Practical Augmented Lagrangian Methods for Constrained Optimization

Fundamentals *of* Algorithms

Editor-in-Chief: Nicholas J. Higham, University of Manchester

The SIAM series on Fundamentals of Algorithms is a collection of short user-oriented books on state-of-the-art numerical methods. Written by experts, the books provide readers with sufficient knowledge to choose an appropriate method for an application and to understand the method's strengths and limitations. The books cover a range of topics drawn from numerical analysis and scientific computing. The intended audiences are researchers and practitioners using the methods and upper level undergraduates in mathematics, engineering, and computational science.

Books in this series not only provide the mathematical background for a method or class of methods used in solving a specific problem but also explain how the method can be developed into an algorithm and translated into software. The books describe the range of applicability of a method and give guidance on troubleshooting solvers and interpreting results. The theory is presented at a level accessible to the practitioner. MATLAB® software is the preferred language for codes presented since it can be used across a wide variety of platforms and is an excellent environment for prototyping, testing, and problem solving.

The series is intended to provide guides to numerical algorithms that are readily accessible, contain practical advice not easily found elsewhere, and include understandable codes that implement the algorithms.

Series Volumes

Birgin, E. G., and Martínez, J. M., *Practical Augmented Lagrangian Methods for Constrained Optimization*
Bini, D. A., Iannazzo, B., and Meini, B., *Numerical Solution of Algebraic Riccati Equations*
Escalante, R. and Raydan, M., *Alternating Projection Methods*
Hansen, P. C., *Discrete Inverse Problems: Insight and Algorithms*
Modersitzki, J., *FAIR: Flexible Algorithms for Image Registration*
Chan, R. H.-F. and Jin, X.-Q., *An Introduction to Iterative Toeplitz Solvers*
Eldén, L., *Matrix Methods in Data Mining and Pattern Recognition*
Hansen, P. C., Nagy, J. G., and O'Leary, D. P., *Deblurring Images: Matrices, Spectra, and Filtering*
Davis, T. A., *Direct Methods for Sparse Linear Systems*
Kelley, C. T., *Solving Nonlinear Equations with Newton's Method*

E. G. Birgin
University of São Paulo
São Paulo, SP, Brazil

J. M. Martínez
State University of Campinas
Campinas, SP, Brazil

Practical Augmented Lagrangian Methods for Constrained Optimization

siam
Society for Industrial and Applied Mathematics
Philadelphia

10 9 8 7 6 5 4 3 2 1

Royalties from the sale of this book are placed in a fund to help students attend SIAM meetings and other SIAM-related activities. This fund is administered by SIAM, and qualified individuals are encouraged to write directly to SIAM for guidelines.

Library of Congress Cataloging-in-Publication Data

Birgin, E. G. (Ernesto G.)

Practical augmented lagrangian methods for constrained optimization / E.G. Birgin, University of São Paulo, São Paulo, Brazil, J. M. Martínez, State University of Campinas, Campinas, São Paulo, Brazil.
pages cm. -- (Fundamentals of algorithms ; 10)
Includes bibliographical references and index.
ISBN 978-1-611973-35-8
1. Constrained optimization. 2. Mathematical optimization. 3. Lagrangian functions. I. Martínez, J. M. (José Mario) II. Title.
QA323.B57 2014
519.6--dc23

2013048382

To our families

Contents

Preface

This book is about the Augmented Lagrangian method, a popular technique for solving constrained optimization problems. It is mainly dedicated to engineers, chemists, physicists, economists, and general users of constrained optimization for solving real-life problems. Nevertheless, it describes in rigorous mathematical terms the convergence theory that applies to the algorithms analyzed. Users often need to understand with precision the properties of the solutions that a practical algorithm finds and the way in which these properties are reflected in practice. Many theorems concerning the behavior of practical algorithms will be found in this book. The geometrical and computational meaning of each theoretical result will be highlighted to make the relevant theory accessible to practitioners. Often, the assumptions under which we prove that algorithms work will not be the most general ones but will be those whose interpretation helps one to understand the computational behavior in real-life problems. Moreover, the plausibility of most assumptions will be discussed, presenting simple sufficient conditions under which assumptions hold. This helps one foresee what can be expected from a practical algorithm and which properties are not expected at all.

Modest mathematical background is required to understand the proofs and less is required for understanding and interpreting statements and definitions. Elementary calculus in $\mathbb{R}^n$ with the basic topological properties concerning convergence of sequences and compact sets are enough. In fact, we have deliberately included only results for which the comprehension of such background is sufficient. The optimality conditions for nonlinear programming, for example, are presented in a concise though rigorous way that demands only resources acquired in good undergraduate engineering courses. In particular, although familiarity is always welcome, no previous knowledge of optimization is required.

Readers are introduced in this book to the employment of a specific constrained optimization package of Augmented Lagrangian type, called Algencan. The software is introduced after the statement and interpretation of all the relevant theory. The book finishes with practical examples. Codes and supplementary materials can be found at www.siam.org/books/fa10.

Acknowledgments Many students, colleagues, and friends have contributed to this book and to the development of Algencan. To all of them we are extremely thankful. In particular, we would like to thank Ana Friedlander, Francisco Magalhães Gomes, Francisco Sobral, Gabriel Haeser, Jair Silva, Jan Gentil, John Gardenghi, Juliano Francisco, Maria Aparecida Diniz Ehrhardt, Márcia Gomes Ruggiero, Margarida Pinheiro Mello, Marina Andretta, Laura Schuverdt, Leandro Prudente, Lucas Pedroso, Lúcio Tunes dos Santos, Luis Felipe Bueno, Luiz Antonio Medeiros, Nataša Krejić, Rafael Lobato, Ricardo Andrade, Roberto Andreani, Rodrigo Lima, Sandra Santos, Sergio Ventura, Vera Lucia da Rocha Lopes, Viviana Ramírez, and Yalcin Kaya. We are especially thankful to Rafael

Lobato for the dedicated and careful reading of the manuscript. We are also thankful to all the Algencan users who have contributed with suggestions over the years. We also would like to thank Marcos Raydan and the other members of the editorial board of the SIAM Fundamentals of Algorithms book series, who encouraged us to write this book. To the anonymous referees of our proposal, whose comments contributed to assembling and giving shape to our initial ideas, many thanks for invigorating and stimulating suggestions and kind words. We also express our gratitude to the Institute of Mathematics and Statistics of the University of São Paulo and to the Institute of Mathematics, Statistics, and Scientific Computing of the State University of Campinas for their support of our research and to the National Council for Scientific and Technological Development (CNPq) and the São Paulo Research Foundation (FAPESP, grants 2013/03447-6, 2013/05475-7, and CEPID/Industrial Mathematics 2013/07375-0) for their financial support. Finally, we wish to thank the SIAM publishing staff, including Bruce Bailey, Gina Rinelli, and Lois Sellers for their hard work. In particular, our thanks to Ann Manning Allen and Elizabeth Greenspan for their patience, understanding, and support.

Ernesto G. Birgin
São Paulo, SP, Brazil

José Mario Martínez
Campinas, SP, Brazil

Nomenclature

- The elements of $\mathbb{R}^n$ will be denoted by columns.
- If $x \in \mathbb{R}^n$, its ith component will be denoted by x_i if this does not lead to confusion.
- If $x \in \mathbb{R}^n$ and $x_i > 0$ (< 0) for all $i = 1, \dots, n$, we write $x > 0$ (< 0).
- If $x, y \in \mathbb{R}^n$ and $x - y > 0$ we write $x > y$.
- If $x, y \in \mathbb{R}^n$, the vector whose components are $\min\{x_1, y_1\}, \dots, \min\{x_n, y_n\}$ will be denoted by $\min\{x, y\}$.
- We will denote $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n \mid x \geq 0\}$ and $\mathbb{R}^n_{++} = \{x \in \mathbb{R}^n \mid x > 0\}$.
- A^T will denote the transpose of the matrix A.
- If $x \in \mathbb{R}^n$, $x = (x_1, \dots, x_n)^T$, we denote $x_+ = (\max\{0, x_1\}, \dots, \max\{0, x_n\})^T$ and $x_- = (\min\{0, x_1\}, \dots, \min\{0, x_n\})^T$.
- We denote $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ and $\mathbb{N} = \{1, 2, 3, \dots\}$.
- The sequence $\{z^1, z^2, z^3, \dots\}$ will be denoted by $\{z^k\}$.
- If $K = \{k_1, k_2, \dots\} \subseteq \mathbb{N}$ (with $k_j < k_{j+1}$ for all j), we denote $K \underset{\infty}{\subseteq} \mathbb{N}$.
- The symbol $\|\cdot\|$ will denote an arbitrary norm. The Euclidean norm will be denoted by $\|\cdot\|_2$ and $\|\cdot\|_\infty$ will be the symbol for the infinity-norm. We will always assume that $\|(x_1, \dots, x_n)^T\| = \|(|x_1|, \dots, |x_n|)^T\|$.
- If $h : \mathbb{R}^n \to \mathbb{R}^m$, we denote $\nabla h(x) = (\nabla h_1(x), \dots, \nabla h_m(x)) \in \mathbb{R}^{n \times m}$ and $h'(x) = \nabla h(x)^T$.
- The Euclidean ball with center $\bar{x}$ and radius Δ given by $\{x \in \mathbb{R}^n \mid \|x - \bar{x}\|_2 \leq \Delta\}$ will be denoted by $B(\bar{x}, \Delta)$.
- Given $x, y \in \mathbb{R}^n$, the segment with extremes x and y will be denoted by $[x, y]$.
- If $\{a_k\}$ and $\{b_k\}$ are nonnegative sequences, we denote by $a_k = o(b_k)$ the fact that there exists a sequence $c_k \to 0$ such that $a_k \leq c_k b_k$ for all $k \in \mathbb{N}$. If there exists $c > 0$ such that $a_k \leq c\, b_k$ for all $k \in \mathbb{N}$, we denote $a_k = O(b_k)$.

Chapter 1
Introduction

Constrained optimization is one of the mathematical problems that most frequently serve for modeling natural phenomena. Large problems of this type appear in many challenging areas of science and technology, such as protein folding, electronic calculations, geophysical prospecting, nonlinear control, data mining, image processing and reconstruction, and structural engineering. Many human activities give rise to the minimization of functions with constraints. Making decisions usually involves optimization (maximizing utilities and profits or minimizing loss), risk analysis requires careful identification of objectives and restrictions, and, in general, every mathematical model of real-life systems requires fitting parameters before its actual application.

