delta_k} \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + \left[ 1 + \frac{1}{\delta_k} \frac{\|y_k\|^2}{y_k^T s_k} \right] \frac{s_k s_k^T}{y_k^T s_k}.$$

(4) Consider

$$B_{k+1} = \delta_k \left[ I - \frac{s_k s_k^T}{\|s_k\|^2} \right] + \gamma_k \frac{y_k y_k^T}{y_k^T s_k},$$

where  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^T s_k > 0$ ,  $s_k \neq 0$  and  $\delta_k, \gamma_k \in \mathbb{R}$  are two known nonzero scalar parameters. Then the inverse of  $B_{k+1}$ , denoted by  $H_{k+1}$ , is computed by twice applying the Sherman-Morrison update formula as

$$H_{k+1} = \frac{1}{\delta_k} I - \frac{1}{\delta_k} \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + \left[ \frac{1}{\gamma_k} + \frac{1}{\delta_k} \frac{\|y_k\|^2}{y_k^T s_k} \right] \frac{s_k s_k^T}{y_k^T s_k}.$$

## Orthogonality

A square matrix  $Q \in \mathbb{R}^{n \times n}$  is *orthogonal* if it has the property  $QQ^T = Q^T Q = I$ , where  $I$  is the  $n \times n$  identity matrix. Therefore, the inverse of an orthogonal matrix is its transpose.

Suppose that  $\|u\| = 1$  and let  $u^\perp$  denote the space consisting of all the vectors that are perpendicular on  $u$ .  $u^\perp$  is called the *orthogonal complement* of  $u$ . The matrix  $P = I - uu^T$  is the *orthogonal projector* onto  $u^\perp$  in the sense that  $P$  maps each  $x$  to its orthogonal projection in  $u^\perp$ . For a subspace  $S \subset \mathbb{R}^n$ , the orthogonal complement  $S^\perp$  of  $S$  is defined as the set of all the vectors in  $\mathbb{R}^n$  that are orthogonal to every vector in  $S$ . In this case,  $\dim S^\perp = n - \dim S$ .

## Eigenvalues

A scalar value  $\lambda$  is an *eigenvalue* of the  $n \times n$  matrix  $A$  if there exists a nonzero vector  $u \in \mathbb{R}^n$  so that  $Au = \lambda u$ . The vector  $u$  is called an *eigenvector* of  $A$ . The *spectrum* of a matrix is the set of all its eigenvalues. Let  $\lambda_1, \dots, \lambda_n$  be the eigenvalues of the matrix  $A$ , real or complex. Then, its *spectral radius*  $\rho(A)$  is defined as  $\rho(A) = \max \{|\lambda_1|, \dots, |\lambda_n|\}$ . Observe that  $\rho(A) \leq \|A\|$  for every matrix norm. The condition number of  $A$  can be expressed as  $\kappa(A) = \rho(A)\rho(A^{-1})$ . A matrix  $A$  is nonsingular if all its eigenvalues are different from zero. The eigenvalues of the symmetric matrices are all real numbers. The nonsymmetric matrices may have imaginary eigenvalues.

Two matrices  $A, B \in \mathbb{R}^{n \times n}$  are *similar* if there exists a nonsingular matrix  $P \in \mathbb{R}^{n \times n}$  so that  $B = P^{-1}AP$ . Similar matrices represent the same linear operator in different bases, with  $P$  being the change of the basis matrix. Two similar matrices have the same eigenvalues, even though they will usually have different eigenvectors.

## Positive Definite Matrices

A square matrix  $A$  is *positive definite* if and only if  $x^T Ax > 0$  for every nonzero  $x \in \mathbb{R}^n$ . For the real symmetric matrices  $A$ , the following statements are equivalent:

1. All the eigenvalues of  $A$  are positive.
2.  $A = B^T B$  for some nonsingular  $B$ . While  $B$  is not unique, there is one and only one upper-triangular matrix  $R$  with positive diagonals so that  $A = R^T R$ . This is the *Cholesky factorization* of  $A$ .
3.  $A$  has an *LU* (or *LDU*) factorization with all the pivots being positive. The *LDU* factorization is of the form  $A = LDL^T = R^T R$ , where  $R = D^{1/2}L^T$  is the Cholesky factor of  $A$ .

Any of the statements above can serve as the definition of a positive definite matrix.

A matrix  $A$  is *positive semidefinite* if for all  $x \in \mathbb{R}^n$ ,  $x^T Ax \geq 0$ . The following statements are equivalent and can serve as the definition of a positive semidefinite matrix:

1. All the eigenvalues of  $A$  are nonnegative.
2.  $A = B^T B$  for some  $B$  with  $\text{rank}(B) = r$ .

If a matrix is symmetric and positive definite, then its eigenvalues are all positive real numbers. A symmetric matrix can be tested if it is positive definite by computing its eigenvalues and by verifying if they are all positive or by performing a Cholesky factorization.

## Gaussian Elimination (LU Factorization)

For solving the system  $Ax = b$ , where  $A$  is nonsingular, the Gaussian elimination consists of the following four steps:

1. Factorize the matrix  $A$  as  $A = PLU$ , where:

$P$  is a *permutation* matrix

$L$  is a *unit lower triangular* matrix

$U$  is a *nonsingular upper triangular* matrix

2. Solve the system  $PLUx = b$  subject to  $LUX$  by permuting the entries of  $b$ , i.e.,  $LUX = P^{-1}b = P^Tb$ .
3. Solve the system  $LUX = P^{-1}b$  subject to  $Ux$  by *forward substitution*, i.e.,  $Ux = L^{-1}(P^{-1}b)$ .
4. Solve the system  $Ux = L^{-1}(P^{-1}b)$  subject to  $x$  by *backward substitution*, i.e.,  $x = U^{-1}(L^{-1}(P^{-1}b))$ .

The following result is central in the Gaussian elimination.

*The following two statements are equivalent:*

1. *There exist a unique unit lower triangular matrix  $L$  and a nonsingular upper triangular matrix  $U$  such that  $A = LU$ . This is called the LU factorization of  $A$ .*
2. *All the leading principal submatrices of  $A$  are nonsingular.*

The LU factorization without pivoting can fail on nonsingular matrices, and therefore we need to introduce permutations into the Gaussian elimination.

*If  $A$  is a nonsingular matrix, then there exist the permutation matrices  $P_1$  and  $P_2$ , a unit lower triangular matrix  $L$ , and a nonsingular upper triangular matrix  $U$  such that  $P_1AP_2 = LU$ . Observe that  $P_1A$  reorders the rows of  $A$ .  $AP_2$  reorders the columns of  $A$ .  $P_1AP_2$  reorders both the rows and the columns of  $A$ .*

The next two results state simple ways to choose the permutation matrices  $P_1$  and  $P_2$  to guarantee that the Gaussian elimination will run on nonsingular matrices.

### Gaussian Elimination with Partial Pivoting

*The permutation matrices  $P'_2 = I$  and  $P'_1$  can be chosen in such a way that  $a_{11}$  is the largest entry in absolute value in its column. More generally, at step  $i$  of the Gaussian elimination, where the  $i$ -th column of  $L$  is computed, the rows  $i$  through  $n$  are permuted so that the largest entry in the column is on the diagonal. This is called “Gaussian elimination with partial pivoting,” or GEPP for short. GEPP guarantees that all the entries of  $L$  are bounded by one in absolute value.*

### Gaussian Elimination with Complete Pivoting

*The permutation matrices  $P'_2$  and  $P'_1$  are chosen in such a way that  $a_{11}$  is the largest entry in absolute value in the whole matrix. More generally, at step  $i$  of the Gaussian elimination, where the  $i$ -th column of  $L$  is computed, the rows and the columns  $i$  through  $n$  are permuted so that the largest entry in this submatrix is on the diagonal. This is called “Gaussian elimination with complete pivoting,” or GECP for short.*

### Cholesky Factorization

The Cholesky factorization of a symmetric and positive definite matrix  $A$ , is defined as  $A = LL^T$ , where  $L$  is a lower triangular matrix as follows:

---

```

For  $k = 1, \dots, n$ 

$$l_{kk} = \sqrt{a_{kk} - \sum_{i=1}^{k-1} l_{ki}^2};$$

For  $j = k+1, \dots, n$ 

$$l_{jk} = \frac{a_{kj} - \sum_{i=1}^{k-1} l_{ki} l_{ji}}{l_{kk}}$$

End for
End for
◆

```

---

The Cholesky factorization method for solving a symmetric positive definite system  $Ax = b$  by using the factorization  $A = LDL^T$  computes the elements of the diagonal matrix  $D$  and the lower triangular matrix  $L$  as follows.

---

```

For  $j = 1, \dots, n$ 

$$c_{jj} = a_{jj} - \sum_{s=1}^{j-1} d_s l_{js}^2, \quad d_j = c_{jj}$$

For  $i = j+1, \dots, n$ 

$$c_{ij} = a_{ij} - \sum_{s=1}^{j-1} d_s l_{is} l_{js}, \quad l_{ij} = c_{ij} / d_j$$

End for
End for
◆

```

---

When  $A$  is symmetric and positive definite, then the Cholesky factorization requires about  $n^3/6$  multiplications per iteration. If  $A$  is indefinite, then the Cholesky factorization may not exist. Even if it does exist, numerically it is unstable when it is applied to such matrices, in the sense that the elements of  $L$  can become arbitrarily large. In this case, the *modified Cholesky factorization* may be used, as described in Gill, Murray, and Wright (1981) or in Moré and Sorensen (1984).

## Modified Cholesky Factorization

In this factorization the matrix  $A$  is modified during the course of the factorization in such a way that all the elements of  $D$  are sufficiently positive and the elements of  $D$  and  $L$  are not too large. To control the quality of the modification, two positive parameters  $\delta$  and  $\beta$  are selected. They require that during the computation of the  $j$ -th columns of  $L$  and  $D$  in the Cholesky factorization, the following bounds be satisfied:  $d_j \geq \delta$  and  $|m_{ij}| \leq \beta$ ,  $i = j+1, \dots, n$ , where  $m_{ij} = l_{ij}\sqrt{d_j}$ . To satisfy these bounds, we only need to change one step in the Cholesky factorization algorithm. Indeed, the formula for computing the diagonal element  $d_j$  is replaced by

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

To verify that the above bounds hold, observe that in the Cholesky factorization,  $c_{ij} = l_{ij}d_j$  and therefore

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

Details on the modified Cholesky factorization can be found in (Gill, Murray, & Wright, 1981).

## QR Decomposition

A QR decomposition of a matrix  $A \in \mathbb{R}^{n \times m}$  is given by  $A = QR$ , where  $Q \in \mathbb{R}^{n \times n}$  is an orthogonal matrix and  $R \in \mathbb{R}^{n \times m}$  is an upper-triangular matrix. If  $n \geq m$ , then  $R$  has the form

$$R = \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix},$$

where  $\hat{R} \in \mathbb{R}^{m \times m}$  is an upper-triangular square matrix. Therefore,  $A$  can be expressed as  $A = \hat{Q}\hat{R}$ , where  $\hat{Q}$  is the matrix formed by the first  $m$  columns of  $Q$ . Now, if  $\hat{Q} = [q_1 \ q_2 \ \cdots \ q_m]$ , then  $A = \hat{Q}\hat{R}$  can be written as

$$Ax = \hat{Q}\hat{R}\hat{x} = \hat{Q}\hat{x} = \sum_{i=1}^m \hat{x}_i q_i.$$

In other words, if  $A$  has full column rank  $m$ , then the first  $m$  columns of  $Q$  form an orthogonal basis for the range of  $A$ .

## Singular Value Decomposition

Suppose  $A \in \mathbb{R}^{m \times n}$  with  $\text{rank}(A) = r$ . Then  $A$  can be factored as  $A = U\Sigma V^T$ , where  $U \in \mathbb{R}^{m \times r}$  satisfies  $U^T U = I$ ,  $V \in \mathbb{R}^{n \times r}$  satisfies  $V^T V = I$  and  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r)$  with  $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ . The columns of  $U$  are called *left singular vectors* of  $A$ , the columns of  $V$  are called *right singular vectors* of  $A$ , and the numbers  $\sigma_i$  are the *singular value*. The matrix decomposition  $A = U\Sigma V^T$  is known as the singular value decomposition (SVD) of  $A$ . Observe that

$$AA^T = U \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} U^T \quad \text{and} \quad A^T A = V \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} V^T.$$

Therefore, the singular values of  $A$  are the positive square roots of the nonzero eigenvalues of  $AA^T$  (or  $A^T A$ ), the  $i$ -th left singular vector  $u_i$  is the  $i$ -th eigenvector of  $AA^T$ , and the  $i$ -th right singular vector  $v_i$  is the  $i$ -th eigenvector of  $A^T A$ . It has many applications in optimization and in other fields. For example:

1. The  $l_2$  norm and the *Frobenius norm* of matrix  $A \in \mathbb{R}^{m \times n}$  of rank  $r$  are given by

$$\|A\|_2 = \sigma_1, \quad \text{and} \quad \|A\|_F = \sqrt{\sum_{i=1}^r \sigma_i^2}.$$

2. The *condition number* of a nonsingular matrix  $A \in \mathbb{R}^{n \times n}$  is defined as

$$\text{cond}(A) = \kappa(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}.$$

3. The *range and the null space* of a matrix  $A \in \mathbb{R}^{m \times n}$  of rank  $r$  have the form

$$R(A) = \text{span}\{u_1, \dots, u_r\} \text{ and } N(A) = \text{span}\{v_{r+1}, \dots, v_n\}.$$

4. The properties and the computation of the *Moore-Penrose pseudo-inverse* of a matrix  $A \in \mathbb{R}^{m \times n}$  is defined as the matrix  $A^+ \in \mathbb{R}^{n \times m}$  that satisfies the following four conditions:

- (i)  $AA^+A = A$ ,
- (ii)  $A^+AA^+ = A^+$ ,
- (iii)  $(AA^+)^T = AA^+$ ,
- (iv)  $(A^+A)^T = A^+A$ .

Using the SVD of  $A$ , the Moore-Penrose pseudo-inverse of  $A$  can be obtained as

$$A^+ = V\Sigma^+U^T, \text{ where } \Sigma^+ = \text{diag}(\sigma_1^{-1}, \dots, \sigma_r^{-1}).$$

5. For an underdetermined system of linear equations  $Ax = b$ , where  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$  with  $m < n$  and  $b \in R(A)$ , *all its solutions* are characterized by

$$x = A^+b + V_r z,$$

where  $A^+$  is the Moore-Penrose pseudo-inverse of  $A$ ,  $V_r = [v_{r+1} \ v_{r+2} \ \cdots \ v_n] \in \mathbb{R}^{n \times (n-r)}$  composed of the last  $n - r$  columns of the matrix  $V$  which is obtained by computing the SVD of  $A$ , and  $z \in \mathbb{R}^{n-r}$  is an *arbitrary* vector. Observe that  $A^+b$  is a solution of the system  $Ax = b$ , while  $V_r z$  belongs to the null space of  $A$ . In other words, the vector  $z$  parameterizes all the solutions of an underdetermined system of linear equations.

## Spectral Decomposition (Symmetric Eigenvalue Decomposition)

Suppose  $A \in \mathbb{R}^{n \times n}$  is a real symmetric matrix. Then  $A$  can be factored as  $A = Q\Lambda Q^T$ , where  $Q = [q_1, \dots, q_n] \in \mathbb{R}^{n \times n}$  is an orthogonal matrix with columns  $q_i$ ,  $i = 1, \dots, n$ , as eigenvectors of  $A$  and  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ , where  $\lambda_i$  are the eigenvalues of  $A$ . When  $A$  is positive definite as well as symmetric, this spectral decomposition is identical to the singular value decomposition. In this case, the singular values  $\sigma_i$  and the eigenvalues  $\lambda_i$  coincide.

## Elementary Matrices

Matrices of the form  $I - uv^T$ , where  $u, v \in \mathbb{R}^n$  such that  $v^T u \neq 1$ , are called elementary matrices. All such matrices are nonsingular and

$$(I - uv^T)^{-1} = I - \frac{uv^T}{v^T u - 1}.$$

The inverses of the elementary matrices are elementary matrices. The elementary matrices are used for (i) interchanging the rows  $i$  and  $j$ , (ii) multiplying the row (column)  $i$  by  $\alpha \neq 0$ , (iii) adding a multiple of the row (column)  $i$  to the row (column)  $j$ .

An *elementary lower-triangular matrix* is defined as an  $n \times n$  triangular matrix of the form

$$T_k = I - c_k e_k^T,$$

where  $c_k$  is a column with zero in the first  $k$  positions. In particular, if

$$c_k = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \mu_{k+1} \\ \vdots \\ \mu_n \end{bmatrix}, \text{ then } T_k = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & -\mu_{k+1} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -\mu_n & 0 & \cdots & 1 \end{bmatrix}.$$

Observe that

$$T_k^{-1} = I + c_k e_k^T = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \mu_{k+1} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mu_n & 0 & \cdots & 1 \end{bmatrix},$$

which is also an elementary lower-triangular matrix.

The importance of the elementary lower-triangular matrices lies in the fact that all the row operations needed to annihilate the entries below the  $k$ -th pivot can be accomplished with one multiplication by  $T_k$ . If

$$A_{k-1} = \begin{bmatrix} * & * & \cdots & \alpha_1 & * & \cdots & * \\ 0 & * & \cdots & \alpha_2 & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \alpha_k & * & \cdots & * \\ 0 & 0 & \cdots & \alpha_{k+1} & * & \cdots & * \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_n & * & \cdots & * \end{bmatrix}$$

is the partially triangularized result after  $k - 1$  elimination steps, then

$$\begin{aligned}
T_k A_{k-1} &= (I - c_k e_k^T) A_{k-1} = A_{k-1} - c_k e_k^T A_{k-1} \\
&= \begin{bmatrix} * & * & \cdots & \alpha_1 & * & \cdots & * \\ 0 & * & \cdots & \alpha_2 & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \alpha_k & * & \cdots & * \\ 0 & 0 & \cdots & 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & * & \cdots & * \end{bmatrix}, \text{ where } c_k = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \alpha_{k+1}/\alpha_k \\ \vdots \\ \alpha_n/\alpha_k \end{bmatrix}
\end{aligned}$$

contains the multipliers used to annihilate the entries below  $\alpha_k$ . Therefore, if no row interchanges are required, then reducing the nonsingular matrix  $A$  to an upper triangular matrix  $U$  by the Gaussian elimination is equivalent to executing a sequence of  $n - 1$  left-hand multiplications with elementary lower-triangular matrices. That is,  $T_{n-1} \cdots T_2 T_1 A = U$ , and hence  $A = T_1^{-1} T_2^{-1} \cdots T_{n-1}^{-1} U$ . Having in view that  $e_j^T c_k = 0$  whenever  $j \leq k$ , it follows that

$$\begin{aligned}
T_1^{-1} T_2^{-1} \cdots T_{n-1}^{-1} &= (I + c_1 e_1^T)(I + c_2 e_2^T) \cdots (I + c_{n-1} e_{n-1}^T) \\
&= I + c_1 e_1^T + c_2 e_2^T + \cdots + c_{n-1} e_{n-1}^T.
\end{aligned}$$

But

$$c_k e_k^T = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & l_{k+1,k} & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{n,k} & 0 & \cdots & 0 \end{bmatrix},$$

where the  $l_{i,k}$ 's are the multipliers used at the  $k$ -th stage to annihilate the entries below the  $k$ -th pivot. Therefore, from all these developments, it follows that

$$A = LU,$$

where

$$L = I + c_1 e_1^T + c_2 e_2^T + \cdots + c_{n-1} e_{n-1}^T = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & 1 \end{bmatrix},$$

is the lower-triangular matrix with 1's on the main diagonal, and where  $l_{i,j}$  is the multiplier used to annihilate the  $(i, j)$ -position during the Gaussian elimination. Therefore, the factorization  $A = LU$  is the matrix formulation of the Gaussian elimination with no row interchanges.

## Conditioning and Stability

These are two terms used in numerical computations when a problem is solved with an algorithm. Conditioning is a property of the problem, irrespective of its being a linear algebra problem, an optimization or a differential equation. A problem is *well-conditioned* if its solution is not greatly affected by small perturbations to the data that define the problem. Otherwise, it is *ill-conditioned*. On the other hand, stability of an algorithm is a property of the algorithm. An algorithm is *stable* if it is guaranteed to generate accurate answers to well-conditioned problems.

The *condition number* of a nonsingular matrix  $A \in \mathbb{R}^{n \times n}$  denoted as  $\text{cond}(A)$  or  $\kappa(A)$  is defined as  $\text{cond}(A) = \|A\| \|A^{-1}\|$ . If the 2-norm is used, then  $\kappa(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$ , where  $\sigma_{\max}(A)$  and  $\sigma_{\min}(A)$  are the largest and the smallest singular values of  $A$ , respectively. For normal matrices,  $\kappa(A) = |\lambda_{\max}(A)|/|\lambda_{\min}(A)|$ , where  $\lambda_{\max}(A)$  and  $\lambda_{\min}(A)$  are the largest and the smallest eigenvalues of  $A$ , respectively. The matrix  $A$  is *well-conditioned* if  $\kappa(A)$  is small (close to 1). The matrix  $A$  is *ill-conditioned* if  $\kappa(A)$  is large.

For general linear systems  $Ax = b$ , where  $A \in \mathbb{R}^{n \times n}$ , the condition number of the matrix can be used to see the conditioning of the system. If the matrix  $A$  is perturbed to  $\bar{A}$  and  $b$  to  $\bar{b}$  and consider  $\bar{x}$  as the solution of the perturbed system  $\bar{A}\bar{x} = \bar{b}$ , it can be shown that (Golub & Van Loan, 1996)

$$\frac{\|x - \bar{x}\|}{\|x\|} \approx \kappa(A) \left[ \frac{\|A - \bar{A}\|}{\|A\|} + \frac{\|b - \bar{b}\|}{\|b\|} \right].$$

Therefore, a large condition number  $\kappa(A)$  indicates that the problem  $Ax = b$  is ill-conditioned, while a small value shows well-conditioning of the problem.

To see the significance of the stability of an algorithm, let us consider the linear system  $Ax = b$  solved by means of the Gaussian elimination with partial pivoting and triangular substitution. It is shown that this algorithm gives a solution  $\bar{x}$  whose relative error is approximately

$$\frac{\|x - \bar{x}\|}{\|x\|} \approx \kappa(A) \frac{\text{gr}(A)}{\|A\|} u,$$

where  $\text{gr}(A)$  is the size of the largest element that arises in  $A$  during the execution of the Gaussian elimination with partial pivoting and  $u$  is the unit roundoff. (In double-precision IEEE arithmetic,  $u$  is about  $1.1 \times 10^{-16}$ .) In the worst case, it can be shown that  $\text{gr}(A)/\|A\|$  may be around  $2^{n-1}$ , which indicates that the Gaussian elimination with partial pivoting is an unstable algorithm (Demmel, 1997). However, in practice, after decades of numerical experience with the Gaussian elimination with partial pivoting, it was noticed that  $\text{gr}(A)$  is growing slowly as a function of  $n$ . In practice,  $\text{gr}(A)$  is almost always  $n$  or less. The average behavior seems to be  $n^{2/3}$  or perhaps even  $n^{1/2}$  (Trefethen & Schreiber, 1990). Therefore, the Gaussian elimination with partial pivoting is stable for all practical purposes. However, the Gaussian elimination without pivoting is definitely unstable. For the system  $Ax = b$  where  $A$  is a symmetric and positive definite matrix, the Cholesky factorization method with triangular substitution is a stable algorithm.

## Determinant of a Matrix

The *determinant* is a scalar defined only for square matrices. A *permutation*  $p = (p_1, p_2, \dots, p_n)$  of the numbers  $(1, 2, \dots, n)$  is simply any rearrangement of these numbers. The *sign of a permutation*  $p$  is defined to be the number

$$\sigma(p) = \begin{cases} +1, & \text{if } p \text{ can be restored to the natural order by an even number of interchanges,} \\ -1, & \text{if } p \text{ can be restored to the natural order by an odd number of interchanges.} \end{cases}$$

Let  $A = (a_{ij}) \in \mathbb{R}^{n \times n}$  be an arbitrary matrix, where all its elements  $a_{ij}$  are real numbers. The *determinant* of  $A$  is defined to be the scalar

$$\det(A) = \sum_p \sigma(p) a_{1p_1} a_{2p_2} \cdots a_{np_n},$$

where the sum is taken over the  $n!$  permutations  $p = (p_1, p_2, \dots, p_n)$  of  $(1, 2, \dots, n)$ . ( $n! = 1 \times 2 \times \dots \times n$ . For example  $3! = 1 \times 2 \times 3 = 6$ .) Each term  $a_{1p_1} a_{2p_2} \cdots a_{np_n}$  contains exactly one entry from each row and from each column of  $A$ .

Some properties of the determinants are as follows:

1. The determinant of a diagonal matrix  $\det[\text{diag}(x_1, x_2, \dots, x_n)] = x_1 x_2 \dots x_n$ .
2. Let  $I_n$  be the identity matrix of order  $n$ . Then  $\det(I_n) = 1$ .
3. The determinant of a triangular matrix is the product of its diagonal entries.
4. For any matrix  $A \in \mathbb{R}^{n \times n}$  and constant  $c$ ,  $\det(cA) = c^n \det(A)$ .
5. Suppose that  $B$  is obtained from  $A$  by swapping two of the rows (columns) of  $A$ . Then  $\det(B) = -\det(A)$ .
6. If there is a row (column) of  $A$  all zero, then  $\det(A) = 0$ .
7. If two rows (columns) of  $A$  are equal, then  $\det(A) = 0$ .
8.  $\det(A^T) = \det(A)$ .
9.  $\det(A^{-1}) = 1/\det(A)$ .
10.  $\det(AB) = \det(A)\det(B)$ .
11. If  $\lambda_1, \lambda_2, \dots, \lambda_n$  are the eigenvalues of  $A \in \mathbb{R}^{n \times n}$ , then  $\det(A) = \lambda_1 \lambda_2 \cdots \lambda_n$ .

For a matrix  $A \in \mathbb{R}^{n \times n}$ , the polynomial  $p(\lambda) = \det(A - \lambda I)$  is called the *characteristic polynomial* of  $A$ . The set of all the eigenvalues of  $A$  is the set of all the roots of its characteristic polynomial. The *Cayley-Hamilton theorem* says that  $p(A) = 0$ .

Let  $I_n$  be the identity matrix of order  $n$  and  $u_1, u_2 \in \mathbb{R}^n$  arbitrary vectors. Then,

$$\det(I_n + u_1 u_2^T) = 1 + u_1^T u_2.$$

Let  $I_n$  be the identity matrix of order  $n$  and  $u_1, u_2, u_3, u_4 \in \mathbb{R}^n$  arbitrary vectors. Then,

$$\det(I_n + u_1 u_2^T + u_3 u_4^T) = (1 + u_1^T u_2)(1 + u_3^T u_4) - (u_1^T u_4)(u_2^T u_3).$$

Indeed,

$$I_n + u_1 u_2^T + u_3 u_4^T = (I_n + u_1 u_2^T) \left[ I_n + (I_n + u_1 u_2^T)^{-1} u_3 u_4^T \right].$$

Therefore,

$$\begin{aligned}
\det(I_n + u_1 u_2^\top + u_3 u_4^\top) &= \det(I_n + u_1 u_2^\top) \det \left[ I_n + (I_n + u_1 u_2^\top)^{-1} u_3 u_4^\top \right] \\
&= (1 + u_1^\top u_2) \left[ 1 + u_4^\top (I_n + u_1 u_2^\top)^{-1} u_3 \right] \\
&= (1 + u_1^\top u_2) \left[ 1 + u_4^\top \left( I_n - \frac{u_1 u_2^\top}{1 + u_1^\top u_2} \right) u_3 \right] \\
&= (1 + u_1^\top u_2) (1 + u_3^\top u_4) - (u_1^\top u_4) (u_2^\top u_3).
\end{aligned}$$

*Determinant of the quasi-Newton BFGS update.*

(1) Let

$$B_{k+1} = B_k - \frac{B_k s_k s_k^\top B_k}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k},$$

be the BFGS updating of the matrix  $B_k$ , where  $B_k \in \mathbb{R}^{n \times n}$  and  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^\top s_k > 0$ . Then,

$$\begin{aligned}
\det(B_{k+1}) &= \det \left[ B_k \left( I - \frac{s_k s_k^\top B_k}{s_k^\top B_k s_k} + \frac{B_k^{-1} y_k y_k^\top}{y_k^\top s_k} \right) \right] \\
&= \det(B_k) \det \left( I - s_k \frac{(B_k s_k)^\top}{s_k^\top B_k s_k} + B_k^{-1} y_k \frac{y_k^\top}{y_k^\top s_k} \right) = \det(B_k) \frac{y_k^\top s_k}{s_k^\top B_k s_k}.
\end{aligned}$$

(2) Consider

$$B_{k+1} = \delta_k \left[ B_k - \frac{B_k s_k s_k^\top B_k}{s_k^\top B_k s_k} \right] + \gamma_k \frac{y_k y_k^\top}{y_k^\top s_k},$$

where  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^\top s_k > 0$  and  $\delta_k, \gamma_k \in \mathbb{R}$  are two known nonzero scalar parameters. Then,

$$\det(B_{k+1}) = \det(B_k) \frac{y_k^\top s_k}{s_k^\top B_k s_k} \delta_k^{n-1} \gamma_k.$$

## Trace of a matrix

The trace of a square matrix  $A = (a_{ij}) \in \mathbb{R}^{n \times n}$  is

$$\text{trace}(A) = \text{tr}(A) = \sum_{i=1}^n a_{ii}.$$

The trace satisfies:

1.  $\text{tr}(A^\top) = \text{tr}(A)$ .
2.  $\text{tr}(AB) = \text{tr}(BA)$ .
3.  $\text{tr}(\alpha A + \beta B) = \alpha \text{tr}(A) + \beta \text{tr}(B)$ ,  $\alpha, \beta \in \mathbb{R}$ .

If  $\lambda_1, \lambda_2, \dots, \lambda_n$  are the eigenvalues of  $A \in \mathbb{R}^{n \times n}$ , then  $\text{trace}(A) = \lambda_1 + \lambda_2 + \dots + \lambda_n$ . If

$$A = (a_{ij}) \in \mathbb{R}^{m \times n}, \text{ then } \text{tr}(A^T A) = \sum_{i=1}^m \sum_{j=1}^n a_{ij}^2.$$

Let  $B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$  be the BFGS updating of the matrix  $B_k$ , where  $B_k \in \mathbb{R}^{n \times n}$  and  $s_k, y_k \in \mathbb{R}^n$  so that  $y_k^T s_k > 0$ . Then,

$$\text{tr}(B_{k+1}) = \text{tr}(B_k) - \frac{\|B_k s_k\|^2}{s_k^T B_k s_k} + \frac{\|y_k\|^2}{y_k^T s_k}.$$

## A2. Elements of Analysis

Let  $\{x_k\}$  be a sequence of points from  $\mathbb{R}^n$ . A sequence  $\{x_k\}$  converges to a point  $x^*$ , written as  $\lim_{k \rightarrow \infty} x_k = x^*$ , if for any  $\epsilon > 0$  there exists an index  $K$  so that  $\|x_k - x^*\| \leq \epsilon$  for all  $k \geq K$ . Given an index set  $\bar{K} \subset \{1, 2, \dots\}$ , a subsequence of  $\{x_k\}$  corresponding to  $\bar{K}$  can be defined and denoted by  $\{x_k\}_{k \in \bar{K}}$ . Consider a convergent sequence  $\{x_k\}$  with the limit  $x^*$ . Then, any subsequence of  $\{x_k\}$  also converges to  $x^*$ . A convergent sequence has only one limit. A sequence  $\{x_k\}$  in  $\mathbb{R}^n$  is bounded if there exists a number  $B \geq 0$  such that  $\|x_k\| \leq B$  for all  $k = 1, 2, \dots$ . Every convergent sequence is bounded. A sequence  $\{x_k\}$  in  $\mathbb{R}^n$  is uniformly bounded away from zero if there exists  $\epsilon > 0$  such that  $|x_k| \geq \epsilon$  for any  $k \geq 1$ .

**Theorem A2.1** (Bolzano-Weierstrass Theorem). *Each bounded sequence in  $\mathbb{R}^n$  has a convergent subsequence.* ◆

The point  $x^* \in \mathbb{R}^n$  is an accumulation point or a limit point or a cluster point for the sequence  $\{x_k\}$  if there is an infinite set of indices  $k_1, k_2, k_3, \dots$  so that the subsequence  $\{x_{k_i}\}_{i=1,2,3,\dots}$  converges to  $x^*$ , i.e.,  $\lim_{i \rightarrow \infty} x_{k_i} = x^*$ . A sequence is a Cauchy sequence if for any  $\epsilon > 0$  there exists an integer  $K > 0$  so that  $\|x_k - x_m\| \leq \epsilon$  for all the indices  $k \geq K$  and  $m \geq K$ . A sequence converges if and only if it is a Cauchy sequence.

A function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  is continuous at  $x \in \mathbb{R}^n$  if for all  $\epsilon > 0$  there exists a  $\delta(\epsilon, x) > 0$  so that for any  $y \in \mathbb{R}^n$ ,  $\|y - x\|_2 \leq \delta(\epsilon, x) \Rightarrow \|f(y) - f(x)\|_2 \leq \epsilon$ . The continuity can be described in terms of limits: whenever the sequence  $\{x_k\}$  in  $\mathbb{R}^n$  converges to a point  $x \in \mathbb{R}^n$ , the sequence  $\{f(x_k)\}$  in  $\mathbb{R}^m$  converges to  $f(x)$ , i.e.,  $\lim_{k \rightarrow \infty} f(x_k) = f(\lim_{k \rightarrow \infty} x_k)$ . A function  $f$  is continuous if it is continuous at every point in  $\mathbb{R}^n$ .

A function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  is uniformly continuous at  $x \in \mathbb{R}^n$  if for all  $\epsilon > 0$  there exists a  $\delta(\epsilon) > 0$  so that for any  $y \in \mathbb{R}^n$ ,  $\|y - x\|_2 \leq \delta(\epsilon) \Rightarrow \|f(y) - f(x)\|_2 \leq \epsilon$ . It is obvious that a uniformly continuous function is continuous.

If  $\{x_k\}$  is a Cauchy sequence and  $f$  is uniformly continuous on a convex domain, then  $\{f(x_k)\}$  is also a Cauchy sequence.

A function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  is bounded if there exists a constant  $C \geq 0$  so that  $\|f(x)\| \leq C$  for all  $x \in \mathbb{R}^n$ .

A continuous function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is coercive if  $\lim_{\|x\| \rightarrow \infty} f(x) = +\infty$ . This means that for any constant  $M$  there must be a positive number  $R_M$  such that  $f(x) \geq M$  whenever  $\|x\| \geq R_M$ . In particular, the values of  $f(x)$  cannot remain bounded on a set in  $\mathbb{R}^n$  which is not bounded. For  $f(x)$  to be coercive, it is not sufficient that  $f(x) \rightarrow \infty$  as each coordinate tends to  $\infty$ . Rather,  $f(x)$  must become infinite

along any path for which  $\|x\|$  becomes infinite. If  $f(x)$  is coercive, then  $f(x)$  has at least one global minimizer, and these minimizers can be found among the critical points of  $f(x)$ .

Let  $f: \mathbb{R} \rightarrow \mathbb{R}$  be a real-valued function of a real variable. The *first derivative* is defined by

$$f'(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}.$$

The *second derivative* is defined by

$$f''(x) = \lim_{\epsilon \rightarrow 0} \frac{f'(x + \epsilon) - f'(x)}{\epsilon}.$$

The *directional derivative* of a function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  in the direction  $p \in \mathbb{R}^n$  is given by

$$D(f(x); p) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon p) - f(x)}{\epsilon}.$$

Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a continuously differentiable function. The conditions which characterize a minimum can be expressed in terms of the *gradient*  $\nabla f(x)$  with the first partial derivatives defined as

$$\nabla f(x) = \left[ \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right]^T$$

and of  $n \times n$  *Hessian* matrix  $\nabla^2 f(x)$  with the second partial derivatives whose  $(i, j)$ -th element is

$$(\nabla^2 f(x))_{ij} = \partial^2 f(x) / \partial x_i \partial x_j, \quad i, j = 1, \dots, n.$$

When  $f$  is twice continuously differentiable, the Hessian matrix is always symmetric. As a simple example, let us consider the quadratic function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $f(x) = (1/2)x^T Ax + b^T x + a$ , where  $A \in \mathbb{R}^{n \times n}$  is a symmetric matrix. Then,  $\nabla f(x) = Ax + b$ . The Hessian of  $f$  is given by  $\nabla^2 f(x) = A$ , i.e., the second order approximation of a quadratic function is itself.

If  $f$  is continuously differentiable in a neighborhood of  $x$ , then

$$D(f(x); p) = \nabla f(x)^T p.$$

**Theorem A2.2 (Mean Value Theorem)** Given a continuously differentiable function  $f: \mathbb{R} \rightarrow \mathbb{R}$  and two real numbers  $x_1$  and  $x_2$  that satisfy  $x_2 > x_1$ , then

$$f(x_2) = f(x_1) + f'(\xi)(x_2 - x_1)$$

for some  $\xi \in (x_1, x_2)$ .

For a multivariate function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  the mean value theorem says that for any vector  $d \in \mathbb{R}^n$ ,

$$f(x + d) = f(x) + \nabla f(x + \alpha d)^T d$$

for some  $\alpha \in (0, 1)$ . ◆

**Theorem A2.3 (Taylor's Theorem).** If  $f$  is continuously differentiable in a domain containing the line segment  $[x_1, x_2]$ , then there is a  $\theta$ ,  $0 \leq \theta \leq 1$ , so that

$$f(x_2) = f(x_1) + \nabla f(\theta x_1 + (1 - \theta)x_2)^T (x_2 - x_1).$$

Moreover, iff is twice continuously differentiable in a domain containing the line segment  $[x_1, x_2]$ , then there is a  $\theta$ ,  $0 \leq \theta \leq 1$ , so that

$$f(x_2) = f(x_1) + \nabla f(x_1)^T (x_2 - x_1) + \frac{1}{2} (x_2 - x_1)^T \nabla^2 f(\theta x_1 + (1 - \theta)x_2) (x_2 - x_1). \quad \blacklozenge$$

For twice continuously differentiable functions  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  and for any vector  $d \in \mathbb{R}^n$ , one form of the Taylor theorem is

$$f(x + d) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x + \alpha d) d,$$

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

The *level set* of a function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  at level  $c$  is the set of points

$$S = \{x : f(x) = c\}.$$

**Theorem A2.4** Suppose that  $f$  is continuously differentiable. Then the vector  $\nabla f(x_0)$  is orthogonal to the tangent vector to an arbitrary smooth curve passing through  $x_0$  on the level set determined by  $f(x) = f(x_0)$ .  $\blacklozenge$

Consider the minimization problem  $\min_{x \in \mathbb{R}^n} f(x)$  with the minimum point  $x^*$ . Let  $e = x - x^*$  be the error and  $B(\delta)$  the ball of the radius  $\delta$  about  $x^*$ .

**Theorem A2.5** Suppose that  $f$  is twice continuously differentiable with  $\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq \gamma \|x - y\|$ ,  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite. Then there is  $\delta > 0$  such that for all  $x \in B(\delta)$ ,

$$\begin{aligned} \|\nabla^2 f(x)\| &\leq 2 \|\nabla^2 f(x^*)\|, \\ \left\| (\nabla^2 f(x))^{-1} \right\| &\leq 2 \left\| (\nabla^2 f(x^*))^{-1} \right\|, \\ \frac{1}{2} \left\| (\nabla^2 f(x^*))^{-1} \right\|^{-1} \|e\| &\leq \|\nabla f(x)\| \leq 2 \|\nabla^2 f(x^*)\| \|e\|. \end{aligned} \quad \blacklozenge$$

In the point  $x_0$  the gradient  $\nabla f(x_0)$  is the direction of the *maximum rate of increase* of  $f$  at  $x_0$ . Since  $\nabla f(x_0)$  is orthogonal to the level set through  $x_0$  determined by  $f(x) = f(x_0)$ , it follows that the direction of the maximum rate of increase of a real-valued differentiable function at a point is orthogonal to the level set of the function through that point.

**Theorem A2.6 (Implicit Function Theorem).** Let  $h: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$  be a function such that:

1.  $h(z^*, 0) = 0$  for some  $z^* \in \mathbb{R}^n$ .
2. The function  $h(., .)$  is continuously differentiable in some neighborhood of  $(z^*, 0)$ .
3.  $\nabla_z h(z, t)$  is nonsingular at the point  $(z, t) = (z^*, 0)$ .

Then, there are the open sets  $N_z \subset \mathbb{R}^n$  and  $N_t \subset \mathbb{R}^m$  containing  $z^*$  and 0, respectively, and a continuous function  $z : N_t \rightarrow N_z$  such that  $z^* = z(0)$  and  $h(z(t), t) = 0$  for all  $t \in N_t$ .  $z(t)$  is uniquely defined. If  $h$  is  $q$  times continuously differentiable with respect to both its arguments for some  $q > 0$ , then  $z(t)$  is also  $q$  times continuously differentiable with respect to  $t$  and

$$\nabla z(t) = -\nabla_t h(z(t), t) [\nabla_z h(z(t), t)]^{-1},$$

for all  $t \in N_t$ . ◆

The implicit function theorem is applied to parameterized systems of linear equations in which  $z$  is obtained as the solution of  $M(t)z = g(t)$ , where  $M(\cdot) \in \mathbb{R}^{n \times n}$  has  $M(0)$  nonsingular and  $g(t) \in \mathbb{R}^n$  (see the algebraic characterization of a tangent space). To apply the theorem, define  $h(z, t) = M(t)z - g(t)$ . If  $M(\cdot)$  and  $g(\cdot)$  are continuously differentiable in some neighborhood of 0, the theorem implies that  $z(t) = M(t)^{-1}g(t)$  is a continuous function of  $t$  in some neighborhood of 0.

### Rates of Convergence

Let  $\{x_k\}$  be a sequence from  $\mathbb{R}^n$  that converges to  $x^* \in \mathbb{R}^n$ . This sequence converges  $Q$ -linear if there is a constant  $r \in (0, 1)$  so that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r$$

for all  $k$  sufficiently large. The convergence is  $Q$ -superlinear if

$$\lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0.$$

The convergence is  $Q$ -quadratic if

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M$$

for all  $k$  sufficiently large, where  $M$  is a positive constant, not necessarily smaller than 1.

Typically, under appropriate assumptions, the quasi-Newton methods for unconstrained optimization converge  $Q$ -superlinearly, whereas Newton's method converges  $Q$ -quadratically. The steepest descent algorithms converge only at a  $Q$ -linear rate, and when the problem is ill-conditioned, the convergence constant  $r$  is close to 1.

### Finite-Difference Derivative Estimates

Let  $f : \mathbb{R} \rightarrow \mathbb{R}$ . Finite-differencing refers to the estimation of  $f'(x)$  using the values of  $f(x)$ . Finite-difference estimates can be obtained from Taylor's series. In one dimension we have

$$f(x + h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(\xi),$$

where  $\xi$  is on the line segment connecting  $x$  and  $x + h$ . From the above equality, we get

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2}hf''(\xi),$$

thus obtaining the approximation

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

This is the most used finite-difference formula. It is called the *forward finite-difference formula*.

Now, let us estimate the error in finite differencing, and determine an acceptable value for  $h$ . For this, observe that part of the error is due to the inaccuracies in the formula itself, called the *truncation error*

$$\text{truncated error} = \frac{1}{2}h|f''(\xi)|.$$

Besides, there are *rounding errors* from the evaluation of  $(f(x+h) - f(x))/h$  which depends on  $\epsilon_{mach}$ , the precision of the computer calculation. Therefore, the rounding error from the evaluation of the function  $f$  in the numerator is

$$(\text{rounding error})_1 \approx |f(x)|\epsilon_{mach}.$$

Hence, the rounding error is obtained as

$$(\text{rounding error})_2 \approx \frac{|f(x)|\epsilon_{mach}}{h} + |f'(x)|\epsilon_{mach},$$

where the first rounding error is magnified by  $1/h$ , and then there is an additional rounding error from the division that is proportional to the result  $f'(x)$ . When  $h$  is small and  $f'(x)$  is not large, then the first term will dominate, thus leading to the estimate

$$\text{rounding error} \approx \frac{|f(x)|\epsilon_{mach}}{h}.$$

Therefore, the total error is a combination of the above truncation error and the rounding error, i.e.,

$$\text{error} \approx \frac{1}{2}h|f''(\xi)| + \frac{|f(x)|\epsilon_{mach}}{h}.$$

The best value of  $h$  is obtained by minimizing the error as a function of  $h$ . Differentiating the error with respect to  $h$  and setting the derivative to zero, we get

$$\frac{1}{2}|f''(\xi)| - \frac{|f(x)|\epsilon_{mach}}{h^2} = 0.$$

Therefore, an estimate for  $h$  is obtained as

$$h = \sqrt{\frac{2|f(x)|\epsilon_{mach}}{|f''(\xi)|}}.$$

When  $f(x)$  and  $f''(\xi)$  are neither large nor small, then a simpler estimation

$$h \approx \sqrt{\epsilon_{mach}}$$

can be used. Observe that if  $h < |x|\epsilon_{mach}$ , then the computed value of  $x + h$  will be equal to  $x$  and the finite-difference estimate will be zero. Thus, in the general case the choice of  $h$  will depend on  $\epsilon_{mach}$ ,  $|x|$  and on the values of  $|f''|$ . Some software packages use this estimation of  $h$  or some simple modification of it which takes into account  $|x|$  or  $|f(x)|$ . see Dennis and Schnabel (1983) and Gill, Murray, and Wright (1981).

A more accurate finite-difference formula is given by the *central finite-difference* formula

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{12}h^2(f'''(\xi_1) + f'''(\xi_2)).$$

It can be derived using the Taylor series for  $f(x+h)$  and  $f(x-h)$  about point  $x$ . Higher derivatives can also be obtained by finite differencing. For example,

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{1}{24}h^2(f^{(4)}(\xi_1) + f^{(4)}(\xi_2))$$

can be obtained from the Taylor series for  $f(x+h)$  and  $f(x-h)$  about point  $x$ .

For multidimensional functions  $F(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , the derivatives can be estimated by applying the finite-difference formulas to each component of the gradient or the Hessian matrix

$$[\nabla f(x)]_j \approx \frac{f(x+he_j) - f(x)}{h},$$

where  $e_j$  is the  $j$ -th column of the unity matrix.

If the gradient is known, then the Hessian can be approximated as

$$[\nabla^2 f(x)]_{j,k} = \frac{\partial^2 f(x)}{\partial x_j \partial x_k} \approx \frac{[\nabla f(x+he_k) - \nabla f(x)]_j}{h}.$$

Special finite-differencing techniques to approximate sparse Hessian matrices were developed by Curtis, Powell, and Reid (1974) and by Powell and Toint (1979).

In this presentation it was assumed that the rounding errors were proportional to the machine precision  $\epsilon_{mach}$ . More generally, the rounding errors will be proportional to the accuracy  $\epsilon_f$  with which the function  $f$  can be computed, which may be larger than the machine precision. In this case, the formula for the estimation of  $h$  will include  $\epsilon_f$  as well as  $\epsilon_{mach}$ .

## Automatic Differentiation

The automatic differentiation is a computational technique implemented in software tools which apply rules for differentiating arithmetic operations, functions of functions, etc. These tools can compute derivatives of the first and higher order. The automatic differentiation is different from software for the symbolic differentiation which operates on the mathematical formulae and produces the corresponding formulae for derivatives with respect to the chosen variables. The automatic differentiation takes a user-supplied program for evaluating the function  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  and, using

the rules of calculus, determines the corresponding derivative values. In the following, a short description of a method for the automatic differentiation which generates the function values, its gradient, and the Hessian in a certain point  $x \in \mathbb{R}^n$  is given.

Every variable  $x_i$ ,  $i = 1, \dots, n$ , is represented by a triplet  $X = (x_i, x'_i, x''_i)$ , where  $x_i \in \mathbb{R}$ ,  $x'_i \in \mathbb{R}^n$  and  $x''_i \in \mathbb{R}^{n \times n}$ . Denote the set of all these triplets by  $T^n$ . On  $T^n$ , for any  $U$  and  $V$  from  $T^n$ , the following arithmetic operations can be defined:

$$\begin{aligned} U + V &= (u + v, \ u' + v', \ u'' + v''), \\ U - V &= (u - v, \ u' - v', \ u'' - v''), \\ U^*V &= \left( uv, \ uv' + vu', \ uv'' + u'v'^T + v'u'^T + vu'' \right), \\ U/V &= \left( u/v, \ (vu' - uv')/v^2, \ \left( v^2u'' - v(v'u'^T + u'v'^T) + 2uv'v'^T - uvv'' \right)/v^3 \right), \end{aligned}$$

for  $v \neq 0$ .