One of the most natural ideas for solving constrained minimization problems consists of adding to the objective function a term that penalizes the lack of fulfillment of the constraints. Quadratic penalty functions measure the nonsatisfaction of equality constraints $h_i(x) = 0$ by their square $h_i(x)^2$. Nonfulfillment of inequality constraints $g_i(x) \leq 0$ is measured by $\max\{0, g_i(x)\}^2$. Many other penalty functions have been considered in the optimization literature, but the simple quadratic penalty term maintains its attractiveness because it preserves differentiability and penalizes linear constraints by convex quadratics (or piecewise convex quadratics), which are very adequate for minimization purposes. In the case of inequality constraints, the theory of semismooth nonlinear systems has revealed that the absence of continuous second derivatives is not a serious drawback, at least for the application of Newtonian and inexact Newton strategies. Moreover, if one uses slack variables, inequality constraints are transformed into equality constraints and bounds, so that the penalty subproblem associated with a quadratic programming problem reduces to the sequential minimization of quadratics with bounds on the variables. Constrained optimization problems may be extremely difficult, and their solvability in the large-scale case depends on clever formulations that, roughly speaking, make them similar to quadratic programming, at least locally. As a consequence, the quadratic penalty approach remains a useful practical choice. Dostál [102] recently published a book reflecting his research in applied mechanical engineering in which he introduces the employment of Augmented Lagrangians with quadratic penalty functions for quadratic programming.

By means of the employment of the quadratic penalty method, one reduces the constrained optimization problem to a sequence of simply constrained (perhaps unconstrained) minimization problems in which the penalty parameters tend to infinity, giving increasing weight to the constraints with respect to the objective function. This is the main

drawback of the penalty approach. As the penalty parameters grow, subproblems become more and more difficult. If the penalty parameters are very large, a point close to the solution of the original problem with modest infeasibility may be considered as being worse than an almost feasible point with poor objective function value that could be far from the solution. To alleviate this inconvenience, one could very cautiously increase the penalty parameters (see [130]), which may lead to solving an excessively large number of subproblems, or one may adopt the shifting strategy that motivates the Augmented Lagrangian approach.

According to the shifting idea, one tries to obtain the solution of the problem employing a moderate value of the penalty parameters. The basic observation is that the effect of getting a subproblem solution close to the true solution of the constrained optimization problem can be obtained by applying penalization not to the true constraints but to displaced constraints whose shifting level corresponds to the infeasibility at the solution of the last subproblem solved. This idea seems to be implicitly or explicitly present in the first historical works in which the Augmented Lagrangian approach with quadratic penalization appears [73, 136, 144, 218, 229].

Augmented Lagrangian methods have been used many times in connection with engineering problems, and for general optimization, its popularity increased with the introduction of the Lancelot package in the late 1980s [82, 83].

In spite of the permanent introduction of new efficient methods for nonlinearly constrained optimization, there are families of problems for which the Augmented Lagrangian approach is recommended, especially in the large-scale case. Many reasons support this point of view:

1. The approach is modular. The efficiency of most practical Augmented Lagrangian methods is linked to the efficiency of the algorithms used for solving the subproblems. Often, the constraints of these subproblems define an n-dimensional box. Box-constrained minimization is a classical problem on which intensive research continues, especially in the large-scale case. As local or global box-constraint algorithms are developed, the effectiveness of the associated Augmented Lagrangian algorithms is increased.

2. The Augmented Lagrangian approach is appealing for situations in which it is natural to divide the constraints into two groups. The constraints to which one applies penalization and shifts belong to the first group. The second group contains constraints that remain in the subproblems and are not penalized or shifted at all. Sometimes, as in the case of boxes, these constraints are simple and we know practical methods for dealing with them. Generally speaking, the Augmented Lagrangian framework allows one to handle specific constraints in a specific way, when the appropriate technology for that purpose exists.

3. The Augmented Lagrangian method may be used for derivative-free optimization, provided that derivative-free methods for solving the subproblems are available [99, 164, 175, 176].

4. The convergence toward global minimizers of constrained optimization problems depends, in the Augmented Lagrangian framework, only on the global minimization properties of the probably simpler subproblems [49, 58].

5. The applicability of the Augmented Lagrangian method is not impaired by an excess of (equality or inequality) constraints, as for other constrained minimization methods.

6. Natural extensions of the method for nonsmooth generalizations of the constrained optimization problems exist and involve minimal modifications of the basic Augmented Lagrangian algorithm [42].

7. The convergence theory of the Augmented Lagrangian methods can be used to improve robustness of algorithms that, in general, are not classified as Augmented Lagrangian [107].

Although dedicated to engineers, social scientists, physicists, chemists, and general practitioners, this book offers a solid theoretical basis for understanding the results that help to explain the practical behavior of algorithms. All proved theorems make use of plausible assumptions with clear geometrical or algorithmic meaning. Even when the user does not wish to read the proofs, he or she will be able to interpret their meaning, the hypotheses on which they are based, and their practical consequences. Appropriate remarks will emphasize these points throughout the text.

Chapter 2
Practical Motivations

The optimization paradigm provides a model for almost every human conscious (and perhaps unconscious) activity. We are always doing things in the best possible way subject to material, economical, technical, or social constraints. We maximize profits, we improve happiness, we minimize the risks of illness and poverty, we adjust our perceptions of reality to observations, and so on. More surprising is the fact that even nature persistently seems to optimize objectives by means of energy minimization, entropy maximization, or the search for equilibrium states. Perhaps some kind of natural selection forces participants in the universe game to optimize criteria in order to survive, or merely to be observable. The problems described below have been addressed by optimization methods, Augmented Lagrangians, and related techniques and illustrate the applicability of these technologies in the real world.

2.1 ▪ Selected examples

Problem 2.1: Electronic structure calculations

For fixed nuclei coordinates, an electronic structure calculation consists of finding the wave functions from which the spatial electronic distribution of the system can be derived. These wave functions are the solutions of the Schrödinger equation [74, 143].

The practical solution of the Schrödinger equation is computationally very demanding. Therefore, simplifications are made that lead to more tractable mathematical problems. The best-known approach consists of approximating the solution by a (Slater) determinant. Such approximation allows for a significant simplification of the Schrödinger postulation, which results in a "one-electron" eigenvalue (Hartree–Fock) equation. The solutions of this new one-electron eigenvalue problem are used to reconstitute the Slater determinant and, therefore, the electronic density of the system.

We will focus on the restricted Hartree–Fock case, for which the number of electrons is $2n_e$. Writing the n_e functions that compose the Slater determinant as linear combinations with respect to a basis with n_b elements, the unknowns of the problem turn out to be the coefficients of the unknown functions with respect to the basis, giving rise to the so-called Hartree–Fock–Roothaan problem. A suitable discretization technique uses plane wave basis or localized basis functions, with compact support or with a Gaussian fall-off. In this way, the unknowns of the problem are represented by a coefficient matrix $C \in \mathbb{R}^{n_b \times n_e}$. The optimal choice of the coefficients comes from the solution of an

optimization problem of the form

$$\text{Minimize } E(P) \text{ subject to } P = P^T,\ P^2 = P,\ \text{Trace}(P) = n_e, \tag{2.1}$$

where $P = CC^T$ is the so-called density matrix in the atomic-orbital basis. Obviously, this problem can also be formulated using as an independent variable the coefficient matrix C.

The form of $E(P)$ in (2.1) is

$$E_{SCF}(P) = \text{Trace}\Big[2HP + G(P)P\Big],$$

where H is the one-electron Hamiltonian matrix, elements $G_{ij}(P)$ of $G(P)$ are given by

$$G_{ij}(P) = \sum_{k=1}^{n_b}\sum_{\ell=1}^{n_b}(2g_{ijk\ell} - g_{i\ell kj})P_{\ell k},$$

$g_{ijk\ell}$ is a two-electron integral in the atomic-orbital basis, n_b is the number of functions in the basis, and, as stated above, $2n_e$ is the number of electrons. For all $i, j, k, \ell = 1, \ldots, n_b$ one has

$$g_{ijk\ell} = g_{jik\ell} = g_{ij\ell k} = g_{k\ell ij}.$$

This problem has been addressed using the Augmented Lagrangian technology in [2] and [202]. See also [57]. Note that for large values of n_e and n_b, the number of variables and constraints of the problem is very big.

Problem 2.2: Packing molecules

Molecular dynamics is a powerful technique for comprehension at the molecular level of a great variety of chemical processes. With enhancement by computational resources, very complex systems can be studied. The simulations need starting points that must have adequate energy requirements. However, if the starting configuration has close atoms, the temperature scaling is disrupted by excessive potentials that accelerate molecules over the accepted velocities for almost any reasonable integration time step. In fact, the starting coordinate file must be reliable in the sense that it must not exhibit overlapping or close atoms, so that temperature scaling can be performed with reasonable time steps for a relatively fast energy equilibration of the system.

In [187, 193] the problem of placing the initial positions of the molecules is represented as a "packing problem." The goal is to place known objects in a finite domain in such a way that the distance between any pair of points of different objects is larger than a threshold tolerance. In our case, objects are molecules and points are atoms. Following this idea, an optimization problem is defined. The mathematical (optimization) problem consists in the minimization of a function of (generally many) variables, subject to constraints that define the region in which the molecules should be placed.

Let us call $nmol$ the total number of molecules that we want to place in a region $\mathscr{R}$ of the three-dimensional space. For each $i = 1, \ldots, nmol$, let $natom(i)$ be the number of atoms of the ith molecule. Each molecule is represented by the orthogonal coordinates of its atoms. To facilitate the visualization, assume that the origin is the barycenter of all the molecules. For all $i = 1, \ldots, nmol$, $j = 1, \ldots, natom(i)$, let

$$A(i, j) = (a_1^{ij}, a_2^{ij}, a_3^{ij})$$

be the coordinates of the jth atom of the ith molecule.

Suppose that one rotates the ith molecule sequentially around the axes x_1, x_2, and x_3, where α_i, β_i, and γ_i are the angles that define such rotations. Moreover, suppose that after these rotations, the whole molecule is displaced so that its barycenter, instead of the origin, becomes $C_i = (c_1^i, c_2^i, c_3^i)$. These movements transform the atom of coordinates $A(i,j)$ in a displaced atom of coordinates

$$P(i,j) = (p_1^{ij}, p_2^{ij}, p_3^{ij}).$$

Observe that $P(i,j)$ is always a function of $(C_i, \alpha_i, \beta_i, \gamma_i)$, but we do not make this dependence explicit in order to simplify the presentation.

In [187, 193] the objective is to find angles α_i, β_i, γ_i and displacements C_i, $i = 1,\ldots,nmol$, in such a way that, for all $j = 1,\ldots,natom(i)$, $j' = 1,\ldots,natom(i')$,

$$||P(i,j) - P(i',j')||_2^2 \geq \varepsilon \text{ whenever } i \neq i', \tag{2.2}$$

and, for all $i, = 1,\ldots,nmol$, $j = 1,\ldots,natom(i)$,

$$P(i,j) \in \mathscr{R}, \tag{2.3}$$

where $\varepsilon > 0$ is a user-specified tolerance. In other words, the rotated and displaced molecules must remain in the specified region and the squared distance between any pair of atoms must not be less than ε.

The objective (2.2) leads us to define the merit function

$$f(C_1,\ldots,C_{nmol},\alpha_1,\beta_1,\gamma_1,\ldots,\alpha_{nmol},\beta_{nmol},\gamma_{nmol})$$

$$= \sum_{i=1}^{nmol-1} \sum_{j=1}^{natom(i)} \sum_{i'=i+1}^{nmol} \sum_{j'=1}^{natom(i')} \max\{0, \varepsilon - ||P(i,j) - P(i',j')||_2^2\}^2. \tag{2.4}$$

Note that $f(C_1,\ldots,C_{nmol},\alpha_1,\beta_1,\gamma_1,\ldots,\alpha_{nmol},\beta_{nmol},\gamma_{nmol})$ is nonnegative for all angles and displacements. Moreover, f vanishes if and only if the objective (2.2) is fulfilled. This means that if we find displacements and angles where $f = 0$, the atoms of the resulting molecules are sufficiently separated. This leads us to define the following minimization problem:

$$\text{Minimize } f(C_1,\ldots,C_{nmol},\alpha_1,\beta_1,\gamma_1,\ldots,\alpha_{nmol},\beta_{nmol},\gamma_{nmol}) \tag{2.5}$$

subject to (2.3).

The objective function f is continuous and differentiable, although their second derivatives are discontinuous. The number of variables is $6 \times nmol$ (three angles and a displacement per molecule). The analytical expression of f is cumbersome, since it involves consecutive rotations, and its first derivatives are not very easy to code. However, optimization experience leads one to pay the cost of writing a code for computing derivatives or to use automatic differentiation, with the expectation that algorithms that take advantage of first-order information are really profitable, especially when the number of variables is large. Having a code that computes f and its gradient, we are prepared to solve (2.5) using constrained optimization techniques.

Problem 2.3: Nonrigid image registration

Nonrigid registration is used to segment images using a prelabeled atlas, to estimate the motion in a sequence of images, and to compress and encode video [22, 201, 206, 234, 236, 245, 248, 249]. The formulation that we present here is, essentially, the one given

by Sdika [236]. Consider two "images" I_f and I_r defined on a box $\Omega \subseteq \mathbb{R}^N$ (N being generally equal to 2 or 3). The function $I_r : \Omega \to \mathbb{R}$ will be called the "reference" image, whereas $I_f : \Omega \to \mathbb{R}$ is the "floating" image. The problem is to find a bijective smooth transformation $\Phi : \Omega \to \Omega$ such that, roughly speaking, $I_f(\Phi(x))$ is as similar as possible to $I_r(x)$ for all $x \in \mathscr{P}$, where $\mathscr{P} \subseteq \Omega$ is a finite set of "voxels." The transformation Φ is chosen to be dependent of a finite vector of parameters c (spline coefficients in [236]). Consequently, we denote $t(c,x) = \Phi(x)$, and the objective function that one wants to minimize is

$$f(c) = \frac{1}{|\mathscr{P}|} \sum_{x \in \mathscr{P}} \rho(I_f(t(c,x)] - I_r(x)),$$

where $|\mathscr{P}|$ denotes the number of voxels and $\rho(z)$ measures dissimilarity (for example, $\rho(z) = z^2/2$). The transformation Φ needs to be bijective, a property that is difficult to guarantee computationally. In [236] the bijectivity of Φ is handled by imposing two conditions:

1. The determinant of the Jacobian $\Phi'(x)$ must be greater than a given tolerance $\varepsilon_d > 0$ for all the voxels $x \in \mathscr{P}$.
2. If, at a voxel x, the determinant of the Jacobian is close to ε_d, the norm of its gradient must be small. (This is guaranteed algebraically by means of a judicious choice of a forcing function.)

Rigorously speaking, the two conditions above do not guarantee bijectivity of Φ but seem to be sufficient to ensure that property in practice. The optimization problem is to minimize f subject to the constraints above (two constraints per voxel). For the registration of a three-dimensional (3D) image of size $256 \times 256 \times 180$, the number of constraints is bigger than 10^7. The number of variables is the dimension of the parameter vector c. In the spline representation used by Sdika [236], employing a node spacing of six voxels, the problem has 2×10^5 variables. These dimensions recommend the use of Augmented Lagrangian methods [236].

Problem 2.4: Optimization of cryopreservation [34]

Cryopreservation of human oocytes is an important technique used in in vitro fertilization. Conventional freezing and "vitrification" involve the use of "cryoprotective agents" (CPAs), which, in both cases, are crucial components of the cryopreservation medium. However, exposure to CPAs can cause cell damage because of toxicity. A model introduced by Benson, Kearsley, and Higgins [34] aims to minimize a toxicity cost function while avoiding osmotic damage by keeping cell volumes constrained. The (dimensionless) dynamical system considered in this model is

$$\frac{dw}{d\tau} = -m_1(\tau) - m_2(\tau) + \frac{1 + s(\tau)}{w(\tau)}, \quad \frac{ds}{d\tau} = b\left(m_2(\tau) - \frac{s(\tau)}{w(\tau)}\right) \tag{2.6}$$

for $\tau \in [0, \tau^f]$, where (i) τ is the temporal variable, (ii) $w(\tau)$ is the intracellular water volume, (iii) $s(\tau)$ is the quantity of intracellular CPA, (iv) $m_1(\tau)$ is the extracellular concentration of nonpermeating solute, (v) $m_2(\tau)$ is the extracellular concentration of CPA, (vi) $w(0) = w^i$ and $s(0) = s^i$ where (w^i, s^i) is the given initial state, and (vii) b is a unitless relative permeability constant.

The total cell volume is given by

$$v(\tau) = w(\tau) + \gamma s(\tau) + v_b,$$

where γ is the product of the isotonic solute concentration and the partial molar volume of CPA and v_b is the osmotically inactive volume normalized to the cell water volume under isotonic conditions. (In fact, all the quantities of the model are conveniently normalized.) Minimal and maximal normalized cell volumes v_* and v^* are imposed in such a way that

$$v_* \le w(\tau) + \gamma s(\tau) + v_b \le v^* \tag{2.7}$$

for all $\tau \in [0, \tau^f]$. The objective function that has to be minimized with respect to $m_1(\tau)$ and $m_2(\tau)$ is given by

$$J_{\alpha,\varepsilon}(m_1, m_2) = \int_0^{\tau^f} [s(\tau)/w(\tau)]^\alpha + (1/\varepsilon)[(w(\tau^f) - w^f]^2 + [s(\tau^f) - s^f]^2 d\tau, \tag{2.8}$$

where $\alpha = 1.6$ describes the concentration dependence of the toxicity rate, ε is a regularization empirically determined parameter according to the scaling of $s(\tau)$ and $w(\tau)$, and (w^f, s^f) is a desired final state.

In [34], given the discretization parameters N and h such that $\tau^f = Nh$ and the independent variables $m_1(0)$, $m_2(0)$, $m_1(h)$, $m_2(h), \ldots, m_1(Nh)$, $m_2(Nh)$, the objective function is computed integrating (2.8) by means of Heun's method [244], and the process of minimizing with constraints (2.7) is conducted with the Augmented Lagrangian code Algencan [8]. (In fact, the authors of [34] prefer to replace the set of $2(N+1)$ constraints (2.7) with only two (more complex) constraints. Constraints

$$w(\tau) + \gamma s(\tau) + v_b - v^* \le 0, \ \tau = 0, h, \ldots, Nh,$$

are replaced by

$$(1/\tau^f) \sum_\tau H(w(\tau) + \gamma s(\tau) + v_b - v^*) \le 0,$$

and constraints

$$w(\tau) + \gamma s(\tau) + v_b - v_* \ge 0, \ \tau = 0, h, \ldots, Nh,$$

are replaced by

$$(1/\tau^f) \sum_\tau -H(w(\tau) + \gamma s(\tau) + v_b - v_*) \le 0,$$

where H is a smooth approximation of the Heaviside step function.)

Problem 2.5: Minimization with value-at-risk constraints

Value-at-risk (VaR) constrained optimization is the problem of finding a portfolio such that the probability a predicted loss will be greater than some given tolerance is smaller than α. Among the portfolios that satisfy that constraint, one wishes to minimize some objective function, like the average loss, the variation with respect to the actual portfolio, the transaction costs, or combinations of those criteria [42]. Assume that the portfolio is composed of n assets. We consider an instant t in the future and m scenarios, and we define that under the scenario i the unitary value of asset j at instant t is given by π_{ij}. Moreover, by simulation, we establish that the probability of scenario i is p_i. Therefore, the predicted value of the portfolio $(x_1, \ldots, x_n)$ at instant t under scenario i is $\sum_{j=1}^n \pi_{ij} x_j$. Consequently, the predicted value of the portfolio will be smaller than M with a probability $\sum_{i=1}^m p_i H(M - \sum_{j=1}^n \pi_{ij} x_j)$, where H is the Heaviside function. This means that the constraint establishing a low probability α for a value of the portfolio smaller than M may be expressed as

$$\sum_{i=1}^m p_i H\Big(M - \sum_{j=1}^n \pi_{ij} x_j\Big) \le \alpha. \tag{2.9}$$

With some abuse of notation, we also define H as a smooth approximation of the Heaviside function such that $H(s) = 1$ for all $s > 0$. Therefore, the fulfillment of (2.9) for the smooth approximation guarantees the fulfillment of the constraint when H is the true step function. Consider now the objective function that imposes the minimal change with respect to a present portfolio $\bar{x}$. Ideally, we would like to change as few assets as possible. Then, in order to approximate that objective, we minimize the ℓ_1-norm of the difference between x and $\bar{x}$. Namely, the objective function will be

$$f(x) = \sum_{i=1}^{n} |x_i - \bar{x}_i|. \tag{2.10}$$

In order to overcome the lack of differentiability of f, we consider the change of variables $x - \bar{x} = u - v$ and formulate the problem of minimizing (2.10) with the constraints (2.9) and $x \geq 0$ in the following way:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n} u_i + v_i \\ \text{subject to} \quad & \sum_{i=1}^{m} p_i H\left(M - \sum_{j=1}^{n} \pi_{ij}(\bar{x}_j + u_j - v_j) \right) \leq \alpha, \\ & \bar{x} + u - v \geq 0, \\ & u \geq 0,\ v \geq 0. \end{aligned}$$

Problem 2.6: Capacity expansion planning [177]

Optimal industrial capacity planning involves selection of processes and timing of capacity expansions in a way that maximizes the "net present value" (NPV) of the project over a planning horizon. The future values of prices, demands, and availabilities of chemical products and raw materials are subject to uncertainties, a feature that gives rise to a "stochastic optimization" problem which is typically handled by using different scenarios. In [177] the corresponding "stochastic integer programming" problem is solved using an Augmented Lagrangian approach.

The decision variables of the model (formulated first in [178]) are as follows: (i) Y_{its}, binary capacity expansion decision for process i, time t, and scenario s (value 1 indicates the expansion); (ii) E_{its}, capacity expansion of process i to be installed in period t under scenario s; (iii) Q_{its}, total capacity of process i in period t under scenario s; (iv) W_{its}, operating level of process i in period t under scenario s; (v) $P_{j\ell ts}$, amount of chemical j purchased from market ℓ in period t under scenario s; (vi) $S_{j\ell ts}$, amount of chemical j sold to market ℓ in period t under scenario s; (vii) Y_{1i}, binary capacity expansion decision for process i in the first time period; (viii) E_{1i}, capacity expansion for process i to be installed in the first time period; (ix) Q_{1i}, total capacity for process i in the first time period.