An independent variable in  $T^n$  is represented as  $X_i = (x_i, \ e_i, \ 0)$ , where  $e_i$  is the  $i$ -th column of the unitary matrix and 0 is an  $n \times n$  matrix. All the constants  $c$  are represented by  $C = (c, \ 0, \ 0)$ . Let  $C = (c, \ 0, \ 0)$  be a constant and the triplet  $U = (u, \ u', \ u'') \in T^n$ . Then,

$$\begin{aligned} C + U &= U + C = (c + u, \ u', \ u''), \\ C - U &= (c - u, \ -u', \ -u''), \\ U - C &= (u - c, \ u', \ u''), \\ C^*U &= U^*C = (c^*u, \ cu', \ cu''), \\ C/U &= \left( c/u, \ -cu'/u^2, \ \left( 2cu'u'^T - cuu'' \right)/u^3 \right), \\ U/C &= (u/c, \ u'/c, \ u''/c), \end{aligned}$$

for  $c \neq 0$  and  $u \neq 0$ .

Consider the following example:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

for which let us compute  $f(x_0)$ ,  $\nabla f(x_0)$  and  $\nabla^2 f(x_0)$ , where  $x_0 = [-1.2, 1]^T$ . For this, the point  $x_0$  is represented in  $T^2$  as

$$\begin{aligned} X_1 &= \left( -1.2, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \right), \\ X_1 &= \left( 1.0, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \right). \end{aligned}$$

Taking into consideration the above arithmetic operations, the following steps determine  $f(x_0)$ ,  $\nabla f(x_0)$  and  $\nabla^2 f(x_0)$ , as in the table below

Step	Operation	Triplet		
		$f(x_0)$	$\nabla f(x_0)$	$\nabla^2 f(x_0)$
1.	$Y = X_1 * X_1$	1.44	$\begin{bmatrix} -2.4 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$
2.	$Y = X_2 - Y$	-0.44	$\begin{bmatrix} -2.4 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -2 & 0 \\ 0 & 0 \end{bmatrix}$
3.	$Y = Y * Y$	0.1936	$\begin{bmatrix} -2.112 \\ -0.88 \end{bmatrix}$	$\begin{bmatrix} 13.28 & 4.8 \\ 4.8 & 2 \end{bmatrix}$
4.	$Y = 100 * Y$	19.36	$\begin{bmatrix} -211.2 \\ -88 \end{bmatrix}$	$\begin{bmatrix} 1328 & 480 \\ 480 & 200 \end{bmatrix}$
5.	$Z = 1 - X_1$	2.2	$\begin{bmatrix} -1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
6.	$Z = Z * Z$	4.84	$\begin{bmatrix} -4.4 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$
7.	$Y = Y + Z$	24.20	$\begin{bmatrix} -215.6 \\ -88 \end{bmatrix}$	$\begin{bmatrix} 1330 & 480 \\ 480 & 200 \end{bmatrix}$

If  $g : \mathbb{R} \rightarrow \mathbb{R}$  is a twice continuous differentiable function, then its extension on  $T^n$  is the triplet

$$g(U) = (g(u, u', u'') = (g(u), g'(u)u', g'(u)u'' + g''(u)u'u'^T)).$$

For example,

$$\sin(U) = \sin(u, u', u'') = (\sin(u), u' \cos(u), u'' \cos(u) - u'u'^T \sin(u)).$$

## Order Notation

*Order notation* is a concept used to see how the members of a sequence behave when we get far enough along in the sequence. Let us consider two nonnegative sequences of scalars  $\{\eta_k\}$  and  $\{\theta_k\}$ .  $\eta_k = o(\theta_k)$  if the sequence of ratios  $\{\eta_k/\theta_k\}$  approaches zero, i.e.,  $\lim_{k \rightarrow \infty} \eta_k/\theta_k = 0$ .  $\eta_k = O(\theta_k)$  if there is a positive constant  $c$  so that  $|\eta_k| \leq c|\theta_k|$  for all  $k$  sufficiently large. If  $\eta : \mathbb{R} \rightarrow \mathbb{R}$  is a function, then  $\eta(t) = o(t)$  to specify that the ratio  $\eta(t)/t$  approaches zero either as  $t \rightarrow 0$  or  $t \rightarrow \infty$ . Similarly,  $\eta(t) = O(t)$  if there is a constant  $c$  so that  $|\eta(t)| \leq ct$  for all  $t \in \mathbb{R}$ . A slight variant of the above definitions is as follows.  $\eta_k = o(1)$  to specify that  $\lim_{k \rightarrow \infty} \eta_k = 0$ . Similarly,  $\eta_k = O(1)$  to indicate that there is a constant  $c$  so that  $|\eta_k| \leq c$  for all  $k$ . Sometimes, in the above definitions, there are vectors or matrices as arguments. In these cases, the definitions apply to the norms of these quantities. For instance, if  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , then  $f(x) = O(\|x\|)$  if there is a positive constant  $c$  so that  $\|f(x)\| \leq c\|x\|$  for all  $x$  in the domain of  $f$ .

### A3. Elements of Topology in the Euclidian Space $\mathbb{R}^n$

The *open ball* of radius  $\varepsilon$  centered at  $x^*$  is defined as the set  $B(x^*, \varepsilon) = \{x \in \mathbb{R}^n : \|x^* - x\| < \varepsilon\}$  in any norm.

A subset  $D \subset \mathbb{R}^n$  is *open* if for every  $x \in D$  there exists a positive number  $\varepsilon > 0$  so that the ball of radius  $\varepsilon$  centered at  $x$  is contained in  $D$ , i.e.,  $\{y \in \mathbb{R}^n : \|y - x\| \leq \varepsilon\} \subset D$ . The intersection of a finite number of open sets is open. Any union of open sets is open.

A point  $x \in \mathbb{R}^n$  is an *interior point* of the set  $D$  if there is an open ball  $B(x, \varepsilon)$  so that  $B(x, \varepsilon) \subset D$ . The *interior* of a set  $D$ , denoted by  $\text{int}D$ , is the set of the interior points of  $D$ . The interior of a set is the largest open set contained in  $D$ .

A point  $x \in \mathbb{R}^n$  is an *exterior point* of  $D$  if it is an interior point of  $\mathbb{R}^n \setminus D$ . Notice that the set  $D$  is open if every point of  $D$  is an interior point of  $D$ . Obviously, if  $D$  is open, then  $\text{int}D = D$ .

A point  $\tilde{x}$  is said to be a *limit point* of the set  $D$  if every open ball  $B(\tilde{x}, \varepsilon)$  contains a point  $x \neq \tilde{x}$  so that  $x \in D$ . Note that  $\tilde{x}$  does not necessarily have to be an element of  $D$  for being a limit point of  $D$ .

The set  $D$  is *closed* if for all the possible sequences of points  $\{x_k\}$  in  $D$ , all the limit points of  $\{x_k\}$  are elements of  $D$ . The union of a finite number of closed sets is closed. Any intersection of closed sets is closed.

The set  $D$  is *bounded* if there is some real number  $M > 0$  so that  $\|x\| \leq M$  for all  $x \in D$ .

The set  $D$  is *compact* if every sequence  $\{x_k\}$  of points in  $D$  has at least one limit point and all such limit points are in  $D$ . A central result in topology is that in  $\mathbb{R}^n$  the set  $D$  is compact if it is both closed and bounded.

**Theorem A3.1** (*Weierstrass Extreme Value Theorem.*) Every continuous function on a compact set attains its extreme values on that set. ◆

The *closure* of the set  $D$  is the set  $\text{cl}(D) = D \cup L$ , where  $L$  denotes the set of all the limit points of  $D$ .

For a given point  $x \in \mathbb{R}^n$ , a *neighborhood* of  $x$  is an open set containing  $x$ . A useful neighborhood is the open ball of radius  $\varepsilon$  centered at  $x$ .

A point  $x \in \mathbb{R}^n$  is a *boundary point* of the set  $D$  if every neighborhood of  $x$  contains points both inside and outside of  $D$ . The set of the boundary points of  $D$  is denoted by  $\partial D$ .

Let  $f: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ . Then  $f$  is *Lipschitz continuous* on an open set  $N \subset D$  if there is a constant  $0 < L < \infty$  so that

$$\|f(x) - f(y)\| \leq L\|x - y\|$$

for all  $x, y \in N$ .  $L$  is called the *Lipschitz constant*. If  $g, h: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  are two Lipschitz continuous functions on a set  $N \subset D$ , then their sum  $g + h$  is also Lipschitz continuous, with the Lipschitz constant equal to the sum of the Lipschitz constants for  $f$  and  $g$ , respectively. If  $g, h: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  are two Lipschitz continuous functions and bounded on a set  $N \subset D$ , i.e., there is a constant  $M > 0$  such that  $|g(x)| \leq M$  and  $|h(x)| \leq M$  for all  $x \in N$ , then the product  $gh$  is Lipschitz continuous on  $N$ .

If  $f$  is Lipschitz continuous on a set  $D \subset \mathbb{R}^n$ , then  $f$  is uniformly continuous on  $D$ . The reverse is not true.

## A4. Elements of Convexity:Convex Sets and Convex Functions

### Convex Sets

A set  $C \subset \mathbb{R}^n$  is a *convex set* if for every point  $x, y \in C$  the point  $z = \lambda x + (1 - \lambda)y$  is also in the set  $C$  for any  $\lambda \in [0, 1]$ . The intersection of any family of convex sets is a convex set. An *affine* set in  $\mathbb{R}^n$  is the set of all the vectors  $\{x\} \oplus S$ , where  $x \in \mathbb{R}^n$  and  $S$  is a subspace of  $\mathbb{R}^n$ . A *cone* is a set  $V$  with the property that for all  $x \in V$  it follows that  $\alpha x \in V$  for all  $\alpha > 0$ . A *cone generated by*  $\{x_1, x_2, \dots, x_m\}$  is the set of all the vectors of the form

$$x = \sum_{i=1}^m \alpha_i x_i, \text{ where } \alpha_i \geq 0 \text{ for all } i = 1, \dots, m.$$

Observe that all the cones of this form are convex sets. A *convex combination* of a finite set of vectors  $\{x_1, x_2, \dots, x_m\}$  in  $\mathbb{R}^n$  is any vector  $x$  of the form

$$x = \sum_{i=1}^m \alpha_i x_i, \text{ where } \sum_{i=1}^m \alpha_i = 1, \alpha_i \geq 0 \text{ for all } i = 1, \dots, m.$$

The following two results on the separation of the convex sets are used in specifying the optimality conditions for nonlinear programming problems. We present them without proof (see Bazaraa, Sherali, & Shetty, 1993).

**Proposition A4.1** (*Separation of a Convex Set and a Point*). *Let  $C \subset \mathbb{R}^n$  be a nonempty and convex set. Consider a point  $y \notin C$ . Then, there exist a nonzero vector  $a \in \mathbb{R}^n$  and a scalar  $c \in \mathbb{R}$  such that  $a^T y > c$  and  $a^T x \leq c$  for any  $x \in C$ .* ◆

**Proposition A4.2** (*Separation of Two Convex Sets*). *Let  $C_1$  and  $C_2$  be two nonempty and convex sets in  $\mathbb{R}^n$ . Suppose that  $C_1 \cap C_2 = \emptyset$ . Then, there exists a hyperplane that separates  $C_1$  and  $C_2$ , i.e., there is a nonzero vector  $p \in \mathbb{R}^n$  such that  $p^T x_1 \geq p^T x_2$  for any  $x_1 \in cl(C_1)$  and for any  $x_2 \in cl(C_2)$ .* ◆

### Convex Functions

A function  $f : C \rightarrow \mathbb{R}$  defined on a convex set  $C \subset \mathbb{R}^n$  is a *convex function* if  $f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$  for every  $x, y \in C$  and every  $\lambda \in (0, 1)$ . Moreover,  $f$  is said to be *strictly convex* if for every  $x, y \in C$  and every  $\lambda \in (0, 1)$ ,  $f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda)f(y)$ . In other words, this means that if we take any two points  $x$  and  $y$ , then  $f$  evaluated at any convex combination of these two points should be no larger than the same convex combination of  $f(x)$  and  $f(y)$ . A function that is not convex is said to be *nonconvex*. A function  $f$  is *concave* if  $-f$  is convex. Any linear function of  $n$  variables is both convex and concave on  $\mathbb{R}^n$ . The following result shows why the convex functions are of interest in optimization problems.

**Theorem A4.1** *Any local minimum of a convex function  $f : C \rightarrow \mathbb{R}$  defined on a convex set  $C \subset \mathbb{R}^n$  is also a global minimum on  $C$ . Any local minimum of a strictly convex function  $f : C \rightarrow \mathbb{R}$  defined on a convex set  $C \subset \mathbb{R}^n$  is the unique strict global minimum off on  $C$ .* ◆

## Strong Convexity

A differentiable function  $f$  is called *strongly convex* on  $S$  with the parameter  $\mu > 0$  if for all the points  $x, y \in S$ ,

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2} \|y - x\|^2.$$

Intuitively, strong convexity means that there exists a quadratic lower bound on the growth of the function. Observe that a strongly convex function is strictly convex since the quadratic lower bound growth is strictly greater than the linear growth. An equivalent condition for the *strong convexity* of the function  $f$  on  $S$  is

$$(\nabla f(x) - \nabla f(y))^T(x - y) \geq \mu \|x - y\|^2$$

for some  $\mu > 0$  and for all  $x, y \in S$ .

For differentiable strongly convex functions, it is easy to prove that:

1.  $\|\nabla f(x)\|^2 \geq 2\mu(f(x) - f(x^*))$  for all  $x \in S$ , where  $x^*$  is a local minimum of the function  $f$ .
2.  $\|\nabla f(x) - \nabla f(y)\| \geq \mu \|x - y\|$  for all  $x \in S$ .
3.  $f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2\mu} \|\nabla f(y) - \nabla f(x)\|^2$ , for all  $x \in S$ .

If the function  $f$  is twice continuously differentiable, then it is strongly convex with the parameter  $\mu > 0$  on  $S$  if and only if  $\nabla^2 f(x) \geq \mu I$  for all  $x \in S$ , where  $I$  is the identity matrix and the inequality  $\geq$  means that  $\nabla^2 f(x) - \mu I$  is positive semidefinite.

**Proposition A4.3** (*Convexity of Level Set*). *Let  $C$  be a convex set in  $\mathbb{R}^n$  and let  $f: C \rightarrow \mathbb{R}$  be a convex function. Then, the level set  $C_\alpha = \{x \in C : f(x) \leq \alpha\}$ , where  $\alpha$  is a real number, is a convex set.*

**Proof** Let  $x_1, x_2 \in C$ . Of course,  $x_1, x_2 \in C_\alpha$ ,  $f(x_1) \leq \alpha$  and  $f(x_2) \leq \alpha$ . Now, let  $\lambda \in (0, 1)$  and consider  $x = \lambda x_1 + (1 - \lambda)x_2$ . By the convexity of  $C$ , it follows that  $x \in C$ . On the other hand, by the convexity of  $f$  on  $C$ ,

$$f(x) \leq \lambda f(x_1) + (1 - \lambda)f(x_2) \leq \lambda\alpha + (1 - \lambda)\alpha = \alpha,$$

i.e.,  $x \in C_\alpha$ . ◆

**Proposition A4.4** (*Convexity of a domain defined by a set of convex functions*). *Let  $C$  be a convex set in  $\mathbb{R}^n$  and let  $c_i: C \rightarrow \mathbb{R}$ ,  $i = 1, \dots, m$ , be convex functions on  $C$ . Then, the set defined by  $X = \{x \in C : c_i(x) \leq 0, i = 1, \dots, m\}$  is convex.*

**Proof** The result follows from Proposition A4.3 and from the property of the intersection of the convex sets. ◆

The following two propositions give *differential criteria* of checking the convexity of a function.

**Proposition A4.5** (*First-Order Condition for Convexity*). *Let  $C$  be a convex set in  $\mathbb{R}^n$  with nonempty interior. Consider the function  $f: C \rightarrow \mathbb{R}$  which is continuous on  $C$  and differentiable on  $\text{int}(C)$ . Then  $f$  is convex on  $\text{int}(C)$  if and only if  $f(y) \geq f(x) + \nabla f(x)^T(y - x)$  for any points  $x, y \in C$ .* ◆

**Proposition A4.6** (*Second-Order Condition for Convexity*). Let  $C$  be a convex set in  $\mathbb{R}^n$  with nonempty interior. Consider the function  $f: C \rightarrow \mathbb{R}$  which is continuous on  $C$  and twice differentiable on  $\text{int}(C)$ . Then,  $f$  is convex on  $\text{int}(C)$  if and only if the Hessian  $\nabla^2 f(x)$  is positive semidefinite at each  $x \in \text{int}(C)$ .  $\blacklozenge$

The convexity of the objective function and of the constraints is crucial in nonlinear optimization. The convex programs have very nice theoretical properties which can be used to design efficient optimization algorithms. Therefore, it is important to know how to detect the convexity and the operations that preserve the convexity of functions.

**Theorem A4.2** (*Farkas' Theorem*). Let  $A$  be an  $m \times n$  matrix and  $c$  an  $n$  vector. Then, just one of the following two statements holds:

System 1. There exists  $x \in \mathbb{R}^n$  such that  $Ax \leq 0$  and  $c^T x > 0$ .

System 2. There exists  $y \in \mathbb{R}^m$  such that  $A^T y = c$  and  $y \geq 0$ .

**Proof** Suppose that System 2 has a solution, i.e., there exists  $y \geq 0$  such that  $A^T y = c$ . Let  $x$  be such that  $Ax \leq 0$ . Then,  $c^T x = y^T A x \leq 0$ . Therefore, System 1 has no solution.

Now, suppose that System 2 has no solution. Consider the set  $X = \{x : x = A^T y, y \geq 0\}$ . Observe that  $X$  is a closed and convex set. By Proposition A4.1, there exist a vector  $p \in \mathbb{R}^n$  and a scalar  $\alpha$  such that  $p^T c > \alpha$  and  $p^T x \leq \alpha$  for all  $x \in X$ . Since  $0 \in X$ , then  $\alpha \geq 0$  and so  $p^T c > 0$ . Also,  $\alpha \geq p^T A^T y = y^T A p$  for all  $y \geq 0$ . Since  $y \geq 0$  can be made arbitrarily large, it follows that the last inequality implies that  $A p \leq 0$ . Therefore, we have constructed a vector  $p \in \mathbb{R}^n$  such that  $A p \leq 0$  and  $c^T p > 0$ , i.e., System 1 has a solution.  $\blacklozenge$

**Theorem A4.3** (*Gordan's Theorem*). Let  $A$  be a  $m \times n$  matrix. Then just one of the following two statements holds:

System 1. There exists  $x \in \mathbb{R}^n$  such that  $Ax < 0$ .

System 2. There exists  $y \in \mathbb{R}^m$ ,  $y \neq 0$  such that  $A^T y = 0$  and  $y \geq 0$ .

**Proof** System 1 can be equivalently written as  $Ax + es \leq 0$  for some  $x \in \mathbb{R}^n$  and  $s > 0$ ,  $s \in \mathbb{R}$ , where  $e$  is a vector of  $m$  ones. Now, rewriting this system in the form of System 1 of Theorem A4.2, we obtain  $[A \ e] \begin{pmatrix} x \\ s \end{pmatrix} \leq 0$  and  $(0, \dots, 0, 1) \begin{pmatrix} x \\ s \end{pmatrix} > 0$  for some  $\begin{pmatrix} x \\ s \end{pmatrix} \in \mathbb{R}^{n+1}$ . By Theorem A4.2, the associated System 2 states that  $\begin{bmatrix} A^T \\ e^T \end{bmatrix} y = (0, \dots, 0, 1)^T$  and  $y \geq 0$  for some  $y \in \mathbb{R}^m$ , that is,  $A^T y = 0$ ,  $e^T y = 1$  and  $y \geq 0$  for some  $y \in \mathbb{R}^m$ . But this is equivalent to System 2.  $\blacklozenge$

**Theorem A4.4** Let  $P$  and  $Q$  be two symmetric matrices such that  $Q \geq 0$  and  $P > 0$  on the null space of  $Q$  (i.e.,  $y^T P y > 0$  for any  $y \neq 0$  with  $Qy = 0$ ). Then, there exists  $\bar{c} > 0$  such that  $P + cQ > 0$  for any  $c > \bar{c}$ .

**Proof** Assume the contrary. Then, for any  $k > 0$  there exists  $x^k$ ,  $\|x^k\| = 1$  such that  $(x^k)^T P(x^k) + k(x^k)^T Q(x^k) \leq 0$ . Consider a subsequence  $\{x^k\}_K$  convergent to some  $\bar{x}$  with  $\|\bar{x}\| = 1$ . Dividing the above inequality by  $k$  and taking the limit as  $k \in K \rightarrow \infty$ , we get  $\bar{x}^T Q \bar{x} \leq 0$ . On the other hand,  $Q$  being semipositive definite we must have  $\bar{x}^T Q \bar{x} \geq 0$ , hence  $\bar{x}^T Q \bar{x} = 0$ . Therefore, using the hypothesis it follows that  $\bar{x}^T P \bar{x} \geq 0$ . But this contradicts the fact that  $\bar{x}^T P \bar{x} + \limsup_{k \rightarrow \infty, k \in K} k(x^k)^T Q(x^k) \leq 0$ .  $\blacklozenge$

**Proposition A4.7** (*Linear combination with nonnegative coefficients*). *Let  $C$  be a convex set in  $\mathbb{R}^n$ . If  $f: C \rightarrow \mathbb{R}$  and  $g: C \rightarrow \mathbb{R}$  are convex functions on  $C$ , then their linear combination  $\lambda f + \eta g$ , where the coefficients  $\lambda$  and  $\eta$  are nonnegative, is also convex on  $C$ .*  $\blacklozenge$

**Proposition A4.8** (*Composition with affine mapping*). *Let  $C$  and  $D$  be convex sets in  $\mathbb{R}^m$  and  $\mathbb{R}^n$ , respectively. If  $g: C \rightarrow \mathbb{R}$  is a convex function on  $C$  and  $h: D \rightarrow \mathbb{R}^m$  is an affine mapping, i.e.,  $h(x) = Ax + b$  with  $\text{range}(h) \subset C$ , then the composite function  $f: D \rightarrow \mathbb{R}$  defined as  $f(x) = g(h(x))$  is convex on  $D$ .*  $\blacklozenge$

## Notes and References

The material in this appendix is covered in (Dennis & Schnabel, 1983; Peressini, Sullivan, & Uhl, 1988; Trefethen & Schreiber, 1990; Bazaraa, Sherali, & Shetty, 1993; Golub & Van Loan, 1996; Demmel, 1997; Trefethen & Bau III, 1997; Meyer, 2000; Nocedal & Wright, 2006; Bartholomew-Biggs, 2008; Griva, Nash, & Sofer, 2009), etc.

---

## Appendix B: The SMUNO Collection

---

### Small-Scale Continuous Unconstrained Optimization Applications

**Application S1** *Weber (1) Locate a central facility* (Kelley, 1999, pp. 118–119; Andrei, 2003, p. 58) [WEBER-1]

$$f(x) = 2\sqrt{(x_1 - 2)^2 + (x_2 - 42)^2} + 4\sqrt{(x_1 - 90)^2 + (x_2 - 11)^2} + 5\sqrt{(x_1 - 43)^2 + (x_2 - 88)^2}.$$

Initial point:  $x_0 = [10, 10]$ .  $f(x_0) = -0.374731E + 02$ .

**Application S2** *Weber (2) Locate a central facility* (Kelley, 1999, pp. 118–119) [WEBER-2]

$$\begin{aligned} f(x) = & 2\sqrt{(x_1 + 10)^2 + (x_2 + 10)^2} - 4\sqrt{(x_1)^2 + (x_2)^2} \\ & + 2\sqrt{(x_1 - 5)^2 + (x_2 - 8)^2} + \sqrt{(x_1 - 25)^2 + (x_2 - 30)^2}. \end{aligned}$$

Initial point:  $x_0 = [1.2, 1]$ .  $f(x_0) = 0.785943E + 02$ .

**Application S3** *Weber (3) Locate a central facility* (Kelley, 1999, pp. 118–119) [WEBER-3]

$$\begin{aligned} f(x) = & 2\sqrt{(x_1 + 10)^2 + (x_2 + 10)^2} - 4\sqrt{(x_1)^2 + (x_2)^2} \\ & + 2\sqrt{(x_1 - 5)^2 + (x_2 - 8)^2} + \sqrt{(x_1 - 25)^2 + (x_2 - 30)^2} \\ & + \sin(0.0035(x_1^2 + x_2^2)). \end{aligned}$$

Initial point:  $x_0 = [1.2, 1]$ .  $f(x_0) = 0.786028E + 02$ .

**Application S4** *Analysis of enzymes reaction* (Andrei, 2003, p. 62) [ENZIMES]

$$f(x) = \sum_{i=1}^{11} \left( y_i - \frac{x_1(u_i^2 + u_i x_2)}{u_i^2 + u_i x_3 + x_4} \right)^2,$$

where  $y_i$  and  $u_i$  have the following values:

$i$	$y_i$	$u_i$	$i$	$y_i$	$u_i$
1	0.1957	4.000	7	0.0456	0.125
2	0.1947	2.000	8	0.0342	0.100
3	0.1735	1.000	9	0.0323	0.0833
4	0.1600	0.500	10	0.0235	0.0714
5	0.0844	0.250	11	0.0246	0.0625
6	0.0627	0.167			

Initial point:  $x_0 = [0.25, 0.39, 0.415, 0.39]$ .  $f(x_0) = 0.531317E - 02$ .

**Application S5** *Stationary solution of a chemical reactor* (Shacham, 1986, pp. 1455–1481) [REACTOR]

$$\begin{aligned} f(x) = & (1 - x_1 - k_1 x_1 x_6 + r_1 x_4)^2 \\ & + (1 - x_2 - k_2 x_2 x_6 + r_2 x_5)^2 \\ & + (-x_3 + 2k_3 x_4 x_5)^2 \\ & + (k_1 x_1 x_6 - r_1 x_4 - k_3 x_4 x_5)^2 \\ & + (1.5(k_2 x_2 x_6 - r_2 x_5) - k_3 x_4 x_5)^2 \\ & + (1 - x_4 - x_5 - x_6)^2 \end{aligned}$$

where:  $k_1 = 31.24$     $k_2 = 0.272$     $k_3 = 303.03$     $r_1 = 2.062$     $r_2 = 0.02$ .

Initial point:  $x_0 = [1.09, 1.05, 3.05, 0.99, 6.05, 1.09]$ .  $f(x_0) = 0.196173E + 08$ .

**Application S6** *Robot kinematics problem* (Kearfott & Novoa, 1990, pp. 152–157; Andrei, 2013e, pp. 101–103; Floudas, Pardalos, Adjiman, Esposito, Gümüs, Harding, Klepeis, Meyer, & Schweiger, 1999, pp. 329–331) [ROBOT]

$$\begin{aligned} f(x) = & (4.731 \cdot 10^{-3} x_1 x_3 - 0.3578 x_2 x_3 - 0.1238 x_1 + x_7 - 1.637 \cdot 10^{-3} x_2 - 0.9338 x_4 - 0.3571)^2 \\ & + (0.2238 x_1 x_3 + 0.7623 x_2 x_3 + 0.2638 x_1 - x_7 - 0.07745 x_2 - 0.6734 x_4 - 0.6022)^2 \\ & + (x_6 x_8 + 0.3578 x_1 + 4.731 \cdot 10^{-3} x_2)^2 \\ & + (-0.7623 x_1 + 0.2238 x_2 + 0.3461)^2 \\ & + (x_1^2 + x_2^2 - 1)^2 \\ & + (x_3^2 + x_4^2 - 1)^2 \\ & + (x_5^2 + x_6^2 - 1)^2 \\ & + (x_7^2 + x_8^2 - 1)^2. \end{aligned}$$

Initial point:  $x_0 = [0.164, -0.98, -0.94, -0.32, -0.99, -0.05, 0.41, -0.91]$ .  
 $f(x_0) = 0.533425E + 01$ .

**Application S7** *Solar spectroscopy* (Andrei, 2003, p. 68) [SPECTR]

$$f(x) = \sum_{i=1}^{13} \left( x_1 + x_2 \exp \left( -\frac{(i+x_3)^2}{x_4} \right) - y_i \right)^2,$$

where  $y_i$ ,  $i = 1, \dots, 13$  are as in the table below:

$i$	$y_i$	$i$	$y_i$
1	0.5	8	2.5
2	0.8	9	1.6
3	1	10	1.3
4	1.4	11	0.7
5	2	12	0.4
6	2.4	13	0.3
7	2.7		

Initial point:  $x_0 = [1, 1, 1, 1]$ .  $f(x_0) = 0.995870E + 01$ .

**Application S8** *Estimation of parameters* (Himmelblau, 1972, p. 430) [ESTIMP]

$$f(x) = \sum_{i=1}^7 \left( \frac{x_1^2 + a_i x_2^2 + a_i^2 x_3^2}{(1 + a_i x_4^2) b_i} - 1 \right)^2,$$

where the parameters  $a_i, b_i$ ,  $i = 1, \dots, 7$  have the following values:

$i$	$a_i$	$b_i$
1	0.0	7.391
2	0.000428	11.18
3	0.0010	16.44
4	0.00161	16.20
5	0.00209	22.20
6	0.00348	24.02
7	0.00525	31.32

Initial point:  $x_0 = [2.7, 90, 1500, 10]$ .  $f(x_0) = 0.290530E + 01$ .

**Application S9** *Propan combustion in air – reduced variant* (Meintjes & Morgan, 1990, pp. 143–151; Averick, Carter, Moré, & Xue, 1992, pp. 18–19; Andrei, 2013e, pp. 54–56; Floudas, Pardalos, Adjiman, Esposito, Gümüs, Harding, Klepeis, Meyer, & Schweiger, 1999, p. 327) [PROPAN]

$$\begin{aligned} f(x) = & (x_1 x_2 + x_1 - 3x_5)^2 \\ & + (2x_1 x_2 + x_1 + 2R_{10} x_2^2 + x_2 x_3^2 + R_7 x_2 x_3 + R_9 x_2 x_4 + R_8 x_2 - Rx_5)^2 \\ & + (2x_2 x_3^2 + R_7 x_2 x_3 + 2R_5 x_3^2 + R_6 x_3 - 8x_5)^2 \\ & + (R_9 x_2 x_4 + 2x_4^2 - 4Rx_5)^2 \\ & + (x_1 x_2 + x_1 + R_{10} x_2^2 + x_2 x_3^2 + R_7 x_2 x_3 + R_9 x_2 x_4 + R_8 x_2 + R_5 x_3^2 + R_6 x_3 + x_4^2 - 1)^2 \end{aligned}$$

where:

$$\begin{aligned} R_5 &= 0.193 & R_6 &= 0.4106217541E - 3 & R_7 &= 0.5451766686E - 3 \\ R_8 &= 0.44975E - 6 & R_9 &= 0.3407354178E - 4 & R_{10} &= 0.9615E - 6 \\ R &= 10 \end{aligned}$$

Initial point:  $x_0 = [10, 10, 0.05, 50.5, 0.05]$ .  $f(x_0) = 0.331226E + 08$ .

**Application S10** *Gear train of minimum inertia* (Sandgren & Ragsdell, 1980; Schittkowski, 1987, Problem 328, p. 149) [GEAR-1]

$$f(x) = 0.1(12 + x_1^2 + (1 + x_2^2)/x_1^2 + (x_1^2 x_2^2 + 100)/x_1^4 x_2^4).$$

Initial point:  $x_0 = [0.5, 0.5]$ .  $f(x_0) = 0.256332E + 04$ .

**Application S11** *Human heart dipole* (Andrei, 2003, p. 65; Averick, Carter, Moré, & Xue, 1992, p. 17; Andrei, 2013e, pp. 51–54; Nelson & Hodgkin, 1981, pp. 817–823)) [HHD]

$$\begin{aligned} f(x) = & (x_1 + x_2 - s_{mx})^2 \\ & + (x_3 + x_4 - s_{my})^2 \\ & + (x_1 x_5 + x_2 x_6 - x_3 x_7 - x_4 x_8 - s_A)^2 \\ & + (x_1 x_7 + x_2 x_8 + x_3 x_5 + x_4 x_6 - s_B)^2 \\ & + (x_1(x_5^2 - x_7^2) - 2x_3 x_5 x_7 + x_2(x_6^2 - x_8^2) - 2x_4 x_6 x_8 - s_C)^2 \\ & + (x_3(x_5^2 - x_7^2) + 2x_1 x_5 x_7 + x_4(x_6^2 - x_8^2) + 2x_2 x_6 x_8 - s_D)^2 \\ & + (x_1 x_5(x_5^2 - 3x_7^2) + x_3 x_7(x_7^2 - 3x_5^2) + x_2 x_6(x_6^2 - 3x_8^2) + x_4 x_8(x_8^2 - 3x_6^2) - s_E)^2 \\ & + (x_3 x_5(x_5^2 - 3x_7^2) - x_1 x_7(x_7^2 - 3x_5^2) + x_4 x_6(x_6^2 - 3x_8^2) - x_2 x_8(x_8^2 - 3x_6^2) - s_F)^2 \end{aligned}$$

where:

$$\begin{aligned} s_{mx} &= 0.485 & s_A &= -0.0581 & s_C &= 0.105 & s_E &= 0.167 \\ s_{my} &= -0.0019 & s_B &= 0.015 & s_D &= 0.0406 & s_F &= -0.399 \end{aligned}.$$

Initial point:

$$\begin{aligned} x_0 &= [0.299 \quad 0.186 \quad -0.0273 \quad 0.0254 \quad -0.474 \quad 0.474 \quad -0.0892 \quad 0.0892]^T. \\ f(x_0) &= 0.190569E + 00. \end{aligned}$$

**Application S12** *Neurophysiology* (Andrei, 2013e, pp. 57–61; Verschelde, Verlinden, & Cools, 1994, pp. 915–930) [NEURO]

$$\begin{aligned} f(x) = & (x_1^2 + x_3^2 - 1)^2 + (x_2^2 + x_4^2 - 1)^2 \\ & + (x_5 x_3^3 + x_6 x_4^3 - 1)^2 + (x_5 x_1^3 + x_6 x_2^3 - 2)^2 \\ & + (x_5 x_1 x_3^2 + x_6 x_2 x_4^2 - 1)^2 + (x_5 x_3 x_1^2 + x_6 x_4 x_2^2 - 4)^2. \end{aligned}$$

Initial point:  $x_0 = [0.01, \dots, 0.001]$ .  $f(x_0) = 0.239991E + 02$ .

**Application S13** *Combustion application* (Morgan, 1987; Andrei, 2013e, pp. 61–63) [COMBUST]

$$\begin{aligned}
f(x) = & (x_2 + 2x_6 + x_9 + 2x_{10} - 10^{-5})^2 \\
& + (x_3 + x_8 - 3 \cdot 10^{-5})^2 \\
& + (x_1 + x_3 + 2x_5 + 2x_8 + x_9 + x_{10} - 5 \cdot 10^{-5})^2 \\
& + (x_4 + 2x_7 - 10^{-5})^2 \\
& + (0.5140437 \cdot 10^{-7}x_5 - x_1^2)^2 \\
& + (0.1006932 \cdot 10^{-6}x_6 - 2x_2^2)^2 \\
& + (0.7816278 \cdot 10^{-15}x_7 - x_4^2)^2 \\
& + (0.1496236 \cdot 10^{-6}x_8 - x_1x_3)^2 \\
& + (0.6194411 \cdot 10^{-7}x_9 - x_1x_2)^2 \\
& + (0.2089296 \cdot 10^{-14}x_{10} - x_1x_2^2)^2.
\end{aligned}$$

Initial point:  $x_0 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$ .  $f(x_0) = 0.017433088$ .

**Application S14** *Circuit design* (Ratschek & Rokne, 1993, p. 501; Andrei, 2009e, pp. 243–244), [CIRCUIT]

$$f(x) = (x_1x_3 - x_2x_4)^2 + \sum_{k=1}^4 (a_k^2 + b_k^2),$$

where

$$a_k = (1 - x_1x_2)x_3 \left\{ \exp [x_5(g_{1k} - g_{3k}x_7 \cdot 10^{-3} - g_{5k}x_8 \cdot 10^{-3})] - 1 \right\} + g_{4k}x_2 - g_{5k}, \quad k = 1, \dots, 4,$$

$$b_k = (1 - x_1x_2)x_4 \left\{ \exp [x_6(g_{1k} - g_{2k} - g_{3k}x_7 \cdot 10^{-3} - g_{4k}x_9 \cdot 10^{-3})] - 1 \right\} + g_{4k} - g_{5k}x_1,$$

$$k = 1, \dots, 4, \quad g = \begin{bmatrix} 0.4850 & 0.7520 & 0.8690 & 0.9820 \\ 0.3690 & 1.2540 & 0.7030 & 1.4550 \\ 5.2095 & 10.0677 & 22.9274 & 20.2153 \\ 23.3037 & 101.7790 & 111.4610 & 191.2670 \\ 28.5132 & 111.8467 & 134.3884 & 211.4823 \end{bmatrix},$$

Initial point:  $x_0 = [0.7, 0.5, 0.9, 1.9, 8.1, 8.1, 5.9, 1, 1.9]$ .  $f(x_0) = 0.296457E + 04$ .

**Application S15** *Thermistor* (Andrei, 2009e, pp. 722–723) [THERMI]

$$f(x) = \sum_{i=1}^{16} \left( y_i - x_1 \exp \left( \frac{x_2}{45 + 5i + x_3} \right) \right)^2$$

where

$i$	$y_i$	$i$	$y_i$
1	34780	9	8261
2	28610	10	7030
3	23650	11	6005
4	19630	12	5147
5	16370	13	4427
6	13720	14	3820
7	11540	15	3307
8	9744	16	2872

Initial point:  $x_0 = [0.01, 6100, 340]$ .  $f(x_0) = 0.233591\text{E} + 10$ .

**Application S16** *Optimal design of a gear train* (Sandgren, 1988, pp. 95–105; Andrei, 2013e, p. 79) [GEAR-2]

$$f(x) = \left( \frac{1}{6.931} - \frac{x_1 x_2}{x_3 x_4} \right)^2.$$

Initial point:  $x_0 = [15, 14, 35, 35]$ .  $f(x_0) = 0.737081\text{E} - 03$ .

### Notes and References

This appendix includes 16 unconstrained optimization applications taken from literature.

---

## Appendix C: The LACOP Collection

---

### Large-Scale Continuous Nonlinear Optimization Applications

#### Application L1. *Chemical equilibrium (ELCH)*

This application is described in (Hock & Schittkowski, 1981, pp. 121; Andrei, 1999a, pp. 804; Andrei, 2015a, pp. 972).

$$\begin{aligned} \min \quad & \sum_{j=1}^{10} x_j \left( c_j + \ln \frac{x_j}{x_1 + \dots + x_{10}} \right) \\ \text{subject to} \quad & x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 = 0, \\ & x_4 + 2x_5 + x_6 + x_7 - 1 = 0, \\ & x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0. \end{aligned} \tag{1}$$

Solution of the application:

Nr.	$c_j$	Initial point	Lower bound	Solution
1	-6.089	0.1	1.e-6	0.0406685
2	-17.164	0.1	1.e-6	0.1477301
3	-34.054	0.1	1.e-6	0.7831533
4	-5.914	0.1	1.e-6	0.001414229
5	-24.721	0.1	1.e-6	0.4852466
6	-14.986	0.1	1.e-6	0.0006931799
7	-24.100	0.1	1.e-6	0.02739941
8	-10.708	0.1	1.e-6	0.01794741
9	-26.662	0.1	1.e-6	0.03731418
10	-22.179	0.1	1.e-6	0.09687152

$$f(x_0) = -20.96029 \quad f(x^*) = -47.761090859$$

#### Application L2. *Optimization of an alkilation process (ALKI)*

This application is taken from (Hock & Schittkowski, 1981, pp. 123; Andrei, 1999a, pp. 803; Andrei, 2015a, pp. 974).

$$\begin{aligned}
& \min (5.04x_1 + 0.035x_2 + 10x_3 + 3.36x_5 - 0.063x_4x_7) \\
& \text{subject to} \tag{2} \\
& 35.82 - 0.222x_{10} - bx_9 \geq 0, \\
& -133 + 3x_7 - ax_{10} \geq 0, \\
& -35.82 + 0.222x_{10} + bx_9 + (1/b - b)x_9 \geq 0, \\
& 133 - 3x_7 + ax_{10} + (1/a - a)x_{10} \geq 0, \\
& 1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - ax_4 \geq 0, \\
& 57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - ax_7 \geq 0, \\
& -1.12x_1 - 0.13167x_1x_8 + 0.00667x_1x_8^2 + ax_4 + (1/a - a)x_4 \geq 0, \\
& -57.425 - 1.098x_8 + 0.038x_8^2 - 0.325x_6 + ax_7 + (1/a - a)x_7 \geq 0, \\
& 1.22x_4 - x_1 - x_5 = 0, \\
& 98000x_3/(x_4x_9 + 1000x_3) - x_6 = 0, \\
& (x_2 + x_5)/x_1 - x_8 = 0,
\end{aligned}$$

where  $a = 0.99$  and  $b = 0.9$ .

Solution of the application:

Nr.	Initial point	Lower bound	Solution	Upper bound
1	1745	0.00001	1698.094	2000
2	12000	0.00001	15818.51	16000
3	110	0.00001	54.10228	120
4	3048	0.00001	3031.226	5000
5	1974	0.00001	2000	2000
6	89.2	85	90.11548	93
7	92.8	90	95	95
8	8	3	10.49324	12
9	3.6	1.2	1.561636	4
10	145	145	153.5354	162

$$f(x_0) = -872.3874 \quad f(x^*) = -1768.80696$$

### Application L3. Optimal design of a reactor as a geometric programming problem (PREC)

This application is presented in (Dembo, 1976; Rijckaert, 1973; Andrei, 1999a, pp. 801; Hock & Schittkowski 1981, pp. 113).

$$\begin{aligned}
& \min (0.4x_1^{0.67}x_7^{-0.67} + 0.4x_2^{0.67}x_8^{-0.67} - x_1 - x_2 + 10) \\
& \text{subject to} \\
& 1 - 0.0588x_5x_7 - 0.1x_1 \geq 0, \\
& 1 - 0.0588x_6x_8 - 0.1x_1 - 0.1x_2 \geq 0, \\
& 1 - 4x_3x_5^{-1} - 2x_3^{-0.71}x_5^{-1} - 0.0588x_3^{-1.3}x_7 \geq 0, \\
& 1 - 4x_4x_6^{-1} - 2x_4^{-0.71}x_6^{-1} - 0.0588x_4^{-1.3}x_8 \geq 0, \\
& 0.4x_1^{0.67}x_7^{-0.67} + 0.4x_2^{0.67}x_8^{-0.67} - x_1 - x_2 + 10 \geq 1, \\
& 0.4x_1^{0.67}x_7^{-0.67} + 0.4x_2^{0.67}x_8^{-0.67} - x_1 - x_2 + 10 \leq 4.2.
\end{aligned} \tag{3}$$

Solution of the application:

Nr.	Initial point	Lower bound	Solution	Upper bound
1	6	0.1	6.465115	10
2	3	0.1	2.232708	10
3	0.4	0.1	0.6673975	10
4	0.2	0.1	0.5957564	10
5	6	0.1	5.932676	10
6	6	0.1	5.527235	10
7	1	0.1	1.013322	10
8	0.5	0.1	0.4006682	10

$$f(x_0) = 3.657366 \quad f(x^*) = 3.951163508$$

#### Application L4. Cost minimization of a transformer design (TRAFO)

The objective function of this application represents the worth of the transformer, including the operating cost. The constraints refer to the rating of the transformer and to the allowable transmission loss. The variables  $x_1, x_2, x_3$  and  $x_4$  are physical dimensions of winding and core. The variables  $x_5$  and  $x_6$  are the magnetic flux density and the current density, respectively (Price, 1983; Ballard, Jelinek, & Schinzinger, 1974). The mathematical model is described in (Hock & Schittkowski, 1981) and (Andrei, 2003, Application A1, pp. 344).

$$\begin{aligned}
& \min 0.0204x_1x_4(x_1 + x_2 + x_3) + 0.0187x_2x_3(x_1 + 1.57x_2 + x_4) + \\
& 0.0607x_1x_4x_5^2(x_1 + x_2 + x_3) + 0.0437x_2x_3x_6^2(x_1 + 1.57x_2 + x_4) \\
& \text{subject to} \\
& 0.001x_1x_2x_3x_4x_5x_6 - 2.07 \geq 0, \\
& 1 - 0.00062x_1x_4x_5^2(x_1 + x_2 + x_3) - 0.00058x_2x_3x_6^2(x_1 + 1.57x_2 + x_4) \geq 0,
\end{aligned} \tag{4}$$

where the variables are bounded as:  $x_i \geq 0$ ,  $i = 1, \dots, 6$ .

The solution of this application is

$$\begin{aligned}
x_1^* &= 5.3326663, & x_2^* &= 4.6567441, & x_3^* &= 10.4329919, \\
x_4^* &= 12.0823063, & x_5^* &= 0.7526074, & x_6^* &= 0.8786509.
\end{aligned}$$

**Application L5. Optimization of a multi-spindle automatic lathe (LATHE)**

The optimization of a multi-spindle automatic lathe is to minimize a nonlinear objective subject to 15 nonlinear constraints with 10 variables. It is taken from (Schittkowski, 1987, pp. 195) and has the following mathematical expression.

$$\begin{aligned}
 & \min \left( -20000 \frac{0.15x_1 + 14x_2 - 0.06}{0.002 + x_1 + 60x_2} \right) \\
 & \text{subject to} \\
 & x_1 - \frac{0.75}{x_3 x_4} \geq 0, \\
 & x_1 - \frac{x_9}{x_4 x_5} \geq 0, \\
 & x_1 - \frac{x_{10}}{x_4 x_6} - \frac{10}{x_4} \geq 0, \\
 & x_1 - \frac{0.19}{x_4 x_7} - \frac{10}{x_4} \geq 0, \\
 & x_1 - \frac{0.125}{x_4 x_8} \geq 0, \\
 & 10000x_2 - 0.00131x_9 x_5^{0.666} x_4^{1.5} \geq 0, \\
 & 10000x_2 - 0.001038x_{10} x_6^{1.6} x_4^3 \geq 0, \\
 & 10000x_2 - 0.000223x_7^{0.666} x_4^{1.5} \geq 0, \\
 & 10000x_2 - 0.000076x_8^{3.55} x_4^{5.66} \geq 0, \\
 & 10000x_2 - 0.000698x_3^{1.2} x_4^2 \geq 0, \\
 & 10000x_2 - 0.00005x_3^{1.6} x_4^3 \geq 0, \\
 & 10000x_2 - 0.00000654x_3^{2.42} x_4^{4.17} \geq 0, \\
 & 10000x_2 - 0.000257x_3^{0.666} x_4^{1.5} \geq 0, \\
 & 30 - 2.003x_4 x_5 - 1.885x_4 x_6 - 0.184x_4 x_8 - 2x_4 x_3^{0.803} \geq 0, \\
 & x_9 + x_{10} - 0.255 = 0,
 \end{aligned} \tag{5}$$

where the variables are bounded as

$$\begin{aligned}
 0 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 0.1, \quad 0.5e - 4 \leq x_3 \leq 0.0081, \quad 10 \leq x_4 \leq 1000, \\
 0.5e - 4 \leq x_5 \leq 0.0017, \quad 0.5e - 4 \leq x_6 \leq 0.0013, \quad 0.5e - 4 \leq x_7 \leq 0.0027, \\
 0.5e - 4 \leq x_8 \leq 0.002, \quad 0.5e - 4 \leq x_9 \leq 1, \quad 0.5e - 4 \leq x_{10} \leq 1.
 \end{aligned}$$

The solution of the problem is

$$\begin{aligned}
 x_1^* &= 0.1472722, & x_2^* &= 0.1, & x_3^* &= 0.0081, & x_4^* &= 628.7173075, & x_5^* &= 0.0017, \\
 x_6^* &= 0.0011816, & x_7^* &= 0.0027, & x_8^* &= 0.00135, & x_9^* &= 0.1574074, \\
 x_{10}^* &= 0.0975926.
 \end{aligned}$$

**Application L6. Static power scheduling (PPSE)**

This application involves two electrical generators connected into a net with three nodes. The variables  $x_1$  and  $x_2$  are the real power outputs from the generators,  $x_3$  and  $x_4$  represent the reactive power outputs, and  $x_5$ ,  $x_6$ , and  $x_7$  are the voltage magnitudes at the nodes of the electrical network. Finally,  $x_8$  and  $x_9$  are the voltage phase angles at two of the nodes. The constraints of the model, other than the simple bounds on variables, are the real and the reactive power balance equations, the constraint stating that the power flowing into a node must balance the power flowing out. The mathematical model, described in (Bartholomew-Biggs, 1976; Hock & Schittkowski, 1981, pp. 116; Andrei, 2003, pp. 347; Andrei, 2013e), is as follows:

$$\begin{aligned}
 & \min (3000x_1 + 1000x_1^3 + 2000x_2 + 666.667x_2^3) \\
 & \text{subject to} \\
 & 0.4 - x_1 + 2Cx_5^2 + x_5x_6(D \sin(-x_8) - C \cos(-x_8)) + \\
 & x_5x_7(D \sin(-x_9) - C \cos(-x_9)) = 0, \\
 & 0.4 - x_2 + 2Cx_6^2 + x_5x_6(D \sin(x_8) - C \cos(x_8)) + \\
 & x_6x_7(D \sin(x_8 - x_9) - C \cos(x_8 - x_9)) = 0, \\
 & 0.8 + 2Cx_7^2 + x_5x_7(D \sin(x_9) - C \cos(x_9)) + \\
 & x_6x_7(D \sin(x_9 - x_8) - C \cos(x_9 - x_8)) = 0, \\
 & 0.2 - x_3 + 2Dx_5^2 + x_5x_6(C \sin(-x_8) + D \cos(-x_8)) - \\
 & x_5x_7(C \sin(-x_9) + D \cos(-x_9)) = 0, \\
 & 0.2 - x_4 + 2Dx_6^2 - x_5x_6(C \sin(x_8) + D \cos(x_8)) - \\
 & x_6x_7(C \sin(x_8 - x_9) + D \cos(x_8 - x_9)) = 0, \\
 & -0.337 + 2Dx_7^2 - x_5x_7(C \sin(x_9) + D \cos(x_9)) - \\
 & x_6x_7(C \sin(x_9 - x_8) + D \cos(x_9 - x_8)) = 0,
 \end{aligned} \tag{6}$$

where  $C = \sin(0.25)48.4/50.176$  and  $D = \cos(0.25)48.4/50.176$ . The simple bounds on variables are  $x_i \geq 0$ ,  $i = 1, 2$  and  $0.90909 \leq x_i \leq 1, 0909$ , for  $i = 5, 6, 7$ .

The solution is

$$\begin{aligned}
 x_1^* &= 0.6670128, & x_2^* &= 1.0223847, & x_3^* &= 0.2282871, & x_4^* &= 0.1848218, \\
 x_5^* &= 1.0909, & x_6^* &= 1.0909, & x_7^* &= 1.069036, & x_8^* &= 0.1066106, \\
 x_9^* &= -0.3387876.
 \end{aligned}$$

**Application L7. Optimization of a separation process in a membrane with three stages (MSP3)**

This application is taken from (Dembo, 1976; Hock & Schittkowski, 1981, pp. 124; Andrei, 1999a, pp. 802).

$$\begin{aligned}
& \min (x_{11} + x_{12} + x_{13}) \\
& \text{subject to} \\
& x_3 - x_2 \geq 0, \\
& x_2 - x_1 \geq 0, \\
& 1 - 0.002x_7 + 0.002x_8 \geq 0, \\
& x_{13} - 1.262626x_{10} + 1.231059x_3x_{10} \geq 0, \\
& x_5 - 0.03475x_2 - 0.975x_2x_5 + 0.00975x_2^2 \geq 0, \\
& x_6 - 0.03475x_3 - 0.975x_3x_6 + 0.00975x_3^2 \geq 0, \\
& x_5x_7 - x_1x_8 - x_4x_7 + x_4x_8 \geq 0, \\
& 1 - 0.002(x_2x_9 + x_5x_8 - x_1x_8 - x_6x_9) - x_5 - x_6 \geq 0, \\
& x_2x_9 - x_3x_{10} - x_6x_9 - 500x_2 + 500x_6 + x_2x_{10} \geq 0, \\
& x_2 - 0.9 - 0.002(x_2x_{10} - x_3x_{10}) \geq 0, \\
& x_4 - 0.03475x_1 - 0.975x_1x_4 + 0.00975x_1^2 \geq 0, \\
& x_{11} - 1.262626x_8 + 1.231059x_1x_8 \geq 0, \\
& x_{12} - 1.262626x_9 + 1.231059x_2x_9 \geq 0, \\
& x_{11} + x_{12} + x_{13} \geq 50, \\
& x_{11} + x_{12} + x_{13} \leq 250.
\end{aligned} \tag{7}$$

The solution of this application is as follows:

Nr.	Initial point	Lower bound	Solution	Upper bound
1	0.5	0.1	0.8037730	1
2	0.8	0.1	0.8999858	1
3	0.9	0.1	0.9709695	1
4	0.099	0.0001	0.09999994	0.1
5	0.899	0.1	0.1908132	0.9
6	0.5	0.1	0.4605417	0.9
7	489	0.1	574.0773	1000
8	80	0.1	74.0776	1000
9	650	500	500.0162	1000
10	450	0.1	0.1	500
11	149.9	1	20.23311	150
12	149.9	0.0001	77.34769	150
13	149.9	0.0001	0.006730541	150

$$f(x_0) = 449.700 \quad f(x^*) = 97.587532422$$

#### Application L8. Optimization of a separation process in a membrane with five stages (MSP5)

This application is taken from (Dembo, 1976; Andrei, 2003, pp. 367).

$$\begin{aligned}
& \min \{ & 1.262626(x_{12} + x_{13} + x_{14} + x_{15} + x_{16}) \\
& - 1.231060(x_1x_{12} + x_2x_{13} + x_3x_{14} + x_4x_{15} + x_5x_{16}) \} \\
& \text{subject to} & (8) \\
& x_6 - 0.03475x_1 - 0.975x_1x_6 + 0.00975x_1^2 \geq 0, \\
& x_7 - 0.03475x_2 - 0.975x_2x_7 + 0.00975x_2^2 \geq 0, \\
& x_8 - 0.03475x_3 - 0.975x_3x_8 + 0.00975x_3^2 \geq 0, \\
& x_9 - 0.03475x_4 - 0.975x_4x_9 + 0.00975x_4^2 \geq 0, \\
& x_{10} - 0.03475x_5 - 0.975x_5x_{10} + 0.00975x_5^2 \geq 0, \\
& x_7x_{11} - x_6x_{11} - x_1x_{12} + x_6x_{12} \geq 0, \\
& x_8 - x_7 - 0.002(x_7x_{12} + x_2x_{13} - x_8x_{13} - x_1x_{12}) \geq 0, \\
& 1 - x_8 - x_9 - 0.002(x_8x_{13} + x_3x_{14} - x_2x_{13} - x_9x_{14}) \geq 0, \\
& x_3x_{14} - x_9x_{14} - x_4x_{15} - 500x_{10} + 500x_9 + x_8x_{15} \geq 0, \\
& x_4x_{15} - x_5x_{16} - x_{10}x_{15} - 500x_4 + 500x_{10} + x_4x_{16} \geq 0, \\
& x_4x_{15} - x_5x_{16} - x_{10}x_{15} - 500x_4 + 500x_{10} + x_4x_{16} \geq 0, \\
& x_4 - 0.002x_4x_{16} + 0.002x_5x_{16} - 0.9 \geq 0, \\
& 1 - 0.002x_{11} + 0.002x_{12} \geq 0, \\
& x_{11} - x_{12} \geq 0, \\
& x_5 - x_4 \geq 0, \\
& x_4 - x_3 \geq 0, \\
& x_3 - x_2 \geq 0, \\
& x_2 - x_1 \geq 0, \\
& x_{10} - x_9 \geq 0, \\
& x_9 - x_8 \geq 0, \\
& 1.262626(x_{12} + x_{13} + x_{14} + x_{15} + x_{16}) \\
& - 1.231060(x_1x_{12} + x_2x_{13} + x_3x_{14} + x_4x_{15} + x_5x_{16}) - 50 \geq 0, \\
& - 1.262626(x_{12} + x_{13} + x_{14} + x_{15} + x_{16}) \\
& + 1.231060(x_1x_{12} + x_2x_{13} + x_3x_{14} + x_4x_{15} + x_5x_{16}) + 250 \geq 0,
\end{aligned}$$

Solution of the application is as follows:

Nr.	Initial point	Lower bound	Solution	Upper bound
1	0.8	0.1	0.8037651	0.9
2	0.83	0.1	0.8161088	0.9
3	0.85	0.1	0.9	0.9
4	0.87	0.1	0.9	0.9
5	0.91	0.9	0.9	1
6	0.09	0.0001	0.0999996	0.1
7	0.12	0.1	0.1070319	0.9
8	0.19	0.1	0.1908370	0.9
9	0.25	0.1	0.1908370	0.9
10	0.29	0.1	0.1908370	0.9
11	512	1	505.0219	1000
12	13.1	0.000001	5.046301	500
13	71.8	1	72.63798	500
14	640	500	500.0	1000
15	650	500	500.0	1000
16	5.7	0.000001	0.1042061e-5	500

$$f(x_0) = 284.6696 \quad f(x^*) = 174.787136606$$

**Application L9. Blending/pooling with five feeds and two products (POOL)** (Andrei, 1999a, pp. 808; Andrei, 2003, pp. 382)

$$\begin{aligned} \min & (1.1x_1 + 1.1x_2 + 1.1x_3 + 0.878x_4 + 0.878x_5 + 0.878x_6 + 0.878x_7 \\ & + 1.6x_8 + 1.6x_9 + 0.5x_{10} + 0.5x_{11} + 0.98x_{12} + 0.998x_{13}) \end{aligned}$$

subject to (9)

$$\begin{aligned} 90x_1 + 89x_2 + 91x_3 - (x_1 + x_2 + x_3)x_{14} &= 0, \\ 87x_4 + 90.5x_5 + 89.5x_6 + 94x_7 - (x_4 + x_5 + x_6 + x_7)x_{15} &= 0, \\ 89x_8 + 89.1x_9 - (x_8 + x_9)x_{16} &= 0, \\ 89.2x_{10} + 89.3x_{11} + x_{14}x_{29} + x_{15}x_{30} - (x_{10} + x_{11} + x_{29} + x_{30})x_{17} &= 0, \\ 89.4x_{12} + 89.5x_{13} + x_{15}x_{31} + x_{16}x_{32} - (x_{12} + x_{13} + x_{31} + x_{32})x_{18} &= 0, \\ 86x_1 + 85.5x_2 + 86.5x_3 - (x_1 + x_2 + x_3)x_{19} &= 0, \\ 83.2x_4 + 86.9x_5 + 85x_6 + 89.8x_7 - (x_4 + x_5 + x_6 + x_7)x_{20} &= 0, \\ 85.1x_8 + 85.3x_9 - (x_8 + x_9)x_{21} &= 0, \\ 85.4x_{10} + 85.2x_{11} + x_{19}x_{29} + x_{20}x_{30} - (x_{10} + x_{11} + x_{29} + x_{30})x_{22} &= 0, \\ 85.5x_{12} + 85.6x_{13} + x_{20}x_{31} + x_{21}x_{32} - (x_{12} + x_{13} + x_{31} + x_{32})x_{23} &= 0, \\ 0.78x_1 + 0.8x_2 + 0.81x_3 - (x_1 + x_2 + x_3)x_{24} &= 0, \\ 0.77x_4 + 0.775x_5 + 0.78x_6 + 0.82x_7 - (x_4 + x_5 + x_6 + x_7)x_{25} &= 0, \\ 0.785x_8 + 0.79x_9 - (x_8 + x_9)x_{26} &= 0, \\ 0.787x_{10} + 0.776x_{11} + x_{24}x_{29} + x_{25}x_{30} - (x_{10} + x_{11} + x_{29} + x_{30})x_{27} &= 0, \\ 0.783x_{12} + 0.779x_{13} + x_{25}x_{31} + x_{26}x_{32} - (x_{12} + x_{13} + x_{31} + x_{32})x_{28} &= 0, \\ x_1 + x_2 + x_3 - x_{29} &= 0, \\ x_4 + x_5 + x_6 + x_7 - x_{30} - x_{31} &= 0, \\ x_8 + x_9 - x_{32} &= 0, \\ x_{10} + x_{11} + x_{29} + x_{30} - x_{33} &= 0, \\ x_{12} + x_{13} + x_{31} + x_{32} - x_{34} &= 0. \end{aligned}$$

Solution of the application:

Nr.	Initial point	Lower bound	Solution	Upper bound
1	182	0	131.9361	11000
2	140	0	132.5016	11000
3	70	0	135.5624	11000
4	1070	0	264.3126	11000
5	47	0	273.9741	11000
6	11	0	270.9078	11000
7	12	0	290.8055	11000
8	700	0	398.3295	11000
9	60	0	401.6705	11000
10	190	0	100.0567	11000
11	0.1	0	99.94331	11000
12	0.1	0	0	11000
13	0.1	0	0	11000
14	88	0	90.00765	11000
15	88	0	90.33802	11000
16	88	0	89.05021	11000
17	88	0	89.98826	11000
18	88	0	89.65119	11000
19	88	0	86.00383	11000
20	88	0	86.30969	11000
21	88	0	85.20042	11000
22	88	0	85.98542	11000
23	88	0	85.71808	11000
24	0.77	0	0.7967923	11000
25	0.77	0	0.7869266	11000
26	0.77	0	0.7875104	11000
27	0.77	0	0.7897882	11000
28	0.77	0	0.7872380	11000
29	401	400	400	11000
30	402	400	400	11000
31	702	700	700	11000
32	810	800	800	11000
33	1001	1000	1000	11000
34	1510	1500	1500	11000

$$f(x_0) = 2743.368 \quad f(x^*) = 2785.80$$

#### Application L10. Distribution of electrons on a sphere (DES)

The problem is described in (Dolan, Moré, & Munson, 2004, pp. 5; see also Andrei, 2001; Andrei, 2003, pp. 348). Given  $n_p$  electrons, find the equilibrium state distribution (of minimal Coulomb potential) of the electrons positioned on a conducting sphere. The problem, known as the Thomson problem (raised in 1904), consists in finding the lowest energy configuration of  $n_p$  point charges on a conducting sphere. This is an important problem in physics and chemistry, which determines a minimal structure with respect to atomic positions.

If  $(x_i, y_i, z_i)$  are the positions of the  $n_p$  points (electrons), then the potential energy is

$$\sum_{i=1}^{n_p-1} \sum_{j=i+1}^{n_p} \left( (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \right)^{-\frac{1}{2}}, \quad (10a)$$

which must be minimized subject to the  $n_p$  constraints

$$x_i^2 + y_i^2 + z_i^2 = 1, \quad i = 1, \dots, n_p. \quad (10b)$$

The problem has a multitude of local minimizers at which the objective value is relatively close to the objective value at the global minimum. Also, the number of the local minimizers grows exponentially with  $n_p$ . Therefore, determining the global minimum is a difficult task, and solvers are usually expected to find only a local minimum.

For  $n_p = 50$  the value of the potential energy is 1055.1823147.

### Application L11. Hanging chain (HANG)

Find the chain of uniform density of length  $L$  suspended between two points with minimal potential energy. This is a classical problem, known as the dog problem (Cesari, 1983). It was suggested by Mittelmann, and it is described in (Dolan, Moré, & Munson, 2004, pp. 9). The problem is to determine a function  $x(t)$ , the shape of the chain that minimizes the potential energy

$$\int_0^1 x(t) \left( 1 + x'(t)^2 \right)^{1/2} dt,$$

subject to the constraints on the length of the chain

$$\int_0^1 \left( 1 + x'(t)^2 \right)^{1/2} dt = L,$$

as well as the end conditions  $x(0) = a$  and  $x(1) = b$ , where  $a$  and  $b$  are given constants.

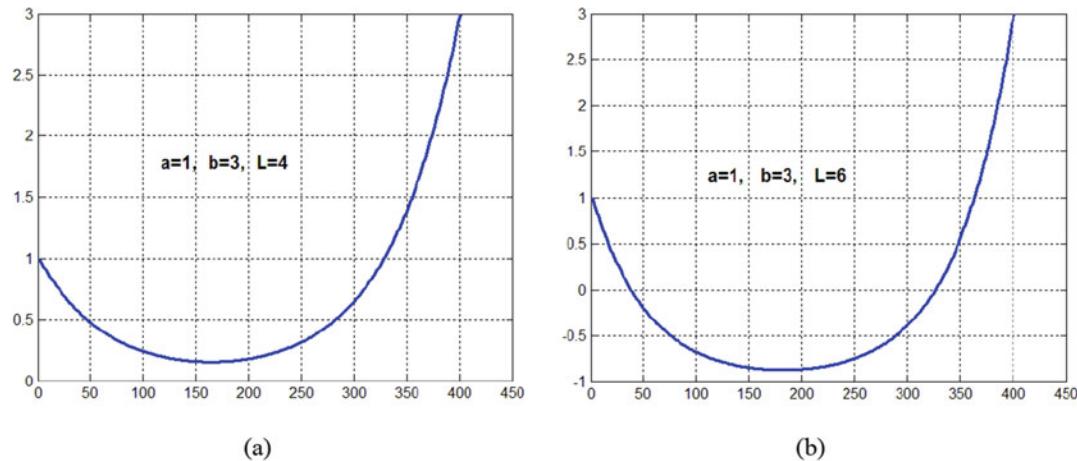
Another formulation of this problem is possible by introducing a control variable  $u(t) = x'_1(t)$  and the potential energy function

$$x_2(t) = \int_0^t x_1(s) \left( 1 + u(s)^2 \right)^{1/2} ds.$$

This formulation leads to minimizing the total potential energy  $x_2(1)$  subject to the differential equations

$$\begin{aligned} x'_1(t) &= u, \\ x'_2(t) &= x_1(t) \left( 1 + u(t)^2 \right)^{1/2}, \\ x'_3(t) &= \left( 1 + u(t)^2 \right)^{1/2}. \end{aligned}$$

By introducing  $n$  discretization points by means of a uniform time step of length  $h = 1/(n + 1)$ , then the discrete variant of the problem is



**Fig. L1** Hanging chain of minimal potential energy of length  $L = 4$  or  $L = 6$

$$\min h \sum_{i=1}^{n+1} \left( \frac{x_i + x_{i-1}}{2} \right) \sqrt{1 + \left( \frac{x_i - x_{i-1}}{h} \right)^2} \quad (11a)$$

subject to

$$\sum_{i=1}^{n+1} \sqrt{1 + \left(\frac{x_i - x_{i-1}}{h}\right)^2} = \frac{L}{h}, \quad (11b)$$

where  $x_0 = a$  and  $x_{n+1} = b$  (Bondarenko, Bortz, & Moré, 1999).

Figure L1a shows the form of the chain for  $a = 1, b = 3, L = 4$ , and  $nh = 400$  while Fig. L1b shows the form of the chain for  $a = 1, b = 3, L = 6$ , and  $nh = 400$

**Application L12. Determine the optimal mixing policy of two catalysts along the length of a tubular plug flow reactor involving several reactions (CAT)**

This application is described in (von Stryk, 1999) and (Dolan, Moré, & Munson, 2004, pp. 33) and has the following form:

$$x'_1(t) \equiv u(t)(10x_2(t) - x_1(t)). \quad (12a)$$

$$x_2'(t) \equiv y(t)(x_1(t) - 10x_2(t)) = (1 - y(t))x_2(t). \quad (12b)$$

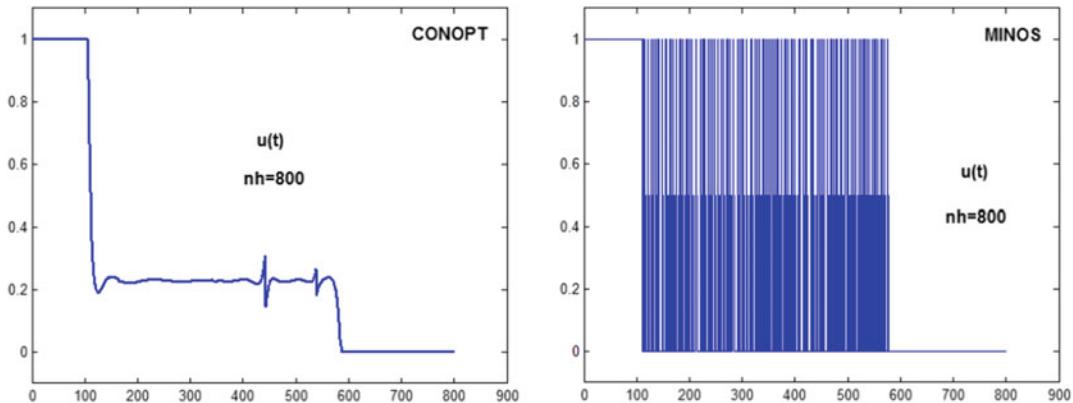
The initial conditions are  $x_1(0) = 1$  and  $x_2(t) = 0$ . The control variable  $u(t)$  represents the mixing ratio of the catalysts and satisfies the bounds

$$0 \leq \mu(t) \leq 1 \quad (13)$$

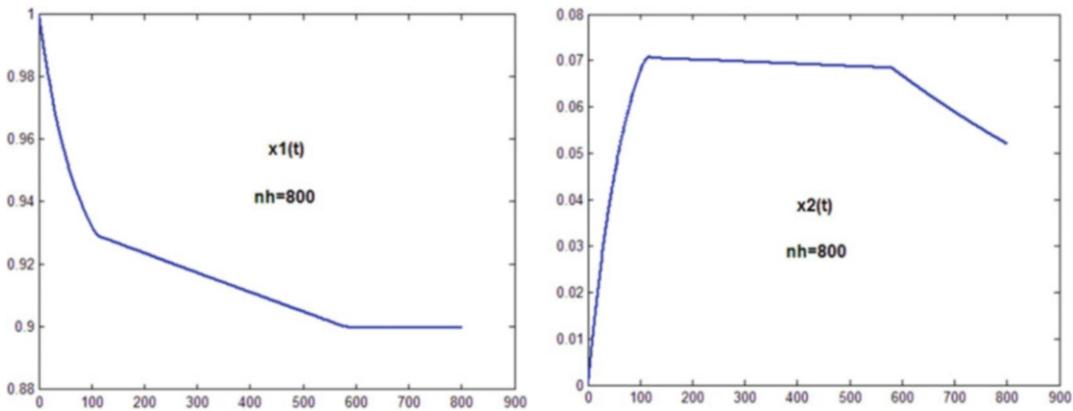
The problem is to minimize

$$-1 + x_1(t_c) + x_2(t_c) \quad t_c \equiv 1 \quad (14)$$

A uniform partition of the interval  $[0, 1]$  with  $n_h$  subintervals is considered and the equations discretized in a natural way. As initial points we take  $u = 0$ ,  $x_1 = 1$ , and  $x_2 = 0$ , evaluated in the grid points



**Fig. L2** Evolution of  $u(t)$  given by CONOPT and MINOS



**Fig. L3** Evolution of  $x_1(t)$  and  $x_2(t)$  given by CONOPT

Figure L2 presents the time evolution of the control variable  $u(t)$  given by CONOPT and MINOS, respectively. We see the bang-bang character of this evolution, which is imposed by the presence of bounds on the control variable.

Figure L3 presents the evolution of the variables  $x_1(t)$  and  $x_2(t)$  given by CONOPT.

#### Application L13. Optimal control of a continuous stirred-tank chemical reactor (CSTR)

Let us consider a continuous stirred-tank chemical reactor where the flow of a coolant through a coil inserted in the reactor is to control the first-order, irreversible exothermic reaction taking place in the reactor (Lapidus & Luus, 1967). The state variables are  $x_1(t)$  and  $x_2(t)$ , which represent the deviation from the steady-state temperature and the deviation from the steady-state concentration.  $u(t)$  is the normalized control variable representing the effect of the coolant flow on the chemical reaction. The state equations are

$$x'_1(t) = -2(x_1(t) + 0.25) + (x_2(t) + 0.5) \exp\left(\frac{25x_1(t)}{x_2(t) + 2}\right) - (x_1(t) + 0.25)u(t), \quad (15a)$$

$$x'_2(t) = (0.5 - x_2(t)) - (x_2(t) + 0.5) \exp\left(\frac{25x_1(t)}{x_2(t) + 2}\right). \quad (15b)$$

The initial conditions are  $x_1(0) = 0.05$  and  $x_2(0) = 0$ . The performance measure to be minimized is

$$J = \int_0^{0.78} (x_1^2(t) + x_2^2(t) + Ru^2(t)) dt, \quad (16)$$

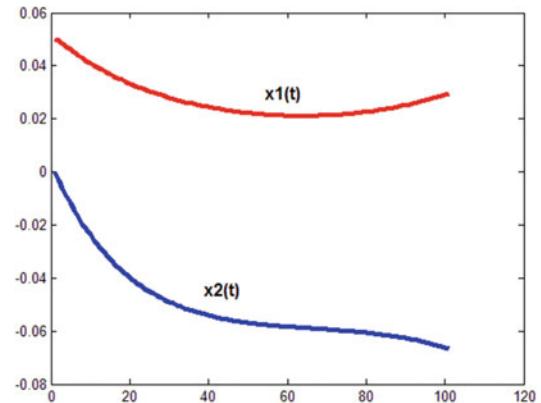
showing that the desired objective is to maintain the temperature and the concentration close to their steady-state values, without expending large amounts of control effort. Here  $R$  is a weighting factor that we arbitrarily shall select as 0.2.

The above optimal control problem can be represented as a nonlinear programming problem, thus avoiding the backward in time solution of the Riccati equation. To solve the problem, we approximate the state differential equations by differences, while the integral term in the performance measure, by summation.

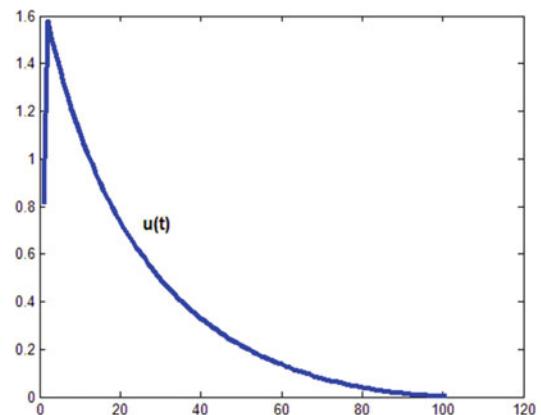
Taking a discretization of the time interval  $[0, t_f]$  in 100 subintervals, Figs. L4 and L5 show the time evolution of the state variables and of the control, respectively.

If for  $t \in [0, 0.78]$  the control variable is bounded as  $0.2 \leq u(t) \leq 1.2$ , then the time evolution of the variables and of the control is as in Figs. L6 and L7, respectively.

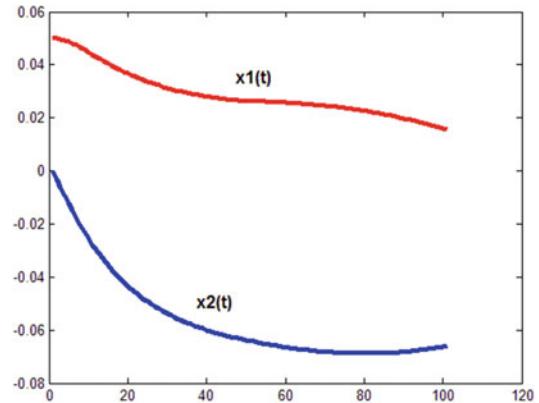
**Fig. L4** Evolution of  $x_1(t)$  and  $x_2(t)$



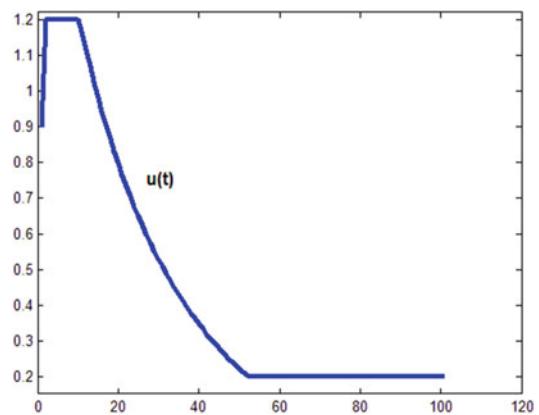
**Fig. L5** Evolution of  $u(t)$



**Fig. L6** Evolution of  $x_1(t)$  and  $x_2(t)$



**Fig. L7** Evolution of  $u(t)$



#### Application L14. Optimal temperature field in a rectangular area (DIFF)

This application considers the problem of determining the stationary temperature field in a rectangular area with heterogeneous thermal conductivity and with some source points of heat inside the area, as well as the fixed temperatures on the boundaries of the solution domain (McKinney & Savitsky, 2003). Therefore, consider an area with the thermal conductivity  $V [Wm^{-1}K^{-1}]$  heterogeneously and symmetrically distributed in the area of interest, i.e., there is a symmetric zone at the center of the solution domain, where  $V$  becomes larger than in the rest of the area. Then the heat transportation equation is

$$\frac{\partial(c\rho T)}{\partial t} = \nabla \cdot (V \nabla(c\rho T)) + I, \quad (17)$$

where  $c$  is the specific heat capacity of the substance (cal/kg degree),  $\rho$  is the density of the substance ( $kg/m^3$ ),  $T$  is the temperature of the substance (degree),  $V$  is the thermal conductivity,  $I$  is the point source of heat ( $cal/m^3 s$ ), and  $t$  is the time in seconds. On the boundary of the considered area, the temperature is fixed as

$$T = T_0(x, y) \text{ for } (x, y) \text{ on the boundary.}$$

In the steady-state case, no characteristics of the problem change in time. Therefore, the transportation equation becomes

$$\nabla \cdot (V \nabla (c\rho T)) + I = 0. \quad (18)$$

Now, if  $c$  and  $\rho$  do not vary spatially, the above equation can be written as follows:

$$\nabla \cdot (V \nabla T) + \frac{I}{c\rho} = 0. \quad (19)$$

To solve this equation, the differential operators must be approximated by algebraic analogues, and a conservative scheme of solving the resulting system of algebraic equations must be used. For this, the rectangular area is subdivided into a number of small rectangles with sides parallel to the  $x$  and  $y$  axes. Thus, a number of intersection points known as nodes are obtained. Using this system of nodes, the algebraic analogues for the differential operators  $\nabla \cdot ()$  and  $\nabla()$  can be constructed.

If  $a$  and  $B$  are differential functions, then

$$\nabla \cdot (a \nabla B) = a \nabla \cdot (\nabla B) + \nabla a \cdot \nabla B.$$

Using this formula, we get the following equation for the heat transportation:

$$V \nabla \cdot (\nabla T) + \nabla T \cdot \nabla V + \frac{I}{c\rho} = 0. \quad (20)$$

Using the finite difference in two dimensions, this equation can be discretized as

$$V_{ij} \frac{(T_{i+1,j} - 2T_{i,j} + T_{i-1,j})}{(\Delta x)^2} + V_{ij} \frac{(T_{i,j+1} - 2T_{i,j} + T_{i,j-1})}{(\Delta y)^2} + \frac{(V_{i+1,j} - V_{i-1,j})}{2(\Delta x)} \frac{(T_{i+1,j} - T_{i-1,j})}{2(\Delta x)} + \frac{(V_{i,j+1} - V_{i,j-1})}{2(\Delta y)} \frac{(T_{i,j+1} - T_{i,j-1})}{2(\Delta y)} + \frac{I}{c\rho} = 0,$$

for  $i = 1, \dots, n_x$  and  $j = 1, \dots, n_y$  where  $n_x$  and  $n_y$  are the numbers of the discretization points along the sides of the area, respectively.

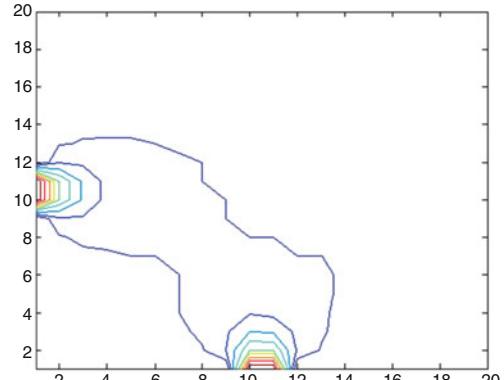
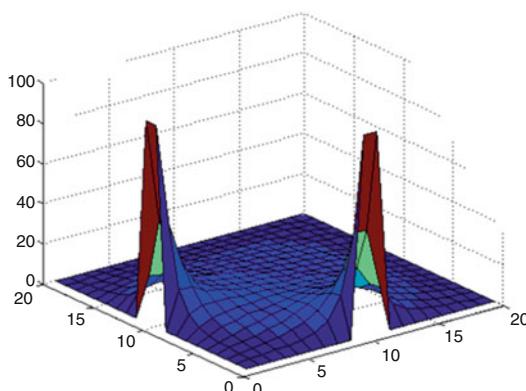
Consider the case in which the temperature is zero on the boundary of the area, except for the cells (I1, J10), (I1, J11), (I10, J1), and (I11, J1), where the temperature is about 100 °C. If  $n_x = 20$  and  $n_y = 20$ , then Fig. L8 presents the solution to the heat transportation problem with two fixed boundary conditions (Andrei, 2013e).

Let us now consider the situation in which an additional source of heat is placed in the center of the cell (I5, J5) of value 10,000 (cal/m³s). The distribution of the temperature and the constant level curves are as in Fig. L9.

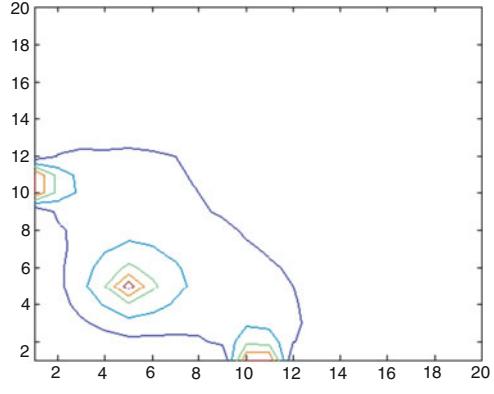
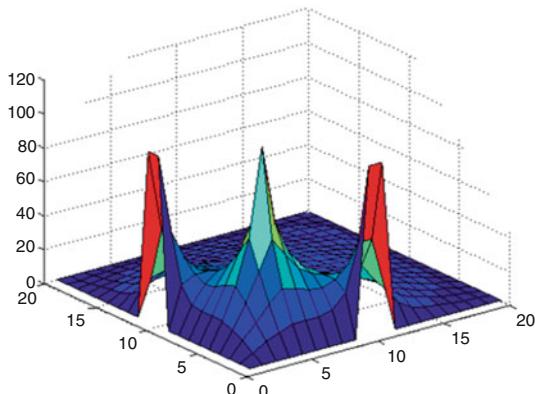
Observe that the symmetry of the problem and the boundary conditions determine the symmetry of the solution.

In the following, let us consider another situation in which, in the center of the cell (I17, J17), an additional source of value 15,000 is introduced. Figure L10 shows the temperature distribution in the rectangular area.

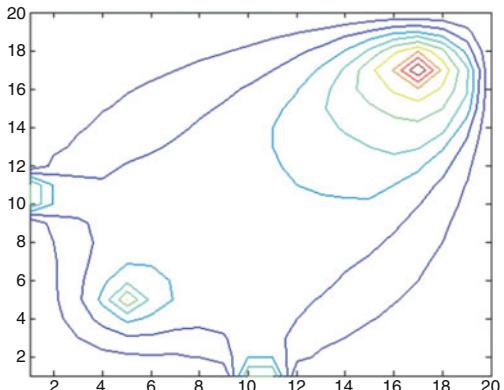
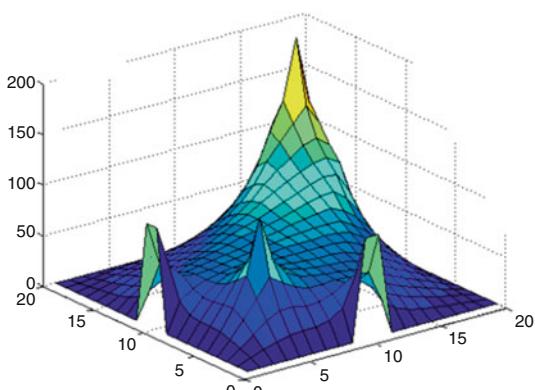
In the following, we consider a *time-dependent temperature field in a rectangular area with heterogeneous thermal conductivity and a point source of heat and heat flows through the border of the solution domain*. Let a point source of heat begin to heat the area in the point (I10, J18) of value 10,000 (cal/m³s). The heat from this source is distributed over the solution domain according to the



**Fig. L8** Solution to the heat transportation problem with two fixed boundary conditions



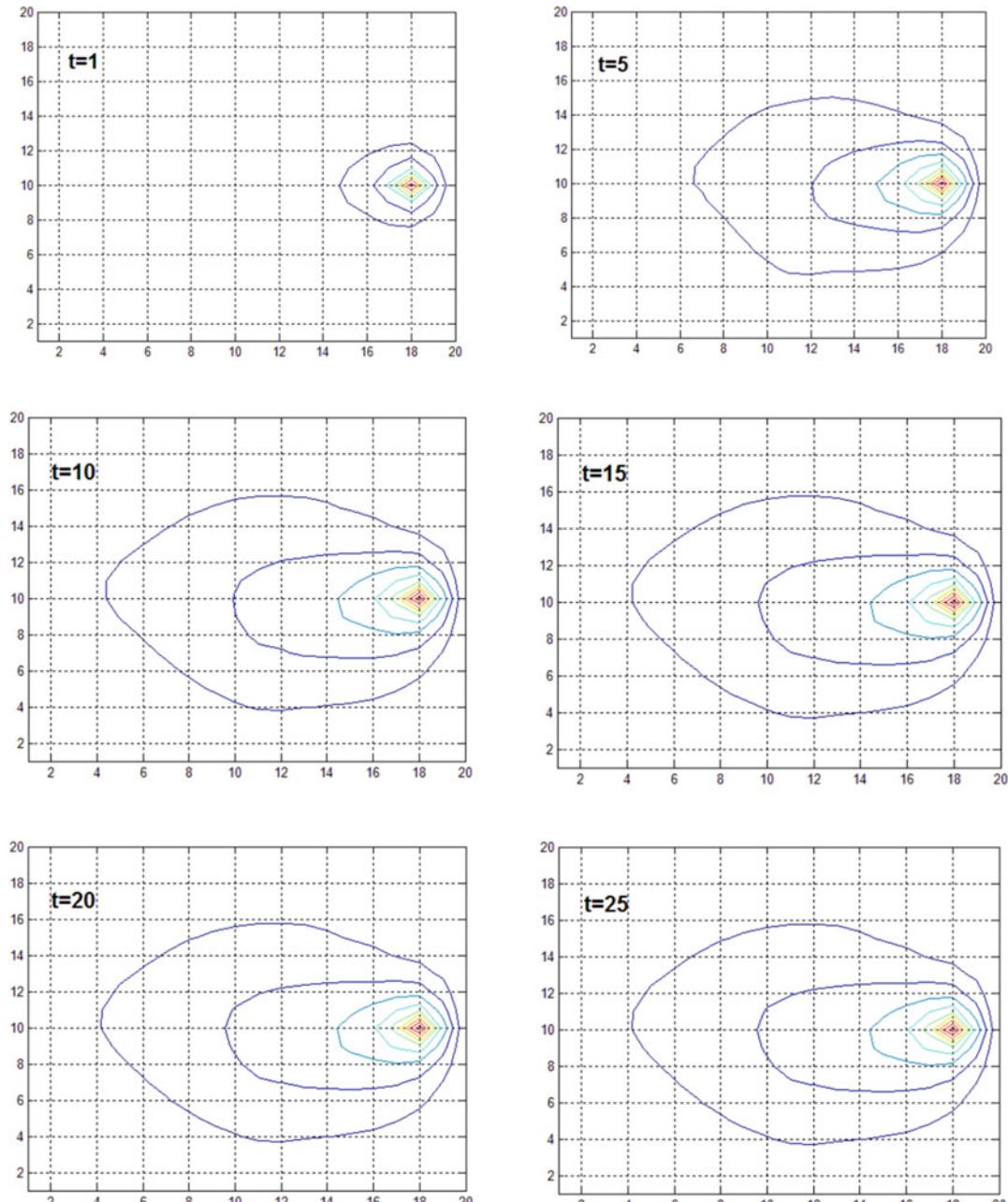
**Fig. L9** Solution to the heat transportation problem with two fixed boundary conditions and one heat source on the cell (15, J5)



**Fig. L10** Solution to the heat transportation problem with two fixed boundary conditions and two heat sources

heat transportation equation (19). The solution of the time-dependent temperature field consists in calculating the changes in the temperature field inside the solution domain at one time step on the basis of the previous temperature field. After the current period temperature field has been computed from the previous time period, it is saved in order to be used in the next time step.

Figure L11 presents the solution of the transient heat transportation problem at a number of some periods:  $t1$ ,  $t5$ ,  $t10$ ,  $t15$ ,  $t20$ , and  $t25$ . A stationary solution to the transient equation is found by using a



**Fig. L11** Solution of the transient heat transportation problem at 6 time periods

large number of iterations under stationary boundary conditions. After a number of iterations, the influence of the initial conditions becomes very small and  $\partial T/\partial t \rightarrow 0$ . In Fig. L11 observe that the field temperature converges to a stationary solution after about 15–20 iterations.

### **Application L15. Stationary flow of an incompressible fluid in a rectangular area (FLOW/FLOWO)**

The equations that describe the flow of a fluid are the conservation of the linear momentum and the mass conservation (McKinney & Savitsky, 2003). The conservation of the linear momentum of a viscous, incompressible fluid in horizontal plane area in the direction  $x$  is as follows:

$$\frac{\partial V_x}{\partial t} + V_x \frac{\partial V_x}{\partial x} + V_y \frac{\partial V_y}{\partial y} = -\frac{1}{\rho} \frac{\partial P}{\partial x} + \mu \nabla^2 V_x, \quad (21)$$

and in the direction  $y$

$$\frac{\partial V_y}{\partial t} + V_y \frac{\partial V_x}{\partial x} + V_y \frac{\partial V_y}{\partial y} = -\frac{1}{\rho} \frac{\partial P}{\partial y} + \mu \nabla^2 V_y. \quad (22)$$

The equation of the mass conservation (continuity equation) for the fluid is

$$\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} = 0, \quad (23)$$

where  $V_x$  is the  $x$  component of velocity [m/s],  $V_y$  is the  $y$  component of velocity [m/s],  $P$  is the pressure [pa],  $\mu$  is the kinematic viscosity [ $m^2/s$ ], and  $\rho$  is the density of the fluid [ $Kg/m^3$ ].

Observe that these equations are valid in the case when the viscosity is constant and the fluid weight acts perpendicularly to the  $x - y$  plane. Therefore, the gravitational force is not present in these equations.

With these preparatives, let us consider a stationary flow of an incompressible fluid in a rectangular area with given inflow of fluid on the borders of the solution domain (McKinney & Savitsky, 2003). In this application, the zones of inflow and outflow of water are on two opposite sides. On the zone of inflow, the boundary conditions are  $V_x = 0.5$  [m/s] and  $V_y = 0$  [m/s]. On the zone of outflow, the boundary conditions are  $\frac{\partial V_x}{\partial x} = 0$  and  $V_y = 0$ . On the other sides of the rectangle, the so-called “non-slip” conditions are taken as  $V_x = 0$  [m/s] and  $V_y = 0$  [m/s].

To compute the flow of water, a finite difference analog of the differential equations will be considered. The solution domain is divided into small rectangles by means of a grid of parallel lines and to each intersection point the indices  $(i, j)$  are associated. At each point, the fluid velocity and the pressure are computed. The fluid velocity is not computed in points, but on the lines connecting the points. On the lines parallel to the  $x$  axis,  $V_x$  is computed. On the other hand, on the lines parallel to the  $y$  axis, we compute  $V_y$ . Therefore, the continuity equation has the following finite difference form:

$$\frac{V_{x,i,j} - V_{x,i-1,j}}{\Delta x} + \frac{V_{y,i,j} - V_{y,i,j-1}}{\Delta y} = 0. \quad (24)$$

The pressure is computed at the node points  $(i, j)$  of each line, where the pressure gradient is computed from the velocity:

$$\left. \frac{\partial P}{\partial x} \right|_{ij} = \frac{P_{i+1,j} - P_{i,j}}{\Delta x}, \quad (25)$$

$$\left. \frac{\partial P}{\partial y} \right|_{ij} = \frac{P_{i,j+1} - P_{i,j}}{\Delta y}. \quad (26)$$

The diffusive terms from (21) and (22) are computed as

$$\mu \nabla^2 V_x \approx \mu \left( \frac{V_{x,i+1,j} - 2V_{x,i,j} + V_{x,i-1,j}}{(\Delta x)^2} + \frac{V_{x,i,j+1} - 2V_{x,i,j} + V_{x,i,j-1}}{(\Delta y)^2} \right), \quad (27)$$

$$\mu \nabla^2 V_y \approx \mu \left( \frac{V_{y,i+1,j} - 2V_{y,i,j} + V_{y,i-1,j}}{(\Delta x)^2} + \frac{V_{y,i,j+1} - 2V_{y,i,j} + V_{y,i,j-1}}{(\Delta y)^2} \right). \quad (28)$$

Since there are errors introduced by the algebraic approximations, it is not always possible to approximate the differentials by finite differences. Analyzing the system of Eqs. (21) and (22), we see that these equations contain the velocity components connected by the pressure gradient  $\nabla P$ . If  $P$  is not connected with boundary conditions, then all the approximation errors will leave the solution domain through the borders. In the Eq. (23) the velocity vector is in strong contact with the boundary conditions. Therefore, in the calculations of the continuity equation, we expect big errors to appear at each point. To overcome this difficulty, the continuity equation is written in a slightly different (relaxed) form as

$$\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} = \delta, \quad (29)$$

where  $\delta$  is the error in the approximation of the differentials, which must be minimized. The final system of equations which have to be solved has the following form:

$$-\frac{1}{\rho} \frac{\partial P}{\partial x} + \mu \nabla^2 V_x = 0, \quad (30)$$

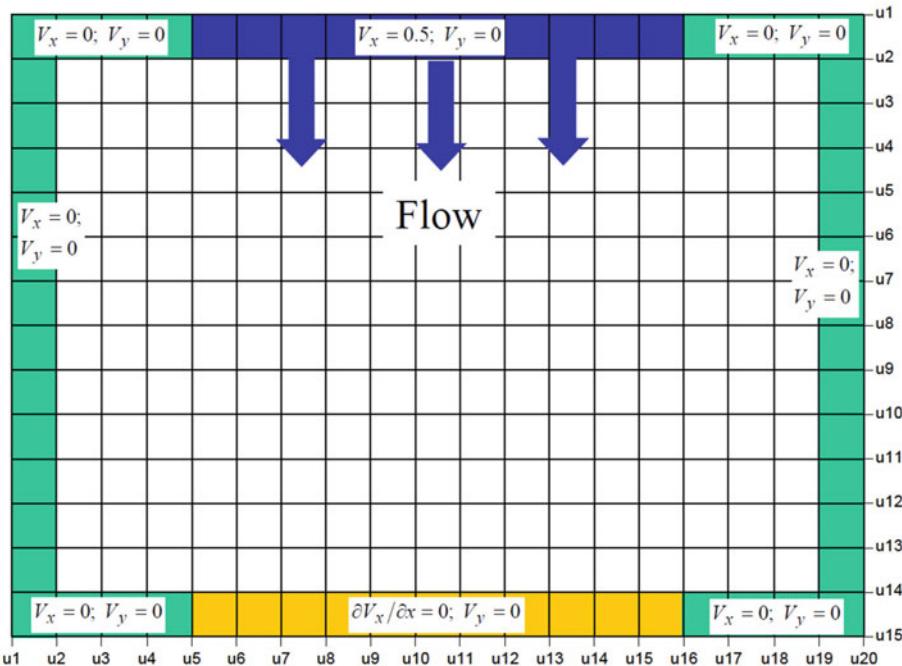
$$-\frac{1}{\rho} \frac{\partial P}{\partial y} + \mu \nabla^2 V_y = 0, \quad (31)$$

$$\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} = \delta. \quad (32)$$

The finite difference form of (30), (31), and (32) is as follows:

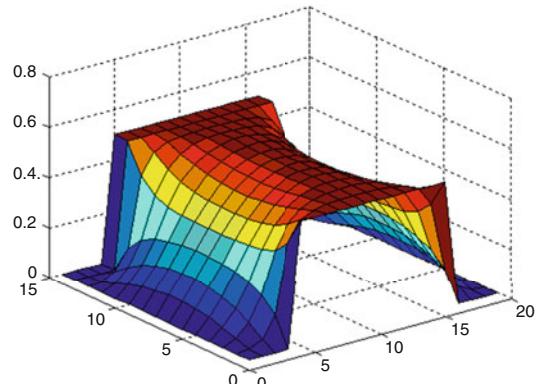
$$\begin{aligned} \frac{1}{\rho} \frac{P_{i+1,j} - P_{i,j}}{\Delta x} &= \mu \left( \frac{V_{x,i+1,j} - 2V_{x,i,j} + V_{x,i-1,j}}{(\Delta x)^2} + \frac{V_{x,i,j+1} - 2V_{x,i,j} + V_{x,i,j-1}}{(\Delta y)^2} \right), \\ \frac{1}{\rho} \frac{P_{i,j+1} - P_{i,j}}{\Delta y} &= \mu \left( \frac{V_{y,i+1,j} - 2V_{y,i,j} + V_{y,i-1,j}}{(\Delta x)^2} + \frac{V_{y,i,j+1} - 2V_{y,i,j} + V_{y,i,j-1}}{(\Delta y)^2} \right), \\ \frac{V_{x,i,j} - V_{x,i-1,j}}{\Delta x} + \frac{V_{y,i,j} - V_{y,i,j-1}}{\Delta y} &= \delta_{i,j}. \end{aligned}$$

In the following, consider a discretization of the rectangular area in which the  $x$  axis is discretized in 15 intervals and the  $y$  axis in 20, where  $\Delta x = \Delta y = 1$ , Fig. L12 (McKinney, & Savitsky, 2003).