The time variable t goes from 1 to T. The objective function NPV to be maximized in the model is

$$NPV = \sum_{s} p_s \left[\sum_{j,\ell,t} (\gamma_{j\ell ts} S_{j\ell ts} - \Gamma_{j\ell ts} P_{j\ell ts}) - \sum_{i,t} (\alpha_{it} E_{its} + \beta_{it} Y_{its} + \delta_{its} W_{its}) \right]. \tag{2.11}$$

In (2.11), p_s denotes the probability of scenario s; $\gamma_{j\ell ts}$ and $\Gamma_{j\ell ts}$ are sales and purchase prices, respectively; α_{it} and β_{it} are investment costs; and δ_{its} is the unit operation cost for process i during time period t under scenario s.

The function NPV must be maximized subject to constraints of the following form:

$$Y_{its}E^L_{it} \le E_{its} \le Y_{its}E^U_{it} \text{ for all } i,t,s, \tag{2.12}$$

$$Q_{its} = Q_{i,t-1,s} + E_{its} \text{ for all } i,t,s, \tag{2.13}$$

$$\sum_t Y_{its} \le NEXP_i \text{ for all } i,s, \tag{2.14}$$

$$\sum_i (\alpha_{it}E_{its} + \beta_{it}Y_{its}) \le CI_t \text{ for all } t,s, \tag{2.15}$$

$$W_{its} \le Q_{its} \text{ for all } i,t,s, \tag{2.16}$$

$$\sum_\ell P_{j\ell ts} + \sum_i \eta_{ij}W_{its} = \sum_\ell S_{j\ell ts} + \sum_i \mu_{ij}W_{its} \text{ for all } j,t,s, \tag{2.17}$$

$$a^L_{j\ell ts} \le P_{j\ell ts} \le a^U_{j\ell ts} \text{ for all } j,\ell,t,s, \tag{2.18}$$

$$d^L_{j\ell ts} \le S_{j\ell ts} \le d^U_{j\ell ts} \text{ for all } j,\ell,t,s, \tag{2.19}$$

$$Y_{i1s} = Y1_i, E_{i1s} = E1_i, Q_{i1s} = Q1_i \text{ for all } i,s, \tag{2.20}$$

$$Y_{its} \in \{0,1\}, E_{its}, Q_{its}, W_{its} \ge 0 \text{ for all } i,t,s, \tag{2.21}$$

$$Y1_i \in \{0,1\}, E1_i, Q1_i \ge 0 \text{ for all } i, \tag{2.22}$$

$$P_{j\ell ts}, S_{j\ell ts} \ge 0 \text{ for all } j,\ell,t,s, \tag{2.23}$$

where E^L_{it}, E^U_{it}, $NEXP_i$, CI_t, η_{ij}, μ_{ij}, $a^L_{j\ell ts}$, $a^U_{j\ell ts}$, $d^L_{j\ell ts}$, and $d^U_{j\ell ts}$ are given constants. (See [177, p. 882] for a detailed description of the meaning of each constant, variable, and constraint.)

The "nonanticipativity" constraints (2.20) say that the first-stage decisions should be the same under all possible scenarios. If these constraints were excluded from the model, the optimization problem would be "separable" in the sense that its solution would come from separately solving a different (mixed integer linear programming (MILP)) problem for each scenario. Including the constraints (2.20), we also have an MILP problem but, due to its large-scale characteristics, its solution is handled in [177] by means of a decomposition Augmented Lagrangian strategy. The idea is to eliminate the constraints (2.20), incorporating them in the objective function as a penalty Lagrangian term. (This idea will be exhaustively developed in this book.) The new objective function will have the form

$$LA = -NPV + \sum_{s,i} \lambda^Y_{is}(\hat{Y}_i - Y_{i1s}) + \lambda^E_{is}(\hat{E}_i - E_{i1s}) + \lambda^Q_{is}(\hat{Q}_i - Q_{i1s})$$
$$+ (\rho/2)\sum_{si} (\hat{Y}_i - Y_{i1s})^2 + (\hat{E}_i - E_{i1s})^2 + (\hat{Q}_i - Q_{i1s})^2$$

and should now be minimized. The solution of the original problem comes from repetitive minimizations of LA, followed by suitable updates of the "Lagrange multipliers" λ_{is} and the "penalty parameter" ρ. Unfortunately, these subproblems are not separable yet, due to the presence of the squared terms $(\hat{Y}_i - Y_{i1s})^2$, $(\hat{E}_i - E_{i1s})^2$, and $(\hat{Q}_i - Q_{i1s})^2$. A technique called diagonal quadratic approximation, introduced by Ruszczyński [233], provides separability using iteratively the approximations

$$ab \approx b_k a + a_k b - a_k b_k$$

and

$$(a-b)^2 \approx (a-b_k)^2 + (b-a_k)^2 - (a_k - b_k)^2.$$

Employing these approximations for the quadratic terms of LA, the subproblem becomes a separable mixed integer quadratic programming (MIQP) problem that can be solved using parallelism. After solving each MIQP problem, the approximations a_k and b_k are updated and the process continues (not without some technicalities) until convergence.

Problem 2.7: Maximum Entropy Reconstruction [223, 257]

Newton diffraction measurements allow one to obtain values of the magnetic structure factor of superconductor materials as $SrFe_2As_2$ for different wavelengths [223]. There is a direct mathematical relationship between the magnetic density distribution and the magnetic structure factor, so that, if enough data are available, reliable magnetic density information can be obtained from Fourier inversion of the structure factor. An alternative to the direct Fourier inversion of the data to obtain the magnetization density is to carry out a maximum entropy reconstruction of the moment density [223].

The basic idea behind maximum entropy is that there may be a number of possible moment densities that fit the form factor data equally well within the experimental uncertainties. Thus, to obtain a representative moment density, the strategy is to search for moment densities which maximize entropy but are constrained to minimize the fit to the data. This technique picks the most likely magnetization density consistent with the data [223].

The structure of the system so far obtained is similar to that of the maximum entropy optimization strategy employed for phase determination in X-ray crystallography, a case in which a pure Newtonian strategy is satisfactory (see Wu et al. [257]). On the other hand, in the case of the determination of magnetic density distribution from structural factors of $SrFe_2As_2$, an Augmented Lagrangian technique is used to maximize the entropy subject to compatibility of the measurements.

Problem 2.8: Structural optimization

In [133], large constrained optimization problems defined by

$$\text{Minimize } f_0(x) \text{ subject to } f_j(x) \le 0, j = 1, \ldots, m, \text{ and } \ell \le x \le u$$

are considered. The focus is on problems in which the evaluation of the objective and the constraint functions is computationally very expensive, as in the case of structural optimization with a finite element type of simulation in the loop. The expensiveness of the evaluation of the functions f_j and the particular form of these functions in structural applications lead the optimization developers in these areas to propose solution of the problems by means of sequential approximate optimization (SAO) methods.

In the SAO philosophy the original problem is replaced by a surrogate model built with information collected at the present major iteration. Therefore, "subproblems" are constrained optimization problems as the original problem, but they are considerably simpler. On the other hand, subproblems should be similar enough to the original problem so that a sequence of subproblem solutions converges to the original desired minimizer.

In [133], given the iterate x^k, the functions f_j are approximated by separable quadratics:

$$f_j^k(x) = f_j(x^k) + \nabla f_j(x^k)^T (x - x^k) + \sum_{i=1}^{n} c_{ik}(x_i - x_i^k)^2,$$

where the coefficients c_{ik} are obtained using information at x^k and x^{k-1}. In the context of finite element calculations, the gradients $\nabla f_j(x^k)$ can be obtained by solving appropriate

adjoint models. The subproblems defined by

$$\text{Minimize } f_0^k(x) \text{ subject to } f_j^k(x) \leq 0, j = 1, \ldots, m, \text{ and } \ell \leq x \leq u$$

are solved using Augmented Lagrangians, the overall procedure able to solve practical problems of many variables in affordable computational time.

2.2 ▪ Review and summary

The examples presented in this chapter concern constrained optimization problems that have been handled in practice using Augmented Lagrangians or related methods. We wish to emphasize that this book is directed to people who deal with nonacademic applications for which a good knowledge of the Augmented Lagrangian framework should be useful. We support the point of view that clear mathematical theory should help practitioners to choose the best alternatives for their practical problems. Mathematical theory is not the only criterion for the choice of a method, although clearly it is one of them.

2.3 ▪ Further reading

Almost every textbook on numerical optimization contains examples of engineering and science applications [40, 113, 125, 182, 213]. We particularly recommend the book by Edgar, Himmelblau, and Lasdon with its applications to chemical engineering [105] and the textbook of Bartholomew-Biggs involving financial applications [29].

2.4 ▪ Problems

2.1 Prove that problem (2.1) can be equivalently formulated using as an independent variable an $n_b \times n_e$ matrix that represents an orthonormal basis of the range-subspace of P. Discuss possible advantages and disadvantages of this formulation.

2.2 Formulate a packing molecule problem maximizing the minimal distance between any pair of molecules. Use an auxiliary variable to avoid nonsmooth inconveniences of the minimum function.

2.3 Define the image registration problem with $N = 1$ and interpret it geometrically.

2.4 Define the cryopreservation problem employing $2(N + 1)$ constraints and discuss possible advantages and disadvantages.

2.5 The problem of minimization with VaR constraints may be formulated by saying that one seeks feasible portfolios such that under a fixed number of scenarios, the expected loss is smaller than some tolerance. This formulation suggests a fixed-point procedure that consists of fixing the required number of scenarios as constraints and iterating until repetition of the scenarios. Develop this idea [42, 122] and discuss possible advantages and disadvantages.

2.6 Develop the ideas of the capacity expansion planning problem for problems in which the constraints have the following structure: block-angular and staircase. In the block-angular case, constraints are divided into blocks which are connected only by a small group of "master" constraints. In the staircase situation, each block is connected to the following one by means of a few variables.

2.7 Formulate the idea of maximum energy reconstruction for general estimation problems with constraints.

2.8 Write a flux diagram to understand the way in which diagonal approximations are used in the structural optimization problem.

Chapter 3
Optimality Conditions

We consider the problem of minimizing a continuous real function f over a closed set D contained in $\mathbb{R}^n$. If D is nonempty, we say that the problem is feasible. We say that $x^* \in D$ is a global solution (or global minimizer) of this problem if $f(x^*) \leq f(x)$ for all $x \in D$. We say that $x^* \in D$ is a strict global minimizer if $f(x^*) < f(x)$ for all $x \in D$. A local minimizer is a point $x^* \in D$ such that for some $\varepsilon > 0$ and for all $x \in D \cap B(x^*, \varepsilon)$ one has that $f(x^*) \leq f(x)$. If, in addition, $f(x^*) < f(x)$ for all $x \in D \cap B(x^*, \varepsilon)$ such that $x \neq x^*$, we say that x^* is a strict local minimizer. The value of f at a (local or global) minimizer will be called (local or global) minimum. If D is compact (closed and bounded in $\mathbb{R}^n$), the Bolzano–Weierstrass theorem guarantees that a global minimizer of f over D necessarily exists (see Apostol [20]). If a local minimizer x^* is an interior point of D and f admits first partial derivatives, we know from elementary calculus that $\nabla f(x^*) = 0$ [20].

In constrained optimization, one aims to find the lowest possible value of an objective function within a given domain. Strictly speaking, this is the goal of global optimization, which is usually very hard, especially in the case of large-scale problems: a full guarantee that a given point is a global minimizer of a continuous function can be obtained, if additional properties of the function are not available, only after visiting a dense set in the domain. Of course, such a search (which evokes the impossible task of evaluating the objective function and the constraints at all the points of $\mathbb{R}^n$) is not affordable except in low-dimensional problems. For solving medium to large-scale constrained optimization problems, one usually relies on "affordable algorithms" which do not need dense set evaluations at all and, in the best case, only guarantee convergence to points that satisfy some necessary optimality condition. As its name suggests, in the present context, a necessary optimality condition is a condition that necessarily holds at every local (or global) minimizer. Unless we have additional information on the problem, points that satisfy necessary optimality conditions are only *probable* global (or local) minimizers.

Although an optimality condition is a formal mathematical property, the concept of *usefulness* is associated with it. An optimality condition should be *strong* in order to maximize the probability that points that fulfill it also possess minimization properties. However, strength is not the only desirable property of optimality conditions. Since we always solve optimization problems by means of iterative methods, *useful* optimality conditions are those that are associated with affordable methods. In this chapter, we will show that all local minimizers of constrained optimization problems satisfy a useful "sequential" optimality condition. We will see that this condition is the one that can be expected (and in some sense, computed) at successful iterations of the main algorithms studied in this book. Points that satisfy necessary optimality conditions are usually said to be *stationary*.

Throughout this chapter, we will assume that the objective function and the functions that define the constraints admit continuous first derivatives for all $x \in \mathbb{R}^n$. The perceptive reader may verify that this assumption can be relaxed in several places. For example, sometimes only existence (not necessarily continuity) of the derivatives is necessary and, many times, even existence is necessary only on a suitable region, instead of the whole $\mathbb{R}^n$.

3.1 ▪ Explicit functional constraints

We start by studying the case in which D is described by a system of continuous equalities and inequalities. Namely, $x \in D$ if and only if the following constraints are fulfilled:

$$h(x) = 0 \text{ and } g(x) \leq 0, \tag{3.1}$$

where $h : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^n \to \mathbb{R}^p$ are continuous. Given $x \in D$, the inequality constraints for which $g_i(x) < 0$ are said to be "inactive." The remaining constraints, which include those defined by equalities, are said to be "active" at the feasible point x.

3.1.1 ▪ Approximate Karush–Kuhn–Tucker condition

In the following theorem, we prove that given a local minimizer x^* of $f(x)$ subject to $x \in D$, there exists a sequence $\{x^k\}$ that converges to x^* such that the gradient of f at x^k is, asymptotically, a linear combination of the gradients of the active constraints at x^k, with the "correct" signs for the coefficients that correspond to inequalities.

Theorem 3.1. *Assume that x^* is a local minimizer of $f(x)$ subject to $x \in D$, where D is given by (3.1) and f, h, and g admit first derivatives in a neighborhood of x^*. Then, there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that*

$$\lim_{k\to\infty} x^k = x^*, \tag{3.2}$$

$$\lim_{k\to\infty} ||\nabla f(x^k) + \nabla h(x^k)\lambda^k + \nabla g(x^k)\mu^k|| = 0, \tag{3.3}$$

and

$$\lim_{k\to\infty} \min\{-g_i(x^k), \mu_i^k\} = 0 \text{ for all } i = 1, \ldots, p.$$

Proof. By the hypothesis, there exists $\varepsilon > 0$ such that x^* is a global minimizer of $f(x)$ on $D \cap B(x^*, \varepsilon)$. Therefore, x^* is the unique global minimizer of $f(x) + ||x - x^*||_2^2$ on $D \cap B(x^*, \varepsilon)$.

For all $k \in \mathbb{N}$, consider the problem

$$\text{Minimize } f(x) + k\left[||h(x)||_2^2 + ||g(x)_+||_2^2\right] + ||x - x^*||_2^2 \text{ subject to } ||x - x^*|| \leq \varepsilon. \tag{3.4}$$

By the Bolzano–Weierstrass theorem, this problem admits a solution $x^k \in B(x^*, \varepsilon)$. Since $||x^k - x^*|| \leq \varepsilon$ for all $k \in \mathbb{N}$ and $B(x^*, \varepsilon)$ is compact, there exists $K \underset{\infty}{\subseteq} \mathbb{N}$ and $z^* \in B(x^*, \varepsilon)$ such that $\lim_{k\in K} x^k = z^*$.

By the definition of x^k, we have that

$$\begin{aligned} f(x^k) + k\left[||h(x^k)||_2^2 + ||g(x^k)_+||_2^2\right] + ||x^k - x^*||_2^2 \\ \leq f(x^*) + k\left[||h(x^*)||_2^2 + ||g(x^*)_+||_2^2\right] + ||x^* - x^*||_2^2. \end{aligned}$$

Therefore, since $||h(x^*)||_2 = ||g(x^*)_+||_2 = 0$,

$$f(x^k) + k\left[||h(x^k)||_2^2 + ||g(x^k)_+||_2^2\right] + ||x^k - x^*||_2^2 \leq f(x^*) \tag{3.5}$$

for all $k \in K$. Dividing by k both sides of (3.5), using continuity of f, h, and g, and taking limits for $k \in K$, we obtain

$$\|h(z^*)\|_2^2 + \|g(z^*)_+\|_2^2 = 0.$$

Thus, $z^* \in D$.

By (3.5) we have that for all $k \in K$,

$$f(x^k) + \|x^k - x^*\|_2^2 \le f(x^*).$$

Taking limits on both sides of this inequality, we obtain that $f(z^*) + \|z^* - x^*\|_2^2 \le f(x^*)$. But, since x^* is the unique global minimizer of $f(x) + \|x - x^*\|_2^2$ subject to $x \in D \cap B(x^*, \varepsilon)$ and $\|z^* - x^*\| \le \varepsilon$, we have that $z^* = x^*$. Therefore,

$$\lim_{k \in K} x^k = x^*. \tag{3.6}$$

Clearly, for $k \in K$ large enough, we have that $\|x^k - x^*\| < \varepsilon$. Then, annihilating the gradient of the objective function in (3.4), we get

$$\nabla f(x^k) + \sum_{i=1}^{m} \Big[2k h_i(x^k)\Big] \nabla h_i(x^k) + \sum_{g_i(x^k) > 0} \Big[2k g_i(x^k)\Big] \nabla g_i(x^k) + 2(x^k - x^*) = 0. \tag{3.7}$$

Define $\lambda^k = 2k h(x^k)$ and $\mu^k = 2k g(x^k)_+$. Then, by (3.6) and (3.7),

$$\lim_{k \in K} \|\nabla f(x^k) + \sum_{i=1}^{m} \lambda_i^k \nabla h_i(x^k) + \sum_{i=1}^{p} \mu_i^k \nabla g_i(x^k)\| = 0,$$

where $\mu_i^k = 0$ if $g_i(x^k) < 0$. Therefore, $\min\{-g_i(x^k), \mu_i^k\} = 0$ for all $i = 1, \dots, p$ and $k \in K$. This completes the proof. □

Observe that Theorem 3.1 can be reformulated saying that for all $\delta > 0$ and $\varepsilon > 0$ there exist $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^m$, and $\mu \in \mathbb{R}^p_+$ such that

$$\|\nabla f(x) + \nabla h(x)\lambda + \nabla g(x)\mu\| \le \varepsilon, \tag{3.8}$$

$$\|h(x)\| \le \delta \text{ and } \|g(x)_+\| \le \delta, \tag{3.9}$$

and

$$\|\min\{-g(x), \mu\}\| \le \varepsilon. \tag{3.10}$$

The properties (3.8), (3.9), and (3.10) can be verified at every iteration of any algorithm that computes "primal-dual" iterates x^k, λ^k, and μ^k. In fact, these properties have the basic form of the stopping criteria used in those algorithms. This is the motivation for the following definition.

Definition 3.1. *We say that $x^* \in D$ satisfies the approximate Karush–Kuhn–Tucker (AKKT) condition with respect to the problem of minimizing $f(x)$ with constraints given by (3.1) if there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that*

$$\lim_{k \to \infty} x^k = x^*,$$

$$\lim_{k \to \infty} \|\nabla f(x^k) + \nabla h(x^k)\lambda^k + \nabla g(x^k)\mu^k\| = 0, \tag{3.11}$$

and

$$\lim_{k \to \infty} \min\{-g_i(x^k), \mu_i^k\} = 0 \text{ for all } i = 1, \dots, p. \tag{3.12}$$

Theorem 3.1 may be rephrased by saying that any local minimizer of $f(x)$ subject to the constraints (3.1) satisfies the AKKT condition.

The following theorem establishes an equivalent form of AKKT that will be useful in convergence proofs.

Theorem 3.2. *A feasible point x^* satisfies AKKT with respect to the minimization of $f(x)$ subject to (3.1) if and only if there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that*

$$\lim_{k\to\infty} x^k = x^*,$$

$$\lim_{k\to\infty} ||\nabla f(x^k) + \nabla h(x^k)\lambda^k + \nabla g(x^k)\mu^k|| = 0, \tag{3.13}$$

and

$$\mu_i^k = 0 \text{ if } g_i(x^*) < 0 \text{ for all } i = 1,\dots,p. \tag{3.14}$$

Proof. Assume that x^* satisfies the AKKT condition and $i \in \{1,\dots,p\}$ is such that $g_i(x^*) < 0$. Then, by (3.12), we have that $\lim_{k\to\infty} \mu_i^k = 0$. Therefore, by (3.11) and the boundedness of $\{\nabla g(x^k)\}$, we obtain

$$\lim_{k\to\infty} ||\nabla f(x^k) + \nabla h(x^k)\lambda^k + \sum_{g_i(x^*)=0} \mu_i^k \nabla g_i(x^k)|| = 0.$$

This implies the fulfillment of (3.13) and (3.14).

Conversely, assume that the sequence $\{x^k\}$ converges to x^* and that (3.13) and (3.14) take place. If $g_i(x^*) = 0$, by continuity, we have that $\lim_{k\to\infty} g_i(x^k) = 0$. Then, since $\mu_i^k \geq 0$, we obtain that (3.12) holds. This completes the proof. □

If $\delta > 0$ and $\varepsilon > 0$ are "small," the fulfillment of (3.8), (3.9), and (3.10) suggests that, probably, x^k is close to a solution of the constrained minimization problem.