**Fig. L12** Flow domain and its discretization

**Fig. L13** Velocity in the direction  $x$

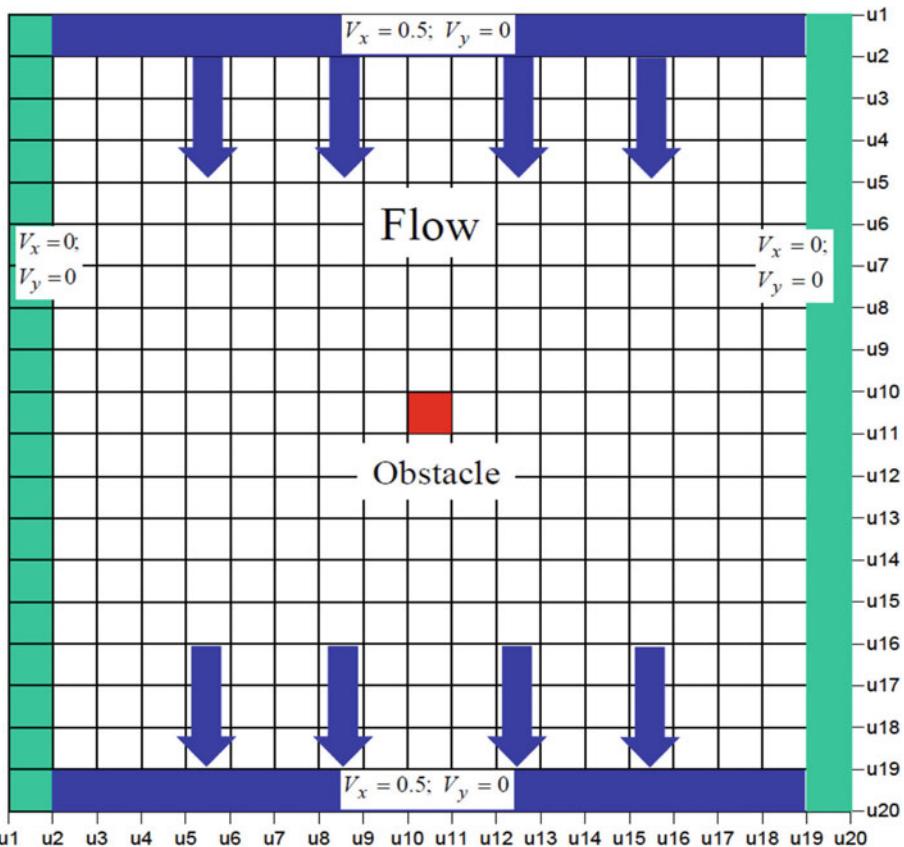
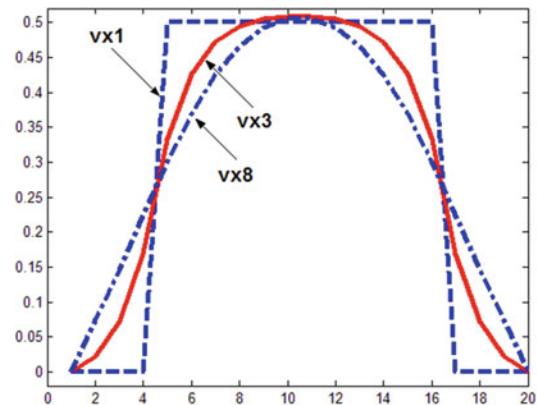


Let the fluid density be equal to 1000 [Kg/m<sup>3</sup>]. The kinematic viscosity of the fluid is 0.005 [m<sup>2</sup>/s]. The flow of the fluid is parallel to the  $x$  axis. Now, consider that in the interval [u5, u15] on the  $y$  axis, the following boundary conditions are active:  $V_x = 0.5$  [m/s] and  $V_y = 0$  [m/s]. On the opposite side, the boundary conditions are  $\frac{\partial V_x}{\partial x} = 0$  and  $V_y = 0$ . On the rest of the intervals of discretization, the boundary conditions are  $V_x = 0$  [m/s] and  $V_y = 0$  [m/s].

The error in the approximation through finite differences is zero. Besides, notice that the optimal value of the objective function is also zero. Figure L13 presents the velocity evolution in the direction  $x$ .

Figure L14 shows three cross-sections of velocity  $V_x$  in the direction  $x$ .

**Fig. L14** Three cross-sections of velocity in the direction  $x$

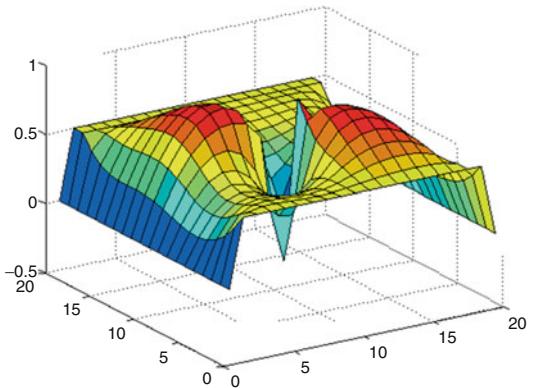


**Fig. L15** Flow domain with an obstacle

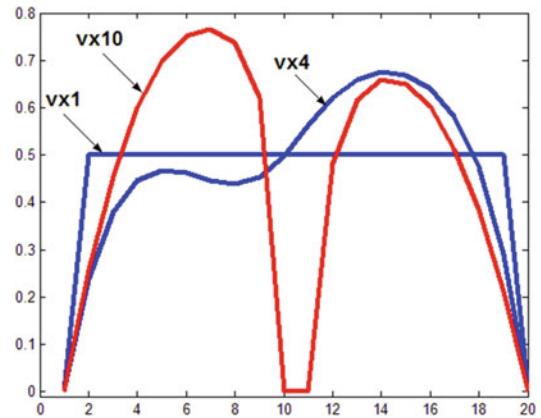
In the following, we compute the stationary flow of water in a rectangular area in the presence of an obstacle (Fig. L15). As above, let us consider a rectangular area. On the top edge of the area an inflow of water with velocity of 0.5 m/s occurs.

The same flow leaves the area through the bottom edge of the domain. The boundary conditions are the same as in the previous application, but in the middle of the domain there is a symmetrically

**Fig. L16** Velocity in the direction  $x$



**Fig. L17** Three cross-sections of velocity in the direction  $x$



located obstacle around which the fluid is compelled to flow as represented in Fig. L15 (McKinney & Savitsky, 2003). Figure L16 presents the evolution of the velocity  $V_x$  in the direction  $x$ .

Figure L17 shows three cross-sections of velocity  $V_x$  in the direction  $x$ .

Observe the asymmetry of the approximations. This happens because the velocities are not determined at the cell centers, but on the cell faces.

If the inertial terms of the equations are considered, then they can be computed by using the “marker and cell” method, as described in (Peyret & Taylor, 1985)

$$V_x \frac{\partial V_x}{\partial x} + V_y \frac{\partial V_x}{\partial y} \cong V_{x,i,j} \frac{V_{x,i+1,j} - V_{x,i,j}}{2\Delta x} + \frac{V_{y,i+1,j} + V_{y,i,j} + V_{y,i,j-1} + V_{y,i+1,j-1}}{4} \frac{V_{x,i,j+1} - V_{x,i,j-1}}{2\Delta y},$$

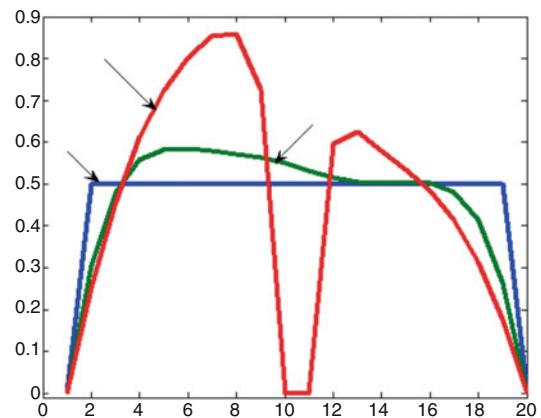
$$V_x \frac{\partial V_y}{\partial x} + V_y \frac{\partial V_y}{\partial y} \cong V_{y,i,j} \frac{V_{y,i,j+1} - V_{y,i,j-1}}{2\Delta y} + \frac{V_{x,i,j} + V_{x,i,j+1} + V_{x,i-1,j+1} + V_{x,i-1,j}}{4} \frac{V_{y,i+1,j} - V_{y,i-1,j}}{2\Delta x}.$$

Figure L18 presents three cross-sections of velocity in the direction  $x$ , with inertial terms.

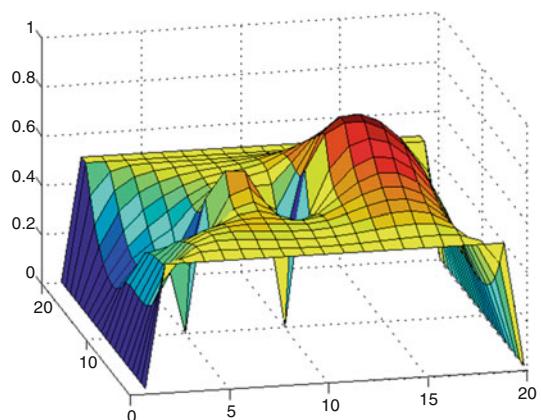
It is worth seeing the solution of the flow in a rectangular area with two obstacles (u10, u5) and (u10, u10). Figure L19 shows the velocity in the direction  $x$ .

Figure L20 shows three cross-sections of velocity  $V_x$  in the direction  $x$ .

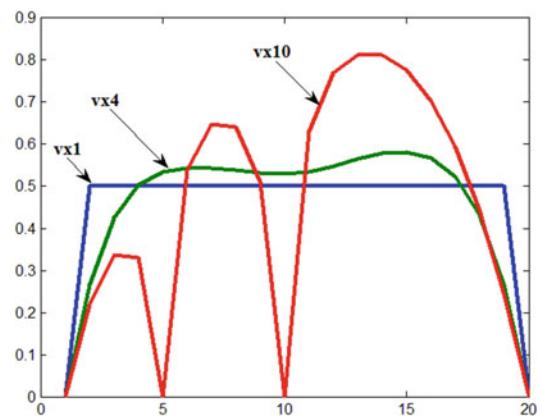
**Fig. L18** Three cross-sections of velocity in the direction  $x$  with inertial effects



**Fig. L19** Velocity in the direction  $x$  for two obstacles



**Fig. L20** Three cross-sections of velocity in the direction  $x$



#### Application L16. Fed-batch fermenter for penicillin production (PENICI)

This problem deals with the dynamic optimization of a fed-batch fermenter for the production of penicillin through anaerobic glucose fermentation. The dynamic optimization of this process with fixed final time was shown in (Bang, Alonso, & Singh, 1997; Cuthrell & Biegler, 1989; Luus, 1993;

Larrosa, 2008). The problem consists in maximizing the total amount of penicillin produced by using the feed rate of substrate as control variable. The mathematical model of this application is as follows:

$$\begin{aligned}
 & \max (y_2(t_f)y_4(t_f)) \\
 & \text{subject to} \\
 & \frac{dy_1}{dt} = h_1 y_1 - \frac{uy_1}{500y_4}, \\
 & \frac{dy_2}{dt} = h_2 y_1 - 0.01 y_2 - \frac{uy_2}{500y_4}, \\
 & \frac{dy_3}{dt} = -\frac{h_1 y_1}{0.47} - \frac{h_2 y_1}{1.2} - \frac{0.029 y_1 y_3}{0.0001 + y_3} + \frac{u}{y_4} \left(1 - \frac{y_3}{500}\right), \\
 & \frac{dy_4}{dt} = \frac{u}{500},
 \end{aligned} \tag{33}$$

where

$$h_1 = \frac{0.11 y_3}{0.006 y_1 + y_3}, \quad h_2 = \frac{0.0055 y_3}{0.0001 + y_3(1 + 10 y_3)}.$$

In this model,  $y_1$  represents the concentration of the biomass,  $y_2$  is the penicillin concentration,  $y_3$  is the concentration of the substrate, and  $y_4$  is the fermenter volume (in L). The initial conditions are  $y_1(0) = 1.5$ ,  $y_2(0) = 0.1$ ,  $y_3(0) = 0.1$  and  $y_4(0) = 7$ . The final product is destined to human consumption. Therefore, the concentrations of the present species are subject to the following path constraints:

$$0 \leq y_1 \leq 40, \quad 0 \leq y_3 \leq 25, \quad 0 \leq y_4 \leq 10.$$

The control variable is bounded as  $0 \leq u \leq 50$  and the total process time is fixed as  $t_f = 100$  h.

For solving this application and for showing some other optimal control examples, let us discuss some aspects of the numerical solutions of the ordinary differential equations

$$y' = f(t, y), \quad t_0 \leq t \leq t_f, \quad y(t_0) = y_0. \tag{34}$$

One of the simplest methods for solving (34) is *Euler's method*. For this, the interval  $[t_0, t_f]$  is divided by the *mesh-points*  $t_k = t_0 + kh$ ,  $k = 0, \dots, N$ , where  $h = (t_f - t_0)/N$  is the *step size* and  $N$  is a positive integer. Now, consider that for each  $k$  we seek a numerical approximation  $y_k$  to  $y(t_k)$ , the value of the analytical solution at the mesh point  $t_k$ . Euler's method proceeds by integrating the differential equation (34) between two consecutive points  $t_k$  and  $t_{k+1}$ :

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t)) dt, \quad k = 0, \dots, N - 1, \tag{35}$$

and then, by applying the numerical integration rule,

$$\int_{t_k}^{t_{k+1}} g(t) dt \approx hg(t_k), \tag{36}$$

called the *rectangle rule*, with  $g(t) = f(t, y(t))$ , to get

$$y(t_{k+1}) \approx y(t_k) + hf(t_k, y(t_k)), \quad k = 0, \dots, N-1, \quad y(t_0) = y_0.$$

This simple derivation motivates the definition of *Euler's method* as

$$y_{k+1} = y_k + hf(t_k, y_k), \quad k = 0, \dots, N-1. \quad (37)$$

This idea can be very easily generalized by replacing the rectangle rule in the derivation of Euler's method with one-parameter family of the integration rule of the form

$$\int_{t_k}^{t_{k+1}} g(t) dt \approx h[(1-\theta)g(t_k) + \theta g(t_{k+1})],$$

where  $\theta \in [0, 1]$  is a parameter. Now, applying it in (35) with  $g(t) = f(t, y(t))$  we get

$$y(t_{k+1}) \approx y(t_k) + h[(1-\theta)f(t_k, y(t_k)) + \theta f(t_{k+1}, y(t_{k+1}))], \quad k = 0, \dots, N-1, \quad y(t_0) = y_0.$$

Therefore, this motivates the introduction of the following one-parameter family of methods: given that  $y_0$  is specified by (34), define

$$y_{k+1} = y_k + h[(1-\theta)f(t_k, y_k) + \theta f(t_{k+1}, y_{k+1})], \quad k = 0, \dots, N-1, \quad (38)$$

known as the  *$\theta$ -method*. Observe that for  $\theta = 0$  we recover Euler's method. Now, for  $\theta = 1$  and  $y_0$  given by (34), we get

$$y_{k+1} = y_k + hf(t_{k+1}, y_{k+1}), \quad k = 0, \dots, N-1, \quad (39)$$

known as the *implicit* or the *backward Euler method*, since, unlike Euler's method defined by (37) and (39) requires the solution of an implicit equation in order to determine  $y_{k+1}$ , given  $y_k$ . In this context, (37) is called the *explicit* or the *forward Euler method*. For  $\theta = 1/2$  in (38), we get another interesting computational scheme: given that  $y_0$  is specified by (34),  $y_{k+1}$  are computed as

$$y_{k+1} = y_k + \frac{1}{2}h[f(t_k, y_k) + f(t_{k+1}, y_{k+1})], \quad k = 0, \dots, N-1, \quad (40)$$

known as the *trapezoidal rule method*. Like the backward Euler method, the trapezoidal rule is implicit: in order to compute  $y_{k+1}$ , a nonlinear system of algebraic equations must be solved.

Each of these methods is *consistent* with the ordinary differential equation (34). That is, if we plug solutions to the exact equation into the numerical method, we get a *small local error*. For example, the forward Euler has consistency of order 1, and the trapezoidal rule has second-order consistency.

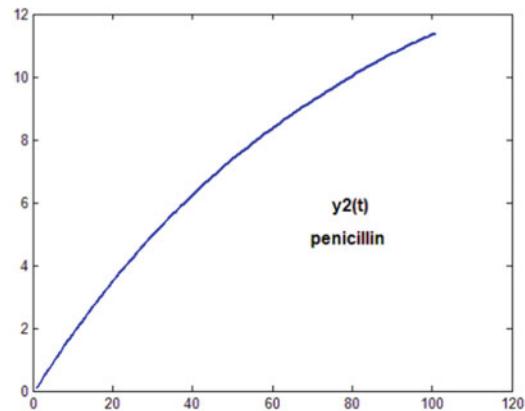
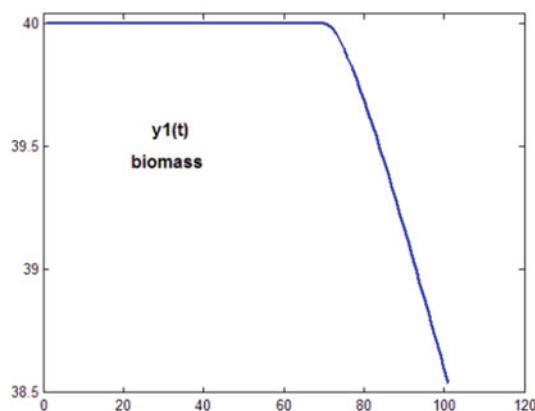
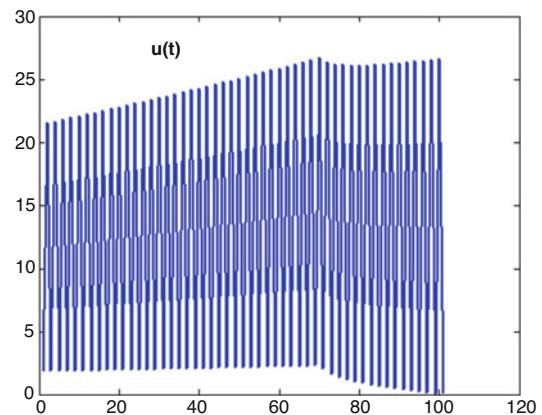
While the trapezoidal rule method leads to more accurate approximations, the forward Euler method is less convenient from the computational viewpoint since it requires the solution of the implicit equations at each mesh point  $x_{k+1}$  to get  $y_{k+1}$ . To overcome this difficulty, an attractive computational scheme lies in using the forward Euler method to compute an initial crude approximation for  $y(t_{k+1})$  and then to use this value within the trapezoidal rule for obtaining a more accurate approximation of  $y(t_{k+1})$ . The resulting numerical method is

$$y_{k+1} = y_k + \frac{1}{2}h[f(t_k, y_k) + f(t_{k+1}, y_k + hf(t_k, y_k))], \quad k = 0, \dots, N-1, \quad (41)$$

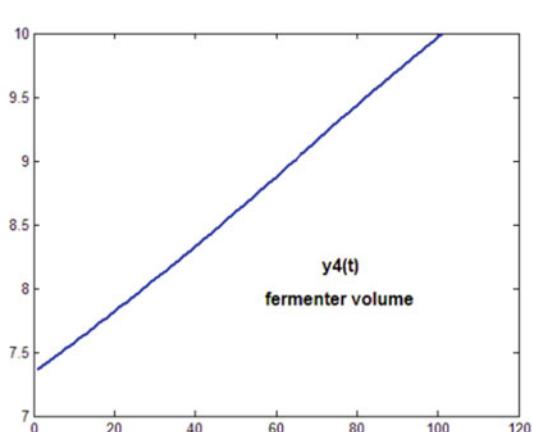
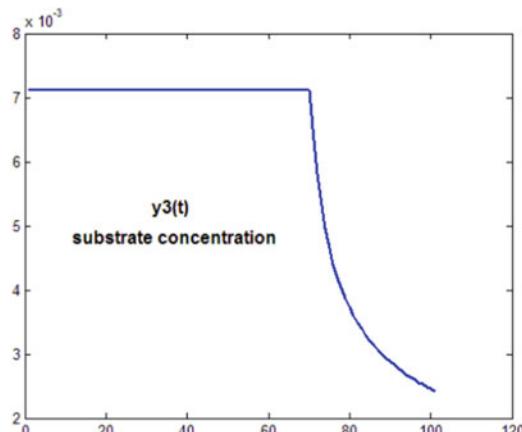
known as the *improved Euler method*. This is an explicit one-step computational scheme.

Using the trapezoidal rule (40), Fig. L21 presents the evolution of the control  $u(t)$ . Observe the bang-bang character of this evolution.

**Fig. L21** Evolution of the control  $u(t)$



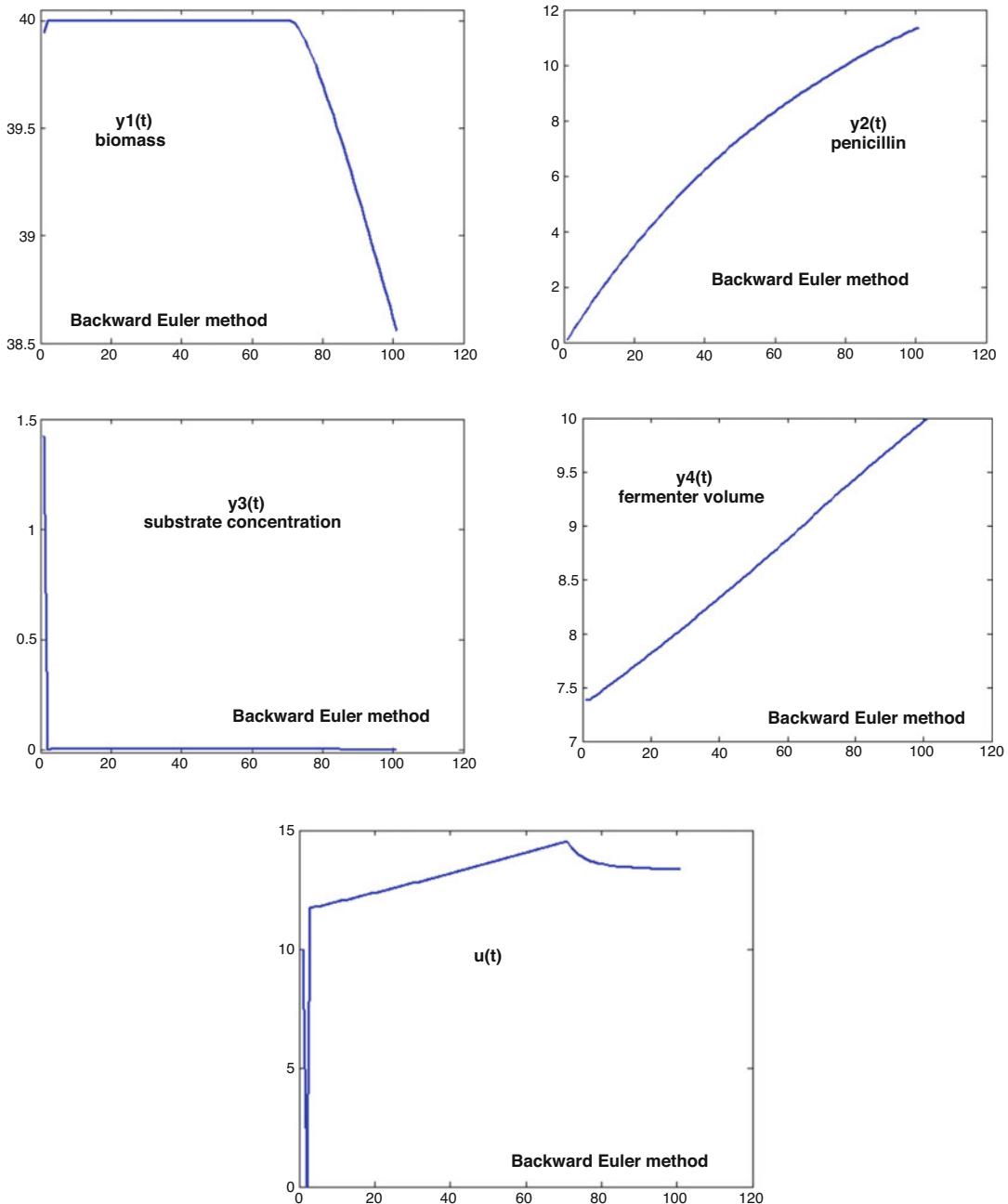
**Fig. L22** State variables  $y_1(t)$  and  $y_2(t)$



**Fig. L23** State variables  $y_3(t)$  and  $y_4(t)$

Figure L22 presents the evolution of the state variables  $y_1(t)$  (biomass) and  $y_2(t)$  (penicillin).

Figure L23 presents the time evolution of the state variables  $y_3(t)$  (substrate concentration) and  $y_4(t)$  (volume).



**Fig. L24** Time evolution of variables (backward Euler method)

It is interesting to see the solution of this application when the backward Euler method is used. Figure L24 presents the time evolution of the variables associated to this application.

The trapezoidal rule method is more accurate, better spotting the characteristics of the optimal control  $u(t)$  as well as of the state variables  $y_i(t)$ ,  $i = 1, \dots, 4$ .

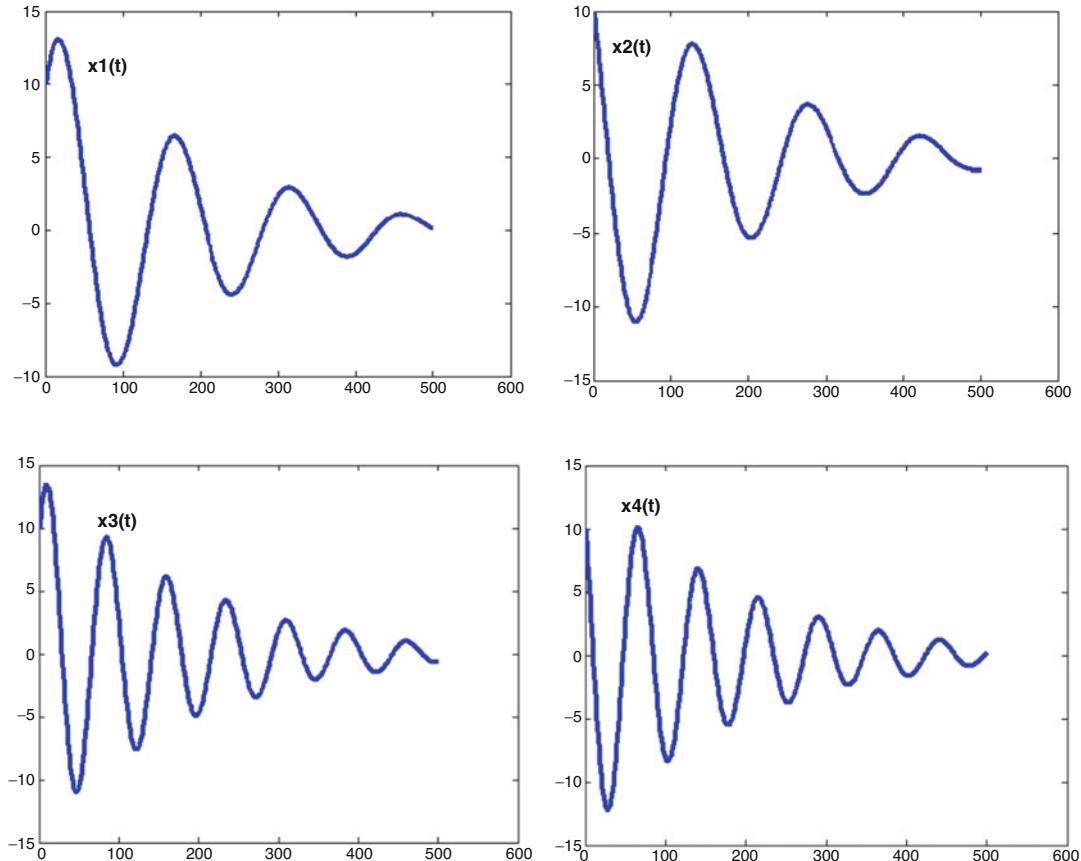
**Application L17. A standard linear lumped parameter system (CONT)**

For a linear dynamic system, it is desired to find the optimal control which will drive the state vector from its initial state to the origin in minimum time. The mathematical model of this application is as follows:

$$\begin{aligned} & \min \sum_{i=1}^4 x_i^2(t) \\ & \text{subject to} \\ & \frac{dx_1}{dt} = -0.5x_1 + 5x_2, \quad \frac{dx_2}{dt} = -5x_1 - 0.5x_2 + u, \\ & \frac{dx_3}{dt} = -0.6x_3 + 10x_4, \quad \frac{dx_4}{dt} = -10x_3 - 0.6x_4 + u, \\ & -1 \leq u(t) \leq 1, \end{aligned} \tag{42}$$

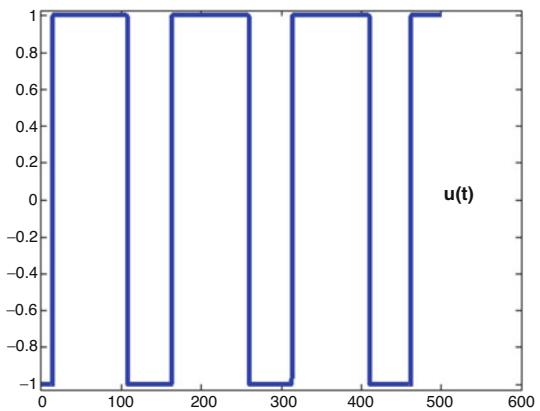
with the initial conditions:  $x(t_0) = [10 \quad 10 \quad 10 \quad 10]$  and the final time  $t_f = 4.2$  (Nishida, Liu, Lapidus, & Hiratsuka, 1976; Irizarry, 2005).

Figures L25 and L26 present the time evolution of the state variables and the control of this application, respectively.

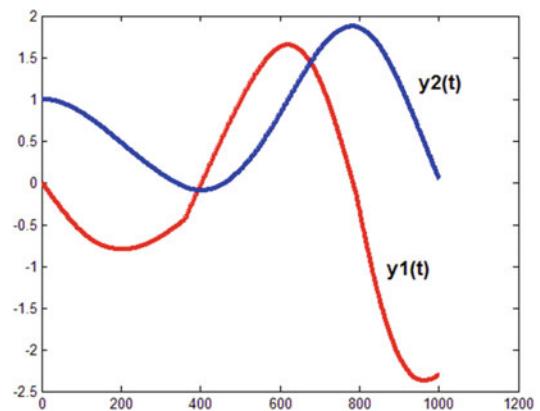


**Fig. L25** Time evolution of the state variables

**Fig. L26** Time evolution of the control  $u(t)$



**Fig. L27** Evolution of  $y_1(t)$  and  $y_2(t)$



### Application L18. Van der Pol oscillator (POL)

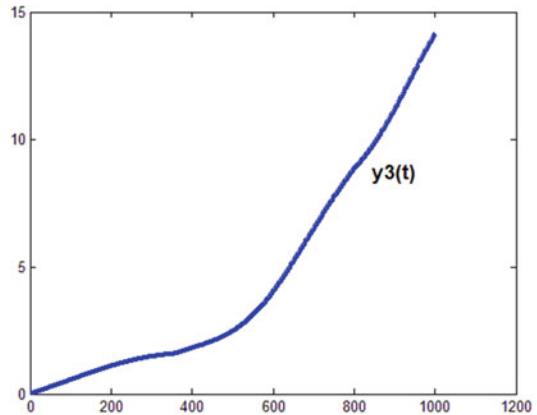
One of the classical equations of nonlinear dynamics was formulated by van der Pol (1927). It was a model for an electrical circuit with a triode. A brief description of this circuit is given in Strogatz (1994). The van de Pol oscillator has the following mathematical expression:

$$\begin{aligned} & \max_{u(t)} y_3(t_f) \\ & \text{subject to} \\ & \frac{dy_1}{dt} = (1 - y_2^2) - y_2 + u, \\ & \frac{dy_2}{dt} = y_1, \\ & \frac{dy_3}{dt} = y_1^2 + y_2^2 + u^2, \\ & -0.3 \leq u(t) \leq 1.0, \end{aligned} \tag{43}$$

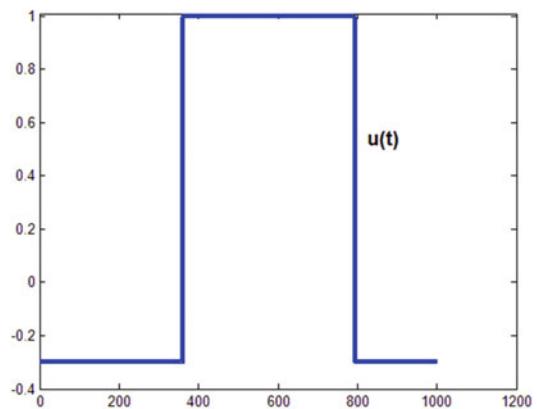
with the initial conditions  $y(0) = [0, 1, 0]$  and the final time  $t_f = 5$ .

Figures L27, L28, and L29 show the evolution of the state variables and of the control for this application.

**Fig. L28** Evolution of  $y_3(t)$



**Fig. L29** Control  $u(t)$



More optimization applications are found in (Andrei, 2003, 2011c, 2013e). Observe that the applications described in this chapter are classified in two classes. The first one is the *static nonlinear optimization*, and the second one is the *dynamic nonlinear optimization*, often called *optimal control*. The strategy for solving optimal control problems is to “kill” the dynamics, i.e., to transform the dynamic nonlinear optimization problem into a static one. In this way, we get a large-scale optimization problem for which the Jacobian of the constraints has a block diagonal or a block triangular structure. Some optimizers (e.g., CONOPT) can take into consideration this structure to solve the problem in an efficient and robust manner.

### Notes and References

The description of all these 18 nonlinear optimization applications (both in algebraic form and in the GAMS technology) can be seen in (Andrei, 2013e, 2015a, and 2017c).

---

## Appendix D: The MINPACK-2 Collection

---

### Large-Scale Unconstrained Optimization Applications

This Appendix includes a number of six applications from the MINPACK-2 collection (Averick, Carter, & Moré, 1991; Averick, Carter, Moré, & Xue, 1992). The mathematical models of these applications are expressed as unconstrained optimization problems.

#### Application A1. Elastic-Plastic Torsion

The description and the physical interpretation of the torsion problem are discussed in Glowinski (1984, pp. 41–55). We follow the presentation of this problem from (Averick, Carter, Moré, & Xue, 1992). The elastic plastic torsion problem arises from the determination of the stress field on an infinitely long cylindrical bar. The infinite-dimensional version of this problem is of the following form:

$$\min \{q(v) : v \in K\},$$

where  $q : K \rightarrow \mathbb{R}$  is a quadratic function

$$q(v) = \frac{1}{2} \int_D \|\nabla v(x)\|^2 dx - c \int_D v(x) dx$$

for some constant  $c$ , and  $D$  is a bounded domain with smooth boundary. The convex set  $K$  is defined as

$$K = \{v \in H_0^1(D) : |v(x)| \leq dist(x, \partial D), x \in D\},$$

where  $dist(., \partial D)$  is the distance function to the boundary of  $D$  and  $H_0^1(D)$  is the Hilbert space of all the functions with compact support in  $D$ , such that  $v$  and  $\|\nabla v\|^2$  belong to  $L^2(D)$ .

A finite element approximation to the torsion problem is obtained by triangulating  $D$  and then by replacing the minimization of  $q$  over  $H_0^1(D)$  with the minimization of  $q$  over the set of piecewise linear functions that satisfy the constraints specified by  $K$ . The finite element approximation thus gives rise to a finite-dimensional minimization problem whose variables are the values of the piecewise linear function at the vertices of the triangulation.

In (Averick, Carter, Moré, & Xue, 1992) a finite element approximation to a minimization problem with a quadratic  $q$  of the general form

$$q(v) = \frac{1}{2} \int_D w_q(x) \|\nabla v(x)\|^2 dx - \int_D w_l(x) v(x) dx \quad (1)$$

is described, where  $w_q : D \rightarrow \mathbb{R}$  and  $w_l : D \rightarrow \mathbb{R}$  are functions defined on the rectangle  $D$ . In the torsion problem,  $w_q = 1$  and  $w_l = c$ .

Let  $D = (\xi_{1,l}, \xi_{1,u}) \times (\xi_{2,l}, \xi_{2,u})$  be a rectangle in  $\mathbb{R}^2$ . The vertices  $z_{i,j} \in \mathbb{R}^2$  for a triangulation of  $D$  are obtained by choosing the grid spacing  $h_x$  and  $h_y$  and by defining the grid points

$$z_{i,j} = (\xi_{1,l} + ih_x, \xi_{2,l} + jh_y), \quad 0 \leq i \leq n_x + 1, \quad 0 \leq j \leq n_y + 1,$$

such that  $z_{n_x+1, n_y+1} = (\xi_{1,u}, \xi_{2,u})$ . The triangulation consists of the triangular elements  $T_L$  with the vertices at  $z_{i,j}$ ,  $z_{i+1,j}$ , and  $z_{i,j+1}$  and the triangular elements  $T_U$  with the vertices  $z_{i,j}$ ,  $z_{i-1,j}$ , and  $z_{i,j-1}$ .

A finite element approximation to the torsion problem is obtained by minimizing  $q$  over the space of piecewise linear functions  $v$  with the values  $v_{i,j}$  at  $z_{i,j}$ . The approximation to the integral

$$\int_D w_q(x) \|\nabla v(x)\|^2 dx$$

over the element  $T_L$  is the quadratic  $q_{i,j}^L(v)$ , where

$$q_{i,j}^L(v) = \mu_{i,j} \left\{ \left( \frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left( \frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right\},$$

$$\mu_{i,j} = \frac{h_x h_y}{6} \{w_q(z_{i,j}) + w_q(z_{i+1,j}) + w_q(z_{i,j+1})\}.$$

Similarly, the approximation over the element  $T_U$  is the quadratic  $q_{i,j}^U(v)$ , where

$$q_{i,j}^U(v) = \lambda_{i,j} \left\{ \left( \frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left( \frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\},$$

$$\lambda_{i,j} = \frac{h_x h_y}{6} \{w_q(z_{i,j}) + w_q(z_{i-1,j}) + w_q(z_{i,j-1})\}.$$

Therefore, the finite element approximation to the quadratic (1) leads to a quadratic programming problem of the following form:

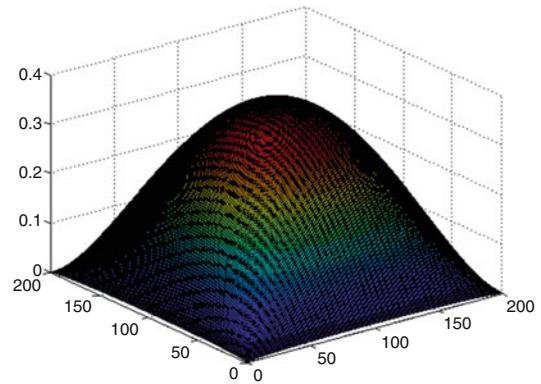
$$\min \{q(v) : v \in \Omega\}, \quad (2)$$

where  $q$  is the quadratic function

$$q(v) = \frac{1}{2} \sum (q_{i,j}^L(v) + q_{i,j}^U(v)) - h_x h_y \sum w_l(z_{i,j}) v_{i,j}. \quad (3)$$

Observe that in this formulation, the quadratic  $q_{i,j}^L$  is defined only when  $0 \leq i \leq n_x$  and  $0 \leq j \leq n_y$ , while  $q_{i,j}^U$  is defined when  $1 \leq i \leq n_x + 1$  and  $1 \leq j \leq n_y + 1$ . Besides, note that for the torsion problem,  $w_q = 1$  and  $w_l = c$ , and the feasible set  $\Omega$ , is  $\Omega = \{v \in R^{n_x n_y} : |v_{i,j}| \leq d_{i,j}\}$ , where  $d_{i,j}$  is the value of  $dist(., \partial D)$  at the vertices  $z_{i,j}$ .

**Fig. A1** Solution of Application A1.  $nx = 200$ ,  $ny = 200$



Considering  $D = (0, 1) \times (0, 1)$ ,  $c = 5$  and  $nx = 200$ ,  $ny = 200$ , then a minimization problem with 40,000 variables is obtained. The solution of this application without simple bounds is illustrated in Fig. A1.

### Application A2. Pressure Distribution in a Journal Bearing

This problem consists in determining the pressure distribution in a thin film of lubricant between two circular cylinders. The infinite-dimensional version of this problem is of the following form:

$$\min \{q(v) : v \in K\}, \quad (4)$$

$$q(v) = \frac{1}{2} \int_D w_q(x) \|\nabla v(x)\|^2 dx - \int_D w_l(x)v(x)dx$$

with

$$w_q(z_1, z_2) = (1 + \varepsilon \cos z_1)^3, \quad w_l(z_1, z_2) = \varepsilon \sin z_1,$$

for some constant  $\varepsilon \in (0, 1)$  and  $D = (0, 2\pi) \times (0, 2b)$ , where  $b > 0$  is again an arbitrary constant. The convex set  $K$  is defined as

$$K = \{v \in H_0^1(D) : v \in D, \quad v \geq 0\}.$$

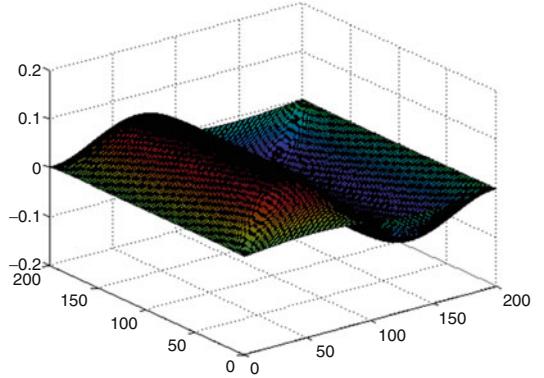
A finite element approximation to the journal bearing problem is obtained as in the torsion problem. Thus, a quadratic programming problem of the form (2) where  $q$  is the quadratic defined by (3) is obtained. In the case of the journal bearing problem,  $w_q(\xi_1, \xi_2) = (1 + \varepsilon \cos \xi_1)^3$  and  $w_l(\xi_1, \xi_2) = \varepsilon \sin \xi_1$ . The feasible set  $\Omega$  is given by  $\Omega = \{v \in \mathbb{R}^{n_x n_y} : v_{i,j} \geq 0\}$ . Considering  $b = 10$  and  $\varepsilon = 0.1$ , as well as a discretization  $n_x \times n_y$  of the domain  $D = (0, 2\pi) \times (0, 2b)$ , where  $nx = 200$  and  $ny = 200$ , then the solution of this application without simple bounds is represented in Fig. A2.

Numerical results for this problem are given, for example, by Lin and Cryer (1985), Cimatti and Menchi (1978), and Moré and Toraldo (1991).

### Application A3. Optimal Design with Composite Materials

This application requires determining the placement of two elastic materials in the cross-section of a rod with maximal torsional rigidity. The formulation of this problem is given in (Averick, Carter, Moré, & Xue, 1992) and follows the presentation from (Goodman, Kohn, & Reyna, 1986).

**Fig. A2** Solution of Application A2.  $nx = 200$ ,  $ny = 200$



Let  $D \subset \mathbb{R}^2$  be a bounded domain and let  $w < |D|$ , where  $|D|$  is the area of  $D$ . The solution of the optimal design problem is a subset  $\Omega$  of  $D$  that solves the problem

$$\min \{F(v, \Omega) : v \in H_0^1(D), |\Omega| = w\}, \quad (5)$$

where

$$F(v, \Omega) = \int_D \left\{ \frac{1}{2} \mu(x) \|\nabla v(x)\|^2 + v(x) \right\} dx,$$

and  $\mu(x) = \mu_1$  for  $x \in \Omega$ , and  $\mu(x) = \mu_2$  for  $x \notin \Omega$ . The reciprocals of the constants  $\mu_1$  and  $\mu_2$  are the shear moduli of the elastic materials in the rod. It is assumed that  $\mu_1 < \mu_2$ .

Goodman, Kohn, and Reyna (1986) formulate the optimal design problem in terms of a family of problems of the form

$$\min \{f_\lambda(v) : v \in H_0^1(D)\},$$

where  $f_\lambda : H_0^1(D) \rightarrow \mathbb{R}$  is the functional

$$f_\lambda(v) = \int_D \{\psi_\lambda(\|\nabla v(x)\|) + v(x)\} dx$$

and  $\psi_\lambda : \mathbb{R} \rightarrow \mathbb{R}$  is a piecewise quadratic function. In this formulation,  $\lambda$  is a Lagrange multiplier associated with the optimal design problem. The piecewise quadratic  $\psi_\lambda : \mathbb{R} \rightarrow \mathbb{R}$  is of the form

$$\psi_\lambda(t) = \begin{cases} \frac{1}{2} \mu_2 t^2, & 0 \leq t \leq t_1, \\ \mu_2 t_1 \left( t - \frac{1}{2} t_1 \right), & t_1 \leq t \leq t_2, \\ \frac{1}{2} \mu_1 (t^2 - t_2^2) + \mu_2 t_1 \left( t_2 - \frac{1}{2} t_1 \right), & t_2 \leq t, \end{cases}$$

with the breakpoints  $t_1$  and  $t_2$  defined by

$$t_1 = \left( 2\lambda \frac{\mu_1}{\mu_2} \right)^{1/2} \text{ and } t_2 = \left( 2\lambda \frac{\mu_2}{\mu_1} \right)^{1/2}.$$

The definition of these breakpoints implies that  $\mu_1 t_2 = \mu_2 t_1$  and thus  $\psi$  is continuously differentiable. The solution of the optimum design problem considered by Averick, Carter, Moré, and Xue (1992) is minimizing  $f_\lambda$  for a fixed value of  $\lambda$ , where  $\mu_1 = 1$  and  $\mu_2 = 2$ , such that  $t_1^2 = \lambda$  and  $t_2^2 = 2\lambda$ .

A finite element approximation to this problem is obtained by minimizing  $f_\lambda$  over the space of the piecewise linear functions  $v$  with the values  $v_{ij}$  at  $z_{ij}$ , where  $z_{ij} \in \mathbb{R}^2$  are the vertices of a triangulation of  $D$  with the grid spacings  $h_x$  and  $h_y$ . The values  $v_{ij}$  are obtained by solving the minimization problem

$$\min \left\{ \sum \left( f_{ij}^L(v) + f_{ij}^U(v) + h_x h_y v_{ij} \right) : v \in \mathbb{R}^n \right\},$$

where the functions  $f_{ij}^L$  and  $f_{ij}^U$  are defined by

$$f_{ij}^L(v) = \frac{h_x h_y}{2} \psi_\lambda \left( d_{ij}^+(v) \right), \quad f_{ij}^U(v) = \frac{h_x h_y}{2} \psi_\lambda \left( d_{ij}^-(v) \right),$$

with

$$d_{ij}^\pm(v) = \left\{ \left( \frac{v_{i\pm 1,j} - v_{ij}}{h_x} \right)^2 + \left( \frac{v_{ij\pm 1} - v_{ij}}{h_y} \right)^2 \right\}^{1/2}.$$

Observe that in this formulation  $f_{ij}^L$  is defined only for  $0 \leq i \leq n_x$  and  $0 \leq j \leq n_y$ , while  $f_{ij}^U$  is defined for  $1 \leq i \leq n_x + 1$  and  $1 \leq j \leq n_y + 1$ .

Figure A3 presents the solution of this application for  $\mu_1 = 1$  and  $\mu_2 = 2$ , with  $n_x = 200$  and  $n_y = 200$ .

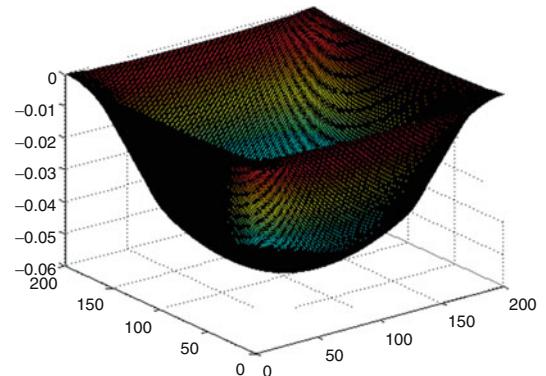
#### Application A4. Steady-State Combustion

This application is taken from (Averick, Carter, Moré, & Xue, 1992). The infinite-dimensional optimization problem is as follows:

$$\min \{f_\lambda(v) : v \in H_0^1(D)\}, \quad (6)$$

where  $f_\lambda : H_0^1(D) \rightarrow \mathbb{R}$  is the functional

**Fig. A3** Solution of Application A3.  $n_x = 200$ ,  $n_y = 200$



$$f_\lambda(v) = \int_D \left\{ \frac{1}{2} \|\nabla v(x)\|^2 - \lambda \exp[v(x)] \right\} dx,$$

and  $\lambda \geq 0$  is a known parameter. This problem is the variational formulation of the boundary value problem

$$\begin{aligned} -\Delta v(x) &= \lambda \exp[v(x)], \quad x \in D, \\ v(x) &= 0, \quad x \in \partial D, \end{aligned}$$

where  $\Delta$  is the Laplacian operator. Aris (1975) and Bebernes and Eberly (1989) discuss this application in the context of combustion problems.