3.1.2 ▪ AKKT and constraint qualifications

When (3.8), (3.9), and (3.10) hold with $\delta = \varepsilon = 0$ we say that the (classical) KKT condition [40, 83, 113, 182] holds.

Definition 3.2. *We say that $x^* \in D$ satisfies the KKT condition with respect to the problem of minimizing $f(x)$ with constraints given by (3.1) if there exist $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^p_+$ such that*

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* + \nabla g(x^*)\mu^* = 0 \tag{3.15}$$

and, for all $i = 1,\dots,p$,

$$\mu_i^* = 0 \text{ whenever } g_i(x^*) < 0. \tag{3.16}$$

AKKT is a "sequential" optimality condition in the sense that KKT is a "pointwise" condition. Points that satisfy the KKT conditions will be called "KKT points." According to this definition, (i) KKT points need to be feasible; (ii) by (3.15), the gradient of f must be a linear combination of the gradients of the constraints with the correct sign for the multipliers associated with the inequalities; and (iii) the complementarity condition (3.16), according to which multipliers corresponding to inactive inequality constraints should be zero, must hold.

Local minimizers of f subject to $h(x) = 0$ and $g(x) \leq 0$ may *not* be KKT points (but they are AKKT points!). For example, consider the problem

$$\text{Minimize } x_1 \text{ subject to } ||x||_2^2 = 0, \tag{3.17}$$

whose unique global solution is $x^* = 0$, which does not satisfy the KKT condition. On the other hand, the AKKT condition holds at x^*. See Figures 3.1 and 3.2.

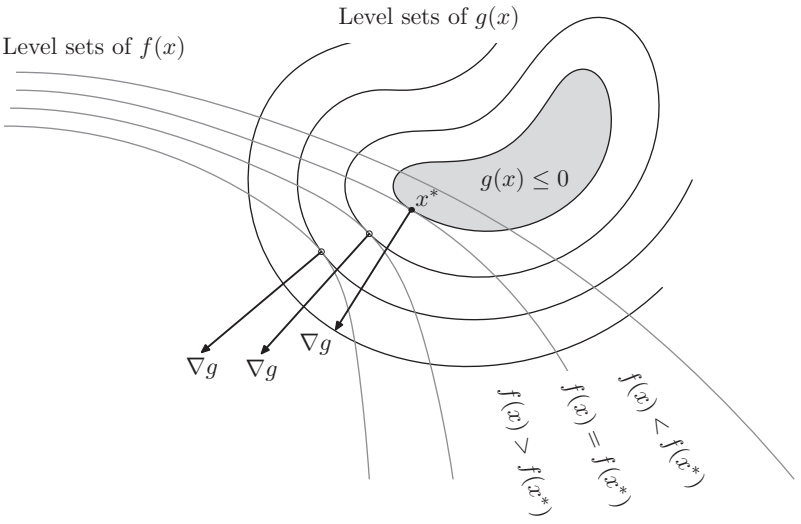

Figure 3.1. *AKKT sequence that converges to a KKT point.*

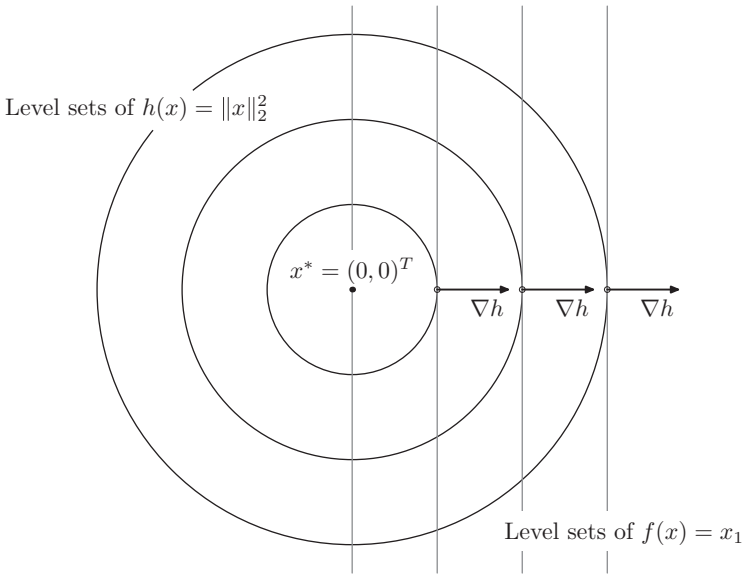

Figure 3.2. *AKKT sequence that converges to a point which does not satisfy the KKT condition.*

Constraint qualifications are properties of the constraints of optimization problems that, when satisfied at a local minimizer x^*, independently of the objective function, imply that x^* fulfills the KKT condition. In other words, if CQ is a constraint qualification, the proposition "KKT or not CQ" is a necessary optimality condition. As a consequence, weak constraint qualifications produce strong optimality conditions. The best-known constraint qualifications are the linear independence of the gradients of active constraints

(LICQ) and Mangasarian–Fromovitz (MFCQ). The constant positive linear dependence (CPLD) condition is a constraint qualification weaker than LICQ and MFCQ and was introduced by Qi and Wei [221]. Andreani, Martínez, and Schuverdt [16] elucidated the status of CPLD with respect to other constraint qualifications.

Note that, unlike KKT, the AKKT condition holds at every local minimizer of an optimization problem independently of the fulfillment of constraint qualifications.

Under which conditions on the constraints of a problem do we have that AKKT implies KKT? Certainly, if P is one of these conditions, P must be a constraint qualification. In fact, if x^* is a local minimizer, by Theorem 3.1, x^* satisfies AKKT. Then, by the property P, x^* is a KKT point. In other words, if the property P is such that P + AKKT $\Rightarrow$ KKT, then P is a constraint qualification. (The reciprocal is not true. For example, the quasi-normality constraint qualification [40] does not satisfy P + AKKT $\Rightarrow$ KKT. Consider the constraints $x_2 = 0$ and $x_1 x_2 = 0$ at the point $(0,1)^T$.)

We will say that a property P is a *strict constraint qualification* (SCQ) if the implication P + AKKT $\Rightarrow$ KKT is true. The most general (weakest) SCQ is given below.

Definition 3.3. *Assume that D is given by (3.1). Let $J \subseteq \{1,\dots,p\}$ be the indices of the active inequality constraints at x^*. We say that the U-condition holds at $x^* \in D$ if, for all $v \in \mathbb{R}^n$, whenever*

$$v + \nabla h(x^k)\lambda^k + \sum_{j\in J} \mu_j^k \nabla g_j(x^k) + E_k = 0$$

with $\lambda^k \in \mathbb{R}^m$, $\mu^k \in \mathbb{R}^p_+$, $x^k \to x^*$, and $E_k \to 0$, there exist $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p_+$ such that

$$v + \nabla h(x^*)\lambda + \sum_{j\in J} \mu_j \nabla g_j(x^*) = 0.$$

Theorem 3.3. *Assume that the U-condition is satisfied by the constraints of (3.1) at $x^* \in D$. Then, for every objective function f, AKKT implies KKT.*

Proof. By AKKT, we have that there exist $x^k \in \mathbb{R}^n$, $E_k \in \mathbb{R}^n$, $\lambda^k \in \mathbb{R}^m$, and $\mu^k \in \mathbb{R}^p_+$ such that, for all $k \in \mathbb{N}$, $x^k \to x^*$,

$$\nabla f(x^k) + \nabla h(x^k)\lambda^k + \sum_{j\in J} \mu_j^k \nabla g_j(x^k) + E_k = 0,$$

and $E_k \to 0$. Therefore,

$$\nabla f(x^*) + \nabla h(x^k)\lambda^k + \sum_{j\in J} \mu_j^k \nabla g_j(x^k) + E_k + \nabla f(x^k) - \nabla f(x^*) = 0.$$

Thus, by the continuity of ∇f and the definition of the U-condition, there exist $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p_+$ such that

$$\nabla f(x^*) + \nabla h(x^*)\lambda + \sum_{j\in J} \mu_j \nabla g_j(x^*) = 0.$$

This completes the proof. □

As mentioned above, this implies that the U-condition is a constraint qualification. Now, we are going to prove that the U-condition is the weakest constraint qualification P for which P + AKKT $\Rightarrow$ KKT.

Theorem 3.4. *Let P be a condition of constraints of the form $h(x) = 0$ and $g(x) \leq 0$ (as in (3.1)) with the property that P + AKKT ⇒ KKT. Then, P implies the U-condition.*

Proof. Let $v \in \mathbb{R}^n$ be such that

$$v + \nabla h(x^k)\lambda^k + \sum_{j \in J} \mu_j^k \nabla g_j(x^k) + E_k = 0$$

with $\mu^k \in \mathbb{R}^p_+$, $\lambda^k \in \mathbb{R}^m$, $x^k \to x^*$, and $E_k \to 0$. Define $f(x) = v^T x$ for all $x \in \mathbb{R}^n$. Then, AKKT is satisfied for this function with the constraints (3.1). Since P + AKKT ⇒ KKT, it follows that there exist $\lambda \in \mathbb{R}^m$, $\mu \in \mathbb{R}^p_+$ such that

$$\nabla f(x^*) + \nabla h(x^*)\lambda + \sum_{j \in J} \mu_j \nabla g_j(x^*) = 0.$$

Since $\nabla f(x^*) = v$ and v was arbitrary, the U-condition holds at x^*. □

We saw that a property P is an SCQ if and only if it implies the U-condition. The results on the U-condition have a clear geometrical meaning. In order to verify that, for a given point x^*, AKKT ⇒ KKT, we only need to verify that this proposition is true for linear objective functions. It is interesting to discover sufficient properties that imply the U-condition and have other geometrical interpretations. These properties are necessarily stronger than (or perhaps equivalent to) the U-condition but their geometrical meaning should provide some insight on the conditions under which AKKT implies KKT. The constant positive generator (CPG) condition was defined in [14], where it was proved that CPG + AKKT ⇒ KKT. Thus, CPG is a constraint qualification too and, by Theorem 3.4, implies the U-condition.