An interesting property of the variational Bratu problem is that  $f_\lambda$  is unbounded below for any  $\lambda > 0$ . This can be seen by observing that if  $v$  is any positive constant function, then  $f_\lambda(\alpha v) \rightarrow -\infty$  as  $\alpha \rightarrow \infty$ . Another interesting property of the variational Bratu problem is that if  $\lambda_{FK} > 0$  is the Frank-Kamenetskii parameter, then  $f_\lambda$  has a unique minimizer for  $\lambda \in [0, \lambda_{FK}]$ , but no minimizers for  $\lambda > \lambda_{FK}$ . If  $D$  is the unit square, then  $\lambda_{FK} \approx 6.81$ , known as the Frank-Kamenetskii parameter.

A finite element approximation to this problem is obtained by minimizing  $f$  over the space of the piecewise linear functions  $v$  with the values  $v_{ij}$  at  $z_{ij}$ , where  $z_{ij} \in \mathbb{R}^2$  are the vertices of a triangulation of  $D$  with the grid spacings  $h_x$  and  $h_y$ . The values of  $v_{ij}$  are computed by solving the following minimization problem:

$$\min \left\{ \sum \left( f_{ij}^L(v) + f_{ij}^U(v) \right) : v \in \mathbb{R}^n \right\},$$

where

$$\begin{aligned} f_{ij}^L(v) &= \frac{h_x h_y}{4} \left\{ \left( \frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left( \frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 - \lambda \mu_{ij}^L \right\}, \\ \mu_{ij}^L &= \frac{2}{3} \{ \exp(v_{i,j}) + \exp(v_{i+1,j}) + \exp(v_{i,j+1}) \}, \end{aligned}$$

and

$$\begin{aligned} f_{ij}^U(v) &= \frac{h_x h_y}{4} \left\{ \left( \frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left( \frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 - \lambda \mu_{ij}^U \right\}, \\ \mu_{ij}^U &= \frac{2}{3} \{ \exp(v_{i,j}) + \exp(v_{i-1,j}) + \exp(v_{i,j-1}) \}. \end{aligned}$$

In this formulation  $f_{ij}^L$  is defined only for  $0 \leq i \leq n_x$  and  $0 \leq j \leq n_y$ , while  $f_{ij}^U$  is defined when  $1 \leq i \leq n_x + 1$  and  $1 \leq j \leq n_y + 1$ .

Considering  $\lambda = 5$ , Fig. A4 shows the solution of this application, where  $n_x = 200$  and  $n_y = 200$ .

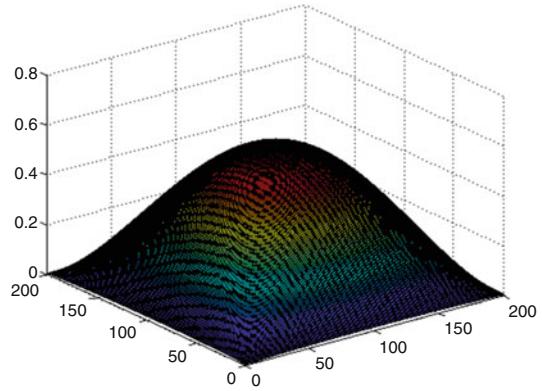
### Application A5. Minimal Surfaces with Enneper Boundary Conditions

Determination of the surface with minimal area and with given boundary values in a convex domain  $D$  is an infinite-dimensional optimization problem of the form

$$\min \{f(v) : v \in K\}, \tag{7}$$

where  $f: K \rightarrow \mathbb{R}$  is the functional

**Fig. A4** Solution of Application A4.  $nx = 200$ ,  $ny = 200$



$$f(v) = \int_D \left(1 + \|\nabla v(x)\|^2\right)^{1/2} dx,$$

and the set  $K$  is defined by

$$K = \{v \in H^1(D) : v(x) = v_D(x) \text{ for } x \in \partial D\}$$

for some boundary data function  $v_D : \partial D \rightarrow \mathbb{R}$ . The boundary function  $v_D$  uniquely defines the solution to the minimal surface problem. An interesting minimal surface, given by Enneper, is obtained by defining  $v_D$  on  $D = (-1/2, 1/2) \times (-1/2, 1/2)$  by

$$v_D(\xi_1, \xi_2) = u^2 - v^2,$$

where  $u$  and  $v$  are the unique solutions to the nonlinear equations

$$\xi_1 = u + uv^2 - \frac{1}{3}u^3, \quad \xi_2 = -v - u^2v + \frac{1}{3}v^3.$$

A finite element approximation to the minimal surface problem is obtained by minimizing  $f$  over the space of the piecewise linear functions  $v$  with the values  $v_{i,j}$  at  $z_{i,j}$ , where  $z_{i,j} \in \mathbb{R}^2$  are the vertices of a triangulation of  $D$  with the grid spacings  $h_x$  and  $h_y$ . The values  $v_{i,j}$  are obtained by solving the minimization problem

$$\min \left\{ \sum \left( f_{i,j}^L(v) + f_{i,j}^U(v) \right) : v \in \mathbb{R}^n \right\},$$

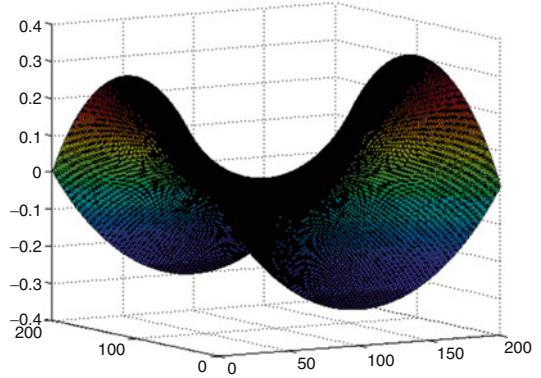
where the functions  $f_{i,j}^L$  and  $f_{i,j}^U$  are defined by

$$f_{i,j}^L(v) = \frac{h_x h_y}{2} \left\{ 1 + \left( \frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left( \frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right\}^{1/2},$$

$$f_{i,j}^U(v) = \frac{h_x h_y}{2} \left\{ 1 + \left( \frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left( \frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\}^{1/2}.$$

Note that in this formulation  $f_{i,j}^L$  is defined only when  $0 \leq i \leq n_x$  and  $0 \leq j \leq n_y$ , while  $f_{i,j}^U$  is defined when  $1 \leq i \leq n_x + 1$  and  $1 \leq j \leq n_y + 1$ . Figure A5 shows the solution of this application, where  $nx = 200$  and  $ny = 200$ .

**Fig. A5** Solution of Application A5.  $nx = 200$ ,  $ny = 200$



Numerical results for this problem are given by Nitsche (1989), Averick, Carter, Moré, and Xue (1992), Andrei (2009e).

#### Application A6. Inhomogeneous Superconductors: 1-D Ginzburg-Landau

This problem arises in the solution of the Ginzburg-Landau equations for inhomogeneous superconductors in the absence of a magnetic field. The formulation of this application is based on the work of Garner and Benedek (1990) and is presented in (Averick, Carter, Moré, & Xue, 1992).

The optimization problem is to minimize the Gibbs free energy as a function of the temperature. The infinite-dimensional version of this problem is of the form

$$\min \{f(v) : v(-d) = v(d), v \in C^1[-d, d]\}, \quad (8)$$

where  $2d$  is the width of the material and  $f$  is the Gibbs free energy function. In this problem,

$$f(v) = \frac{1}{2d} \int_{-d}^d \left\{ \alpha(\xi)|v(\xi)|^2 + \frac{1}{2}\beta(\xi)|v(\xi)|^4 + \frac{\hbar^2}{4m}|v'(\xi)|^2 \right\} d\xi,$$

the functions  $\alpha$  and  $\beta$  are piecewise constants for a fixed value of the temperature,  $\hbar$  is Planck's constant ( $1.05459e-27$  erg-s), and  $m$  is the mass of the electron ( $9.11e-28$  grams).

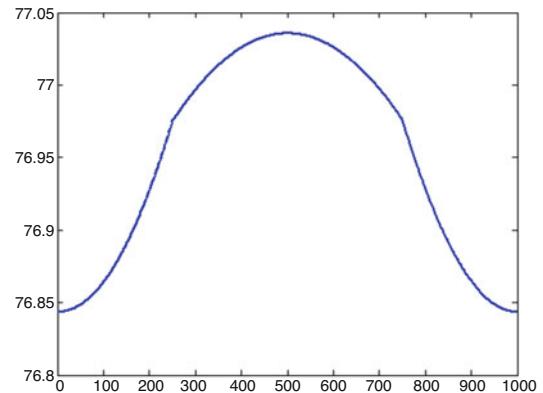
The functions  $\alpha$  and  $\beta$  are constant in the intervals that correspond to lead and tin. Since in this problem the lead in the material corresponds to the interval  $[-ds, ds]$  and the tin in the remaining part of the interval  $[-d, d]$ , the functions  $\alpha$  and  $\beta$  are defined by

$$\alpha(\xi) = \begin{cases} \alpha_N, & -d \leq \xi \leq -ds, \\ \alpha_S, & -ds < \xi \leq ds, \\ \alpha_N, & ds < \xi \leq d, \end{cases} \quad \beta(\xi) = \begin{cases} \beta_N, & -d \leq \xi \leq -ds, \\ \beta_S, & -ds < \xi \leq ds, \\ \beta_N, & ds < \xi \leq d. \end{cases}$$

The constants  $\alpha_S$  and  $\alpha_N$  are negative, but  $\beta_S$  and  $\beta_N$  are positive.

A finite element approximation to the superconductivity problem is obtained by minimizing  $f$  over the space of the piecewise linear functions  $v$  with the values  $v_i$  at  $t_i$ , where  $-d = t_1 < t_2 < \dots < t_n < t_{n+1} = d$ . It is assumed that there are the indices  $n_1$  and  $n_2$  such that  $t_{n_1} = -ds$  and  $t_{n_2} = ds$ , where  $1 < n_1 < n_2 < n$ . This guarantees that the  $t_i$  does not straddle a point of discontinuity of the functions  $\alpha$  and  $\beta$ . The values  $v_i$  are obtained by solving the following minimization problem:

**Fig. A6** Solution of Application A6.  $n = 1000$



$$\min \left\{ \frac{1}{2d} \sum_{i=1}^n f_i(v) : v \in \mathbb{R}^n \right\},$$

where

$$f_i(v) = h_i \left\{ \frac{\alpha_i}{3} \frac{v_{i+1}^3 - v_i^3}{v_{i+1} - v_i} + \frac{\beta_i}{10} \frac{v_{i+1}^5 - v_i^5}{v_{i+1} - v_i} + \frac{\hbar^2}{4m} \left( \frac{v_{i+1} - v_i}{h_i} \right)^2 \right\},$$

with  $h_i = t_{i+1} - t_i$  the length of the  $i$ -th interval and the constants  $\alpha_i$  and  $\beta_i$  as the values of the functions  $\alpha$  and  $\beta$  in the interval  $[t_i, t_{i+1}]$ . The constraint that  $v(-d) = v(d)$  is taken into account by requiring that  $v_{n+1} = v_1$ .

Considering  $d = 3.2\text{\AA}$  and the temperature  $T = 5$ , for  $n = 1000$  the solution of this application is presented in Fig. A6.

### Notes and References

The applications included in this Appendix are taken from the MINPACK-2 collection (Averick, Carter, & Moré, 1991; Averick, Carter, Moré, & Xue, 1992). They are also described in (Andrei, 2017c, Chapter 4).

---

## References

- Abadie, J.: Un nouvel algorithme pour la programmation non linéaire. RAIRO/Rech. Oper. **12**(2), 233–238 (1978)
- Abadie, J.: Une modification de la méthode GRG. RAIRO/Rech. Oper. **13**(3), 323–326 (1979)
- Abadie, J., Carpentier, J.: Generalization of the Wolfe reduced gradient method to the case of nonlinear constraints. In: Fletcher, R. (ed.) Optimization, pp. 37–47. Academic Press, London (1969)
- Abadie, J., Guerrero, G.: Méthode du GRG, Méthode de Newton globale et application à la programmation mathématique. RAIRO/Rech. Oper. **18**(4), 319–351 (1984)
- Abadie, J., Guigou, J.: Numerical experiments with GRG method. In: Abadie, J. (ed.) Integer and Nonlinear Programming, pp. 529–536. North-Holland, Amsterdam (1970)
- Abadie, J., Haggag, A.: Performance du gradient réduit généralisé avec une méthode quasi Newtonienne pour la programmation non linéaire. RAIRO/Rech. Oper. **13**(2), 209–216 (1979)
- Al-Baali, M.: Descent property and global convergence of the Fletcher-Reeves method with inexact linesearch. IMA J. Numer. Anal. **5**, 121–124 (1985)
- Al-Baali, M.: Numerical experience with a class of self-scaling quasi-Newton algorithms. J. Optim. Theory Appl. **96**, 533–553 (1998)
- Al-Baali, M., Grandinetti, L.: On practical modifications of the quasi-Newton BFGS method. AMO-Adv. Model. Optim. **11**(1), 63–76 (2009)
- Al-Baali, M., Narushima, Y., Yabe, H.: A family of three-term conjugate gradient methods with sufficient descent property for unconstrained optimization. Comput. Optim. Appl. **60**, 89–110 (2015)
- Andersen, E.D., Andersen, K.D.: The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In: Frenk, T.T.H., Roos, K., Zhang, S. (eds.) High Performance Optimization, pp. 197–232. Kluwer Academic Publishers, New York (2000)
- An, H.-B., Mo, Z.-Y., Liu, X.-P.: A choice of forcing terms in inexact Newton method. J. Comput. Appl. Math. **200**(1), 47–80 (2007)
- Andreani, R., Birgin, E., Martínez, J., Schuverdt, M.: On augmented Lagrangian methods with general lower-level constraints. SIAM J. Optim. **18**, 1286–1309 (2007)
- Andreani, R., Birgin, E., Martínez, J., Schuverdt, M.: Augmented Lagrangian methods under the constant positive linear dependence constraint qualification. Math. Program. **111**, 5–32 (2008)
- Andrei, N.: A simple algorithm for computing a zero of a nonlinear function of a variable in a given interval [a,b]. (Technical Report, Research Institute for Informatics, Bucharest, Romania, March 28, 1–9) (1975a)
- Andrei, N.: A simple algorithm for computing all zeros of a nonlinear function of a variable in a given interval [a,b]. (Technical Report, Research Institute for Informatics, Bucharest, Romania, April 16, 1–21) (1975b)
- Andrei, N.: RP: a package for efficient calculation of sparse Jacobian matrix for nonlinear systems of equations using finite differences. (Technical Report, Research Institute for Informatics, Bucharest, Romania, April 15, 1–31) (1983)
- Andrei, N.: Application of sparse matrix techniques to the GRG algorithm for large-scale non-linear programming. Rev. Roumaine Sci. Techn. Electrotechn. et Energy. **30**(2), 175–186 (1985)
- Andrei, N.: Application of sparse matrix techniques in GRG algorithm for very large-scale non-linear programming. Rev. Roumaine Sci. Techn. Electrotechn. et Energy. **32**(4), 457–464 (1987)
- Andrei, N.: Computational experience with conjugate gradient algorithms for large-scale unconstrained optimization. (Technical Report, Research Institute for Informatics-ICI, Bucharest, July 21) (1995)
- Andrei, N.: Computational experience with a modified penalty-barrier method for large-scale nonlinear constrained optimization. (Working Paper No. AMOL-96-1, Research Institute for Informatics-ICI, Bucharest, February 6) (1996a)

- Andrei, N.: Computational experience with a modified penalty-barrier method for large-scale nonlinear, equality and inequality constrained optimization. (Technical Paper No. AMOL-96-2, Research Institute for Informatics-ICI, Bucharest, February 12) (1996b)
- Andrei, N.: Computational experience with “SPENBAR” a sparse variant of a modified penalty-barrier method for large-scale nonlinear, equality and inequality, constrained optimization. (Working Paper No. AMOL-96-3, Research Institute for Informatics-ICI, Bucharest, March 10) (1996c)
- Andrei, N.: Penalty-barrier algorithms for nonlinear optimization. Preliminary computational results. *Stud. Inform. Control.* **7**(1), 15–36 (1998a)
- Andrei, N.: Predictor-Corrector interior point methods for linear constrained optimization. *Stud. Inform. Control.* **7**(2), 155–177 (1998b)
- Andrei, N.: An interior point algorithm for nonlinear programming. *Stud. Inform. Control.* **7**(4), 365–395 (1998c)
- Andrei, N.: Programarea matematică avansată. Teorie, Metode computaționale, Aplicații. [Advanced Mathematical Programming. Theory, Computational Methods, Applications]. Editura Tehnică, București (1999a)
- Andrei, N.: Programarea matematică. Metode de punct interior [Mathematical Programming. Interior Point Methods]. Editura Tehnică, București (1999b)
- Andrei, N.: Optimizare fără Restricții – Metode de direcții conjugate [Unconstrained Optimization – Conjugate Direction Methods]. MATRIXROM Publishing House, Bucharest (2000)
- Andrei, N.: Numerical examples solved with SPENBAR – modified penalty barrier method for large-scale nonlinear programming problems. (Technical Report No. 1/2001, Research Institute for Informatics ICI – Bucharest, February) (2001)
- Andrei, N.: Modele, Probleme de Test și Aplicații de Programare Matematică. [Models, Test Problems and Applications for Mathematical Programming]. Editura Tehnică, București (2003)
- Andrei, N.: Convergența algoritmilor de optimizare. [Convergence of the Optimization Algorithms] Editura Tehnică, București (2004a)
- Andrei, N.: Teorie versus empirism în analiza algoritmilor de optimizare. [Theory Versus Empiricism in Analysis of the Optimization Algorithms] Editura Tehnică, București (2004b)
- Andrei, N.: Relaxed gradient descent method with backtracking for unconstrained optimization. În: Works 2005, Manuscript. Biblioteca Academiei Române, pp. 1–10, March 2, 2005a
- Andrei, N.: An acceleration of gradient descent algorithm with backtracking for unconstrained optimization. În: Works 2005, Manuscript. Biblioteca Academiei Române, pp. 1–13, April 20, 2005b
- Andrei, N.: An acceleration of gradient descent algorithm with backtracking for unconstrained optimization. *Numer. Algorithms.* **42**, 63–73 (2006a)
- Andrei, N.: Performance of conjugate gradient algorithms on some MINPACK-2 unconstrained optimization applications. *Stud. Inform. Control.* **15**(2), 145–168 (2006b)
- Andrei, N.: Scaled conjugate gradient algorithms for unconstrained optimization. *Comput. Optim. Appl.* **38**(3), 401–416 (2007a)
- Andrei, N.: A scaled BFGS preconditioned conjugate gradient algorithm for unconstrained optimization. *Appl. Math. Lett.* **20**, 645–650 (2007b)
- Andrei, N.: Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization. *Optim. Methods Softw.* **22**(4), 561–571 (2007c)
- Andrei, N.: Numerical comparison of conjugate gradient algorithms for unconstrained optimization. *Stud. Inform. Control.* **16**(4), 333–352 (2007d)
- Andrei, N.: A scaled nonlinear conjugate gradient algorithm for unconstrained optimization. *Optimization.* **57**(4), 549–570 (2008a)
- Andrei, N.: Another hybrid conjugate gradient algorithm for unconstrained optimization. *Numer. Algorithms.* **47**, 143–156 (2008b)
- Andrei, N.: A Dai-Yuan conjugate gradient algorithm with sufficient descent and conjugacy condition for unconstrained optimization. *Appl. Math. Lett.* **21**(2), 165–171 (2008c)
- Andrei, N.: New hybrid conjugate gradient algorithms for unconstrained optimization. In: Floudas, C.A., Pardalos, P. (eds.) *Encyclopedia of Optimization*, 2nd edn, pp. 2560–2571. Springer Science + Business Media, New York (2008d)
- Andrei, N.: Performance profiles of conjugate gradient algorithms for unconstrained optimization. In: Floudas, C.A., Pardalos, P. (eds.) *Encyclopedia of Optimization*, 2nd edn, pp. 2938–2953. Springer Science + Business Media, New York (2008e)
- Andrei, N.: A hybrid conjugate gradient algorithm for unconstrained optimization as a convex combination of Hestenes-Stiefel and Dai-Yuan. *Stud. Inform. Control.* **17**(1), 55–70 (2008f)
- Andrei, N.: 40 conjugate gradient algorithms for unconstrained optimization – a survey on their definition. (Technical Report, Research Institute for Informatics-ICI, Bucharest, Romania, March 14, 1–13) (2008g)

- Andrei, N.: Noether theorem and fundamentals of mathematical modeling. *Revista Română de Informatică și Automatică*. **18**(4), 11–22 (2008h)
- Andrei, N.: Hybrid conjugate gradient algorithm for unconstrained optimization. *J. Optim. Theory Appl.* **141**(2), 249–264 (2009a)
- Andrei, N.: Another nonlinear conjugate gradient algorithm for unconstrained optimization. *Optim. Methods Softw.* **24**(1), 89–104 (2009b)
- Andrei, N.: Acceleration of conjugate gradient algorithms for unconstrained optimization. *Appl. Math. Comput.* **213**(2), 361–369 (2009c)
- Andrei, N.: Accelerated conjugate gradient algorithm with finite difference Hessian/vector product approximation for unconstrained optimization. *J. Comput. Appl. Math.* **230**, 570–582 (2009d)
- Andrei, N.: Critica Rațiunii Algoritmilor de Optimizare fără Restricții. [Criticism of the Unconstrained Optimization Algorithms Reasoning]. Editura Academiei Române, București (2009e)
- Andrei, N.: Metode Avansate de Gradient Conjugat pentru Optimizare fără Restricții. [Advanced Conjugate Gradient Methods for Unconstrained Optimization]. Editura Academiei Oamenilor de Știință din România, București (2009f)
- Andrei, N.: Accelerated conjugate gradient algorithm with modified secant condition for unconstrained optimization. *Stud. Inform. Control.* **18**(3), 211–232 (2009g)
- Andrei, N.: Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization. *Numer. Algorithms*. **54**, 23–46 (2010a)
- Andrei, N.: Accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization. *Eur. J. Oper. Res.* **204**, 410–420 (2010b)
- Andrei, N.: New accelerated conjugate gradient algorithms as a modification of Dai-Yuan's computational scheme for unconstrained optimization. *J. Comput. Appl. Math.* **234**, 3397–3410 (2010c)
- Andrei, N.: A modified Polak-Ribiere-Polyak conjugate gradient algorithm for unconstrained optimization. *Optimization*. **60**(12), 1457–1471 (2011a)
- Andrei, N.: Open problems in conjugate gradient algorithms for unconstrained optimization. *Bull. Malaysian Math. Sci. Soc.* **34**(2), 319–330 (2011b)
- Andrei, N.: CAON: A collection of nonlinear optimization applications in GAMS language. (Technical Report, No.1/2011, Research Institute for Informatics-ICI, Bucharest, January 31, (105 pages and CD)) (2011c)
- Andrei, N.: Critica Rațiunii Algoritmilor de Programare Liniară. [Criticism of the Linear Programming Algorithms Reasoning]. Editura Academiei Române, București (2011d)
- Andrei, N.: A accelerated conjugate gradient algorithm with guaranteed descent and conjugacy conditions for unconstrained optimization. *Optim. Methods Softw.* **27**(4–5), 583–604 (2012)
- Andrei, N.: A simple three-term conjugate gradient algorithm for unconstrained optimization. *J. Comput. Appl. Math.* **241**, 19–29 (2013a)
- Andrei, N.: On three-term conjugate gradient algorithms for unconstrained optimization. *Appl. Math. Comput.* **219**, 6316–6327 (2013b)
- Andrei, N.: Another conjugate gradient algorithm with guaranteed descent and conjugacy conditions for large-scale unconstrained optimization. *J. Optim. Theory Appl.* **159**, 159–182 (2013c)
- Andrei, N.: A numerical study on efficiency and robustness of some conjugate gradient algorithms for large-scale unconstrained optimization. *Stud. Inform. Control.* **22**(4), 259–284 (2013d)
- Andrei, N.: Nonlinear Optimization Applications using the GAMS Technology. Springer Optimization and its Applications Series, vol. 81. Springer Science + Business Media, New York (2013e)
- Andrei, N.: An accelerated subspace minimization three-term conjugate gradient algorithm for unconstrained optimization. *Numer. Algorithms*. **65**(4), 859–874 (2014)
- Andrei, N.: Critica Rațiunii Algoritmilor de Optimizare cu Restricții. [Criticism of the Constrained Optimization Algorithms Reasoning]. Editura Academiei Române, București (2015a)
- Andrei, N.: A new three-term conjugate gradient algorithm for unconstrained optimization. *Numer. Algorithms*. **68**(2), 305–321 (2015b)
- Andrei, N.: An adaptive conjugate gradient algorithm for large-scale unconstrained optimization. *J. Comput. Appl. Math.* **292**, 83–91 (2016)
- Andrei, N.: Eigenvalues versus singular values study in conjugate gradient algorithms for large-scale unconstrained optimization. *Optim. Methods Softw.* **32**(3), 534–551 (2017a)
- Andrei, N.: Accelerated adaptive Perry conjugate gradient algorithms based on the self-scaling memoryless BFGS update. *J. Comput. Appl. Math.* **325**, 149–164 (2017b)
- Andrei, N.: Continuous Nonlinear Optimization for Engineering Applications in GAMS Technology. Springer Optimization and Its Applications Series, vol. 121. Springer Science + Business Media, New York (2017c)
- Andrei, N.: An adaptive scaled BFGS method for unconstrained optimization. *Numer. Algorithms*. **77**(2), 413–432 (2018a)

- Andrei, N.: A Dai-Liao conjugate gradient algorithm with clustering the eigenvalues. *Numer. Algorithms.* **77**(4), 1273–1282 (2018b)
- Andrei, N.: A double parameter scaled BFGS method for unconstrained optimization. *J. Comput. Appl. Math.* **332**, 26–44 (2018c)
- Andrei, N.: A double parameter scaling Broyden-Fletcher-Goldfarb-Shanno based on minimizing the measure function of Byrd and Nocedal for unconstrained optimization. *J. Optim. Theory Appl.* **178**, 191–218 (2018d)
- Andrei, N.: A diagonal quasi-Newton method based on minimizing the measure function of Byrd and Nocedal for unconstrained optimization. *Optimization.* **67**(9), 1553–1568 (2018e)
- Andrei, N.: A double parameter scaled modified Broyden-Fletcher-Goldfarb-Shanno method for unconstrained optimization. *Stud. Inform. Control.* **27**(2), 135–146 (2018f)
- Andrei, N.: A diagonal quasi-Newton updating method for unconstrained optimization. *Numer. Algorithms.* **81**(2), 575–590 (2019a)
- Andrei, N.: A new diagonal quasi-Newton updating method with scaled forward finite differences directional derivative for unconstrained optimization. *Numer. Funct. Anal. Optim.* **40**(13), 1467–1488 (2019b)
- Andrei, N.: Conjugate gradient algorithms closest to self-scaling memoryless BFGS method based on clustering the eigenvalues of the self-scaling memoryless BFGS iteration matrix or on minimizing the Byrd-Nocedal measure function with different Wolfe line searches for unconstrained optimization. (Technical Report No.2/2019. Academy of Romanian Scientists, Bucharest, Romania, April 18, 1–119) (2019c)
- Andrei, N.: Acceleration by modifying the stepsize versus preconditioning with diagonalized quasi-Newton of self-scaling memoryless BFGS conjugate gradient methods of Hager-Zhang and of Dai-Kou. (Technical Report No.3/2019. Academy of Romanian Scientists, Bucharest, Romania, October 14, 1–20) (2019d)
- Andrei, N.: Nonlinear Conjugate Gradient Methods for Unconstrained Optimization. Springer Optimization and Its Applications Series, vol. 158. Springer Science + Business Media, New York (2020a)
- Andrei, N.: Diagonal approximation of the Hessian by finite difference for unconstrained optimization. *J. Optim. Theory Appl.* **185**, 859–879 (2020b)
- Andrei, N.: A new accelerated diagonal quasi-Newton updating method with scaled forward finite differences directional derivative for unconstrained optimization. *Optimization.* **70**(2), 345–360 (2020c)
- Andrei, N.: New conjugate gradient algorithms based on self-scaling memoryless Broyden-Fletcher-Goldfarb-Shanno method. *Calcolo.* **57**–**17**, 1–17 (2020d)
- Andrei, N.: A Derivative-Free Two Level Random Search Method for Unconstrained Optimization. Springer Briefs in Optimization. Springer, New York (2021a)
- Andrei, N.: More on the efficiency and robustness of conjugate gradient methods subject to procedures for stepsize computation. (Technical Report 4/2021, March 3) (2021b)
- Andrei, N.: Accelerated memory-less SR1 with generalized secant equation method for unconstrained optimization. (Technical Report 8/2021, May 26) (2021c)
- Andrei, N.: A note on memory-less SR1 and memory-less BFGS methods for large-scale unconstrained optimization. *Numer. Algorithms.* **90**(1), 223–240 (2021d)
- Andrei, N.: Accelerated memory-less SR1 method with generalized secant equation for unconstrained optimization. *Calcolo.* **59**–**16**, 1–20 (2022)
- Andrei, N., Bărbulescu, M.: Balance constrained reduction of large-scale linear programming problems. *Ann. Oper. Res.* **43**, 149–170 (1993)
- Antoniou, A., Lu, W.-S.: Practical Optimization. Algorithms and Engineering Applications. Springer Science + Business Media, LLC, New York (2007)
- Aris, R.: The Mathematical Theory of Diffusion and Reaction in Permeable Catalysts. Clarendon Press, Oxford (1975)
- ARKI Consulting and Development A/S: CONOPT version 3 (2004)
- Armijo, L.: Minimization of functions having Lipschitz continuous first partial derivatives. *Pac. J. Math.* **16**(1), 1–3 (1966)
- Arrow, K.J., Sollow, R.M.: Gradient methods for constrained maxima, with weakened assumptions. In: Arrow, K.J., Hurwicz, L., Uzawa, H. (eds.) Studies in Linear and Nonlinear Programming, pp. 166–176. Stanford University Press, Stanford (1958)
- Arzam, M.R., Babaie-Kafaki, S., Ghanbari, R.: An extended Dai-Liao conjugate gradient method with global convergence for nonconvex functions. *Glasnik Matematicki.* **52**(72), 361–375 (2017)
- Averick, B.M., Carter, R.G., Moré, J.J.: The MINPACK-2 test problem collection (Preliminary Version). (Technical Memorandum No.150, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois, Technical Memorandum No. 150, May) (1991)
- Averick, B.M., Carter, R.G., Moré, J.J., Xue, G.L.: The MINPACK-2 test problem collection. (Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois, Preprint MCS-P153-6092, June) (1992)

- Axelsson, O., Lindskog, G.: On the rate of convergence of the preconditioned conjugate gradient method. *Numer. Math.* **48**, 499–523 (1986)
- Babaie-Kafaki, S.: A modified BFGS algorithm based on a hybrid secant equation. *Sci. China Math.* **54**(9), 2019–2036 (2011)
- Babaie-Kafaki, S.: A note on the global convergence theorem of the scaled conjugate gradient algorithms proposed by Andrei. *Comput. Optim. Appl.* **52**(2), 409–414 (2012)
- Babaie-Kafaki, S.: A modified scaled memoryless BFGS preconditioned conjugate gradient method for unconstrained optimization. *4OR* **11**(4), 361–374 (2013)
- Babaie-Kafaki, S.: Two modified scaled nonlinear conjugate gradient methods. *J. Comput. Appl. Math.* **261**(5), 172–182 (2014)
- Babaie-Kafaki, S., Ghanbari, R.: A modified scaled conjugate gradient method with global convergence for nonconvex functions. *Bull. Belg. Math. Soc. Simon Stevin* **21**(3), 465–477 (2014)
- Babaie-Kafaki, S., Ghanbari, R.: A hybridization of the Hestenes–Stiefel and Dai–Yuan conjugate gradient methods based on a least-squares approach. *Optim. Methods Softw.* **30**(4), 673–681 (2015)
- Babaie-Kafaki, S., Rezaee, S.: Two accelerated nonmonotone adaptive trust region line search methods. *Numer. Algorithms* **78**, 911–928 (2018)
- Ballard, D.H., Jelinek, C.A., Schinzinger, R.: An algorithm for the solution of constrained polynomial programming problems. *Comput. J.* **17**, 261–266 (1974)
- Banga, J.R., Alonso, A.A., Singh, R.P.: Stochastic dynamic optimization of batch and semicontinuous bioprocesses. *Biotech. Progress* **13**, 326–335 (1997)
- Bartels, R.H.: A penalty linear programming method using reduced-gradient basis-exchange techniques. *Linear Algebra Appl.* **29**, 17–32 (1980)
- Bartlett, R., Biegler, L.T.: rSQO++: An object oriented framework for successive quadratic programming. In: Biegler, L.T., Ghattas, O., Heinkenschloss, M., van Bloemen Waanders, B. (eds.) *Large-Scale PDE-Constrained Optimization Lecture Notes in Computational Science and Engineering*, vol. 30, pp. 316–330. Springer, New York (2003)
- Bartholomew-Biggs, M.C.: A numerical comparison between two approaches to nonlinear programming problems. (Technical Report, No. 77, Numerical Optimization Centre, Hatfield, England) (1976)
- Bartholomew-Biggs, M.: *Nonlinear Optimization with Engineering Applications*. Springer Science + Business Media, New York (2008)
- Barzilai, J., Borwein, J.M.: Two point step size gradient methods. *IMA J. Numer. Anal.* **8**, 141–148 (1988)
- Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: *Nonlinear Programming Theory and Algorithms*, 2nd edn. Wiley, New York (1993)
- Beale, E.M.L.: On quadratic programming. *Naval Res. Logist. Q.* **6**, 227–244 (1959)
- Beale, E.M.L.: A derivation of conjugate gradients. In: Lotsma, F.A. (ed.) *Numerical Methods for Nonlinear Optimization*, pp. 39–43. Academic Press, New York (1972)
- Bebernes, J., Eberly, D.: *Mathematical Problems from Combustion Theory*. Applied Mathematical Sciences, vol. 83. Springer, New York (1989)
- Ben-Tal, A., El Ghaoui, L., Nemirovski, A.: *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton (2009)
- Ben-Tal, A., Yuzefovich, I., Zibulevsky, M.: Penalty/barrier multiplier methods for minimax and constrained smooth convex problems. (Research Report 9/92, Optimization Laboratory, Faculty of Industrial Engineering and Management, Technion, Haifa, Israel) (1992)
- Ben-Tal, A., Zibulevsky, M.: Penalty-barrier multipier methods for large-scale convex programming problems. (Research Report 6/93, Optimization Laboratory, Faculty of Industrial Engineering and Management, Technion, Haifa, Israel) (1993)
- Benson, H.Y., Shanno, D.F.: Cubic regularization in symmetric rank-1 quasi-Newton methods. *Math. Program. Comput.* **10**, 457–486 (2018)
- Benson, H.Y., Shanno, D.F., Vanderbei, R.J.: Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions. *Comput. Optim. Appl.* **23**, 257–272 (2002a)
- Benson, H.Y., Shanno, D.F., Vanderbei, R.J.: A comparative study of large-scale nonlinear optimization algorithms. (Technical Report ORFE-01-04, Operations Research and Financial Engineering, Princeton University, July 17) (2002b)
- Bergman, L.: Energy policy modeling: a survey of general equilibrium approaches. *J. Policy Model.* **10**(3), 377–399 (1988)
- Bertsekas, D.P.: On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Trans. Automat. Control* **21**, 174–184 (1976)
- Bertsekas, D.P.: Projected Newton mothods for optimization problems with simple constraints. *SIAM J. Control Optim.* **20**, 221–246 (1982a)
- Bertsekas, D.P.: *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, New York (1982b)

- Bertsekas, D.P.: Nonlinear Programming, 2nd edn. Athena Scientific, Belmont (1999)
- Betts, J., Eldersveld, S.K., Frank, P.D., Lewis, J.G.: An interior-point nonlinear programming algorithm for large scale optimization. (Technical Report MCT TECH-003, Mathematics and Computing Technology, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207, USA) (2000)
- Bicanic, N., Johnson, K.: Who was "Raphson"? *Int. J. Numer. Methods Eng.* **14**, 148–152 (1979)
- Biegler, L.T., Nocedal, J., Schmid, C.: A reduced Hessian method for large-scale constrained optimization. *SIAM J. Optim.* **5**, 314–347 (1995)
- Biggs, M.C.: Minimization algorithms making use of non-quadratic properties of the objective function. *J. Inst. Math. Appl.* **8**, 315–327 (1971)
- Biggs, M.C.: Constrained minimization using recursive equality quadratic programming. In: Lootsma, F.A. (ed.) *Numerical Methods for Nonlinear Optimization*, pp. 411–428. Academic Press, London (1972)
- Biggs, M.C.: A note on minimization algorithms making use of non-quadratic properties of the objective function. *J. Inst. Math. Appl.* **12**, 337–338 (1973)
- Birge, J.R., Louveaux, F.: *Introduction to Stochastic Programming*. Springer, New York (1997)
- Birgin, E.G., Martínez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. (Technical Paper, June 7) (1999)
- Birgin, E.G., Martínez, J.M.: A spectral conjugate gradient method for unconstrained optimization. *Appl. Math. Optim.* **43**, 117–128 (2001)
- Birgin, E.G., Martínez, J.M.: Practical Augmented Lagrangian Methods for Constrained Optimization. SIAM, Philadelphia (2014)
- Birgin, E.G., Martínez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM J. Optim. **10**, 1196–1211 (2000)
- Birgin, E.G., Martínez, J.M., Raydan, M.: Algorithm 813: SPG – software for convex-constrained optimization. ACM Trans. Math. Softw. **27**, 340–349 (2001)
- Bisschop, J., Meeraus, A.: On the development of the general algebraic modeling systems in a strategic planning environment. *Math. Program. Study.* **20**, 1–29 (1982)
- Boggs, P.T., Tolle, J.W.: A strategy for global convergence in a sequential quadratic programming algorithm. *SIAM J. Numer. Anal.* **21**, 600–623 (1989)
- Boggs, P.T., Tolle, J.W.: Convergence properties of a class of rank-two updates. *SIAM J. Optim.* **4**, 262–287 (1994)
- Boggs, P.T., Tolle, J.W.: Sequential quadratic programming. *Acta Numer.* **4**, 1–51 (1995)
- Bondarenko, A.S., Bortz, D.M., Moré, J.J.: COPS: Large-scale nonlinearly constrained optimization problems. (Technical Report ANL/MCS-TM-237) (1999)
- Bongartz, I., Conn, A.R., Gould, N.I.M., Toint, P.L.: CUTE: Constrained and unconstrained testing environment. ACM Trans. Math. Softw. **21**(1), 123–160 (1995)
- Bonnans, J., Gilbert, J., Lemaréchal, C., Sagastizábal, C.: Numerical Optimization: Theory and Practical Aspects, 2nd edn. Springer, Berlin (2006)
- Bonnans, J.F., Panier, E.R., Tits, A.L., Zhou, J.L.: Avoiding the Maratos effect by means of a nonmonotone line search. II. Inequality constrained problems – feasible iterates. *SIAM J. Numer. Anal.* **29**, 1187–1202 (1992)
- Boyd, S., El Ghaoui, L., Feron, E., Balakrishnan, V.: *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics, Philadelphia (1994)
- Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
- Breitfeld, M.G., Shanno, D.F.: Preliminary computational experience with modified log-barrier functions for large-scale nonlinear programming. In: Hager, W.W., Hearn, D.W., Pardalos, P.M. (eds.) *Large Scale Optimization, State of the Art*, vol. 1994, pp. 45–67. Kluwer Academic Publishers, Dordrecht-Boston-London (1994a)
- Breitfeld, M.G., Shanno, D.F.: Computational experience with penalty-barrier methods for nonlinear programming. (RUTCOR Research Report, RRR 17-93, August 1993, Revised March 1994. Rutgers Center for Operations Research, Rutgers University, New Brunswick, New Jersey 08903, March) (1994b)
- Breitfeld, M.G., Shanno, D.F.: A globally convergent penalty-barrier algorithm for nonlinear programming and its computational performance. (RUTCOR Research Report, RRR 12-94, April 1994, Rutgers Center for Operations Research, Rutgers University, New Brunswick, New Jersey 08903, March) (1994c)
- Brent, R.P.: Algorithms for Minimization Without Derivatives. Prentice Hall, Englewood Cliffs (1973)
- Brooke, A., Kendrick, D., Meeraus, A., Raman, R., Rosenthal, R.E.: GAMS: A User's Guide. GAMS Development Corporation (1998)
- Brooke, A., Kendrick, D., Meeraus, A., Raman, R.: GAMS: A User Guide. GAMS Development Corporation (2005)
- Brown, A.A., Bartholomew-Biggs, M.C.: ODE vs SQP methods for constrained optimisation. (Technical Report No. 179, Numerical Optimisation Centre, The Hatfield Polytechnic, Hatfield, June) (1987)
- Brown, A.A., Bartholomew-Biggs, M.C.: Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations. *J. Optim. Theory Appl.* **62**, 211–224 (1989)

- Broyden, C.G.: The convergence of a class of double-rank minimization algorithms. I. General considerations. *J. Inst. Math. Appl.* **6**, 76–90 (1970)
- Broyden, C.G., Dennis, J.E., Moré, J.J.: On the local and superlinear convergence of quasi-Newton methods. *J. Inst. Math. Appl.* **12**, 223–246 (1973)
- Buckley, A.G., Lenir, A.: Algorithm 630-BBVSCG: A variable storage algorithm for function minimization. *ACM Trans. Math. Softw.* **11**, 103–119 (1985)
- Bulirsch, R., Stoer, J.: *Introduction to Numerical Analysis*. Springer, New York (1980)
- Bunch, J.R., Kaufman, L.: Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comput.* **31**, 163–179 (1977)
- Bunch, J.R., Parlett, B.N.: Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.* **8**, 639–655 (1971)
- Byrd, R.H., Curtis, F.E., Nocedal, J.: An inexact Newton method for nonconvex equality constrained optimization. *Math. Program. Serie A.* **122**, 273–299 (2010)
- Byrd, R.H., Gilbert, J.-C., Nocedal, J.: A trust region method based on interior point techniques for nonlinear programming. *Math. Program.* **89**, 149–185 (2000)
- Byrd, R.H., Gould, N.I.M., Nocedal, J., Waltz, R.A.: On the convergence of successive linear-quadratic programming algorithms. (Technical Report OTC 2002/5, Optimization Technology Center, Northwestern University, Evanston, IL) (2002)
- Byrd, R.H., Gould, N.I.M., Nocedal, J., Waltz, R.A.: An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Math. Program. Ser. B.* **100**, 27–48 (2004a)
- Byrd, R.H., Hribar, M.E., Nocedal, J.: An interior point method for large scale nonlinear programming. *SIAM J. Optim.* **9**, 877–900 (1999)
- Byrd, R.H., Liu, G., Nocedal, J.: On the local behavior of an interior point method for nonlinear programming. In: Griffiths, D.F., Higham, D.J. (eds.) *Numerical Analysis*, pp. 37–56. Addison-Wesley Longman, Reading (1997)
- Byrd, R.H., Liu, D.C., Nocedal, J.: On the behavior of Broyden's class of quasi-Newton methods. *SIAM J. Optim.* **2**, 533–557 (1992)
- Byrd, R.H., Lu, P., Nocedal, J.: A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Stat. Comput.* **16**(5), 1190–1208 (1995)
- Byrd, R.H., Marazzi, M., Nocedal, J.: On the convergence of Newton iterations to non-stationary points. *Math. Program. Ser. A.* **99**, 127–148 (2004b)
- Byrd, R.H., Nocedal, J.: A tool for the analysis of quasi-Newton methods with application to unconstrained minimization. *SIAM J. Numer. Anal.* **26**, 727–739 (1989)
- Byrd, R.H., Nocedal, J., Waltz, R.A.: Feasible interior methods using slacks for nonlinear optimization. *Comput. Optim. Appl.* **26**(1), 35–61 (2003)
- Byrd, R.H., Nocedal, J., Waltz, R.A.: KNITRO: An integrated package for nonlinear optimization. In: Di Pillo, G., Roma, M. (eds.) *Large-Scale Nonlinear Optimization*, pp. 35–59 (2006)
- Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. (Technical Report NAM-08, [Revised May 1994], Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208) (1994)
- Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.* **16**, 1190–1208 (1995)
- Byrd, R.H., Nocedal, J., Schnabel, R.B.: Representation of quasi-Newton matrices and their use in limited memory methods. *Math. Program.* **63**(2), 129–156 (1994)
- Byrd, R.H., Nocedal, J., Yuan, Y.: Global convergence of a class of quasi-Newton methods on convex problems. *SIAM J. Numer. Anal.* **24**, 1171–1190 (1987a)
- Byrd, R.H., Schnabel, R.B., Shultz, G.A.: A trust region algorithm for nonlinearly constrained optimization. *SIAM J. Numer. Anal.* **24**, 1152–1170 (1987b)
- Byrd, R.H., Schnabel, R.B., Shultz, G.A.: Approximate solution of the trust-region problem by minimization over two-dimensional subspaces. *Math. Program.* **40**, 247–263 (1988)
- Byrd, R.H., Tapia, R.A., Zhang, Y.: An SQP augmented Lagrangean BFGS algorithm for constrained optimization. (Technical Report, University of Colorado at Boulder) (1990)
- Carolan, W.J., Hill, J.E., Kennington, J.L., Niemi, S., Wochmann, S.J.: An empirical evaluation of the KORBX algorithms for military airlift applications. *Oper. Res.* **38**, 240–248 (1990)
- Cauchy, A.: Méthodes générales pour la résolution des systèmes d'équations simultanées. *Comptes Rendus de l'Académie des Sciences Paris.* **25**(1), 536–538 (1847)
- Castillo, E., Conejo, A.J., Pedregal, P., García, R., Alguacil, N.: *Building and Solving Mathematical Programming Models in Engineering and Science*. Wiley, New York (2001)
- Cesari, L.: *Optimization – Theory and Applications*. Springer, Bonn (1983)

- Chachuat, B.C.: Nonlinear and Dynamic Optimization – from Theory to Practice, IC-31: Winter Semester 2006/2007. École Polytechnique Fédérale de Lausanne (2007)
- Chen, H., Lam, W.H., Chan, S.C.: On the convergence analysis of cubic regularized symmetric rank-1 quasi-Newton method and the incremental version in the application of large-scale problems. *IEEE Access.* **7**, 114042–114059 (2019)
- Cheng, S.H.: Symmetric indefinite matrices: Linear system solvers and modified inertia problems. Ph.D. Thesis, University of Manchester, Faculty of Science and Engineering, January 1998
- Cheng, S.H., Higham, N.J.: A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM J. Matrix Anal. Appl.* **19**, 1097–1110 (1998)
- Cheng, W.Y., Li, D.H.: Spectral scaling BFGS method. *J. Optim. Theory Appl.* **146**, 305–319 (2010)
- Chin, C.M.: A global convergence theory of a filter line search method for nonlinear programming. (Technical Report, Department of Electrical Engineering, University of Malaya, Kuala Lumpur, Malaysia, August) (2002)
- Chin, C.M., Fletcher, R.: On the local convergence of an SLP-filter algorithm that takes EQP steps. *Math. Program. Ser. A.* **96**, 161–177 (2003)
- Cimatti, G., Menchi, O.: On the numerical solution of a variational inequality connected with the hydrodynamic lubrication of a complete journal bearing. *Calcolo.* **15**, 249–258 (1978)
- Choi, T.D., Kelley, C.T.: Superlinear convergence and implicit filtering. *SIAM J. Optim.* **10**, 1149–1162 (2000)
- Cohen, A.: Rate of convergence of several conjugate gradient algorithms. *SIAM J. Numer. Anal.* **9**, 248–259 (1972)
- Coleman, T.F., Conn, A.R.: Nonlinear programming via an exact penalty function: Asymptotic analysis. *Math. Program.* **24**, 123–136 (1982a)
- Coleman, T.F., Conn, A.R.: Nonlinear programming via an exact penalty function: Global analysis. *Math. Program.* **24**, 137–161 (1982b)
- Coleman, T.F., Hulbert, L.A.: A direct active set algorithm for large sparse quadratic programs with simple bounds. *Math. Program. Ser. A.* **45**, 373–406 (1989)
- Coleman, T.F., Li, Y.: On the convergence of interior-reflective Newton methods for nonlinear minimization subject to bounds. *Math. Program.* **67**, 189–224 (1994)
- Coleman, T.F., Li, Y.: An interior trust region approach for nonlinear minimization subject to bounds. *SIAM J. Optim.* **6**, 418–445 (1996)
- Coleman T.F., Li, Y.: A trust region and affine scaling interior point method for nonconvex minimization with linear inequality constraints. (Technical Report, Cornell University, Ithaca, NY) (1997)
- Concus, P., Golub, G., O’Leary, D.P.: A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In: Bunch, J., Rose, D. (eds.) *Sparse Matrix Computation*, pp. 309–332. Academic Press, New York (1976)
- Conejo, A.J., Castillo, E., Minguez, R., Garcia-Bertrand, R.: *Decomposition Techniques in Mathematical Programming: Engineering and Science Applications*. Springer, Heidelberg (2006)
- Conn, A.R.: Linear programming via a nondifferentiable penalty function. *SIAM J. Numer. Anal.* **13**, 145–154 (1976)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM J. Numer. Anal.* **25**(2), 433–460 (1988a)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: Testing a class of methods for solving minimization problems with simple bounds on the variables. *Math. Comput.* **50**, 399–430 (1988b)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: A globally convergent augmented Lagrangean algorithm for optimization with general constraints and simple bounds. *SIAM J. Numer. Anal.* **28**, 545–572 (1991a)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: Convergence of quasi-newton matrices generated by the symmetric rank one update. *Math. Program.* **50**(1–3), 177–195 (1991b)
- Conn, A.R., Gould, N.I.M., Toint, Ph.L.: A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. (Technical Report 92/07, Department of Mathematics, Faculté Universitaires de Namur, Namur, Belgium) (1992a)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: *LANCELOT – A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Series in Computational Mathematics, vol. 17. Springer, Berlin (1992b)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Math. Comput.* **66**, 261–288 (1997a)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Math. Program. Ser. A.* **73**(1), 73–110 (1996a)
- Conn, A.R., Gould, N.I.M., Toint, P.L.: Trust-Region Methods. MPS-SIAM Series on Optimization. SIAM, Philadelphia (2000)
- Conn, A.R., Gould, N.I.M., Sartenaer, A., Toint, P.L.: Convergence properties of an augmented Lagrangean algorithm for optimization with a combination of general equality and linear constraints. *SIAM J. Optim.* **6**(3), 674–703 (1996b)

- Conn, A.R., Scheinberg, K., Toint, P.L.: Recent progress in unconstrained nonlinear optimization without derivatives. *Math. Program. Ser. B.* **79**, 397–414 (1997b)
- Conterras, M., Tapia, R.A.: Sizing the BFGS and DFP updates: A numerical study. *J. Optim. Theory Appl.* **78**, 93–108 (1993)
- Cottle, R.W.: William Karush and the KKT theorem. In: Grötschel, M. (ed.) *Optimization stories*. Documenta Mathematica, Journal der Deutschen Mathematiker-Vereinigung, Extra volume, 21st International Symposium on Mathematical Programming, pp. 255–269, Berlin, August 19–24, 2012
- Crowder, H.P., Wolfe, P.: Linear convergence of the conjugate gradient method. *IBM J. Res. Dev.*, 431–433 (1969)
- Courant, R.: Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Am. Math. Soc.* **49**, 1–23 (1943)
- Curtis, A., Powell, M.J.D., Reid, J.K.: On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.* **13**, 117–119 (1974)
- Cuthrell, J.E., Biegler, L.T.: Simultaneous optimization and solution methods for batch reactor control profiles. *Comput. Chem. Eng.* **13**, 49–62 (1989)
- Dai, Y.H.: Analyses of conjugate gradient methods. Ph.D. Thesis, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences (1997)
- Dai, Y.H.: A nonmonotone conjugate gradient algorithm for unconstrained optimization. *J. Syst. Sci. Complex.* **15**(2), 139–145 (2002a)
- Dai, Y.H.: On the nonmonotone line search. *J. Optim. Theory Appl.* **112**, 315–330 (2002b)
- Dai, Y.H.: Convergence properties of the BFGS algorithm. *SIAM J. Optim.* **13**, 693–701 (2003)
- Dai, Y.H.: Chapter 8: Convergence analysis of nonlinear conjugate gradient methods. In: Wang, Y., Yagola, A.G., Yang, C. (eds.) *Optimization and Regularization for Computational Inverse Problems and Applications*, pp. 157–181. Higher Education Press/Springer, Beijing/Berlin, Heidelberg (2010)
- Dai, Y.H.: Nonlinear conjugate gradient methods. Wiley Encyclopedia of Operations Research and Management Science. <https://doi.org/10.1002/9780470400531.eorms0183>. Published on line, February 15, 2011
- Dai, Y.H., Hager, W.W., Schittkowski, K., Zhang, H.: The cyclic Barzilai-Borwein method for unconstrained optimization. *IMA J. Numer. Anal.* **26**, 604–627 (2006)
- Dai, Y.H., Han, J.Y., Liu, G.H., Sun, D.F., Yin, H.X., Yuan, Y.X.: Convergence properties of nonlinear conjugate gradient methods. *SIAM J. Optim.* **10**(2), 345–358 (1999)
- Dai, Y.H., Kou, C.X.: A nonlinear conjugate gradient algorithm with an optimal property and an improved Wolfe line search. *SIAM J. Optim.* **23**(1), 296–320 (2013)
- Dai, Y.H., Liao, L.Z.: New conjugate conditions and related nonlinear conjugate gradient methods. *Appl. Math. Optim.* **43**, 87–101 (2001)
- Dai, Y.H., Liao, L.Z.: R-linear convergence of the Barzilai and Borwein gradient method. *IMA J. Numer. Anal.* **22**(1), 1–10 (2002)
- Dai, Y.H., Liao, L.Z., Li, D.: On restart procedures for the conjugate gradient method. *Numer. Algorithms* **35**, 249–260 (2004)
- Dai, Y.H., Ni, Q.: Testing different conjugate gradient methods for large-scale unconstrained optimization. *J. Comput. Math.* **22**(3), 311–320 (2003)
- Dai, Z., Wen, F.: Another improved Wei-Yao-Liu nonlinear conjugate gradient method with sufficient descent property. *Appl. Math. Comput.* **218**, 7421–7430 (2012)
- Dai, Y.H., Yuan, Y.X.: Convergence properties of the conjugate descent method. *Adv. Math. (China)* **26**, 552–562 (1996)
- Dai, Y.H., Yuan, Y.: A nonlinear conjugate gradient method with strong global convergence property. *SIAM J. Optim.* **10**, 177–182 (1999)
- Dai, Y.H., Yuan, Y.: An efficient hybrid conjugate gradient method for unconstrained optimization. *Ann. Oper. Res.* **103**, 33–47 (2001)
- Dai, Y.H., Yuan, Y.: A class of globally convergent conjugate gradient methods. *Sci. China Math. Ser. A* **46**(2), 251–261 (2003)
- Daniel, J.W.: The conjugate gradient method for linear and nonlinear operator equations. *SIAM J. Numer. Anal.* **4**, 10–26 (1967)
- Daniel, J.W., Gragg, W.B., Kaufman, L., Stewart, G.W.: Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorizations. *Math. Comput.* **30**, 772–795 (1976)
- Dantzig, G.B.: *Linear Programming and Extensions*. Princeton University Press, Princeton (1963)
- Davidon, W.C.: Variable metric method for minimization. (Research and Development Report ANL-5990. Argonne National Laboratories) (1959)
- Davidon, W.C.: Conic approximation and collinear scalings for optimizers. *SIAM J. Numer. Anal.* **17**(2), 268–281 (1980)
- Debreu, G.: Definite and semidefinite quadratic forms. *Econometrica* **20**, 295–300 (1952)

- Dehmiry, A.H.: The global convergence of the BFGS method under a modified Yuan-Wei-Lu line search technique. *Numer. Algorithms.* (2019). <https://doi.org/10.1007/s11075-019-00779-7>
- Dembo, R.S.: A set of geometric programming test problems and their solutions. *Math. Program.* **10**, 192–213 (1976)
- Dembo, R.S., Eisenstat, S.C., Steihaug, T.: Inexact Newton methods. *SIAM J. Numer. Anal.* **19**, 400–408 (1982)
- Dembo, R.S., Steihaug, T.: Truncated Newton algorithms for large-scale unconstrained optimization. *Math. Program.* **26**, 190–212 (1983)
- Dembo, R.S., Tulowitzki, U.: On the minimization of quadratic functions subject to box constraints. (Technical Report, School of Organization and Management, Yale University, New Haven, CT) (1983)
- Demmel, J.W.: *Applied Numerical Linear Algebra*. SIAM, Philadelphia (1997)
- Dener, A., Denchfield, A., Munson, T.: Preconditioning nonlinear conjugate gradient with diagonalized quasi-Newton. (Mathematics and Computer Science Division, Preprint ANL/MCS-P9152-0119, January 2019, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60430) (2019)
- Deng, N.Y., Xiao, Y., Zhou, F.J.: Nonmonotonic trust-region algorithm. *J. Optim. Theory Appl.* **26**, 259–285 (1993)
- Dennis, J.E., Heinkenschlos, M., Vicente, L.N.: Trust-region interior-point algorithms for a class of nonlinear programming problems. *SIAM J. Control. Optim.* **36**, 1750–1794 (1998)
- Dennis, J.E., Moré, J.J.: Quasi-Newton methods, motivation and theory. *SIAM Rev.* **19**(1), 46–89 (1977)
- Dennis, J.E., Schnabel, R.B.: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Reprinted as Classics in Applied Mathematics, vol. 16. Prentice-Hall/SIAM, Englewood Cliffs/Philadelphia (1983)
- Dennis, J.E., Schnabel, R.B.: A view of unconstrained optimization. In: *Optimization*, Vol. 1 of Handbooks in Operations Research and Management, pp. 1–72, Elsevier Science Publishers, Amsterdam (1989)
- Dennis, J.E., Wolkowicz, H.: Sizing and least-change secant methods. *SIAM J. Numer. Anal.* **30**(5), 1291–1314 (1993)
- Deuflhard, P.: Global inexact Newton methods for very large scale nonlinear problems. In: Proceedings of the Cooper Mountain Conference on Iterative Methods. Cooper Mountain, Colorado, April 1–5 (1990)
- Dikin, I.I.: Iterative solution of problems of linear and quadratic programming. *Soviet Math. Doklady.* **8**, 674–675 (1967)
- Dikin, I.I.: On the convergence of an iterative process. *Upravlyayemye Sistemi.* **12**, 54–60 (1974)
- Dixon, L.C.W., Price, R.C.: Numerical experience with truncated Newton method for unconstrained optimization. *J. Optim. Theory Appl.* **56**, 245–255 (1988)
- Dolan, E.D., Moré, J.J.: Benchmarks optimization software with performance profiles. *Math. Program.* **91**, 201–213 (2002)
- Dolan, E.D., Moré, J.J., Munson, T.S.: Benchmarking Optimization Software with COPS 3.0. Preprint ANL/MCS-TM-273, Argonne National Laboratory, Argonne (2004)
- Drud, A.: Application of sparse matrix techniques in large scale nonlinear programming. In: Prekopa, A. (ed.) *Survey of Mathematical Programming*, vol. 1, part 3, Nonlinear Programming. (Proceedings of the 9th International Mathematical Programming Symposium, Budapest, August 23–27, 1976, pp. 429–445) (1976)
- Drud, A.: CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. (Report No. DRD59, Development Research Department, Economics and Research Staff, The World Bank, 1818 H Street, Washington, D.C. 20433, August) (1983)
- Drud, S.A.: CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. Technical Note No.21, Development Research Center, The World Bank, 1818 H Street, Washington, D.C. 20433, March (1984)
- Drud, S.A.: CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Math. Program.* **31**, 153–191 (1985)
- Drud, S.A.: CONOPT – A large-scale GRG code. *ORSA J. Comput.* **6**, 207–216 (1994)
- Drud, S.A.: CONOPT – A system for large-scale nonlinear optimization. (Tutorial for CONOPT subroutine library, 16p. ARKI Consulting and Development A/S, Bagsvaerd, Denmark) (1995)
- Drud, S.A.: CONOPT: A system for large-scale nonlinear optimization. (Reference Manual for CONOPT subroutine library, 69p. ARKI Consulting and Development A/S, Bagsvaerd, Denmark) (1996)
- Drud, S.A.: CONOPT. In: *GAMS the Solver Manuals*, pp. 39–82. GAMS Development Corporation, Washington, DC (2005)
- Drud, S.A.: CONOPT – A system for large scale nonlinear optimization. Tutorial for CONOPT dynamic link library. Version 3.15 (Fortran 90 examples). (ARKI Consulting and Development A/S, Bagsvaerd, Denmark, August) (2011)
- Duran, M., Grossmann, I.E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program.* **36**, 307–339 (1986)
- Ecker, J.G., Kupferschmid, M.: An ellipsoid algorithm for nonlinear programming. *Math. Program.* **27**, 83–106 (1983)
- Ecker, J.G., Kupferschmid, M.: A computational comparison of the ellipsoid algorithm with several nonlinear programming algorithms. *SIAM J. Control. Optim.* **23**(5), 657–674 (1985)
- Eisenstat, S.C., Walker, H.F.: Choosing the forcing terms in an inexact Newton method. *SIAM J. Sci. Comput.* **17**(1), 16–32 (1996)

- El-Bakry, A.S., Tapia, R.A., Tsuchiya, T., Zhang, Y.: On the formulation and theory of the Newton interior-point method for nonlinear programming. *J. Optim. Theory Appl.* **89**(3), 507–541 (1996)
- Eldersveld, S.K.: Large-scale sequential quadratic programming algorithms. Ph.D. Thesis, Department of Operations Research, Stanford University, Stanford, CA (1991)
- Facchinei, F., Júdice, J., Soares, J.: An active set Newton's algorithm for large-scale nonlinear programs with box constraints. *SIAM J. Optim.* **8**, 158–186 (1998)
- Facchinei, F., Lucidi, S.: A class of penalty functions for optimization problems with bounds constraints. *Optimization*. **26**, 239–259 (1992a)
- Facchinei, F., Lucidi, S.: A class of methods for optimization problems with simple bounds. Part 2: Algorithms and numerical results. (Technical Report R.336, IASI-CNR, Roma, Italy) (1992b)
- Facchinei, F., Lucidi, S.: Convergence to second order stationary points in inequality constrained optimization. *Math. Oper. Res.* **23**, 746–766 (1998)
- Facchinei, F., Lucidi, S., Palagi, L.: A truncated Newton algorithm for large-scale box constrained optimization. *SIAM J. Optim.* **4**, 1100–1125 (2002)
- Fiacco, A.V., McCormick, G.P.: The sequential unconstrained minimization technique for nonlinear programming. A primal-dual method. *Manag. Sci.* **10**, 360–366 (1964)
- Fiacco, A.V., McCormick, G.P.: Extensions of SUMT for nonlinear programming: equality constraints and extrapolation. *Manag. Sci.* **12**, 816–828 (1966)
- Fiacco, A.V., McCormick, G.P.: Nonlinear Programming: Sequential Unconstrained Minimization Technique. Wiley, New York (1968). [Republished in 1990 by SIAM, Philadelphia]
- Finkel, D.E.: DIRECT Optimization Algorithm User Guide. Center for Research in Scientific Computation, North Carolina State University, Raleigh (2003)
- Fletcher, R.: A new approach to variable metric algorithms. *Comput. J.* **13**, 317–322 (1970)
- Fletcher, R.: A general quadratic programming algorithm. *J. Inst. Math. Appl.*, 76–91 (1971)
- Fletcher, R.: Second order corrections for nondifferentiable optimization. In: Griffith, D. (ed.) *Numerical Analysis*, pp. 85–114. Springer (1982). Proceedings Dundee 1981
- Fletcher, R.: Practical Methods of Optimization, 2nd edn. Wiley, New York (1987)
- Fletcher, R.: A new variational result for quasi-Newton formulae. *SIAM J. Optim.* **1**, 18–21 (1991)
- Fletcher, R.: An optimal positive definite update for sparse Hessian matrices. *SIAM J. Optim.* **5**, 192–218 (1995)
- Fletcher, R.: Stable reduced Hessian updates for indefinite quadratic programming. *Math. Program.* **87**, 251–264 (2000)
- Fletcher, R.: A package of subroutines for NLP and LCP. Open Source Initiative OSI – Eclipse Public License 1.0 (ELP-1.0), Release 1.0, 2011 (2011)
- Fletcher, R., Sainz de la Maza, E.: Nonlinear programming and nonsmooth optimization by successive linear programming. *Math. Program.* **43**, 235–256 (1989)
- Fletcher, R., Freeman, T.L.: A modified Newton method for minimization. *J. Optim. Theory Appl.* **23**, 357–372 (1977)
- Fletcher, R., Gould, N.I.M., Leyffer, S., Toint, P.T., Wächter, A.: Global convergence of a trust-region SQP filter algorithm for general nonlinear programming. *SIAM J. Optim.* **13**, 635–659 (2002a)
- Fletcher, R., Grothey, A., Leyffer, S.: Computing sparse Hessian and Jacobian approximations with optimal hereditary properties. (Technical Report, Department of Mathematics, University of Dundee) (1996)
- Fletcher, R., Leyffer, S.: User Manual for FilterSQP. (Technical Report NA/181, Department of Mathematics, University of Dundee, Scotland, April. [Updated, March 1999]) (1998)
- Fletcher, R., Leyffer, S.: A bundle filter method for nonsmooth nonlinear optimization. (Numerical Analysis Report NA/195, Dundee University, April) (1999)
- Fletcher, R., Leyffer, S.: Nonlinear programming without a penalty function. *Math. Program. Ser. A* **91**, 239–269 (2002)
- Fletcher, R., Leyffer, S.: Filter-type algorithms for solving systems of algebraic equations and inequalities. In: di Pillo, G., Murli, A. (eds.) *High Performance Algorithms and Software for Nonlinear Optimization*, pp. 259–278. Kluwer Academic Publishers, Central and South America (2003)
- Fletcher, R., Leyffer, S., Toint, Ph.: On the global convergence of an SLP-filter algorithm. (Numerical Analysis Report NA/183, August 1998, revised October 1999) (1999)
- Fletcher, R., Leyffer, S., Toint, P.: On the global convergence of a filter-SQP algorithm. *SIAM J. Optim.* **13**, 44–59 (2002b)
- Fletcher, R., Leyffer, S., Toint, Ph.: A brief history of filter methods. (Argonne National Laboratory, Mathematics and Computer Science Division, Preprint ANL/MCS-P1372-0906, September 26, 2006, revised October 9, 2006) (2006)
- Fletcher, R., Leyffer, S., Shen, C.: Nonmonotone filter method for nonlinear optimization. (Argonne National Laboratory, Mathematics and Computer Science Division, Preprint ANL/MCS-P1679-0909, October 14) (2009)
- Fletcher, R., Powell, M.J.D.: A rapidly convergent descent method for minimization. *Comput. J.*, 163–168 (1963)
- Fletcher, R., Reeves, C.M.: Function minimization by conjugate gradient. *Comput. J.* **7**, 149–154 (1964)
- Floudas, C.A., Pardalos, P.M.: *Recent Advances in Global Optimization*. Princeton University Press, Princeton (1992)

- Floudas, C.A., Pardalos, M.P., Adjiman, C.S., Esposito, W.R., Gümüs, Z.H., Harding, S.T., Klepeis, J.L., Meyer, C.A., Schweiger, C.A.: Handbook of Test Problems in Local and Global Optimization. Kluwer Academic Publishers, Dordrecht (1999)
- Ford, J.A., Moghrabi, I.A.: Multi-step quasi-Newton methods for optimization. *J. Comput. Appl. Math.* **50**(1–3), 305–323 (1994)
- Ford, J.A., Moghrabi, I.A.: Minimum curvature multi-step quasi-Newton methods. *Comput. Math. Appl.* **31**(4–5), 179–186 (1996a)
- Ford, J.A., Moghrabi, I.A.: Using function-values in multi-step quasi-Newton methods. *J. Comput. Appl. Math.* **66**(1–2), 201–211 (1996b)
- Ford, J.A., Narushima, Y., Yabe, H.: Multi-step nonlinear conjugate gradient methods for unconstrained minimization. *Comput. Optim. Appl.* **40**(2), 191–216 (2008)
- Forsgren, A., Gill, P.E., Murray, W.: Computing modified Newton directions using a partial Cholesky factorization. *SIAM J. Sci. Comput.* **16**(1), 139–150 (1995)
- Forsgren, A., Gill, P.E., Wright, M.H.: Interior points for nonlinear optimization. *SIAM Rev.* **44**, 525–597 (2002)
- Forsythe, G.E., Hestenes, M.R., Rosser, J.B.: Iterative methods for solving linear equations. *Bull. Am. Math. Soc.* **57**, 480 (1951)
- Fourer, R.: Modeling languages versus matrix generators for linear programming. *ACM Trans. Math. Softw.* **9**, 143–183 (1983)
- Fourer, R., Gay, M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2nd edn. Duxbury Press, Scituate (2002)
- Frank, M., Wolfe, P.: An algorithm for quadratic programming. *Naval Res. Logist. Q.* **3**, 95–110 (1956)
- Friedlander, A., Martínez, J.M., Santos, S.A.: A new trust region algorithm for bound constrained minimization. *Appl. Math. Optim.* **30**, 235–266 (1994)
- Friedlander, M.P., Saunders, M.: A globally convergent linearly constrained Lagrangian method for nonlinear optimization. *SIAM J. Optim.* **15**, 863–897 (2005)
- Frisch, K.R.: The logarithmic potential method for convex programming. (Manuscript. Institute of Economics, University of Oslo, Oslo, May) (1955)
- Fukushima, M.: A successive quadratic programming algorithm with global and superlinear convergence properties. *Math. Program.* **35**, 253–264 (1986)
- Gabay, D.: Reduced quasi-Newton methods with feasibility improvement for nonlinearly constrained optimization. *Math. Program. Study.* **16**, 18–44 (1982)
- Gabriele, G.A., Ragsdell, K.M.: Large scale nonlinear programming using the generalized reduced gradient method. *Trans. ASME J. Mech. Des.* **102**, 566–573 (1980)
- Garner, J., Benedek, R.: Solution of Ginzburg-Landau equations for inhomogeneous superconductors by nonlinear optimization. *Phys. Rev. B.* **42**, 376–385 (1990)
- Gay, D.M.: Computing optimal locally constrained steps. *SIAM J. Sci. Stat. Comput.* **2**, 186–197 (1981)
- Gay, D.M.: Electronic mail distribution of linear programming test problems. *Math. Program. Soc. COAL Newsl.* **13**, 10–12 (1985)
- Gay, D.M., Overton, M.L., Wright, M.H.: A primal-dual interior method for nonconvex nonlinear programming. (Technical Report 97-4-08, Bell Lab. Murray Hill, July 29) (1997)
- Ge, R.-P., Powell, M.J.D.: The convergence of variable metric matrices in unconstrained optimization. *Math. Program.* **27**, 123–143 (1983)
- George, A., Liu, J.W.-H.: The evolution of the minimum degree ordering. *SIAM Rev.* **31**, 1–19 (1989)
- Gilbert, J.C., Lemaréchal, C.: Some numerical experiments with variable-storage quasi-Newton algorithms. *Math. Program. Ser. B.* **45**, 407–435 (1989)
- Gilbert, J.C.: SQPlab – A MATLAB software for solving nonlinear optimization problems and optimal control problems. (Technical Report, INRIA-Rocquencourt, BP 105, F-78153 Le Chesnay Cedex, France) (2009)
- Gilbert, J.C., Nocedal, J.: Global convergence properties of conjugate gradient methods for optimization. *SIAM J. Optim.* **2**, 21–42 (1992)
- Gill, P.E., Kungurtsev, V., Robinson, D.P.: A stabilized SQP method: Global convergence. *IMA J. Numer. Anal.* **37**, 407–443 (2017a)
- Gill, P.E., Kungurtsev, V., Robinson, D.P.: A stabilized SQP method: Superlinear convergence. *Math. Program.* **163**, 369–410 (2017b)
- Gill, P.E., Leonard, M.W.: Limited-memory reduced-Hessian methods for unconstrained optimization. *SIAM J. Optim.* **14**, 380–401 (2003)
- Gill, P.E., Murray, W.: Quasi-Newton methods for unconstrained optimization. *J. Inst. Math. Appl.* **9**, 91–108 (1972)
- Gill, P.E., Murray, W.: Safeguarding steplength algorithms for optimization using descent methods. (Technical Report NAC 37, National Physical Laboratory, Teddington, UK) (1974a)

- Gill, P.E., Murray, W.: Methods for large-scale linearly constrained problems. In: Gill, P.E., Murray, W. (eds.) Numerical Methods for Constrained Optimization, pp. 93–147. Academic Press, London, New York, San Francisco (1974b)
- Gill, P.E., Murray, W.: Numerically stable methods for quadratic programming. *Math. Program.* **14**, 349–372 (1978)
- Gill, Ph.E., Murray, W.: Conjugate gradient methods for large-scale nonlinear optimization. (Report SOL 79-15. Department of Operations Research, Stanford University, Stanford) (1979)
- Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: Sparse matrix methods in optimization. (Technical Report SOL82-17, Department of Operations Research, Stanford University, Stanford, California) (1982)
- Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: User's guide for SOL/QPSOL. (Technical Report SOL84-6, Department of Operations Research, Stanford University, Stanford, California) (1984)
- Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming. (Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, CA) (1986)
- Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: Maintaining LU factors of a general sparse matrix. *Linear Algebra Appl.* **88**(89), 239–270 (1987)
- Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: Some theoretical properties of an augmented Lagrangean merit function. In: Pardalos, P.M. (ed.) Advances in Optimization and Parallel Computing, pp. 101–128. North-Holland, Amsterdam (1992)
- Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. Academic Press, London (1981)
- Gill, P.E., Murray, W., Saunders, M.A.: User's guide for SNOPT (version 5.3): A Fortran package for large-scale nonlinear programming. (Technical Report NA 97-4, Department of Mathematics, University of California, San Diego) (1997)
- Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: A SQP algorithm for large-scale constrained optimization. *SIAM J. Optim.* **12**, 979–1006 (2002)
- Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.* **47**, 99–131 (2005)
- Gill, P.E., Murray, W., Saunders, M.A.: User's guide for SQOPT Version 7: Software for large-scale nonlinear programming. (Report, Department of Mathematics, University of California, San Diego) (2006)
- Gill, P.E., Golub, G.H., Murray, W., Saunders, M.A.: Methods for modifying matrix factorizations. *Math. Comput.* **28**, 505–535 (1974)
- Glowinski, R.: Numerical Methods for Nonlinear Variational Problems. Springer, Berlin (1984)
- Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik.* **38**, 173–199 (1931)
- Goldfarb, D.: A family of variable metric method derived by variation mean. *Math. Comput.* **23**, 23–26 (1970)
- Goldfarb, D.: Matrix factorizations in optimization of nonlinear functions subject to linear constraints. *Math. Program.* **10**, 1–31 (1975)
- Goldfarb, D.: Using the steepest-edge simplex algorithm to solve sparse linear programs. In: Bunch, J.R., Rose, D. (eds.) Sparse Matrix Computations, pp. 227–240. Academic Press, New York (1976)
- Goldfarb, D.: Curvilinear path steplength algorithms for minimization which use directions of negative curvature. *Math. Program.* **18**, 31–40 (1980)
- Goldfarb, D., Liu, S., Wang, S.: A logarithmic barrier function algorithm for quadratically constrained convex quadratic programming. *SIAM J. Optim.* **1**(2), 252–267 (1991)
- Goldfarb, D., Mu, C., Wright, J., Zhou, C.: Using negative curvature in solving nonlinear programs, arXiv preprint arXiv:1706.00896 (2017)
- Goldfarb, D., Idnani, A.: A numerically stable dual method for solving strictly convex quadratic programs. *Math. Program.* **27**, 1–33 (1983)
- Goldfarb, D., Reid, J.K.: A practicable steepest-edge simplex algorithm. *Math. Program.* **12**, 361–371 (1977)
- Goldfeld, S.M., Quandt, R.E., Trotter, H.F.: Maximization by quadratic hill-climbing. *Econometrica.* **34**, 541–551 (1966)
- Goldstein, A.A.: On steepest descent. *SIAM J. Control.* **3**, 147–151 (1965)
- Goldstein, A.A., Price, J.: An effective algorithm for minimization. *Numer. Math.* **10**, 184–189 (1967)
- Golub, G.H., Van Loan, C.F.: Matrix Computation, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
- Goncalves, A.S.: A primal-dual method for quadratic programming with bounded variables. In: Lootsma, F.A. (ed.) Numerical Methods for Nonlinear Optimization, pp. 255–263. Academic Press, London (1972)
- Goodman, J., Kohn, R., Reyna, L.: Numerical study of a relaxed variational problem from optimal design. *Comput. Methods Appl. Mech. Eng.* **57**, 107–127 (1986)
- Gould, N.I.M.: On the accurate determination of search directions for simple differentiable penalty functions. *IMA J. Numer. Anal.* **6**, 357–372 (1986)
- Gould, N.I.M.: An algorithm for large scale quadratic programming. *IMA J. Numer. Anal.* **11**, 299–324 (1991)

- Gould, N.I.M.: An Introduction to Algorithms for Continuous Optimization. Oxford University Computing Laboratory and Rutherford Appleton Laboratory, UK (2006)
- Gould, N.I.M., Hribar, M.E., Nocedal, J.: On the solution of equality constrained quadratic problems arising in optimization. *SIAM J. Sci. Comput.* **23**, 1375–1394 (2001)
- Gould, N.I.M., Orban, D., Toint, P.L.: GALAHAD, a library of thread-safe Fortran 90 package for large-scale nonlinear optimization. *ACM Trans. Math. Softw.* **29**(4), 353–372 (2004)
- Gould, N.I.M., Orban, D., Toint, P.L.: Numerical methods for large-scale nonlinear optimization. *Acta Numer.* **14**, 299–361 (2005a)
- Gould, N.I.M., Robinson, D.P.: A second derivative SQP methods: Local convergence. (Numerical Analysis Report 08/21, Oxford University Computing Laboratory) (2008)
- Gould, N.I.M., Robinson, D.P.: A second derivative SQP methods: Global convergence. *SIAM J. Optim.* **20**, 2023–2048 (2010)
- Gould, N.I.M., Sainvitu, C., Toint, P.L.: A filter-trust-region method for unconstrained optimization. *SIAM J. Optim.* **16**, 341–357 (2005b)
- Gould, N.I.M., Toint, P.L.: Numerical methods for large-scale non-convex quadratic programming. In: Siddiqi, A.H., Kocvara, M. (eds.) Trends in Industrial and Applied Mathematics, pp. 149–179. Kluwer Academic Publishers, Dordrecht (2002a)
- Gould, N.I.M., Toint, P.L.: An iterative working-set method for large-scale non-convex quadratic programming. *Appl. Numer. Math.* **43**, 09–128 (2002b)
- Gould, N.I.M., Toint, P.L.: Preprocessing for quadratic programming. *Math. Program.* **100**(1), 95–132 (2004)
- Gould, N.I.M., Toint, Ph.L.: A Quadratic Programming Bibliography. (RAL Numerical Analysis Group Internal Report 2000–1, March 28) (2012)
- Greenbaum, A.: Iterative Methods for Solving Linear Systems. Frontiers in Applied Mathematics. SIAM, Philadelphia (1997)
- Greenbaum, A., Strakoš, Z.: Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM J. Matrix Anal. Appl.* **13**, 121–137 (1992)
- Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia (2000)
- Griewank, A., Toint, P.L.: Partitioned variable metric updates for large structured optimization problems. *Numer. Math.* **39**, 119–137 (1982a)
- Griewank, A., Toint, P.L.: Local convergence analysis of partitioned quasi-Newton updates. *Numer. Math.* **39**, 429–448 (1982b)
- Griffith, R.E., Stewart, R.A.: A nonlinear programming technique for the optimization of continuous processing systems. *Manag. Sci.* **7**(4), 379–392 (1961)
- Grippo, L., Lampariello, F., Lucidi, S.: A nonmonotone line search technique for Newton’s method. *SIAM J. Numer. Anal.* **23**, 707–716 (1986)
- Grippo, L., Lampariello, F., Lucidi, S.: A truncated Newton method with nonmonotone line search for unconstrained optimization. *J. Optim. Theory Appl.* **60**, 401–419 (1989)
- Grippo, L., Lampariello, F., Lucidi, S.: A class of nonmonotone stabilization methods in unconstrained optimization. *Numer. Math.* **59**, 779–805 (1991)
- Grippo, L., Sciandrone, M.: Nonmonotone globalization techniques for the Barzilai-Borwein gradient method. *Comput. Optim. Appl.* **23**, 143–169 (2002)
- Griva, I., Nash, S.G., Sofer, A.: Linear and Nonlinear Optimization, 2nd edn. SIAM, Philadelphia (2009)
- Grötschel, M.: Optimization Stories. *Documenta Mathematica, Journal der Deutschen Mathematiker-Vereinigung*, Berlin (2012)
- Gu, N.Z., Mo, J.T.: Incorporating nonmonotone strategies into the trust region method for unconstrained optimization. *Comput. Math. Appl.* **55**, 2158–2172 (2008)
- Guérat, C., Prins, C., Sevaux, M.: Applications of Optimization with Xpress-MP. Dash Optimization (2002)
- Guo, Q., Liu, J.G., Wang, D.H.: A modified BFGS method and its superlinear convergence in nonconvex minimization with general line search rule. *J. Appl. Math. Comput.* **28**(1–2), 435–446 (2008)
- Hager, W.W., Zhang, H.: A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM J. Optim.* **16**(2005), 170–192 (2005)
- Hager, W.W., Zhang, H.: Algorithm 851: CG-Descent, a conjugate gradient method with guaranteed descent. *ACM Trans. Math. Softw.* **32**(1), 113–137 (2006a)
- Hager, W.W., Zhang, H.: A survey of nonlinear conjugate gradient methods. *Pac. J. Optim.* **2**(1), 35–58 (2006b)
- Han, S.P.: Superlinearly convergent variable metric algorithms for general nonlinear programming problems. *Math. Program.* **11**, 263–282 (1976)
- Han, S.P.: A globally convergent method for nonlinear programming. *J. Optim. Theory Appl.* **22**, 297–309 (1977)

- Han, J.Y., Liu, G.H., Yin, H.X.: Convergence of Perry and Shanno's memoryless quasi-Newton method for nonconvex optimization problems. *OR Trans.* **1**, 22–28 (1997)
- Han, S.P., Mangasarian, O.L.: Exact penalty functions in nonlinear programming. *Math. Program.* **17**, 251–269 (1979)
- Han, X., Zhang, J., Chen, J.: A new hybrid conjugate gradient algorithm for unconstrained optimization. *Bull. Iranian Math. Soc.* **43**(6), 2067–2084 (2017)
- Heinkenschlos, M., Ulbrich, M., Ulbrich, S.: Superlinear and quadratic convergence of affine-scaling interior-point Newton methods for problems with simple bounds without strict complementarity assumption. *Math. Program.* **86**, 615–635 (1999)
- Hellerman, E., Rarick, D.: Reinversion with the preassigned pivot procedure. *Math. Program.* **1**, 195–216 (1971)
- Hellerman, E., Rarick, D.: The partitioned preassigned pivot procedure (P4). In: Rose, D.J., Willoughby, R.A. (eds.) *Sparse Matrices and Their Applications*, pp. 67–76. Plenum Press, New York (1972)
- Herskovits, J.: A view on nonlinear optimization. In: Herskovits, J. (ed.) *Advances in Structural Optimization*. Kluwer Academic Publishers, Dordrecht (1995)
- Hestenes, M.R.: Iterative methods for solving linear equations. *J. Optim. Theory Appl.* **11**, 323–334 (1951)
- Hestenes, M.R.: Iterative computational methods. *Commun. Pure Appl. Math.* **8**, 85–96 (1955)
- Hestenes, M.R.: The conjugate-gradient method for solving linear systems. In: *Proceedings of the Sixth Symposium in Applied Mathematics* 1953, pp. 83–102. McGraw-Hill, New York (1956a)
- Hestenes, M.R.: Hilbert space methods in variational theory and numerical analysis. In: *Proceedings of the International Congress of Mathematicians* 1954, pp. 229–236, North-Holland, Amsterdam (1956b)
- Hestens, M.R.: Multiplier and gradient methods. *J. Optim. Theory Appl.* **4**, 303–320 (1969)
- Hestenes, M.R.: *Conjugate-Gradient Methods in Optimization*. Springer, Berlin (1980)
- Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.* **49**, 409–436 (1952)
- Himmelblau, D.M.: *Applied Nonlinear Programming*. McGraw-Hill, New York (1972)
- Hock, W., Schittkowski, K.: *Test Examples for Nonlinear Programming Codes* Lecture Notes in Economics and Mathematical Systems, vol. 187. Springer, Berlin (1981)
- Hooke, R., Jeeves, T.A.: Direct search solution of numerical and statistical problems. *J. Assoc. Comput. Mach.* **8**, 212–229 (1961)
- Horst, R., Pardalos, P.M., Thoai, N.V.: *Introduction to Global Optimization*, 2nd edn. Kluwer Academic Publishers, Dordrecht (2000)
- Hough, P., Kolda, T., Torczon, V.: Asynchronous parallel pattern search for nonlinear optimization. *SIAM J. Optim.* **23**, 134–156 (2013)
- Hu, Y.F., Storey, C.: Global convergence result for conjugate gradient methods. *J. Optim. Theory Appl.* **71**, 399–405 (1991)
- Huang, S., Wan, Z., Chen, X.: A new nonmonotone line search technique for unconstrained optimization. *Numer. Algorithms.* **68**, 671–689 (2014)
- Huang, H., Wei, Z., Yao, S.: The proof of the sufficient descent condition of the Wei-Yao-Liu conjugate gradient method under the strong Wolfe-Powell line search. *Appl. Math. Comput.* **189**, 1241–1245 (2007)
- Hürlimann, T.: *Mathematical Modeling and Optimization: An Essay for the Design of Computer-Based Modeling Tools*. Springer, Berlin (1999)
- ILOG CPLEX 8.0: User's Manual, ILOG SA Gentilly, France (2002)
- Irizarry, R.: A generalized framework for solving dynamic optimization problems using the artificial chemical process paradigm: Applications to particulate processes and discrete dynamic systems. *Chem. Eng. Sci.* **60**, 5663–5681 (2005)
- Jensen, D.L., Polyak, R.: The convergence of a modified barrier method for convex programming. (Research Report RC 18570, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York) (1992)
- Jiao, B.C., Chen, L.P., Pan, C.Y.: Convergence properties of a hybrid conjugate gradient method with Goldstein line search. *Math. Numer. Sin.* **29**(2), 137–146 (2007)
- Jian, J., Han, L., Jiang, X.: A hybrid conjugate gradient method with descent property for unconstrained optimization. *Appl. Math. Comput.* **39**(3–4), 1281–1290 (2015)
- Jittorntrum, K., Osborne, M.: A modified barrier function method with improved rate of convergence for degenerate problems. *J. Aust. Math. Soc. Ser. B* **21**, 305–329 (1980)
- John, F.: Extremum problems with inequalities as subsidiary conditions. In: *Studies and Essays, Presented to R. Courant on his 60th birthday, January 8, 1948*, pp. 187–204. Interscience, New York (1948)
- Jones, D., Perttunen, C., Stuckman, B.: Lipschitzian optimization without the Lipschitz constant. *J. Optim. Theory Appl.* **79**, 157–181 (1993)
- Kalan, J.E.: Aspects of large-scale in-core linear programming. In: *Proceedings of the ACM*, pp. 304–313 (1971)
- Kall, P., Wallace, S.W.: *Stochastic Programming*. Wiley, New York (1994)

- Kallrath, J., Wilson, J.M.: Business Optimization Using Mathematical Programming. Macmillan (Palgrave), Basingstoke (1997)
- Karmarkar, N.: A new polynomial time algorithm for linear programming. *Combinatorica*. **4**, 373–395 (1984)
- Karush, W.: Minima of Functions of Several Variables with Inequalities as Side Constraints. M.Sc. Dissertation, Department of Mathematics, University of Chicago, Chicago, Illinois (1939)
- Kearfott, R., Novoa, M.: INTBIS, a portable interval Newton bisection package. *ACM Trans. Math. Softw.* **16**, 152–157 (1990)
- Keller, C., Gould, N.I.M., Wathen, A.J.: Constraint preconditioning for indefinite linear systems. *SIAM J. Matrix Anal. Appl.* **21**, 1300–1317 (2000)
- Kelley, C.T.: Iterative Methods for Linear and Nonlinear Equations. Frontiers in Applied Mathematics. SIAM, Philadelphia (1995)
- Kelley, C.T.: Iterative Methods for Optimization, No 18 in Frontiers in Applied Mathematics. SIAM Publications, Philadelphia (1999)
- Kelley, C.T., Sachs, E.W.: Local convergence of the symmetric rank one iteration. *Comput. Optim. Appl.* **9**, 43–63 (1998)
- Khalfan, H.F., Byrd, R.H., Schnabel, R.B.: A theoretical and experimental study of the symmetric rank-one update. *SIAM J. Optim.* **3**(1), 1–24 (1993)
- Kim, K.: The effective integration of simplex and interior point techniques. Part I: Decomposition. Part II: Null-space affine scaling. Ph.D. Thesis, Department of Pure and Applied Mathematics, Washington State University, Pullman, Washington (1991)
- Kim, K., Nazareth, J.L.: Implementation of a primal null-space affine scaling method and its extensions. (Technical Report 92-1, Washington State University, Pullman, Washington) (1992)
- Kim, K., Nazareth, J.L.: A primal null-space affine-scaling method. *ACM Trans. Math. Softw.* **20**(3), 373–392 (1994)
- Kocvara, M., Stingl, M.: PENNON – A code for convex nonlinear and semidefinite programming. *Optim. Methods Softw.* **18**(3), 317–333 (2003)
- Kolda, T.G., Lewis, R.M., Troczon, V.: Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Rev.* **45**, 385–482 (2003)
- Kollerstrom, N.: Thomas Simpson and “Newton’s method of approximation”: An enduring myth. *Br. J. Hist. Sci.* **25**(3), 347–354 (1992)
- Kortanek, K.O., Potra, F.A., Ye, Y.: On some efficient interior point methods for nonlinear convex programming. *Linear Algebra Appl.* **152**, 169–189 (1991)
- Kou, C.X., Dai, Y.H.: A modified self-scaling memoryless Broyden-Fletcher-Goldfarb-Shanno method for unconstrained optimization. *J. Optim. Theory Appl.* **165**, 209–224 (2015)
- Kuhn, H.W., Tucker, A.W.: Nonlinear programming. In: Proceedings of 2nd Berkeley Symposium, pp. 481–492. University of California Press, Berkeley (1951)
- Lagarias, J.C., Reeds, J.A., Wright, M.H., Wright, P.E.: Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM J. Optim.* **9**, 112–147 (1998)
- Lalee, M., Nocedal, J., Plantenga, T.: On the implementation of an algorithm for large-scale equality constrained optimization. *SIAM J. Optim.* **8**, 682–706 (1998)
- Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand.* **45**, 252–282 (1950)
- Lanczos, C.: Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand.* **49**, 33–53 (1952)
- Lapidus, L., Luus, R.: The control of nonlinear systems: Part II: Convergence by combined first and second variations. *AIChE J.* **13**, 108–113 (1967)
- Larrosa, J.A.E.: New heuristics for global optimization of complex bioprocesses. Ph.D. Thesis, Universidade de Vigo, Departamento de Enxeñería Química, Vigo (2008)
- Larson, J., Menickelly, M., Wild, S.M.: Derivative-free optimization methods. (Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA) (2019)
- Lasdon, L.S., Fox, R.L., Ratner, M.W.: Nonlinear optimization using the generalized reduced gradient method. *RAIRO. 3*, 73–104 (1974)
- Lasdon, L.S., Waren, A.D.: Generalized reduced gradient software for linearly and nonlinearly constrained problems. In: Greenberg, H.J. (ed.) Design and Implementation of Optimization Software, pp. 335–362. Sijhoff and Noordhoff, Holland (1978)
- Lasdon, L.S., Waren, A.D.: A survey of nonlinear programming applications. *Oper. Res.* **28**, 34–50 (1980)
- Lasdon, L.S., Waren, A.D.: GRG2 User’s Guide. (Department of General Business, School of Business Administration, University of Texas, Austin, May 1982) (1982)
- Lasdon, L.S., Waren, A.D., Jain, A., Ratner, M.: Design and testing of a generalized reduced gradient code for nonlinear programming. *ACM Trans. Math. Softw.* **4**, 34–50 (1978)

- Lemaréchal, C.: A view of line search. In: Auslander, A., Oettli, W., Stoer, J. (eds.) Optimization and Optimal Control, pp. 59–78. Springer, Berlin (1981)
- Lemaréchal, C., Nemirovskii, A., Nesterov, Y.: New variants of bundle methods. *Math. Program.* **69**, 111–147 (1995)
- Lemke, C.E.: A method of solution for quadratic programs. *Manag. Sci.* **8**, 442–453 (1962)
- Lenstra, J.K., Rinnooy Kan, A.H.G., Schrijver, A. (eds.): History of Mathematical Programming. A Collection of Personal Reminiscences. CWI Amsterdam and North-Holland, Amsterdam (1991)
- Lescrenier, M.: Convergence of trust region algorithms for optimization with bounds when strict complementarity does not hold. *SIAM J. Numer. Anal.* **28**, 476–495 (1991)
- Levenberg, K.: A method for the solution of certain problems in least squares. *Q. Appl. Math.* **2**, 164–168 (1944)
- Levitin, E.S., Polyak, B.T.: Constrained minimization problems. *USSR Comput. Math. Math. Phys.* **6**, 1–50 (1966)
- Lewis, R.M., Torczon, V., Trosset, M.W.: Direct search methods: then and now. *J. Comput. Appl. Math.* **124**(1–2), 191–207 (2000)
- Leyffer, S., Mahajan, A.: Foundations of Constrained Optimization. (Preprint ANL/MCS-P1767-0610. Argonne National Laboratory, Mathematics and Computer Science Division, June 17) (2010)
- Li, D.H., Fukushima, M.: A modified BFGS method and its global convergence in nonconvex minimization. *J. Comput. Appl. Math.* **129**(1–2), 15–35 (2001a)
- Li, D.H., Fukushima, M.: On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM J. Optim.* **11**(4), 1054–1064 (2001b)
- Liao, A.: Modifying BFGS method. *Oper. Res. Lett.* **20**, 171–177 (1997)
- Lin, C.-J., Moré, J.J.: Newton's method for large bound-constrained optimization problems. *SIAM J. Optim.* **9**, 1100–1127 (1999)
- Lin, Y., Cryer, C.W.: An alternating direction implicit algorithm for the solution of linear complementarity problems arising from free boundary problems. *Appl. Math. Optim.* **13**, 1–7 (1985)
- Liu, J.K., Li, S.J.: New hybrid conjugate gradient method for unconstrained optimization. *Appl. Math. Comput.* **245**, 36–43 (2014)
- Liu, H.W., Liu, Z.X.: An efficient Barzilai-Borwein conjugate gradient method for unconstrained optimization. *J. Optim. Theory Appl.* **180**(3), 879–906 (2019)
- Liu, D.C., Nocedal, J.: On the limited-memory BFGS method for large optimization. *Math. Program.* **45**, 503–528 (1989)
- Liu, Y., Storey, C.: Efficient generalized conjugate gradient algorithms. Part 1: Theory. *J. Optim. Theory Appl.* **69**, 129–137 (1991)
- Liuzzi, G., Lucidi, S., Sciandrone, M.: Sequential penalty derivative-free methods for nonlinear constrained optimization. *SIAM J. Optim.* **20**, 2614–2635 (2010)
- Livieris, I.E., Tampakas, V., Pintelas, P.: A descent hybrid conjugate gradient method based on the memoryless BFGS update. *Numer. Algorithms.* **79**(4), 1169–1185 (2018)
- Lucia, A.: An explicit quasi-Newton update for sparse optimization calculations. *Math. Comput.* **40**(161), 317–322 (1983)
- Luenberger, D.G.: Introduction to Linear and Nonlinear Programming. Addison-Wesley Publishing Company, Reading (1973)
- Luenberger, D.G.: Introduction to Linear and Nonlinear Programming, 2nd edn. Addison-Wesley Publishing Company, Reading (1984)
- Luenberger, D.G., Ye, Y.: Linear and Nonlinear Programming. International Series in Operations Research & Management Science 228, 4th edn. Springer, New York (2016)
- Lustig, I.J., Marsten, R.E., Shanno, D.F.: The primal-dual interior point method on the Cray supercomputer. In: Coleman, T.F., Li, Y. (eds.) Large-Scale Numerical Optimization, pp. 70–80. SIAM, Philadelphia (1990)
- Lustig, I.J., Marsten, R.E., Shanno, D.F.: Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra Appl.* **152**, 191–222 (1991)
- Lustig, I.J., Marsten, R.E., Shanno, D.F.: On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM J. Optim.* **2**(3), 435–449 (1992)
- Lustig, I.J., Marsten, R.E., Shanno, D.F.: Computational experience with a globally convergent primal-dual predictor-corrector algorithm for linear programming. *Math. Program.* **66**, 123–135 (1994)
- Luus, R.: Picewise linear continuous optimal control by iterative dynamic programming. *Ind. Eng. Chem. Res.* **32**, 859–865 (1993)
- Mangasarian, O.L.: Nonlinear Programming. McGraw-Hill, New York. [Reprinted by SIAM Publications, 1995] (1969)
- Mangasarian, O.: Nonlinear Programming. McGraw-Hill, New York. [Reprinted by SIAM Publications, Philadelphia, USA, 1969] (1995)
- Manne, A.S.: ETA-MACRO: A model of energy-economy interactions. In: Hitch, C.J. (ed.) Modeling Energy-Economy Interactions: Five Approaches. Resources for the Future, Washington, DC (1977)