Definition 3.4. *Assume that D is given by (3.1) and $x^* \in D$. Define $I = \{1, \ldots, m\}$. Let $J \subseteq \{1, \ldots, p\}$ be the indices of the active inequality constraints at x^*. Let J_- be the set of indices $\ell \in J$ such that there exist $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ and $\mu_j \in \mathbb{R}_+$ for all $j \in J$ satisfying*

$$-\nabla g_\ell(x^*) = \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) + \sum_{j \in J} \mu_j \nabla g_j(x^*). \tag{3.18}$$

Define $J_+ = J \setminus J_-$. We say that the CPG condition holds at x^* if there exists $I' \subseteq I$ and $J' \subseteq J_-$ such that the following hold:

1. *The gradients $\nabla h_i(x^*)$ and $\nabla g_j(x^*)$ indexed by $i \in I'$ and $j \in J'$ are linearly independent.*
2. *For all x in a neighborhood of x^*, if*

$$z = \sum_{i=1}^{m} \lambda_i' \nabla h_i(x) + \sum_{j \in J} \mu_j' \nabla g_j(x)$$

with $\mu_j' \geq 0$ for all $j \in J$, then for all $i \in I'$, $\ell \in J'$, and $j \in J_+$, there exist $\lambda_i'' \in \mathbb{R}$, $\lambda_\ell''' \in \mathbb{R}$, and $\mu_j'' \in \mathbb{R}_+$ such that

$$z = \sum_{i \in I'} \lambda_i'' \nabla h_i(x) + \sum_{\ell \in J'} \lambda_\ell''' \nabla g_\ell(x) + \sum_{j \in J_+} \mu_j'' \nabla g_j(x).$$

Theorem 3.5. *Assume that $x^* \in D$ satisfies CPG and fulfills the AKKT condition corresponding to the minimization of $f(x)$ with the constraints (3.1). Then, x^* is a KKT point of this problem.*

Proof. By the AKKT condition and Theorem 3.2, there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that x^k tends to x^* and $\lim_{k\to\infty} ||E_k|| = 0$, where

$$E_k = \nabla f(x^k) + \nabla h(x^k)\lambda^k + \sum_{g_i(x^*)=0} \mu_i^k \nabla g(x^k). \tag{3.19}$$

By the second item of the definition of CPG, there exist subsets $I' \subseteq I$ and $J' \subseteq J_-$ such that, for all k large enough, $i \in I'$, $\ell \in J'$, and $j \in J_+$, there exist $\lambda_i''(k) \in \mathbb{R}$, $\lambda_\ell'''(k) \in \mathbb{R}$, and $\mu_j''(k) \in \mathbb{R}_+$ such that

$$E_k - \nabla f(x^k) = \sum_{i\in I'} \lambda_i''(k)\nabla h_i(x^k) + \sum_{\ell\in J'} \lambda_\ell'''(k)\nabla g_\ell(x^k) + \sum_{j\in J_+} \mu_j''(k)\nabla g_j(x^k). \tag{3.20}$$

Define $M_k = \max\{|\lambda_i''(k)|, |\lambda_\ell'''(k)|, \mu_j''(k), i \in I', \ell \in J', j \in J_+\}$.

If the sequence $\{M_k\}$ is bounded, taking limits for an appropriate subsequence, we deduce that for all $i \in I'$, $\ell \in J'$, and $j \in J_+$, there exist $\lambda_i'' \in \mathbb{R}$, $\lambda_\ell''' \in \mathbb{R}$, and $\mu_j'' \in \mathbb{R}_+$ such that

$$-\nabla f(x^*) = \sum_{i\in I'} \lambda_i''\nabla h_i(x^*) + \sum_{\ell\in J'} \lambda_\ell'''\nabla g_\ell(x^*) + \sum_{j\in J_+} \mu_j''\nabla g_j(x^*).$$

Since $J' \subseteq J_-$, using (3.18) for all the indices ℓ such that $\lambda_\ell''' < 0$, we obtain the KKT condition.

Consider now the case in which $\{M_k\}$ is unbounded. Dividing (3.20) by M_k and taking limits for an appropriate subsequence, we obtain that there exist $\tilde\lambda_i \in \mathbb{R}$, $\tilde\lambda_\ell \in \mathbb{R}$, and $\tilde\mu_j \in \mathbb{R}_+$ such that

$$\sum_{i\in I'} \tilde\lambda_i \nabla h_i(x^*) + \sum_{\ell\in J'} \tilde\lambda_\ell \nabla g_\ell(x^*) + \sum_{j\in J_+} \tilde\mu_j \nabla g_j(x^*) = 0 \tag{3.21}$$

and

$$\max\{|\tilde\lambda_i|, |\tilde\lambda_\ell|, \tilde\mu_j, i \in I', \ell \in J', j \in J_+\} > 0. \tag{3.22}$$

Assume for a moment that there exists $\bar j \in J_+$ such that $\tilde\mu_{\bar j} > 0$. Then,

$$-\tilde\mu_{\bar j}\nabla g_{\bar j}(x^*) = \sum_{i\in I'} \tilde\lambda_i \nabla h_i(x^*) + \sum_{\ell\in J'} \tilde\lambda_\ell \nabla g_\ell(x^*) + \sum_{j\in J_+, j\neq \bar j} \tilde\mu_j \nabla g_j(x^*).$$

Therefore, $\bar j$ should belong to J_-, contradicting the fact that $\bar j \in J_+$. Therefore, $\tilde\mu_j = 0$ for all $j \in J_+$. But, by the first item of the definition of CPG, the gradients $\nabla h_i(x^*), \nabla g_\ell(x^*)$ with $i \in I'$ and $\ell \in J'$ are linearly independent. Then, all the coefficients in (3.21) are null, which contradicts (3.22). □

In [14], it is also proved that the CPLD condition implies CPG. This property implies that CPLD is a constraint qualification and that CPLD + AKKT ⇒ KKT. Since the proof that CPLD ⇒ CPG is rather involved, we will give here a direct proof of CPLD + AKKT ⇒ KKT.

Definition 3.5. *Assume that D is given by (3.1). We say that the CPLD condition holds at $x^* \in D$ if, whenever there exist $I_1 \subseteq \{1,\dots,m\}$, $I_2 \subseteq \{1,\dots,p\}$, $\lambda^* \in \mathbb{R}^m$, and $\mu^* \in \mathbb{R}^p_+$ satisfying*

$$g_i(x^*) = 0 \text{ for all } i \in I_2,$$

$$\lambda_i^* = 0 \text{ if } i \notin I_1,\ \mu_i^* = 0 \text{ if } i \notin I_2,\ \|\lambda^*\| + \|\mu^*\| > 0, \text{ and}$$

$$\sum_{i\in I_1} \lambda_i^* \nabla h_i(x^*) + \sum_{i\in I_2} \mu_i^* \nabla g_i(x^*) = 0,$$

we have that there exists $\varepsilon > 0$ such that the gradients $\nabla h_i(x)$ and $\nabla g_j(x)$ with $i \in I_1$ and $j \in I_2$ are linearly dependent for all $x \in B(x^*, \varepsilon)$.

When a subset of gradients of active constraints at x^* is linearly dependent with nonnegative coefficients corresponding to the inequality constraints, these gradients used to be called "positively linearly dependent." Roughly speaking, the CPLD condition says that, whenever a subset of gradients of active constraints is positively linearly dependent at x^*, the same gradients are linearly dependent in a neighborhood of x^*. MFCQ states that there are no positively linearly dependent gradients at x^* and LICQ says that the gradients of active constraints at x^* are linearly independent. Therefore, both LICQ and MFCQ imply CPLD.

We aim to prove that CPLD is a constraint qualification. This will be a direct consequence of Theorem 3.6 below. For completeness, the classical Carathéodory's lemma will be stated first.

Lemma 3.1. *Assume that*

$$u = \sum_{i=1}^{m} \lambda_i v^i + \sum_{j=1}^{p} \mu_j w^j,$$

where $v^i \in \mathbb{R}^n$ for all $i = 1,\dots,m$, $w^j \in \mathbb{R}^n$ for all $j = 1,\dots,p$, and $\mu_j \geq 0$ for all $j = 1,\dots,p$. Then, there exist $I \subseteq \{1,\dots,m\}$, $J \subseteq \{1,\dots,p\}$, $\{\lambda_i'\}_{i\in I} \subseteq \mathbb{R}$, and $\{\mu_j'\}_{j\in J} \subseteq \mathbb{R}_+$ such that

$$u = \sum_{i\in I} \lambda_i' v^i + \sum_{j\in J} \mu_j' w^j$$

and the vectors v^i and w^j with $i \in I$ and $j \in J$ are linearly independent.

Proof. Without loss of generality, assume that $\lambda_i \neq 0$ and $\mu_j > 0$ for all $i = 1,\dots,m$ and $j = 1,\dots,p$.

Assume that the vectors $v^1,\dots,v^m,w^1,\dots,w^p$ are linearly dependent. Then, there exist scalars α_i and β_j such that $\sum_{i=1}^m |\alpha_i| + \sum_{j=1}^p |\beta_j| > 0$ and

$$\sum_{i=1}^{m} \alpha_i v^i + \sum_{j=1}^{p} \beta_j w^j = 0.$$

Thus, by the hypothesis, we have that

$$u = \sum_{i=1}^{m} (\lambda_i - t\alpha_i) v^i + \sum_{j=1}^{p} (\mu_j - t\beta_j) w^j$$

for all $t \in \mathbb{R}$. For $t = 0$, all the coefficients in the equality above are nonnull. Let $t_{\min}$ be the minimum modulus t such that at least one of the coefficients $\lambda_i - t\alpha_i$ or $\mu_j - t\beta_j$ is

null. Then,

$$u = \sum_{i=1}^{m} (\lambda_i - t_{\min}\alpha_i)v^i + \sum_{j=1}^{p} (\mu_j - t_{\min}\beta_j)w^j.$$

Clearly, $\mu_j - t_{\min}\beta_j \geq 0$ for all j, but we were able to write u as a linear combination of, at most, $m + p - 1$ vectors. This process can be repeated while the vectors whose linear combination is u are linearly dependent. So, at the end, u can be written as a combination of linearly independent vectors with nonnegative coefficients corresponding to each w^j, as we wanted to prove. □

Theorem 3.6. *Assume that $x^* \in D$ satisfies CPLD and fulfills the AKKT condition corresponding to the minimization of $f(x)$ with the constraints (3.1). Then, x^* is a KKT point of this problem.*

Proof. By the AKKT condition and Theorem 3.2, there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that x^k tends to x^* and $\lim_{k\to\infty} ||E_k|| = 0$, where

$$E_k = \nabla f(x^k) + \nabla h(x^k)\lambda^k + \sum_{g_i(x^*)=0} \mu_i^k \nabla g(x^k). \tag{3.23}$$

By Lemma 3.1, for all $k \in \mathbb{N}$, there exist subsets

$$I_1^k \subseteq \{1, \ldots, m\},$$

$$I_2^k \subseteq \{i \in \{1, \ldots, p\} \mid g_i(x^*) = 0\},$$

and scalars $\hat{\lambda}_i^k$, $i \in I_1^k$, and $\hat{\mu}_i^k \geq 0$, $i \in I_2^k$, such that the gradients $\nabla h_i(x^k)$, $i \in I_1^k$, $\nabla g_i(x^k)$, $i \in I_2^k$, are linearly independent and

$$E_k - \nabla f(x^k) = \sum_{i\in I_1^k} \hat{\lambda}_i^k \nabla h_i(x^k) + \sum_{i\in I_2^k} \hat{\mu}_i^k \nabla g_i(x^k).$$

Moreover, there exist I_1 and I_2 such that $I_1 = I_1^k$ and $I_2 = I_2^k$ infinitely many times. Thus, there exists $K \underset{\infty}{\subseteq} \mathbb{N}$ such that

$$E_k - \nabla f(x^k) = \sum_{i\in I_1} \hat{\lambda}_i^k \nabla h_i(x^k) + \sum_{i\in I_2} \hat{\mu}_i^k \nabla g_i(x^k) \tag{3.24}$$

for all $k \in K$.