- Marquardt, D.W.: An algorithm for least-squares estimation of nonlinear parameters. *SIAM*. **11**(2), 431–441 (1963)
- Matthews, A., Davies, D.: A comparison of modified Newton methods for unconstrained optimization. *Comput. J.* **14**, 293–294 (1971)
- Mayne, D.Q., Polak, E.: A superlinearly convergent algorithm for constrained optimization problems. *Math. Program. Stud.* **16**, 45–61 (1982)
- Maratos, N.: Exact penalty function algorithms for finite-dimensional and control optimization problems. Ph. D. Thesis, University of London (1978)
- Marazzi, M., Nocedal, J.: Wedge trust region methods for derivative free optimization. *Math. Program. Ser. A* **91**, 289–305 (2002)
- Markowitz, H.M.: Portofolio selection. *J. Financ.* **8**, 77–91 (1952)
- Markowitz, H.M.: The elimination form of the inverse and its application to linear programming. *Manag. Sci.* **3**, 255–269 (1957)
- McCormick, P., Ritter, K.: Alternative proofs of the convergence properties of the conjugate gradient method. *J. Optim. Theory Appl.* **13**(5), 497–518 (1974)
- McKinney, D.C., Savitsky, A.G.: Basic optimization models for water and energy management. June 1999 (revision 6, February 2003) (2003)
- Megiddo, N.: Chapter 8: Pathways to the optimal set in linear programming. In: Megiddo, N. (ed.) *Progress in Mathematical Programming: Interior-Point and Related Methods*, pp. 131–158. Springer, New York (1989)
- Mehrotra, S.: On the implementation of a (primal-dual) interior point method. (Technical Report 90-03, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois 60208) (1989)
- Mehrotra, S.: On the implementation of a primal-dual interior point method. *SIAM J. Optim.* **2**, 575–601 (1992)
- Meintjes, K., Morgan, A.P.: Chemical-equilibrium systems as numerical test problems. *ACM Trans. Math. Softw.* **16**, 143–151 (1990)
- Mészáros, C.: The inexact minimum local fill-in ordering algorithm. Working Paper 95-7, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Hungary (1995)
- Meyer, C.D.: *Matrix Analysis and Applied Linear Algebra*. SIAM, Philadelphia (2000)
- Mizuno, S., Todd, M.J., Ye, Y.: On adaptive-step primal-dual interior-point algorithms for linear programming. *Math. Oper. Res.* **18**, 964–981 (1993)
- Moré, J.J.: Recent developments in algorithms and software for trust-region methods. In: Bachem, A., Grötschel, M., Korte, B. (eds.) *Mathematical Programming: State of the Art*, pp. 258–287. Springer, Berlin (1983)
- Moré, J.J., Sorensen, D.C.: On the use of direction of negative curvature in modified Newton method. *Math. Program.* **16**, 1–20 (1979)
- Moré, J.J., Sorensen, D.C.: On the use of directions of negative curvature in a modified Newton method. *Math. Program.* **16**, 1–20 (1983)
- Moré, J.J., Sorensen, D.C.: Newton's method. In: *Studies in Numerical Analysis*, vol. 24 of *MAA Studies in Mathematics*, pp. 29–82. The Mathematical Association of America (1984)
- Moré, J.J., Thuente, D.J.: On the line search algorithms with guaranteed sufficient decrease. (Mathematics and Computer Science Division Preprint MCS-P153-0590, Argonne National Laboratory, Argonne, IL) (1990)
- Moré, J.J., Thuente, D.J.: Line search algorithms with guaranteed sufficient decrease. (Mathematics and Computer Science Division, Preprint MCS-P330-1092, Argonne National Laboratory, October) (1992)
- Moré, J.J., Thuente, D.J.: Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Softw.* **20**, 286–307 (1994)
- Moré, J.J., Toraldo, G.: Algorithms for bound constrained quadratic programming problems. *Numer. Math.* **55**, 377–400 (1989)
- Moré, J.J., Toraldo, G.: On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optim.* **1**, 93–113 (1991)
- Morgan, A.P.: *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice Hall, Englewood Cliff (1987)
- Murtagh, B.A., Saunders, M.A.: Large-scale linearly constrained optimization. *Math. Program.* **14**, 41–72 (1978)
- Murtagh, B.A., Saunders, M.A.: MINOS/AUGMENTED User's Manual. (Technical Report SOL 80-14, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, CA 94305) (1980)
- Murtagh, B.A., Saunders, M.A.: A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. *Math. Program. Study.* **16**, 84–117 (1982)
- Murtagh, B.A., Saunders, M.A.: MINOS 5.0. User's Guide. (Technical Report SOL 83-20, Department of Operations Research, Stanford University, 1983, [Revised as MINOS 5.1 User's Guide, Report SOL 83-20R, 1987]) (1987)
- Murtagh, B.A., Saunders, M.A.: MINOS 5.4 User's Guide. (Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, CA 94305, February) (1995)

- Murray, W., Overton, M.L.: Steplength algorithms for minimizing a class of nondifferentiable functions. *Computing.* **23**, 309–331 (1979)
- Murty, K.G., Kabadi, S.N.: Some NP-complete problems in quadratic and nonlinear programming. *Math. Program.* **19**, 200–212 (1987)
- Naiman, A.E., Babuska, I.M., Elman, H.C.: A note on conjugate gradient convergence. *Numer. Math.* **76**, 209–230 (1997)
- NAG – The Numerical Algorithms Group Ltd: Mathematical optimization software. <https://www.nag.com/>
- Narushima, Y., Wakamatsu, T., Yabe, H.: Extended Barzilai-Borwein method for unconstrained optimization problems. *Pac. J. Optim.* **6**(3), 591–614 (2008)
- Narushima, Y., Yabe, H., Ford, J.A.: A three-term conjugate gradient method with sufficient descent property for unconstrained optimization. *SIAM J. Optim.* **21**, 212–230 (2011)
- Nash, S.G.: Newton-type minimization via the Lanczos method. *SIAM J. Numer. Anal.* **21**, 770–788 (1984a)
- Nash, S.G.: User's guide for TN/TNBC: Fortran routines for nonlinear optimization. (Report 397, Baltimore, MD: Mathematical Sciences Department, The John Hopkins University) (1984b)
- Nash, S.G.: Preconditioning of truncated-Newton methods. *SIAM J. Sci. Stat. Comput.* **6**, 599–616 (1985)
- Nash, S.G.: SUMT (revisited). *Oper. Res.* **46**, 763–775 (1998)
- Nash, S.G.: A survey of truncated-Newton methods. *J. Comput. Appl. Math.* **124**, 45–59 (2000)
- Nash, S.G., Nocedal, J.: A numerical study of the limited memory BFGS method and the truncated-Newton method for large-scale optimization. *SIAM J. Optim.* **1**, 358–372 (1991)
- Nash, S.G., Polyak, R., Sofer, A.: A numerical comparison of barrier and modified-barrier methods for large-scale bound-constrained optimization. In: Hager, W.W., Hearn, D.W., Pardlos, P.M. (eds.) *Large Scale Optimization: State of the Art*, pp. 319–338. Kluwer Academic Publishers, Dordrecht, Boston, London (1994)
- Navon, I.M., Legler, D.M.: Conjugate gradient methods for large-scale minimization in meteorology. *Mon. Weather Rev.* **115**, 1479–1502 (1987)
- Nazareth, J.L.: A conjugate direction algorithm without line search. *J. Optim. Theory Appl.* **23**, 373–387 (1977)
- Nazareth, J.L.: Conjugate gradient methods less dependent on conjugacy. *SIAM Rev.* **28**(4), 501–511 (1986)
- Nazareth, J.L.: *Computer Solution of Linear Programs*. Oxford University Press, New York (1987)
- Nelder, J.A., Mead, R.: A simplex method for function minimization. *Comput. J.* **7**(4), 308–313 (1965)
- Nelson, C.V., Hodgkin, B.C.: Determination of magnitudes, directions and locations of two independent dipoles in a circular conducting region from boundary potential measurements. *IEEE Trans. Biomed. Eng.* **28**, 817–823 (1981)
- Nemirovskii, A., Todd, M.: Interior point methods for optimization. *Acta Numer.* **17**, 181–234 (2008)
- Nemirovsky, A.S., Yudin, D.B.: *Problem Complexity and Method Efficiency in Optimization*. Wiley-Interscience Series in Discrete Mathematics, New York (1983)
- Nemhauser, G.L., Wolsey, L.A.: *Integer and Combinatorial Optimization*. Wiley, New York (1988)
- Nesterov, Y., Nemirovskii, A.: *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, Philadelphia (1994)
- Ng, E., Peyton, B.W.: Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.* **14**, 1034–1056 (1993)
- Ni, Q., Yuan, Y.: A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization. *Math. Comput.* **66**, 1509–1520 (1997)
- Nishida, N., Liu, Y.A., Lapidus, L., Hiratsuka, S.: An effective computational algorithm for suboptimal singular and/or bang-bang control. *AICHE J.* **22**(3), 505–513 (1976)
- Nitsche, J.C.C.: *Lectures on Minimal Surfaces*, vol. 1. Cambridge University Press, Cambridge, UK (1989)
- Nocedal, J.: Updating quasi-Newton matrices with limited storage. *Math. Comput.* **35**, 773–782 (1980)
- Nocedal, J.: Theory of algorithms for unconstrained optimization. *Acta Numer.* **1**, 199–242 (1992)
- Nocedal, J.: Conjugate gradient methods and nonlinear optimization. In: Adams, L., Nazareth, J.L. (eds.) *Linear and Nonlinear Conjugate Gradient-Related Methods*, pp. 9–23. SIAM, Philadelphia (1996)
- Nocedal, J., Wächter, A., Waltz, R.A.: Adaptive barrier strategies for nonlinear interior methods. (Technical Report RC 23563, IBM Watson Research Center, Yorktown Heights, NY, USA) (2005)
- Nocedal, J., Wright, S.J.: *Numerical Optimization*. Springer Series in Operations Research, 2nd edn. Springer Science + Business Media, New York (2006)
- Nocedal, J., Yuan, Y.X.: Analysis of self-scaling quasi-Newton method. *Math. Program.* **61**, 19–37 (1993)
- Omojokun, E.: Trust region algorithms for optimization with nonlinear equality and inequality constraints. Ph.D. Dissertation, Department of Computer Science, University of Colorado (1989)
- O'Neill, R.: Algorithm AS 47: Function minimization using a simplex procedure. *Appl. Stat.* **20**(3), 338–345 (1971)
- Oren, S.S.: Self-scaling variable metric algorithms for unconstrained optimization. Ph.D. Thesis, Department of Engineering-Economic Systems, Stanford University, Stanford (1972)
- Oren, S.S.: Self-scaling variable metric algorithm. Part II. *Manag. Sci.* **20**, 863–874 (1974)

- Oren, S.S., Luenberger, D.G.: Self-scaling variable metric (SSVM) algorithms, part I: criteria and sufficient conditions for scaling a class of algorithms. *Manag. Sci.* **20**, 845–862 (1974)
- Oren, S.S., Spedicato, E.: Optimal conditioning of self-scaling variable metric algorithm. *Math. Program.* **10**, 70–90 (1976)
- Ortega, J.M., Rheinboldt, W.C.: Iterative Solutions of Nonlinear Equations in Several Variables. SIAM Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia (2000)
- Ou, Y., Liu, Y.: A memory gradient method based on the nonmonotone technique. *J. Ind. Manag. Optim.* **13**(2), 857–872 (2017)
- Ou, Y., Lin, H.: A class of accelerated conjugate-gradient-like methods based on a modified secant equation. *J. Ind. Manag. Optim.* **16**(3), 1503–1518 (2020)
- Panier, E.R., Tits, A.L.: Avoiding the Maratos effect by means of a nonmonotone line search. I: General constrained problema. *SIAM J. Numer. Anal.* **28**, 1183–1195 (1991)
- Pant, M., Thangaraj, R., Singh, V.P.: Particle swarm optimization with crossover operator and its engineering applications. *IAENG Int. J. Comput. Sci.* **36**, 112–121 (2009)
- Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice Hall, Englewood Cliffs (1982)
- Pardalos, P.M., Rosen, J.B.: Constrained Global Optimization: Algorithms and Applications. Springer, Berlin (1987)
- Peressini, A.L., Sullivan, F.E., Uhl, J.J.: The Mathematics of Nonlinear Programming. Springer, New York Inc (1988)
- Perry, A.: A modified conjugate gradient algorithm. (Discussion Paper No. 229, Center for Mathematical Studies in Economics and Management Science, Northwestern University) (1976)
- Perry, A.: A class of conjugate gradient algorithms with two step variable metric memory. (Discussion paper 269, Center for Mathematical Studies in Economics and Management Science. Northwestern University, IL, USA) (1977)
- Peyret, R., Taylor, T.D.: Computational Methods for Fluid Flow. Springer, New York (1985)
- Polak, E., Ribiére, G.: Note sur la convergence de méthodes de direction conjugées. *Revue Francaise d'Informatique et de Recherche Opérationnelle.* **16**, 35–43 (1969)
- Polyak, B.T.: The conjugate gradient method in extremal problems. *USSR Comput. Math. Math. Phys.* **9**, 94–112 (1969)
- Polyak, R.: Modified barrier functions (theory and methods). *Math. Program.* **54**, 177–222 (1992)
- Potra, F.A., Rheinboldt, W.C.: On the monotone convergence of Newton's method. *Computing.* **36**(1), 81–90 (1986)
- Potra, F.A., Shi, Y.: Efficient line search algorithm for unconstrained optimization. *J. Optim. Theory Appl.* **85**, 677–704 (1995)
- Powell, M.J.D.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.* **17**, 155–162 (1964)
- Powell, M.J.D.: A method for nonlinear constraints in optimization problems. In: Fletcher, R. (ed.) Optimization, pp. 283–297. Academic Press, New York (1969)
- Powell, M.J.D.: Convergence properties of a class of minimization algorithms. In: Mangasarian, O., Meyer, R., Robinson, S. (eds.) Nonlinear Programming 2, pp. 1–27. Academic Press, New York (1975)
- Powell, M.J.D.: Some global convergence properties of a variable-metric algorithm for minimization without exact line searches. In: Cottle, R.W., Lemke, C.E. (eds.) Nonlinear Programming, SIAM-AMS Proceedings, vol. 9, pp. 53–72, Philadelphia, PA, USA (1976)
- Powell, M.J.D.: Restart procedures of the conjugate gradient method. *Math. Program.* **2**, 241–254 (1977)
- Powell, M.J.D.: A fast algorithm for nonlinearly constrained optimization calculations. In: Watson, G.A. (ed.) Numerical Analysis, Dundee 1977, Lecture Notes in Mathematics, vol. 630, pp. 144–157. Springer, Berlin (1978a)
- Powell, M.J.D.: The convergence of variable metric methods of nonlinearly constrained optimization calculations. In: Mangasarian, O.L., Meyer, R.R., Robinson, S.M. (eds.) Nonlinear Programming 3, pp. 27–63. Academic Press, New York (1978b)
- Powell, M.J.D.: Algorithms for nonlinear constraints that use Lagrangian functions. *Math. Program.* **14**, 224–248 (1978c)
- Powell, M.J.D.: ZQPCVX: A Fortran subroutine for convex quadratic programming. (Technical Report, Department of Applied Mathematics and Theoretical Physics, Cambridge University) (1983)
- Powell, M.J.D.: Nonconvex minimization calculations and the conjugate gradient method. In: Griffiths, D.F. (ed.) Numerical Analysis (Dundee, 1983), Lecture Notes in Mathematics, vol. 1066, pp. 122–141 (1984)
- Powell, M.J.D.: How bad are the BFGS and DFP methods when the objective function is quadratic? *Math. Program.* **34**, 34–47 (1986a)
- Powell, M.J.D.: Convergence properties of algorithms for nonlinear optimization. *SIAM Rev.* **28**(4), 487–500 (1986b)
- Powell, M.J.D.: Updating conjugate directions by the BFGS formula. *Math. Program.* **38**, 693–726 (1987)

- Powell, M.J.D.: A direct search optimization method that models the objective and constraint functions by linear interpolation. (Department of Applied Mathematics and Theoretical Physics, University of Cambridge (revised August 1993)) (1993)
- Powell, M.J.D.: A direct search optimization method that models the objective and constraint functions by linear interpolation. In: Gomez, S., Hennart, J.P. (eds.) *Advances in Optimization and Numerical Analysis*, pp. 51–67. Kluwer Academic Publishers (1994)
- Powell, M.J.D.: Direct search algorithms for optimization calculations. *Acta Numer.* **7**, 287–336 (1998)
- Powell, M.J.D.: UOBYQA: Unconstrained optimization by quadratic approximation. *Math. Program.* **92**, 555–582 (2002)
- Powell, M.J.D.: On trust region methods for unconstrained minimization without derivatives. *Math. Program.* **97**, 605–623 (2003)
- Powell, M.J.D.: The NEWUOA software for unconstrained optimization without derivatives. (Department of Applied Mathematics and Theoretical Physics, Centre for Mathematical Sciences, Cambridge) (2004)
- Powell, M.J.D.: The NEWUOA software for unconstrained minimization without derivatives. In: Di Pillo, G., Roma, M. (eds.) *Large-Scale Nonlinear Optimization*, pp. 255–297. Springer (2006)
- Powell, M.J.D.: The BOBYQA algorithm for bound constrained optimization without derivatives. (Technical Report, Department of Applied Mathematics and Theoretical Physics, Cambridge, England) (2009)
- Powell, M.J.D.: On the convergence of trust region algorithms for unconstrained minimization without derivatives. (Technical Report, Department of Applied Mathematics and Theoretical Physics, Cambridge, England) (2011)
- Powell, M.J.D., Toint, P.L.: On the estimation of sparse Hessian matrices. *SIAM J. Numer. Anal.* **16**, 1060–1074 (1979)
- Pulkkinen, S.: A review of methods for unconstrained optimization: Theory, implementation and testing. Master's Thesis, University of Helsinki, Department of Mathematics and Statistics (2008)
- Price, W.L.: Global optimization by controlled random search. *J. Optim. Theory Appl.* **55**(1983), 333–348 (1983)
- Price, C.J., Coope, D., Byatt, D.: A convergent variant of the Nelder-Mead algorithm. *J. Optim. Theory Appl.* **113**, 5–19 (2002)
- Ratschek, H., Rokne, J.: A circuit design problem. *J. Global Optim.* **3**, 501 (1993)
- Raydan, M.: On the Barzilai and Borwein choice of steplength for the gradient method. *IMA J. Numer. Anal.* **13**, 321–326 (1993)
- Raydan, M.: The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM J. Optim.* **7**, 26–33 (1997)
- Raydan, M., Svaiter, B.F.: Relaxed steepest descent and Cauchy-Barzilai-Borwein method. *Comput. Optim. Appl.* **21**, 155–167 (2002)
- Reid, J.K.: Sparse in core linear programming. In: Watson, G. (ed.) *Numerical Analysis Dundee, 1975*, Lecture Notes in Mathematics, 506, pp. 176–189 (1975)
- Reid, J.K.: A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Math. Program.* **24**, 55–69 (1982)
- Rijckaert, M.J.: Engineering applications of geometric programming. In: Avriel, M., Rijckaert, M.J., Wilde, M. (eds.) *Optimization in Design*. Prentice-Hall, Englewood Cliffs (1973)
- Rios, L.M., Shainidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Glob. Optim.* **56**, 1247–1293 (2013)
- Robinson, S.M.: A quadratically convergent algorithm for general nonlinear programming problems. *Math. Program.* **3**, 145–156 (1972)
- Robinson, S.M.: Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. *Math. Program.* **7**, 1–16 (1974)
- Rockafellar, R.T.: *Convex Analysis*. Princeton University Press, Princeton (1970)
- Rose, D.J., Tarjan, R.E.: Algorithmic aspects of vertex elimination on graphs. In: Proceedings of the 7th Annual Symposium on the Theory of Computing, pp. 245–254. Association for Computing Machinery, New York (1975)
- Rosen, J.B., Kreuser, J.: A gradient projection algorithm for nonlinear constraints. In: Lootsma, F. (ed.) *Numerical Methods for Nonlinear Optimization*, pp. 297–300. Academic Press, London (1972)
- Rosen, J.B.: The gradient projection method for nonlinear programming. Part I – linear constraints. *J. SIAM Appl. Math.* **8**(1), 181–217 (1960)
- Rosen, J.B.: The gradient projection method for nonlinear programming. Part II – nonlinear constraints. *J. SIAM Appl. Math.* **9**(4), 414–432 (1961)
- Rosenbrock, H.H.: An automatic method for finding the greatest or least value of a function. *Comput. J.* **3**, 175–184 (1960)
- Rosser, J.B.: Rapidly converging iterative methods for solving linear equations. In: Paige, L.J., Taussky, O. (eds.) *Simultaneous Linear Equations and the Determination of Eigenvalues*. Applied Mathematics Series 29, pp. 59–64. National Bureau of Standards, U.S. Government Printing Office, Washington, DC (1953)

- Rothberg, E., Hendrickson, B.: Sparse matrix ordering methods for interior-point linear programming. (Technical Report SAND96-0475J, Sandia National Laboratory) (1996)
- Sainvitu, C., Toint, Ph.L.: A filter-trust-region method for simple-bound constrained optimization. (Technical Report, Department of Mathematics, University of Namur, Belgium) (2006)
- Sandgren, E.: Nonlinear integer and discrete programming in mechanical design. In: Proceedings of the ASME Design Technology Conference, pp. 95–105, Kissimmee (1988)
- Sandgren, E., Ragsdell, K.M.: The utility of nonlinear programming algorithms: A comparative study – Part I. *J. Mech. Des.* **102**(3), 540–546 (1980)
- Sargent, E., Murtagh, B.A.: Projection methods for nonlinear programming. *Math. Program.* **4**, 245–268 (1973)
- Saunders, M.: Augmented Lagrangian methods. (Notes 9. Stanford University, Management Science & Engineering. Spring 2015) (2015a)
- Saunders, M.: NPSOL and SNOPT – SQP Methods. (Notes 11. Stanford University, Management Science & Engineering. Spring 2015) (2015b)
- Schlick, T.: Modified Cholesky factorizations for sparse preconditioners. *SIAM J. Sci. Comput.* **14**, 424–445 (1993)
- Schlick, T., Fogelson, A.: TNPACK – A truncated Newton minimization package for large-scale problems: I Algorithm and usage. *ACM Trans. Math. Softw.* **18**, 46–70 (1992a)
- Schlick, T., Fogelson, A.: TNPACK – A truncated Newton minimization package for large-scale problems: II implementation examples. *ACM Trans. Math. Softw.* **18**, 71–111 (1992b)
- Schittkowski, K.: The nonlinear programming method of Wilson, Han and Powell with an augmented Lagrangean type line search function. Part I: Convergence analysis. *Numer. Math.* **38**, 83–114 (1981)
- Schittkowski, K.: On the convergence of a sequential quadratic programming method with an augmented Lagrange line search function. *Math. Operationsforschung Statistik, Ser. Optim.* **14**(2), 197–216 (1983)
- Schittkowski, K.: NLQPL: A Fortran subroutine for solving constrained nonlinear programming problems. *Ann. Oper. Res.* **5**(2), 485–500 (1985)
- Schittkowski, K.: NLQPL: A Fortran subroutine solving constrained nonlinear programming problems. *Ann. Oper. Res.* **5**, 485–500 (1986)
- Schittkowski, K.: More Test Examples for Nonlinear Programming Codes. Springer, Berlin (1987)
- Schittkowski, K.: NLQPLP: A Fortran implementation of a sequential quadratic programming algorithm. User's guide. (Report, Department of Mathematics, University of Bayreuth) (2002)
- Schittkowski, K.: QL: A Fortran code for convex quadratic programming. User's guide, Version 2.11. (Technical Report, Department of Mathematics, University of Bayreuth, July) (2005)
- Schittkowski, K.: NLQPO: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search. User's guide, version 3.0. (Technical Report, Department of Computer Science, University of Bayreuth) (2009)
- Schittkowski, K.: A robust implementation of a sequential quadratic programming algorithm with successive error restoration. (Technical Report, Department of Computer Science, University of Bayreuth) (2010)
- Schnabel, R.B., Chow, T.T.: Tensor methods for unconstrained optimization using second derivatives. *SIAM J. Optim.* **1**, 293–315 (1991)
- Schnabel, R.B., Frank, P.D.: Tensor methods for nonlinear equations. *SIAM J. Numer. Anal.* **21**, 815–843 (1984)
- Schreiber, R., Keller, H.: Driven cavity flows by efficient numerical techniques. *J. Comput. Phys.* **49**, 310–333 (1983)
- Schultz, G.A., Schnabel, R.B., Byrd, R.H.: A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties. *SIAM J. Numer. Anal.* **22**, 47–67 (1985)
- Shacham, M.: Numerical solution of constrained nonlinear algebraic equations. *Int. J. Numer. Methods Eng.* **23**, 1455–1481 (1986)
- Shanno, D.F.: Conditioning of quasi-Newton methods for function minimization. *Math. Comput.* **24**, 647–656 (1970)
- Shanno, D.F.: Conjugate gradient methods with inexact searches. *Math. Oper. Res.* **3**, 244–256 (1978a)
- Shanno, D.F.: On the convergence of a new conjugate gradient algorithm. *SIAM J. Numer. Anal.* **15**, 1247–1257 (1978b)
- Shanno, D.F.: CONMIN – A Fortran subroutine for minimizing an unconstrained nonlinear scalar valued function of a vector variable  $x$  either by the BFGS variable metric algorithm or by a Beale restarted conjugate gradient algorithm. Private communication, October 17, 1983
- Shanno, D.F.: Globally convergent conjugate gradient algorithms. *Math. Program.* **33**, 61–67 (1985)
- Shanno, D.F.: Who invented the interior-point method? In: Grötschel, M. (ed.) Optimization stories. Documenta Mathematica, Journal der Deutschen Mathematiker-Vereinigung, Extra volume, 21st International Symposium on Mathematical Programming, pp. 55–64, Berlin, August 19–24, 2012
- Shanno, D.F., Breitfeld, M.G., Simantiraki, E.M.: Implementing barrier methods for nonlinear programming. In: Terlaky, T. (ed.) Interior Point Methods of Mathematical Programming, pp. 399–414. Kluwer Academic Publishers (1996)

- Shanno, D.F., Phua, K.H.: Algorithm 500. Minimization of unconstrained multivariable functions. *ACM Trans. Math. Softw.* **2**, 87–94 (1976)
- Shanno, D.F., Phua, K.H.: Matrix conditioning and nonlinear optimization. *Math. Program.* **14**, 149–160 (1978)
- Shanno, D.F., Phua, K.H.: Remark on algorithm 500. *ACM Trans. Math. Softw.* **6**, 618–622 (1980)
- Shanno, D.F., Simantiraki, E.M.: Interior-point methods for linear and nonlinear programming. In: Duff, I.S., Watson, G.A. (eds.) *The State of the Art in Numerical Analysis*, pp. 339–362. Oxford University Press, New York (1997)
- Shi, Z.-J.: Convergence of line search methods for unconstrained optimization. *Appl. Math. Comput.* **157**, 393–405 (2004)
- Shi, Z.-J., Shen, J.: Step-size estimation for unconstrained optimization methods. *Comput. Appl. Math.* **24**(3), 399–416 (2015)
- Sorensen, D.C.: Newton's method with a model trust region modification. *SIAM J. Numer. Anal.* **19**, 409–426 (1982)
- Spall, J.C.: *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley Series in Discrete Mathematics and Optimization. Wiley, New York (2003)
- Spellucci, P.: An SQP method for general nonlinear programs using only equality constrained subproblems. *Math. Program.* **3**, 413–448 (1998)
- Spendley, W., Hext, G.R., Himsworth, F.R.: Sequential applications of simplex designs in optimization and evolutionary operation. *Technometrics* **4**, 441–461 (1962)
- Spedicato, E., Zhao, J.: Explicit general solution of the Quasi-Newton equation with sparsity and symmetry. *Optim. Methods Softw.* **2**(3–4), 311–319 (1993)
- Stewart, G.W.: A modification of Davidon's method to accept difference approximation of derivatives. *ACM* **14**, 72–83 (1967)
- Steihaug, T.: The conjugate gradient method and trust-regions in large-scale optimization. *SIAM J. Numer. Anal.* **20**, 626–637 (1983)
- Stiefel, E.L.: Kernel polynomials in linear algebra and their numerical applications. In: *Further Contributions to the Determination of Eigenvalues*. National Bureau of Standards, Applied Mathematical Series, 49, pp. 1–22 (1958)
- Stoer, J., Yuan, Y.X.: A subspace study on conjugate gradient algorithms. *ZAMP – J. Appl. Math. Mech.* **75**, 69–77 (1995)
- Strogatz, S.H.: *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books, Cambridge, MA (1994)
- Suhl, U.H., Suhl, L.M.: Computing sparse LU factorizations for large-scale linear programming bases. *ORSA J. Comput.* **2**, 325–335 (1990)
- Suhl, L.M., Suhl, U.H.: A fast LU-update for linear programming. (Arbeitspapier des Instituts für Wirtschaftsinformatik, Freie Universität – Berlin, August) (1991)
- Sun, W., Liu, H., Liu, Z.: A class of accelerated subspace minimization conjugate gradient methods. *J. Optim. Theory Appl.* **190**, 811–840 (2021)
- Sun, W., Yuan, Y.X.: *Optimization Theory and Methods. Nonlinear Programming*. Springer Science + Business Media, New York (2006)
- Surry, P.D., Radcliffe, N.J., Boyd, I.D.: A multi-objective approach to constrained optimization of gas supply networks: The COMOGA method. In: Fogarty, T.C. (ed.) *Evolutionary Computing: AISB Workshop*, Number 993. Lecture Notes in Computer Science, pp. 166–180. Springer, Berlin (1995)
- Tapia, R.A.: A stable approach to Newton's method for general mathematical programming problems in  $\mathbb{R}^n$ . *J. Optim. Theory Appl.* **14**, 453–476 (1974)
- Tits, A.L., Wächter, A., Bakhtiari, S., Urban, T.J., Lawrence, C.T.: A primal-dual interior-point method for nonlinear programming with strong global and local convergence properties. *SIAM J. Optim.* **14**, 173–199 (2003)
- Toint, P.L.: On sparse and symmetric matrix updating subject to a linear equation. *Math. Comput.* **31**, 954–961 (1977)
- Toint, P.L.: Towards an efficient sparsity exploiting Newton method for minimization. In: *Sparse Matrices and Their Uses*, pp. 57–87. Academic Press, New York (1981)
- Toint, P.L.: An assessment of nonmonotone line search techniques for unconstrained optimization. *SIAM J. Sci. Comput.* **17**, 725–739 (1996)
- Toint, P.L.: A nonmonotone trust-region algorithm for nonlinear optimization subject to convex constraints. *Math. Program.* **77**, 69–94 (1997)
- Topkis, D.M., Veinott, A.F.: On the convergence of some feasible direction algorithms for nonlinear programming. *SIAM J. Control.* **5**, 268–279 (1967)
- Touati-Ahmed, D., Storey, C.: Efficient hybrid conjugate gradient techniques. *J. Optim. Theory Appl.* **64**, 379–397 (1990)
- Traub, J.: *Iterative Methods for the Solution of Equations*. Prentice-Hall, Englewood Cliffs (1964)
- Trefethen, L.N., Bau III, D.: *Numerical Linear Algebra*. SIAM, Philadelphia (1997)
- Trefethen, L.N., Schreiber, R.S.: Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. Appl.* **11**(3), 335–360 (1990)

- Tröltzsch, A.: Benchmarking of bound-constrained optimization software. (CERFACS working note: WN/PA/07/143, pp. 1–39) (2007)
- Ulbrich, S.: On the superlinear local convergence of a filter-SQP method. *Math. Program.* **100**(1), 217–245 (2004)
- Ulbrich, M., Ulbrich, S., Heinkenschloss, M.: Global convergence of affine-scaling interior-point Newton methods for infinite-dimensional nonlinear problems with pointwise bounds. *SIAM J. Control. Optim.* **37**, 731–764 (1999)
- Ulbrich, M., Ulbrich, S., Vicente, L.N.: A globally convergent primal-dual interior-point filter method for nonconvex nonlinear programming. *Math. Program.* **100**, 379–410 (2004)
- Vanderbei, R.J.: ALPO: Another linear program optimizer. (Technical Report AT&T Bell Laboratories) (1990)
- Vanderbei, R.J.: An interior point code for quadratic programming. (Technical Report SOR 94-15, Princeton University) (1994)
- Vanderbei, R.J.: LOQO: An interior point code for quadratic programming. (Technical Report SOR 94-15, Princeton University) (1995)
- Vanderbei, R.: Linear Programming Foundations and Extensions. Kluwer Academic Publishers, Boston, London, Dordrecht (1996)
- Vanderbei, R.J.: Linear Programming: Foundations and Extensions, 2nd edn. Springer, New York (2001)
- Vanderbei, R.J., Shanno, D.F.: An interior-point algorithm for nonconvex nonlinear programming. (Technical Report SOR 97-21, Princeton University) (1997)
- Vanderbei, R.J., Shanno, D.F.: An interior point algorithm for nonconvex nonlinear programming. *Comput. Optim. Appl.* **13**, 231–252 (1999)
- Vanderplaats, G.N.: DOT Users Manual. Version 4.20, Vanderplaats Research & Development, Inc, Colorado Springs (1995)
- Van der Pol, B.: Forced oscillations in a circuit with nonlinear resistance (receptance with reactive triode). *Lond. Edinb. Dublin Philos. Mag.* **3**, 65–80 (1927)
- Van der Vorst, H.A.: Lecture Notes on Iterative Methods. (Report Mathematical Institute, University of Utrecht) (1993)
- Vardi, A.: A trust region algorithm for equality constrained minimization: convergence properties and implementation. *SIAM J. Numer. Anal.* **22**, 575–591 (1985)
- Vassiliadis, V.S., Floudas, C.A.: The modified barrier function approach for large-scale optimization. *Comp. Chem. Engng.* **21**(8), 855–874 (1997)
- Vavasis, S.A.: Quadratic programming is NP. *Inf. Process. Lett.* **36**, 73–77 (1990)
- Vavasis, S.A.: Nonlinear Optimization. Oxford University Press, New York and Oxford (1991)
- Verschelde, J., Verlinden, P., Cools, R.: Homotopies exploiting Newton polytopes for solving sparse polynomial systems. *SIAM J. Numer. Anal.* **31**, 915–930 (1994)
- Von Stryk, O.: User's guide for DIRCOL (Version 2.1): A direct collocation method for the numerical solution of optimal control problems. (Technical Report, Technische Universität München) (1999)
- Wächter, A.: An interior point algorithm for large scale nonlinear optimization with applications in process engineering. Ph.D. Thesis, Carnegie Mellon University, Pittsburg, PA, January 2002
- Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* **106**, 25–57 (2000)
- Wächter, A., Biegler, L.T.: Line search filter methods for nonlinear programming: Motivation and global convergence. (Technical Report RC 23036, Yorktown Heights, NY: IBM T.J. Watson Research Center (revised 2004)) (2001)
- Wächter, A., Biegler, L.T.: Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM J. Optim.* **16**, 1–31 (2005a)
- Wächter, A., Biegler, L.T.: Line search filter methods for nonlinear programming: Local convergence. *SIAM J. Optim.* **16**, 32–48 (2005b)
- Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* **106**, 25–57 (2006)
- Walsh, G.R.: Methods of Optimization. Wiley, London, New York, Sydney, Toronto (1975)
- Waltz, R.A.: KNITRO 4.0 User's Manual. (Technical Report, Ziena Optimization Inc., Evanston, IL, October) (2004)
- Waltz, R.A., Morales, J.L., Nocedal, J., Orban, D.: An interior algorithm for nonlinear optimization that combines line search and trust region step. (Technical Report 2003-6, Optimization Technology Center, Northwestern University, Evanston, IL, USA, June). [Math. Program. 107, 2006, 391–408] (2003)
- Wan, Z., Huang, S., Zheng, X.D.: New cautious BFGS algorithm based on modified Armijo-type line search. *J. Inequal. Appl.* **241**, 1–10 (2012)
- Wan, Z., Teo, K.L., Shen, X.L., Hu, C.M.: New BFGS method for unconstrained optimization problem based on modified Armijo line search. *Optimization* **63**(2), 285–304 (2014)
- Wang, H.J., Yuan, Y.X.: A quadratic convergence method for one-dimensional optimization. *Chin. J. Oper. Res.* **11**, 1–10 (1992)
- Wei, Z., Li, G., Qi, L.: New nonlinear conjugate gradient formulas for large-scale unconstrained optimization problems. *Appl. Math. Comput.* **179**, 407–430 (2006a)

- Wei, Z., Yao, S., Liu, L.: The convergence properties of some new conjugate gradient methods. *Appl. Math. Comput.* **183**, 1341–1350 (2006b)
- Wei, Z., Yu, G., Yuan, G., Lian, Z.: The superlinear convergence of a modified BFGS-type method for unconstrained optimization. *Comput. Optim. Appl.* **29**, 315–332 (2004)
- Wilkinson, J.H.: The Algebraic Eigenvalue Problem. Oxford University Press, London (1965)
- Wilson, R.B.: A simplicial algorithm for concave programming. Ph.D Thesis, Harvard University (1963)
- Wolfe, P.: A duality theorem for nonlinear programming. *Q. Appl. Math.* **19**, 239–244 (1961)
- Wolfe, P.: Methods of nonlinear programming. In: Abadie, J. (ed.) Nonlinear Programming, pp. 97–131. North-Holland, Amsterdam (1967)
- Wolfe, P.: Convergence conditions for ascent methods. *SIAM Rev.* **11**, 226–235 (1969)
- Wolfe, P.: Convergence conditions for ascent methods. II: Some corrections. *SIAM Rev.* **13**, 185–188 (1971)
- Wolsey, L.A.: Integer Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York (1998)
- Wright, M.H.: Numerical methods for nonlinearly constrained optimization. (SLAC Report No.193, 1976, Stanford University, California. (Ph.D. Dissertation)) (1976)
- Wright, M.H.: Direct search methods: Once scorned, now respectable. In: Numerical Analysis 1996 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis), pp. 191–208. Addison Wesley Longman (1996)
- Wright, S.J.: Implementing proximal point methods for linear programming. *J. Optim. Theory Appl.* **65**, 531–554 (1990)
- Wright, M.H.: Interior methods for constrained optimization. *Acta Numer.* **1**, 341–407 (1991)
- Wright, S.J.: Primal-Dual Interior-Point Methods. SIAM, Philadelphia (1997)
- Wu, G., Liang, H.: A modified BFGS method and its convergence. *Comput. Model. New Technol.* **18**(11), 43–47 (2014)
- Xavier, A.E.: Hyperbolic penalty: A new method for nonlinear programming with inequalities. *Int. Trans. Oper. Res.* **8**, 659–671 (2001)
- Yabe, H., Martínez, H.J., Tapia, R.A.: On sizing and shifting the BFGS update within the sized Broyden family of secant updates. *SIAM J. Optim.* **15**(1), 139–160 (2004)
- Yabe, H., Ogasawara, H., Yoshino, M.: Local and superlinear convergence of quasi-Newton methods based on modified secant conditions. *J. Comput. Appl. Math.* **205**, 717–632 (2007)
- Yamashita, H.: A globally convergent primal-dual interior-point method for constrained optimization. *Optim. Methods Softw.* **10**, 443–469 (1998)
- Yamashita, N.: Sparse Quasi-Newton Updates with Positive Definite Matrix Completion. Department of Applied Mathematics of Physics, Graduate School of Informatics, Kyoto University (2005)
- Yao, S., Wei, Z., Huang, H.: A note about WYL's conjugate gradient method and its application. *Appl. Math. Comput.* **191**, 381–388 (2007)
- Yang, X., Luo, Z., Dai, X.: A global convergence of LS-CD hybrid conjugate gradient method. *Adv. Numer. Anal.* **2013**, 517452 (2013). <https://doi.org/10.1155/2013/517452>
- Yang, E.K., Tolle, J.W.: A class of methods for solving large convex quadratic programs subject to box constraints. *Math. Program.* **51**, 223–228 (1991)
- Ye, Y.: Interior Point Algorithms: Theory and Analysis. Wiley, New York (1997)
- Ye, Y., Todd, M.J., Mizuno, S.: An  $O(nl^{1/2})$ -iterations homogeneous and self-dual linear programming algorithm. *Math. Oper. Res.* **19**, 53–67 (1994)
- Ypma, T.J.: Historical development of the Newton-Raphson method. *SIAM Rev.* **37**, 531–551 (1995)
- Yuan, Y.X.: A modified BFGS algorithm for unconstrained optimization. *IMA J. Numer. Anal.* **11**, 325–332 (1991)
- Yuan, Y.X.: Analysis on the conjugate gradient method. *Optim. Methods Softw.* **2**, 19–29 (1993)
- Yuan, Y.: A new stepsize for the steepest descent method. *J. Comput. Math.* **24**, 149–156 (2006)
- Yuan, Y.: Advances in trust region algorithms. *Math. Program.* **151**, 249–281 (2015)
- Yuan, Y.X., Byrd, R.: Non-quasi-Newton updates for unconstrained optimization. *J. Comput. Math.* **13**(2), 95–107 (1995)
- Yuan, G., Sheng, Z., Wang, B., Hu, W., Li, C.: The global convergence of a modified BFGS method for nonconvex functions. *J. Comput. Appl. Math.* **327**, 274–294 (2018)
- Yuan, G., Wei, Z.: Convergence analysis of a modified BFGS method on convex minimizations. *Comput. Optim. Appl.* **47**, 237–255 (2010)
- Yuan, G., Wei, Z., Lu, X.: Global convergence of BFGS and PRP methods under a modified weak Wolfe–Powell line search. *Appl. Math. Model.* **47**, 811–825 (2017)
- Zangwill, W.I.: The convex simplex method. *Manag. Sci.* **14**(3), 221–238 (1967)
- Zhang, Y.: Interior-point gradient methods with diagonal-scaling for simple-bound constrained optimization. (Technical Report TR04-06, Department of Computational and Applied Mathematics, Rice University, Houston, Texas) (2004)
- Zhang, L.: Two modified Dai-Yuan nonlinear conjugate gradient methods. *Numer. Algorithms.* **50**(1), 1–16 (2009a)

- Zhang, L.: New versions of the Hestenes-Stiefel nonlinear conjugate gradient method based on the secant condition for optimization. *Comput. Appl. Math.* **28**(1), 1–23 (2009b)
- Zhang, J., Deng, N.Y., Chen, L.H.: New quasi-Newton equation and related methods for unconstrained optimization. *J. Optim. Theory Appl.* **102**, 147–167 (1999)
- Zhang, H., Hager, W.W.: A nonmonotone line search technique and its application to unconstrained optimization. *SIAM J. Optim.* **14**, 1043–1056 (2004)
- Zhang, Y., Tapia, R.A., Dennis, J.E.: On the superlinear and quadratic convergence of primal-dual interior-point linear programming algorithms. *SIAM J. Optim.* **2**, 304–324 (1992)
- Zhang, J., Xu, C.: Properties and numerical performance of quasi-Newton methods with modified quasi-Newton equations. *J. Comput. Appl. Math.* **137**, 269–278 (2001)
- Zhang, L., Zhou, W.: Two descent hybrid conjugate gradient methods for optimization. *J. Comput. Appl. Math.* **216**, 251–264 (2008)
- Zhang, L., Zhou, W., Li, H.: Some descent three-term conjugate gradient methods and their global convergence. *Optim. Methods Softw.* **22**(4), 697–711 (2007)
- Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B, FORTRAN subroutines for large scale bound constrained optimization. *ACM Trans. Math. Softw.* **23**, 550–560 (1997)
- Zhu, M., Nazareth, J.L., Wolkowicz, H.: The quasi-Cauchy relation and diagonal updating. *SIAM J. Optim.* **9**(4), 1192–1204 (1999)
- Zhu, H., Wen, S.: A class of generalized quasi-Newton algorithms with superlinear convergence. *Int. J. Nonlinear Sci.* **2**(3), 140–146 (2006)
- Zou, X., Navon, I.M., Berger, M., Phua, K.H., Schlick, T., Le Dimet, F.X.: Numerical experience with limited-memory quasi-Newton and truncated Newton methods. *SIAM J. Optim.* **3**(3), 582–608 (1993)
- Zoutendijk, G.: Methods of Feasible Directions. Elsevier, Amsterdam (1960)
- Zoutendijk, G.: Nonlinear programming, computational methods. In: Abadie, J. (ed.) Integer and Nonlinear Programming, pp. 38–86. North-Holland, Amsterdam (1970)
- Zwart, P.B.: Nonlinear Programming: A Quadratic Analysis of Ridge Paralysis. Washington University, Report COO-1493-21. St. Louis (1969)

---

## Author Index

### A

- Abadie, J., 15, 569, 579, 597  
Adjiman, C.S., 722, 723  
Al-Baali, M., 204, 206, 208, 253, 260, 278, 280, 287  
Alguacil, N., 8  
Alonso, A.A., 749  
Andersen, E.D., 474, 642  
Andersen, K.D., 474, 642  
Andreani, R., 518  
Andrei, N., 1, 3, 6, 9, 13, 15–17, 19, 22, 68, 75, 77, 78, 80, 88, 92, 95, 97, 98, 100, 101, 104, 106, 107, 133, 135, 146, 157, 163, 165, 169, 181, 183, 184, 213, 214, 217–220, 222, 225, 229, 231, 235–238, 241, 245, 250, 253, 256, 259, 260, 267, 272, 278, 280–283, 288–290, 302, 304, 313, 314, 355, 359, 365, 366, 369, 370, 378, 380, 382, 383, 420, 430, 452, 461, 473, 475, 488, 494, 501, 502, 506, 510, 516, 519, 567, 576, 577, 579, 593, 597, 611, 615, 621, 624, 625, 644, 679, 721–729, 731, 732, 735, 741, 756, 764, 765  
Antoniou, A., 80  
Aris, R., 120, 762  
Armijo, L., 38, 68, 380  
Arrow, K.J., 511  
Arzam, M.R., 278  
Averick, B.M., 13, 120, 121, 430, 723, 724, 757, 759, 761, 764  
Axelsson, O., 181

### B

- Babaie-Kafaki, S., 107, 220, 278, 285, 286  
Babuska, I.M., 184  
Bakhtiari, S., 661  
Balakrishnan, V., 382  
Ballard, D.H., 729  
Bang, J.R., 749  
Bartels, R.H., 543  
Bartholomew-Biggs, M.C., 9, 15, 19, 22, 80, 382, 444, 454, 455, 501, 719, 731  
Bartlett, R., 567  
Barzilai, J., 53, 280, 416, 417  
Bau III, D., 719  
Bazaraa, M.S., 9, 19, 22, 382, 383, 409, 716, 719  
Bărbulescu, M., 473

- Beale, E.M.L., 246, 260, 450  
Bebernes, J., 762  
Benedek, R., 764  
Benson, H.Y., 302, 659, 661  
Ben-Tal, A., 11, 490, 492, 499, 519  
Berger, M., 314  
Bergman, L., 557  
Bertsekas, D.P., 9, 19, 22, 382, 383, 409, 411, 415–417, 419, 484, 485, 490, 516  
Betts, J., 642  
Bicanic, N., 168  
Biegler, L.T., 18, 299, 314, 437, 544, 567, 606, 633, 642, 645, 661, 663–667, 669, 671–673, 678, 749  
Biggs, M.C., 278–280, 282, 544  
Birge, J.R., 11  
Birgin, E., 16, 378, 411, 416, 417, 419, 437, 494, 518  
Bisschop, J., 6, 8  
Boggs, P.T., 313, 375, 566  
Bondarenko, A.S., 737  
Bongartz, I., 5  
Bonnans, J., 556, 567  
Bortz, D.M., 737  
Borwein, J.M., 53, 280, 416, 417  
Boyd, I.D., 659  
Boyd, S., 19, 382, 383  
Breitfeld, M.G., 488, 490, 493, 495, 497, 499, 614  
Brent, R.P., 369  
Brooke, A., 6, 145  
Brown, A.A., 15, 501  
Broyden, C.G., 264, 273  
Buckley, A.G., 314  
Bulirsch, R., 69  
Bunch, J.R., 160, 162, 450, 604  
Byatt, D., 357  
Byrd, R.H., 16–18, 269–271, 278, 279, 282, 285, 287, 297, 299, 300, 302, 309, 312–314, 330, 340, 353, 374, 378, 411, 416, 417, 421–424, 426, 427, 437, 494, 542, 557, 558, 561, 563, 566, 631, 633–635, 637, 639, 642, 643, 661–663

### C

- Carolan, W.J., 510  
Cauchy, A., 9, 58  
Carpentier, J., 569, 579, 597

- Carter, R.G., 13, 120, 121, 430, 723, 724, 757, 759, 761, 764, 765  
 Castillo, E., 8  
 Cesari, L., 736  
 Chachuat, B.C., 383, 409  
 Chan, S.C., 302  
 Chen, J., 220  
 Chen, H., 260, 278, 285, 302  
 Chen, L.P., 219, 220  
 Chen, X., 50  
 Cheng, S.H., 604  
 Cheng, W.Y., 280, 282  
 Chin, C.M., 374, 648  
 Choi, T.D., 369  
 Chow, T.T., 15  
 Cimatti, G., 759  
 Cohen, A., 202  
 Coleman, T.F., 415, 416, 536, 566  
 Colombani, Y., 6  
 Concus, P., 327  
 Conejo, A.J., 8  
 Conn, A.R., 5, 9, 17, 299, 300, 302, 311, 314, 330, 353, 369, 370, 378, 379, 382, 411, 416, 421, 437, 499, 500, 518, 519, 536, 543, 566, 643, 644  
 Contreras, M., 278  
 Cools, R., 724  
 Coope, D., 357  
 Cottle, R.W., 409  
 Courant, R., 516  
 Crowder, H.P., 202  
 Cryer, C.W., 759  
 Curtis, A., 712  
 Curtis, F.E., 330  
 Cuthrell, J.E., 749
- D**  
 Dai, X., 220  
 Dai, Y.H., 15, 46, 49, 54, 172, 188, 192, 195, 196, 198, 199, 201, 202, 208, 210, 218–220, 232, 245, 253–255, 259, 260, 287, 304  
 Dai, Z., 219, 220  
 Daniel, J.W., 201, 459  
 Dantzig, G.B., 1, 9, 504  
 Davidon, W.C., 14, 80, 264, 313  
 Davies, D., 141  
 Debreu, G., 487  
 Dehmiry, A.H., 278, 287  
 Dembo, R.S., 315, 322, 415, 428, 728, 731, 732  
 Demmel, J.W., 704, 719  
 Denchfield, A., 106  
 Dener, A., 106  
 Deng, N.Y., 260, 278, 285, 556  
 Dennis, J.E., 9, 68, 75, 80, 165, 273, 288, 313, 353, 382, 416, 417, 615, 622, 712, 719  
 Deuflhard, P., 315  
 Dikin, I.I., 9, 644  
 Dixon, L.C.W., 330  
 Dolan, E.D., 13, 78, 420, 502, 735–737
- Drud, A., 17, 567, 590, 591, 593, 594, 597  
 Duran, M., 651
- E**  
 Eberly, D., 762  
 Ecker, J.G., 15  
 Eisenstat, S.C., 315, 322, 330  
 El-Bakry, A.S., 611, 614, 622, 661  
 Eldersveld, S.K., 544, 546, 642  
 El Ghaoui, L., 11, 382  
 Elman, H.C., 184  
 Esposito, W.R., 722, 723
- F**  
 Facchinei, F., 141, 411, 416, 437  
 Feron, E., 382  
 Fiacco, A.V., 141, 157, 159, 300, 376, 490, 599, 601, 645, 685  
 Finkel, D.E., 370  
 Fletcher, R., 12, 18, 68, 80, 141, 157, 160, 171, 201, 208, 249, 264, 272, 309, 313, 314, 353, 374, 375, 379, 381, 450, 453, 454, 474, 482, 526, 536, 544, 566, 567, 599, 610, 647, 648, 650, 651, 653–659, 661, 678  
 Floudas, C.A., 10, 519, 722, 723  
 Fogelson, A., 9, 326, 330, 416  
 Ford, J.A., 260, 286  
 Forsgren, A., 141, 376, 603, 644  
 Forsythe, G.E., 169  
 Fourer, R., 6, 8, 146  
 Fox, R.L., 579  
 Frank, M., 15, 570  
 Frank, P.D., 15, 642  
 Freeman, T.L., 141, 157, 160  
 Friedlander, A., 416  
 Friedlander, M.P., 378  
 Frisch, K.R., 490, 599, 645  
 Fukushima, M., 285, 287, 536
- G**  
 Gabay, D., 566  
 Gabriele, G.A., 579  
 García-Bertrand, R., 8  
 Garner, J., 764  
 Gay, M., 6, 146, 352, 510, 611  
 Ge, R.-P., 313  
 George, A., 605  
 Ghanbari, R., 220, 278, 286  
 Gilbert, J., 201, 213, 219–221, 254, 288, 292, 312, 314, 567, 631, 634, 661  
 Gill, P.E., 9, 12, 17, 22, 80, 141, 143, 144, 157, 159, 278, 288, 309, 314, 327, 375, 376, 382, 383, 429, 453, 459, 474, 507, 519, 542, 544, 546, 549–551, 566, 567, 603, 643, 644, 699, 700, 712  
 Glowinski, R., 757  
 Gödel, K., 3  
 Goldfarb, D., 141, 264, 450, 456–459, 593, 611  
 Goldfeld, S.M., 141

- Goldstein, A.A., 40, 68, 141  
Golub, G., 186, 327, 459, 467, 562, 704, 719  
Goncalves, A.S., 450  
Goodman, J., 759, 760  
Gould, N.I.M., 5, 9, 17, 80, 299, 300, 302, 311, 314, 330,  
    353, 374, 378, 379, 382, 411, 416, 417, 421, 437,  
    450, 474, 487, 499, 500, 518, 519, 542, 557, 558,  
    561–563, 566, 643, 644, 659  
Graggs, W.B., 459  
Grandinetti, L., 278, 287  
Greenbaum, A., 181, 184  
Griewank, A., 12, 314  
Griffith, R.E., 15, 373  
Grippo, L., 48, 54, 416, 417, 556  
Griva, I., 9, 22, 54, 80, 333, 409, 719  
Grothey, A., 314  
Grossmann, I.E., 651  
Grötschel, M., 19  
Gu, N.Z., 50  
Guéret, C., 474  
Guerrero, G., 15  
Guigou, J., 579  
Gümüs, Z.H., 722, 723  
Guo, Q., 286
- H**  
Hager, W.W., 15, 44, 46, 49, 50, 54, 68, 78, 208, 231, 233,  
    234, 238, 259, 260, 411, 415, 417, 437  
Haggag, A., 579  
Han, J.Y., 210, 220, 234  
Han, L., 219, 220  
Han, S.P., 479, 480, 524, 528, 544, 548, 566  
Han, X., 220  
Harding, S.T., 722, 723  
Heinkenschlos, M., 416  
Heipcke, S., 6  
Hellerman, E., 509, 591  
Hendrickson, B., 605  
Heng-Bin, A., 330  
Herskovits, J., 566  
Hestenes, M.R., 9, 169, 171, 172, 201, 280, 482, 511, 517  
Hext, G.R., 369  
Higham, N.J., 604  
Hill, J.E., 510  
Himmelblau, D.M., 723  
Himsworth, F.R., 369  
Hiratsuka, S., 754  
Hock, W., 624, 727–729, 731  
Hodgkin, B.C., 724  
Hooke, R., 369  
Horst, R., 10  
Hough, P., 370  
Hribar, M.E., 18, 299, 314, 353, 562, 563, 631, 634, 637,  
    642, 661  
Hu, C.M., 278, 286  
Hu, W., 278, 287  
Hu, Y.F., 218, 219, 221, 222  
Huang, H., 219, 220  
Huang, S., 50, 278, 286
- Hulbert, L.A., 415  
Hürlimann, T., 6, 8
- I**  
Idnani, A., 450, 456–459  
Iriaray, R., 754
- J**  
Jeeves, T.A., 369  
Jelinek, C.A., 729  
Jensen, D.L., 499  
Jain, A., 579  
Jian, J., 219, 220  
Jiang, X., 219, 220  
Jiao, B.C., 219, 220  
Jittorntrum, K., 490  
John, F., 409  
Jones, D., 370  
Johnson, K., 168  
Júdice, J., 416
- K**  
Kabadi, S.N., 474  
Kalan, J.E., 591  
Kall, P., 11  
Kallrath, J., 6  
Karmarkar, N., 9, 382, 644  
Karush, W., 409  
Kaufman, L., 162, 450, 459, 604  
Kearfott, R., 722  
Keller, C., 562  
Keller, H., 122  
Kelley, C.T., 80, 155, 168, 181, 300, 312, 369, 382, 411,  
    413, 414, 437, 721  
Kendrick, D., 6, 145  
Kennington, J.L., 510  
Kernighan, B.W., 6, 146  
Khalfan, H.F., 299, 300, 302  
Kim, K., 9  
Klepeis, J.L., 722, 723  
Kocvara, M., 519  
Kohn, R., 759, 760  
Kolda, T., 355, 370, 686  
Kollerstrom, N., 168  
Kortanek, K.O., 611  
Kou, C.X., 15, 46, 245, 253–255, 259, 260  
Kreuser, J., 503, 518  
Kristjansson, B., 6  
Kuhn, H.W., 409  
Kungurtsev, V., 141  
Kupferschmid, M., 15
- L**  
Lagarias, J.C., 357  
Lalee, M., 539, 544  
Lam, W.H., 302  
Lampariello, F., 48, 416, 417, 556  
Lanczos, C., 169  
Lapidus, L., 738, 754

- Larrosa, J.A.E., 750  
 Larson, J., 22, 355, 689  
 Lasdon, L.S., 579  
 Lawrence, C.T., 661  
 Le Dimet, F.X., 314  
 Legler, D.M., 259  
 Lemaréchal, C., 68, 80, 288, 292, 312, 314, 567, 659  
 Lemke, C.E., 450  
 Lenir, A., 314  
 Lenstra, J.K., 18  
 Leonard, M.W., 278, 314  
 Lescrenier, M., 416  
 Levenberg, K., 141, 352  
 Levitin, E.S., 415  
 Lewis, J.G., 642  
 Lewis, R.M., 355, 370, 686  
 Leyffer, S., 18, 314, 353, 371, 375, 379–382, 482, 544, 566, 567, 599, 610, 647, 648, 650, 651, 654–659, 661, 678  
 Li, C., 278, 287  
 Li, D.H., 202, 280, 282, 285, 287  
 Li, G., 219, 220, 283, 285  
 Li, H., 220  
 Li, S.J., 227  
 Li, Y., 416  
 Liao, A., 278, 281, 282  
 Liao, L.Z., 54, 172, 202, 232, 304  
 Lian, Z., 278, 283  
 Liang, H., 278  
 Lin, C.-J., 16, 107, 351, 352, 411, 416, 437, 759  
 Lindskog, G., 181  
 Liu, D.C., 9, 279, 292, 294, 312, 314, 327, 556, 608  
 Liu, G., 662, 663  
 Liu, G.H., 210, 220, 234  
 Liu, H.W., 54, 107  
 Liu, J.G., 286  
 Liu, J.K., 227  
 Liu, J.W.-H., 605  
 Liu, L., 219, 220  
 Liu, S., 611  
 Liu, Y., 51, 201, 214  
 Liu, Y.A., 754  
 Liu, Z., 107  
 Liu, Z.X., 54  
 Liuzzi, G., 18, 680, 684–688  
 Livieris, I.E., 253  
 Louveaux, F., 11  
 Lu, P., 16, 299, 314, 378, 411, 416, 417, 421, 424, 426, 427, 437, 494  
 Lu, X., 278, 287  
 Lucia, A., 309  
 Lucidi, S., 18, 48, 141, 411, 416, 417, 437, 556, 680, 684–688  
 Luenberger, D.G., 9, 19, 22, 253, 278, 280, 281, 382, 383, 394, 409, 576, 577  
 Luo, Z., 220  
 Lustig, I.J., 9, 613  
 Luus, R., 738, 749
- M**  
 Mahajan, A., 371, 375, 380, 382  
 Mangasarian, O.L., 9, 409, 479, 480  
 Manne, A.S., 557  
 Maratos, N., 379, 534  
 Marazzi, M., 370, 633  
 Markowitz, H.M., 474, 519, 591  
 Marsten, R.E., 9, 613  
 Marquardt, D.W., 141, 352  
 Martínez, H.J., 16, 278, 378, 411, 416, 417, 419, 437, 494, 518  
 Matthews, A., 141  
 Mayne, D.Q., 536, 566  
 McCormick, G.P., 141, 157, 159, 202, 300, 376, 490, 599, 601, 645, 685  
 McKinney, D.C., 740, 744, 745, 748  
 Mead, R., 9, 16, 355, 369, 679  
 Meeraus, A., 6, 8, 145  
 Megiddo, N., 558, 639  
 Mehrotra, S., 165, 465  
 Meintjes, K., 723  
 Menchi, O., 759  
 Menickelly, M., 22, 355, 689  
 Mészáros, C., 605  
 Meyer, C.A., 722, 723  
 Meyer, C.D., 719  
 Minguez, R., 8  
 Mizuno, S., 9  
 Moghrabi, I.A., 286  
 Mo, J.T., 50  
 Morales, J.L., 18, 631, 633, 642  
 Moré, J.J., 9, 13, 16, 68, 75, 78, 120, 121, 141, 273, 292, 313, 327, 333, 344, 351–353, 378, 411, 415, 416, 420, 421, 430, 437, 502, 699, 723, 724, 735–737, 757, 759, 761, 764, 765  
 Morgan, A.P., 723, 724  
 Mu, C., 141  
 Munson, T., 106, 502, 735–737  
 Murray, W., 9, 12, 17, 22, 80, 141, 143, 144, 157, 159, 288, 309, 314, 327, 375, 383, 429, 453, 459, 474, 507, 519, 542, 544, 546, 549–551, 566, 567, 643, 699, 700, 712  
 Murtagh, B.A., 17, 378, 475, 488, 503–505, 509, 511, 512, 514, 515, 518, 519, 544, 557, 643  
 Murty, K.G., 474
- N**  
 Naiman, A.E., 184  
 Narushima, Y., 54, 260, 286  
 Nash, S.G., 9, 15, 16, 22, 54, 80, 255, 315, 326, 327, 329, 330, 333, 378, 409, 416, 417, 428–430, 437, 490, 494, 645, 719  
 Navon, I.M., 259, 314  
 Nazareth, J.L., 9, 260, 288  
 Nelder, J.A., 9, 16, 355, 369, 679  
 Nelson, C.V., 724  
 Nemirovskii, A., 11, 88, 376, 377, 659  
 Nemhauser, G.L., 10

- Nesterov, Y., 377, 659  
Ng, E., 605  
Niemi, S., 510  
Nishida, N., 754  
Ni, Q., 219, 416  
Nitsche, J.C.C., 764  
Nocedal, J., 9, 16–19, 22, 75, 80, 195, 201, 204, 213, 219–  
221, 253–255, 259, 269–271, 278–282, 287, 291,  
292, 294, 297, 299, 309, 312–314, 323, 325, 327,  
330, 331, 343, 344, 346, 349, 351–353, 370, 374,  
378, 381–383, 409, 411, 416, 417, 421–424, 426,  
427, 437, 448, 449, 452, 463, 465, 474, 477, 479,  
482, 483, 494, 516, 534, 539, 542, 544, 556–558,  
561–563, 566, 599, 603, 605, 606, 608, 627, 629,  
631–635, 637, 639, 642–644, 661–663, 685, 719  
Novoa, M., 722
- O**  
Ogasawara, H., 278  
O’Leary, D.P., 327  
Omojokun, E., 381, 539  
O’Neill, R., 366, 367, 370  
Orban, D., 18, 450, 518, 566, 631, 633, 642, 644  
Oren, S.S., 253, 278, 280, 281, 327  
Ortega, J.M., 164  
Osborne, M., 490  
Ou, Y., 51, 107  
Overton, M.L., 80, 611
- P**  
Palagi, L., 416, 437  
Pan, C.Y., 219, 220  
Panier, E.R., 556  
Pant, M., 462  
Papadimitriou, C.H., 10  
Pardalos, P.M., 10, 722, 723  
Parlett, B.N., 160, 604  
Pedregal, P., 8  
Peressini, A.L., 382, 383, 719  
Perry, A., 233, 244, 245, 248, 252, 259  
Perttunen, C., 370  
Peyret, R., 748  
Peyton, B.W., 605  
Phua, K.H., 75, 234, 259, 278, 314  
Pintelas, P., 253  
Plantenga, T., 539, 544  
Polak, E., 171, 201, 210, 536, 566  
Polyak, R., 171, 201, 210, 414, 415, 490, 499  
Potra, F.A., 68, 168, 611  
Powell, M.J.D., 16, 18, 68, 80, 201, 202, 204, 210, 213,  
232, 234, 247, 259, 260, 264, 270, 279, 313, 352,  
353, 355, 358, 369, 370, 417, 461, 482, 500, 511,  
517, 524, 528, 534, 544, 547, 555, 566, 679, 680,  
684, 712  
Price, C.J., 141, 357  
Price, R.C., 330  
Price, W.L., 729  
Prins, C., 474  
Pulkkinen, S., 80
- Q**  
Qi, L., 219, 220, 283, 285  
Quandt, R.E., 141
- R**  
Radcliffe, N.J., 659  
Ragsdell, K.M., 579, 724  
Raman, R., 6, 145  
Rarick, D., 509, 591  
Ratner, M.W., 579  
Ratschek, H., 725  
Raydan, M., 16, 49, 54, 93, 95, 378, 411, 416, 417, 419,  
437, 494, 556  
Reeds, J.A., 357  
Reeves, C.M., 171, 201  
Reid, J.K., 591, 593, 712  
Reyna, L., 759, 760  
Rezaee, S., 107  
Rheinboldt, W.C., 164, 168  
Ribière, G., 171, 201, 210  
Rijckaert, M.J., 728  
Rinnooy Kan, A.H.G., 18  
Rios, L.M., 22, 355, 370, 689  
Ritter, K., 202  
Robinson, D.P., 141, 374  
Robinson, S.M., 503, 511, 518, 525  
Rockafellar, R.T., 11  
Roelofs, M., 6  
Rokne, J., 725  
Rose, D.J., 605  
Rosen, J.B., 10, 15, 415, 503, 518, 569, 597  
Rosenbrock, H.H., 369  
Rosenthal, R.E., 6, 145  
Rosser, J.B., 169  
Rothberg, E., 605
- S**  
Sachs, E.W., 300  
Sagastizábal, C., 567  
Sainvitu, C., 417  
Sainz de la Maza, E., 374, 566  
Sandgren, E., 724, 726  
Santos, S.A., 416  
Sargent, E., 511  
Sartenaer, A., 518  
Saunders, M.A., 17, 309, 375, 378, 459, 474, 475, 488,  
503–505, 509, 511, 512, 514–516, 518, 519, 542,  
544, 546, 549–551, 557, 566, 567, 643  
Savitsky, A.G., 740, 744, 745, 748  
Scheinberg, K., 369, 370  
Schinzingher, R., 729  
Schittkowski, K., 17, 54, 461, 500, 533, 544, 553–555,  
566, 624, 724, 728–731  
Schlick, T., 9, 314, 326, 330, 416  
Schmid, C., 544  
Schnabel, R.B., 9, 15, 68, 75, 80, 297, 299, 300, 302, 309,  
312, 313, 340, 353, 382, 417, 421–423, 566, 615,  
622, 712, 719  
Schrage, L., 6

- Schreiber, R., 122, 704, 719  
 Schrijver, A., 18  
 Schultz, G.A., 340, 353  
 Schuverdt, M., 518  
 Schweiger, C.A., 722, 723  
 Sciandrone, M., 18, 54, 680, 684–688  
 Sevaux, M., 474  
 Shacham, M., 722  
 Shainidis, N.V., 22, 355, 370, 698  
 Shanno, D.F., 9, 68, 75, 233, 234, 244–247, 249, 250, 252,  
     259, 260, 264, 277, 278, 302, 307, 488, 490, 493,  
     495, 497, 499, 611, 613, 614, 632, 633, 642, 645,  
     659, 661  
 Shen, C., 659  
 Shen, J., 80  
 Shen, X.L., 278, 286  
 Sheng, Z., 278, 287  
 Shi, Y., 68  
 Shi, Z.-J., 63, 80  
 Sherali, H.D., 9, 19, 22, 382, 383, 409, 716, 719  
 Shetty, C.M., 9, 19, 22, 382, 383, 409, 716, 719  
 Shultz, G.A., 566  
 Simantiraki, E.M., 614  
 Singh, R.P., 749  
 Singh, V.P., 462  
 Soares, J., 416  
 Sofer, A., 9, 22, 54, 80, 333, 409, 490, 719  
 Sollow, R.M., 511  
 Sorensen, D.C., 141, 344, 352, 699  
 Spall, J.C., 11  
 Spedicato, E., 278, 309, 327  
 Spellucci, P., 474, 544  
 Spendley, W., 369  
 Steiglitz, K., 10  
 Steihaug, T., 315, 322, 340, 429  
 Stewart, G.W., 154, 459  
 Stewart, R.A., 15, 373  
 Stiefel, E., 169, 171, 172, 201, 280  
 Stingl, M., 519  
 Stoer, J., 69, 238  
 Storey, C., 201, 214, 218, 219, 221, 222  
 Strakoš, Z., 184  
 Strogatz, S.H., 755  
 Stuckman, B., 370  
 Suhl, L.M., 591  
 Suhl, U.H., 591  
 Sullivan, F.E., 382, 383, 719  
 Sun, D.F., 210, 220  
 Sun, W., 9, 19, 22, 80, 107, 141, 157, 168, 187, 382, 383,  
     409, 411, 474, 603  
 Surry, P.D., 659  
 Svaiter, B.F., 93, 95
- T**  
 Tampakas, V., 253  
 Tapia, R.A., 165, 278, 548, 566, 611, 614, 622, 661  
 Tarjan, R.E., 605  
 Taylor, T.D., 740  
 Teo, K.L., 278, 286
- Thangaraj, R., 462  
 Thoai, N.V., 10  
 Thuente, D.J., 68, 75, 292, 327  
 Tits, A.L., 556, 661  
 Todd, M.J., 9, 376  
 Toint, Ph.L., 5, 9, 17, 299, 300, 302, 309, 311, 314, 330,  
     353, 370, 378, 379, 382, 411, 416, 417, 421, 437,  
     450, 474, 500, 518, 519, 556, 566, 643, 644, 648,  
     651, 659, 712  
 Tolle, J.W., 313, 375, 415, 566  
 Topkis, D.M., 571  
 Toraldo, G., 378, 415, 421, 759  
 Torczon, V., 355, 370, 686  
 Touati-Ahmed, D., 218, 219, 222  
 Tsuchiya, T., 611, 614, 622, 661  
 Traub, J., 164  
 Trefethen, L.N., 704, 719  
 Tröltzscher, A., 437  
 Trosset, M.W., 370  
 Trotter, H.F., 141  
 Tucker, A.W., 409  
 Tulowitzki, U., 415
- U**  
 Uhl, J.J., 382, 383, 719  
 Ulbrich, M., 416, 644, 645, 659, 661  
 Ulbrich, S., 416, 644, 645, 659, 661  
 Urban, T.J., 661
- V**  
 Vandenberghe, L., 19, 383  
 Vanderbei, R.J., 1, 9, 474, 504, 611, 613, 614, 632, 633,  
     642, 645, 659, 661  
 Vanderplaats, G.N., 437  
 Van der Pol, B., 755  
 Van der Vorst, H.A., 181  
 Van Loan, C.F., 186, 467, 562, 704, 719  
 Vardi, A., 375  
 Vassiliadis, V.S., 519  
 Vavasis, S.A., 9, 474  
 Veinott, A.F., 571  
 Verlinden, P., 724  
 Verschelde, J., 724  
 Vicente, L.N., 416, 644, 645, 659, 661  
 Von Stryk, O., 737
- W**  
 Wächter, A., 18, 299, 314, 437, 606, 632, 633, 642, 644,  
     645, 659, 661, 663–667, 669, 671–673, 678  
 Wakamatsu, T., 54  
 Walker, H.F., 330  
 Wallace, S.W., 11  
 Walsh, G.R., 80  
 Waltz, R.A., 17, 18, 374, 542, 557, 558, 561, 563, 566,  
     631–633, 635, 638, 639, 642, 643  
 Wan, Z., 50, 278, 286  
 Wang, B., 278, 287  
 Wang, D.H., 286  
 Wang, H.J., 279

- Wang, S., 611  
Waren, A.D., 579  
Wathen, A.J., 562  
Wei, Z., 219, 220, 278, 283, 285, 287  
Wen, F., 219, 220  
Wen, S., 278  
Wild, S.M., 22, 355, 689  
Wilkinson, J.H., 278  
Wilson, J.M., 8, 566  
Wilson, R.B., 524  
Wochmann, S.J., 510  
Wolfe, P., 15, 41, 68, 189, 202, 407, 504, 569, 570, 576, 597  
Wolkowicz, H., 288  
Wolsey, L.A., 10  
Wright, J., 141  
Wright, M.H., 9, 12, 22, 80, 309, 357, 369, 376, 377, 382, 383, 474, 514, 515, 519, 533, 544, 546, 551, 567, 603, 611, 614, 644, 699, 700, 712  
Wright, P.E., 357  
Wright, S.J., 9, 19, 22, 75, 80, 204, 291, 309, 323, 325, 331, 343, 344, 346, 349, 351, 352, 381–383, 409, 411, 415, 448, 449, 452, 463, 465, 474, 477, 479, 482, 483, 516, 534, 539, 566, 599, 603, 605, 606, 608, 617, 627, 629, 644, 685, 719  
Wu, G., 278  
Wu-Sheng, L., 80
- X**  
Xavier, A.E., 519  
Xiao, Y., 556  
Xing-Ping, L., 330  
Xu, C., 278, 280  
Xue, G.L., 13, 120, 121, 430, 723, 724, 757, 759, 761, 764, 765
- Y**  
Yabe, H., 54, 260, 278, 286  
Yamashita, H., 309, 661
- Yang, E.K., 415  
Yang, X., 220  
Yao, S., 219, 220  
Ye, Y., 9, 19, 382, 409, 576, 577, 611, 614  
Yin, H.X., 210, 220, 234  
Yoshino, M., 278  
Ypma, T.J., 168  
Yu, G., 278, 283, 287  
Yuan, G., 278, 283, 287  
Yuan, Y.X., 9, 19, 22, 80, 141, 157, 168, 187, 188, 192, 198, 201, 202, 208, 210, 218–220, 238, 253, 260, 269–271, 278–282, 285, 313, 352, 382, 383, 409, 411, 416, 474, 603  
Yudin, D.B., 88  
Yuzefovich, I., 490, 492
- Z**  
Ze-Yao, M., 330  
Zhang, H., 15, 44, 46, 49, 50, 54, 68, 78, 208, 231, 233, 234, 238, 259, 260, 411, 415, 417, 437  
Zhang, J., 220, 260, 278, 280, 285  
Zhang, L., 220  
Zhang, Y., 165, 416, 566, 611, 614, 622, 661  
Zangwill, W.I., 15, 569, 578, 597  
Zhao, J., 309  
Zheng, X.D., 278, 286  
Zhou, C., 141  
Zhou, F.J., 556  
Zhou, J.L., 556  
Zhou, W., 220  
Zhu, C., 16, 299, 314, 378, 411, 416, 417, 421, 424, 426, 427, 437  
Zhu, H., 278  
Zhu, M., 288  
Zibulevsky, M., 490, 492, 499, 519  
Zou, X., 314  
Zoutendijk, G., 15, 57, 58, 189, 203, 569, 570, 597  
Zwart, P.B., 141

---

# Subject Index

## A

### Active

- constraint, 390
- set, 390
- methods, 571

Accelerated steepest descent algorithm, 98

Acceptable line-search, 30

Accumulation point, 707

Accuracy of algorithms, 12

A-conjugate directions, 170

Advantages of the Newton method, 140

Algebraic characterization of a

- descent direction, 387
- feasible direction, 390
- tangent space, 395

Algebraic modeling languages, 6

ALGENCAN, 518

Angle between two nonzero vectors, 692

Angle condition, 48, 54

A-orthogonal directions, 170

Approximate Wolfe line-search, 44

Armijo line-search, 38

Augmented Lagrange function, 482

Augmented Lagrangian methods, 377, 482

Automatic differentiation, 712

## B

Backtracking, 46, 47, 68

Balance between function and constraints, 394

Banach lemma, 695

Barzilai-Borwein line-search, 53

Barrier method, 600

Basic assumptions (conjugate gradient), 189

Basic matrix (GRG), 579

Basis, 691

Binding

- constraint, 404
- simple bound constraint, 404

Biobjective optimization problem, 648

Bolzano-Weierstrass theorem, 707

Bound-constrained Lagrangian methods, 378

Bounded deterioration property, 273

Bounded set, 715

Bracketing phase in line-search, 68

Bratu's problem, 120

Broyden class, 267

Broyden-Fletcher-Goldfarb-Shanno (BFGS), 261, 264, 265

memoryless, 277

with modified line-search, 286

with modified secant equation, 283

## C

Cauchy point, 337, 469

- quadratic programming simple bounds, 460

- sequence, 707

- trust-region, 337

Cauchy-Schwarz inequality, 692

Cautious BFGS, 285

Cayley-Hamilton theorem, 705

Central finite-difference formula, 147

Central path, 464

Centroid, 356

CD-method (conjugate gradient) (Fletcher), 208

CG-DESCENT (conjugate gradient), 233, 252

Chance-constrained optimization, 11

Chebyshev polynomials, 179

Circuit design, 120, 150, 166, 725

Circularity danger, 3

Cholesky factorization, 143, 698

- modified, 699

Closed set, 715

COBYLA (direct search), 680

Combinatorial difficulty, 392

Compact representation of the L-BFGS updating, 296, 422

Compactifying matrix, 310

Compact set, 715

Comparison of TN versus

- conjugate gradient algorithms, 325

- L-BFGS, 327

Complementarity

- measure, 464

- slackness, 391

Complexity analysis, 12

Composite Newton method

- of order 1, 164

- of order m, 164

- Computational sciences, 4  
 Computer science, 4  
 Conditioning and stability, 704  
 Condition number, 704  
 Conjugate directions, 170  
 Conjugate gradient methods memoryless BFGS  
     preconditioned, 244  
 CONMIN (conjugate gradient), 245, 249  
 CONOPT, 567, 587, 589  
     linear mode, 593  
     nonlinear mode, 593  
 Consistency of quasi-Newton method, 273  
 Constrained  
     optimization, 9  
     qualification, 392  
 Constraint, 1  
     normal, 397  
 Continuous optimization, 9  
 Convergence from remote starting points, 377  
 Convergence of the MM-SR1gen method, 305  
 Convex  
     function, 10, 716  
     nonconvex, 439  
     program, 386  
     programming, 11  
     quadratic program, 439  
         equality constraints, 467  
     set, 10, 716  
     strictly convex, 439  
 Convex simplex method (Zangwill), 578  
 Criticism of the penalty and augmented Lagrangian  
     methods, 486  
 Crossover technique, 638  
 Curvature condition (line-search), 42, 262
- D**
- Dai and Kou line-search, 46  
 Dai-Liao conjugate gradient method, 232  
 Damped Newton method, 129  
 Day-Yuan method (conjugate gradient), 208  
 Debreu theorem, 488  
 DEEPS algorithm (direct search), 359, 361  
     reduction phase, 363  
     stalling phase, 363  
 DFG (direct search), 684  
 Degeneracy, 404  
 Dennis-Moré condition, 274  
 Departure from linearity, 503  
 Derivative-free  
     constrained optimization, 679  
     unconstrained optimization, 355  
 DFP (Davidon-Fletcher-Powell), 261, 263, 264  
 DK/CGOPT (conjugate gradient), 251, 253, 254  
 Descent  
     condition, 29  
     direction, 22, 188, 387  
 DESCON (conjugate gradient), 236  
     accelerated, 240  
 Determinant of a matrix, 704  
 Determinant of different BFGS updatings, 706  
 Deterministic optimization, 11  
 Diagonal updating of the Hessian, 287  
 Differentiable manifold, 394  
 Dilemma, 5  
 Direction of negative curvature, 157  
 Direct search method  
     constrained optimization, 679  
     unconstrained optimization, 355  
 Disadvantages of the Newton method, 140  
 Discarding the inactive constraints, 402  
 Discrete optimization, 9  
 Dogleg method, 338  
 Dual  
     algorithm for quadratic programming, 457  
     feasibility, 391  
     problem, 406  
 Duality, 406  
 Dynamic restart strategy, 255  
 DYNAMIC (quadratic programming application), 462
- E**
- Efficiency of algorithms, 12, 14, 78  
 Eigenvalues, 697  
 Eigenvalues in tangent space, 399  
 Elastic-plastic torsion application, 294, 430, 757  
 Elastic programming, 543  
 Element functions, 310  
 Elements of  
     analysis, 707  
     applied numerical linear algebra, 691  
     convexity, 716  
     topology in the Euclidian space, 715  
 Ellipsoid norm, 693  
 Errors in functions, gradients and hessians, 154  
 Euclidean norm, 692  
 ETA-MACRO application (NLPQLP), 557  
 Exact line-search, 30, 37  
 Expansion (Nelder-Mead), 356
- F**
- Factorization of the full KKT system (quadratic  
     programming), 440  
 Farkas theorem, 718  
 Feasible  
     direction, 389  
     method (Zoutendijk), 569  
     domain, 1, 383  
     interior-point methods, 609  
     solution, 383  
 Filter  
     algorithm  
         sequential linear programming, 649, 650  
         sequential quadratic programming, 658  
     envelope, 648  
     methods, 379, 647  
         with line search, 665  
 Finite-difference approximation, 146, 710  
 Finite termination property, 175

- First-order  
     necessary conditions, 396  
     optimality conditions, 2, 25, 387  
     sufficient conditions, 388
- First-and second-order necessary conditions, 403
- Fletcher-Reeves (conjugate gradient), 171, 202
- Flow in a drive cavity, 121
- Forcing sequence, 315, 316, 322
- Forward finite-difference formula, 146
- Fraction to the boundary rule, 602, 665
- Frank-Wolfe Method, 570
- Frobenius norm, 693, 700
- Function  
     bounded, 707  
     continuous, 707  
     first derivative, 708  
     uniformly continuous, 707  
     second derivative, 708
- Fundamental  
     ratio, 88  
     theorem of calculus, 23  
     theorem of linear algebra, 694
- G**
- GALAHAD, 518
- Gaussian Elimination (LU Factorization), 697  
     with complete pivoting, 698  
     with partial pivoting, 698
- Gauss-Newton method, 124
- General barrier, 489
- General nonlinear  
     conjugate gradient algorithm, 202  
     memoryless BFGS preconditioned, 244  
     optimization, 1
- General penalty-barrier function (SPENBAR), 490
- Generalized  
     Cauchy point, 352  
     computation, 426  
     reduced gradient method, 579, 580, 586  
     secant equation, 302  
     Wolfe line-search, 43
- Geometric necessary condition, 389, 395
- Gill and Murray method, 143
- Global  
     convergence analysis, 12  
     convergence of BFGS, 270  
     maximum, 385  
     minimization, 22, 385  
     solution, 10
- Globalization strategy, 381
- Goldfeld, Quandt and Trotter method, 141
- Goldstein and Price method, 142
- Goldstein line-search, 40
- Gordan theorem, 718
- Gradient, 22, 708  
     projected, 415  
         for simple bounds, 417  
         method (Rosen), 573
- Greatest lower bound, 384
- Grippo-Lampariello-Lucidi line-search, 48
- Gu-Mo line-search, 50
- H**
- Hager-Zhang  
     conjugate gradient, 78  
     line-search, 44
- Hessian, 22, 125, 708
- Hestenes-Stiefel method (conjugate gradient), 75, 171, 213
- Hypersurface, 394
- Hölder  
     continuity, 27, 117  
     inequality, 692
- Homotopy methods, 600
- Huang-Wan-Chen line-search, 50
- Hybrid conjugate gradient methods, 219  
     convex combination, 218, 225  
     projection concept, 218
- Hybrid convex combination of LS and DY, 227
- I**
- Inactive constraint, 402
- Indefinite quadratic programming, 450
- Implicit function theorem, 709
- Improved Wolfe line-search, 46, 255
- Incomplete Cholesky factorization, 186
- Inertia and singularity, 605
- Inertia correction and regularization, 606, 671
- Inexact  
     line-search, 30, 38  
     Newton method, 315, 316
- Infeasible  
     constraints, 543  
     point, 372
- Infimum, 383
- Inhomogeneous Superconductors 1-D Ginzburg-Landau  
     application, 296, 436, 764
- Inverse of a matrix, 694
- Inverse of some BFGS updatings, 695
- IPOPT, 662  
     feasibility restoration phase, 672  
     primal-dual barrier approach, 662
- Inside contraction (Nelder-Mead), 356
- Integer programming, 10
- Interior point algorithm  
     line-search, 610  
     prototype, 602
- Interior point methods, 375, 376, 599
- J**
- Jacobian matrix, 27, 109, 124
- Jamming, 204
- K**
- Kantorovich inequality, 84
- KKT matrix (quadratic programming), 440
- KKT point, 391
- KKT necessary conditions, 391
- KKT sufficient conditions, 393, 405

## KNITRO

- ACTIVE (Active-Set Sequential Linear-Quadratic Programming), 557, 559, 560
- crossover algorithm, 639
- INTERIOR-CG (Interior-Point Sequential Linear-Quadratic), 631, 636
- INTERIOR-DIRECT (Interior-Point Sequential Linear-Quadratic), 631, 632, 634

## L

- $l_1$  penalty method, 481
- LACOP collection, 13, 727
  - comparison: MINOS, KNITRO/ACTIVE, SNOPT and CONOPT, 597
  - solution with COBYLA, 684
  - solution with CONOPT, 594
  - solution with DFL, 688
  - solution with filterSD, 652
  - solution with IPOPT, 675
  - solution with KNITRO/ACTIVE, 564
  - solution with KNITRO/INTERIOR, 640
  - solution with MINOS, 516, 517
  - solution with NLPQLP, 557
  - solution with PDIP, 626
  - solution with SNOPT, 551
  - solution with SPENBAR, 502
- Lagrange duality, 408
- Lagrange multipliers, 391, 397
  - interpretation, 400, 401, 405, 406
- LANCELOT, 518
- Large-scale optimization, 11
- L-BFGS, 292
- L-BFGS-B (simple bounds), 421, 427
- L-BFGS-B versus SPG, 427
- Least upper bound, 384
- Levenberg-Marquardt method, 141
- LICO constrained qualification, 393, 397
- Limited-memory quasi-Newton methods, 290
- Line-search
  - algorithm, 172
  - method, 380
  - strategy, 22
- Linear combination, 691
- Linear conjugate gradient algorithm, 177
- Linear constraint qualification, 403
- Linearly constrained Lagrangian methods, 377
- Linear programming, 1, 9, 382
- Linguistic models, 2
- Lipschitz continuity, 26, 111, 715
- Liu-Storey Method (conjugate gradient), 213
- Local
  - affine model, 109
  - convergence analysis, 12
  - minimizer, 21
  - models (constrained optimization), 373, 375
  - solution, 10
- Lower bound, 2, 383, 411

## M

- Mathematical
  - programming, 18
  - model, 3, 7
- Maratos effect, 534
- Powell example, 534
- Matrices, 692
  - elementary, 701
  - determinant, 704
  - identity, 692
  - inverse, 694
  - lower triangular, 693
  - norms, 693
  - pentadiagonal, 693
  - symmetric, 692
  - upper triangular, 693
  - trace, 706
  - transpose, 692
  - tridiagonal, 693
- Matthews and Davies method, 142
- Maximum, 384
- Mean value theorem, 708
- Memoryless
  - BFGS method, 277
  - SR1 method with generalized secant equation, 302
- Mental models, 2
- Merit function, 532, 616, 637
- MFCO constraint qualification, 393
- Minimal surfaces with Enneper boundary conditions
  - application, 296, 762
- Minimum, 384
- MINOS algorithm, 503
  - linear constraints, 504, 508
  - nonlinear constraints, 510, 514
- MINPACK-2 collection, 757
  - solution with CG-DESCENT, 237
  - solution with CONMIN, 251
  - solution with DESCON, 244
  - solution with DK+w, DK+aw, DK+iw, 257
  - solution with hybrid conjugate gradient, 224
  - solution with L-BFGS, 294
  - solution with L-BFGS-B, SPG, TNBC, 430
  - solution with MM-SR1gen and MM-BFGS, 308
  - solution with standard conjugate gradient, 217
  - solution with TRON, 351
- Mixed integer programming, 10
- Modern modeling scheme, 7
- Modified
  - augmented Lagrangian (MINOS), 503
  - Cholesky factorization, 145, 167
  - Lagrangian (MINOS), 503
  - log-barrier function, 490
  - secant equation, 285
  - Wolfe line search, 239
- Modified augmented Lagrangian (MINOS), 503, 511
- Moore-Penrose pseudo-inverse, 701

**N**

Necessary optimality conditions, 2, 24

## Nonlinear

algebraic systems, 27

optimization, 9

programming, 9, 383

NELMED algorithm, 356, 357

## Newton

decrement, 133

method, 125, 140

with central-difference, 152

with finite differences, 145

with line-search, 129, 134

step, 133

system, 110, 126, 145, 613

Newton-CG method, 177, 324, 340

NEWUOA algorithm (direct method), 357, 358

NLPQLP (successive error restoration), 553, 556

linear search in, 555

Nocedal condition, 195

Nonlinear elasting mode (SNOPT), 543

Nonconvex quadratic programming, 450

Nondegenerate stationary point, 414

## Nonsmooth

exact penalty functions, 479

penalty method, 479

NPSOL, 567

## Null-space

of a matrix, 694

method (quadratic programming), 442

**O**

Objective function, 1, 21, 383

Open questions (conjugate gradient), 260

## Open

ball, 715

set, 715

Optimal design with composite materials application,

295, 433, 759

Optimality conditions, 2

Optimization based on simulation, 679

Order notation, 714

Orthogonality, 696

Outside contraction (Nelder-Mead), 356

Ou-Liu line-search, 51

**P**

Path-following primal-dual methods (quadratic programming), 464

Parameter, 1

## Penalty

barrier method, 491

parameter, 476

Penalty and merit function methods, 379

Penalty-barrier method, 491

PENNON, 519

Perfect line-search, 30

Performance

ratio, 14

profile, 13, 14

Perpendicular, 691

Perry/Shanno search direction, 248, 252

Perturbed KKT (quadratic programming), 464

Polak-Ribi  re-Polyak conjugate gradient, 171, 210

Positive

definite matrices, 697

semidefinite matrices, 697

Preconditioned conjugate gradient algorithm, 186

Predictor-corrector algorithm for quadratic Programming, 466

Pressure distribution in a journal bearing application, 294, 432, 759

Primal

feasibility, 391

problem, 406

Primal-dual

active set method (quadratic programming), 456

central path, 600, 613

interior-point algorithm, 620

system, 602, 603

Projected conjugate gradient method (quadratic

programming), 448

augmented system approach, 449

normal equations approach, 449

Propan combustion in air, 119, 150, 166, 723

Prototype of the interior-point algorithm, 602

**Q**

Quadratic

model of function f, 125, 262, 331

penalty

function, 475

method, 475, 476

programming, 1, 9

active-set method, 451, 452

convex, 439

elimination of variables, 471

equality constrained, 439

indefinite, 450

inequality constrained, 449

interior point method, 463

nonconvex, 439, 450

primal-dual active-set method, 456

strictly convex, 439

with simple bounds, 468

Quadratic programming solver SQOPT, 549

QR decomposition, 700

Quasi-Newton methods assumptions, 271

**R**

Range of a

function, 693

matrix, 693

Rate of convergence, 34, 710

Reduced gradient, 445, 545, 577, 588

inequality constraints (quadratic programming), 454

linear equality constraints (quadratic programming), 445

- Reduced gradient (*cont.*)  
     method (Wolfe), 576  
     simple bounds (quadratic programming), 455
- Reduced Hessian, 442, 545  
     quadratic programming, 531
- Reduced Newton system, 614
- Reduced system (quadratic programming), 442  
     conjugate gradient applied, 446  
     preconditioned conjugate gradient, 447
- Refining mechanism, 381
- Reflection (Nelder-Mead), 356
- Regular point  
     equality constraints, 394  
     general case, 402  
     inequality constraints, 391
- Regularity assumption, 397
- Regularization techniques, 381
- Relative nonlinearity, 116
- Relaxed steepest descent algorithm, 92
- Residual  
     conjugate gradient, 173  
     truncated Newton, 315, 316
- Restart criterion (Powell), 202
- Restart vectors of Beale, 247
- Restricted Broyden class, 269
- Robot kinematics problem, 118, 149, 166, 722
- Robust optimization, 11
- Robustness of algorithms, 12, 14, 78
- S**
- Scalar product, 691
- Scaling of BFGS method, 278
- Schur complement, 442  
     method, 441
- Secant equation, 262, 264
- Second-order  
     necessary conditions, 387, 398  
     sufficient conditions, 388, 399, 405
- Second-order conditions, 2
- Second-order correction (Maratos), 534, 536
- Selection phase in line-search, 68
- Self-correcting property, 273
- Sensitivity analysis, 2
- Separable unconstrained minimization, 310
- Sequential  
     linear programming, 373  
     linear-quadratic programming, 374, 541  
     quadratic programming, 373  
         equality constrained, 521, 523  
         inequality constrained, 524, 525, 528  
         simple approach, 526  
         with approximate Hessian, 529  
         with line search, 537  
         with successive error restoration, 553  
         with trust-region, 538, 539
- Sequential  $l_1$  quadratic programming for the equality and inequality constraints, 540
- Sherman-Morrison formula, 695
- Sherman-Morrison-Woodbury, 695
- Shrink operation (Nelder-Mead), 356
- Simple bounds, 1, 411  
     optimization, 9, 411  
         necessary conditions, 412  
         sufficient conditions, 414
- Simplified Newton method, 163
- Singular value decomposition, 700
- Slater condition, 403
- Small-scale optimization, 11
- SMUNO collection, 12, 117, 721  
     solution with DEEPS, 368, 369  
     solution with NELMED, 367  
     solution with NEWUOA, 367, 368
- Smooth manifold, 394
- SNOPT (sequential quadratic programming), 542, 551
- Solid fuel ignition, 120
- Span of a set of vectors, 691
- Sparse quasi-Newton updates, 309
- Spectral projected gradient (SPG) algorithm, 417, 418  
     cubic interpolation, 420  
     quadratic interpolation, 420
- SPENBAR algorithm, 488, 495
- SQOPT (quadratic programming solver), 549
- SQPlab, 567
- Spectral decomposition, 701
- SR1 method, 299  
     with generalized secant equation, 302
- Stable Cholesky factorization, 143
- Stable Newton method, 157
- Stable Newton negative curvature method  
     Fiacco and McCormick, 159  
     Fletcher and Freeman, 160  
     Gill and Murray, 158
- Standard conjugate gradient methods, 201
- Stationary point, 387, 487
- Stationary solution of a chemical reactor application, 117, 149, 165, 722
- Steady-State combustion application, 295, 434, 761
- Steepest descent algorithm, 75, 81, 82
- Stochastic optimization, 11
- Strict complementarity, 404
- Strict global  
     maximum, 385  
     minimum, 385
- Strictly convex function, 11
- Strong convex function, 62, 65, 717
- Strongly active (binding) constraint, 404
- Strongly active simple bound constraints, 404
- Sufficient reduction (line search), 42
- Subroutine  
     back, 69  
     LSbis, 71  
     WolfeLS, 72
- Subspaces, 693
- Sufficient  
     descent condition, 22, 63, 189, 270  
     optimality conditions, 2, 25
- Superbasic variables, 505
- Supremum, 384

**T**

- Tangent  
set, 394  
space, 394  
Taylor's theorem, 24, 708  
TNBC (TN with simple bounds), 428  
TNBC versus LBFGS-B and versus SPG, 429, 430  
Transversality, 391  
Trace of BFGS, 707  
TRON, 351  
Truncated Newton (TN) method, 315, 316  
Trust-region  
algorithm, 333  
for barrier problems, 628  
interior-point algorithm, 627, 630  
method, 380  
radius, 23, 332  
sequential quadratic programming algorithm, 539  
strategy, 22

**U**

- Unconstrained optimization, 9  
Uniformly convex function, 62  
UOP collection, 12  
solution with ASDB, 104  
solution with BFGS, 267  
solution with CG-DESCENT, 235  
solution with CONMIN, 250  
solution with DESCN, 243  
solution with DK/CGOPT, 255  
solution with hybrid conjugate gradient methods, 222  
solution with L-BFGS-B, 427  
solution with MM-BFGS, 307  
solution with MM-SR1gen, 307  
solution with SDB, SDW, 88  
solution with RSDB, 97  
solution with SPG, 420  
standard conjugate gradient, 214

- solution with TN, 325  
solution with TNBC, 429  
Upper bound, 2, 384, 411

**V**

- Variables, 1  
artificial (CONOPT), 588  
basic (GRG), 579  
nonbasic (GRG), 579  
slack (CONOPT), 588  
Vectors, 691  
length, 692  
linearly independent, 691  
magnitude, 692  
orthogonal, 691  
Von Neumann lemma, 695

**W**

- Warm start, 525  
Weakly active constraints, 404  
Weierstrass theorem, 386  
Wolfe duality, 408  
Wolfe line-search (strong), 42, 188, 263  
Wolfe line-search (weak), 41, 188, 262  
Wolfe line-search with  
safeguarding and cubic interpolation, 68  
simple bisection, 52, 68  
Word rectification, 3  
Working set, 392, 450  
Working matrix (SNOPT), 545  
Working surface, 572

**Z**

- Zhang-Hager line search, 49  
Zoutendijk  
condition, 58, 63, 190  
method (constrained optimization), 57