Define, for all $k \in K$,

$$M_k = \max\{||\hat{\lambda}^k||_\infty, ||\hat{\mu}^k||_\infty\}.$$

If $\{M_k\}$ is bounded, taking limits on both sides of (3.24) for $k \in K$, we obtain the desired KKT result.

Consider now the case in which M_k tends to infinity. Dividing both sides of (3.24) by M_k and using the boundedness of $\{\nabla f(x^k)\}$, we obtain that

$$E_k' = \sum_{i\in I_1} \tilde{\lambda}_i^k \nabla h_i(x^k) + \sum_{i\in I_2} \tilde{\mu}_i^k \nabla g_i(x^k), \tag{3.25}$$

where $\lim_{k\to\infty} ||E_k'|| = 0$, the coefficients $\tilde{\lambda}_i^k \in \mathbb{R}$ and $\tilde{\mu}_i^k \in \mathbb{R}_+$ are bounded, and, for each k, the modulus of at least one of these coefficients is equal to 1.

Taking limits on both sides of (3.25) for $k \in K$, we obtain that a nontrivial linear combination (with nonnegative coefficients corresponding to inequalities) of the gradients involved in the right-hand side of (3.25) vanishes at x^*. Therefore, by the CPLD condition, these gradients should be linearly dependent at x^k for k large enough, which is a contradiction. □

As in the case of the U-condition and the CPG condition, from the property CPLD + AKKT ⇒ KKT, it can be deduced that CPLD is a constraint qualification.

Very recently, constraint qualifications weaker than CPLD but generally stronger than CPG were introduced in [13, 14, 180, 181, 203, 204]. It has been proved in [11] that the U-condition is strictly weaker than CPG.

3.2 ▪ Including abstract constraints

We wish to consider now a more general form of the feasible set D. Instead of (3.1) we will define

$$D = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0,\ \text{and } x \in \Omega\}, \tag{3.26}$$

where h and g are as in (3.1) and Ω is closed and convex. Recall that a convex set is defined by the property that $[x, y] \subseteq \Omega$ whenever $x, y \in \Omega$. The (Euclidean) projection $P_\Omega : \mathbb{R}^n \to \Omega$ is defined by

$$\|x - P_\Omega(x)\|_2 \leq \|x - y\|_2 \text{ for all } y \in \Omega. \tag{3.27}$$

The property (3.27) univocally defines $P_\Omega(x)$ and, thanks to the contraction property

$$\|P_\Omega(x) - P_\Omega(y)\|_2 \leq \|x - y\|_2, \tag{3.28}$$

the function P_Ω turns out to be continuous. The following nondecreasing property of projections will be used in the proof of Theorem 6.5 in Chapter 6:

$$\text{If } x \in \Omega, v \in \mathbb{R}^n, \text{and } 0 \leq t_1 < t_2, \text{then } \|P_\Omega(x + t_1 v) - x\|_2 \leq \|P_\Omega(x + t_2 v) - x\|_2. \tag{3.29}$$

In the deduction of a sequential optimality condition for the minimization of $f(x)$ subject to $x \in D$, given by (3.26), we will need to consider "subproblems" of the form

$$\text{Minimize } F(x) \text{ subject to } x \in \Omega. \tag{3.30}$$

The following lemma gives a necessary first-order optimality condition for (3.30). Points that satisfy this condition will be called stationary with respect to the problem (3.30).

Lemma 3.2. *Assume that x^* is a local minimizer of $F(x)$ subject to $x \in \Omega$, where Ω is closed and convex and F admits continuous first derivatives. Then,*

$$P_\Omega(x^* - \nabla F(x^*)) = x^*. \tag{3.31}$$

Equivalently,

$$\nabla F(x^*)^T d \geq 0 \text{ for all } d \in \mathbb{R}^n \text{ such that } x^* + d \in \Omega. \tag{3.32}$$

Proof. Assume that $x^* + d \in \Omega$. Since Ω is convex, we have that $x^* + td \in \Omega$ for all $t \in [0, 1]$. Since x^* is a local minimizer, for t small enough, we have that

$$F(x^* + td) \geq F(x^*).$$

Then, the directional derivative $\nabla F(x^*)^T d$ is nonnegative. This proves (3.32).

Define $z = P_\Omega(x^* - \nabla F(x^*))$. Then $z \in \Omega$ solves the problem

$$\text{Minimize } \frac{1}{2}||x - (x^* - \nabla F(x^*))||_2^2 \text{ subject to } x \in \Omega. \tag{3.33}$$

The gradient of the objective function of (3.33) is $x - x^* + \nabla F(x^*)$. Therefore, the scalar product between this gradient evaluated at z and $x^* - z$ (directional derivative) must be nonnegative. Thus,

$$0 \le (z - x^* + \nabla F(x^*))^T (x^* - z) = -||x^* - z||_2^2 + \nabla F(x^*)^T (x^* - z). \tag{3.34}$$

By the hypothesis and (3.32), we have that $\nabla F(x^*)^T (z - x^*) \ge 0$. Then, by (3.34), $z = x^*$. This completes the proof. □

3.2.1 ▪ AKKT in the presence of abstract constraints

Theorem 3.7. *Assume that x^* is a local minimizer of $f(x)$ subject to $x \in D$, where D is given by (3.26) and f, h, and g admit first derivatives in a neighborhood of x^*. Then, there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that*

$$\lim_{k\to\infty} x^k = x^*,$$

$$\lim_{k\to\infty} \left\| P_\Omega\left[x^k - \left(\nabla f(x^k) + \nabla h(x^k)\lambda^k + \nabla g(x^k)\mu^k\right)\right] - x^k \right\| = 0, \tag{3.35}$$

and

$$\lim_{k\to\infty} \min\{-g_i(x^k), \mu_i^k\} = 0 \text{ for all } i = 1, \ldots, p. \tag{3.36}$$

Proof. The proof is identical to that of Theorem 3.1 up to the obtention of (3.6). As in that theorem, for $k \in K$ large enough, we have that $||x^k - x^*|| < \varepsilon$. Therefore, by Lemma 3.2 (with $F(x) = f(x) + k\left[||h(x)||_2^2 + ||g(x)_+||_2^2\right] + ||x - x^*||_2^2$), we have that

$$P_\Omega\Big[x^k - \Big(\nabla f(x^k) + \sum_{i=1}^m \Big[2k h_i(x^k)\Big]\nabla h_i(x^k) + \sum_{g_i(x^k)>0} \Big[2k g_i(x^k)\Big]\nabla g_i(x^k) + 2(x^k - x^*)\Big)\Big] - x^k = 0. \tag{3.37}$$

Defining $\lambda^k = 2kh(x^k)$ and $\mu^k = g(x^k)_+$, and using that $\lim_{k\in K} ||x^k - x^*|| = 0$, the continuity of P_Ω, and (3.37), we obtain

$$\lim_{k\in K} \left\| P_\Omega\left[x^k - \left(\nabla f(x^k) + \sum_{i=1}^m \lambda_i^k \nabla h_i(x^k) + \sum_{i=1}^p \mu_i^k \nabla g_i(x^k)\right)\right] - x^k \right\| = 0,$$

where $\mu_i^k = 0$ if $g_i(x^k) < 0$. Therefore $\min\{-g_i(x^k), \mu_i^k\} = 0$ for all $i = 1, \ldots, p$ and $k \in K$. This completes the proof. □

Extending Definition 3.1, we will define the AKKT condition for the case in which the domain D is given by (3.26).

Definition 3.6. *We say that $x^* \in D$ satisfies the AKKT condition with respect to the problem of minimizing $f(x)$ with constraints given by (3.26) if there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that*

$$\lim_{k\to\infty} x^k = x^*,$$

$$\lim_{k\to\infty} \left\| P_\Omega\left[x^k - \left(\nabla f(x^k) + \nabla h(x^k)\lambda^k + \nabla g(x^k)\mu^k\right)\right] - x^k \right\| = 0, \tag{3.38}$$

and

$$\lim_{k\to\infty} \min\{-g_i(x^k), \mu_i^k\} = 0 \text{ for all } i = 1, \ldots, p. \tag{3.39}$$

3.3 ▪ Remarks on sequential optimality conditions

It is a common belief that practical problems in which constraint qualifications are not fulfilled are rare, and, as a consequence, the case in which KKT does not hold is not considered at all in algorithms and textbooks. However, the following observations are pertinent:

1. Nonfulfillment of KKT is "ignored" in algorithms probably because AKKT, which is the condition that is tested in practice, holds for all local minimizers and, in general, algorithms generate iterates for which the fulfillment of the AKKT-like stopping criterion is verified. Perhaps the occurrence of practical cases in which constraint qualifications and the KKT condition do not hold is not so rare, and we are not aware of their occurrence because the algorithms stop with a successful (and not surprising) AKKT stopping criterion.

2. For many reasons (including lack of sophistication of the user and extreme complexity of the models) constraints like $h_i(x)^2 = 0$ or $g_i(x)_+^2 = 0$ may appear, instead of their respective equivalent and better-posed counterparts $h_i(x) = 0$ and $g_i(x) \leq 0$. In those cases, constraint qualifications do not hold and, probably, the KKT condition does not hold either. It is fortunate that, even in such situations, we are able to stop close to the correct minimizers thanks to the satisfaction of AKKT-like stopping criteria.

3. Sometimes we need to solve a sequence of optimization problems depending on some physical parameter. In this *parametric optimization* case, the occurrence of "linear independence loss" is not a rarity at all [237]. Path following methods used to deal satisfactorily with the corresponding bifurcation and turning points [134, 157, 162, 226, 256] and optimization algorithms able to effectively locate the singularities (converging to AKKT) should be useful in practice.

4. Reformulations of bilevel and equilibrium problems (see, for example, [96, 106]) are used to generate constrained optimization problems in which constraint qualifications do not hold. For example, the set of constraints $x_1 \geq 0$, $x_2 \geq 0$, $x_1 x_2 = 0$ induces the lack of fulfillment of most popular constraint qualifications. These types of "complementarity constraints" are associated with the optimality conditions of the problems that define the lower level of bilevel and equilibrium problems.

3.4 ▪ Review and summary

The objective of constrained optimization is to find solutions to minimization problems with constraints, but algorithms that are guaranteed to provide global minimizers are very expensive, especially in the large-scale case. In this case, affordable algorithms only guarantee solutions that satisfy some optimality condition. Sequential optimality conditions (such as AKKT) provide practical stopping criteria for iterative minimization algorithms, independently of constraint qualifications. When some weak constraint qualification holds, AKKT implies the fulfillment of the KKT condition.

3.5 ▪ Further reading

The KKT condition was introduced in the master's thesis of Karush [158] and rediscovered in the classical paper by Kuhn and Tucker [170]. The penalty method for constrained minimization goes back to pioneering work by Courant [88] and was systematized by Fiacco and McCormick [109]. The penalty approach was used to prove the KKT condition in [197], in Hestenes' book [145], and in Bertsekas' book [40]. The approximate gradient projection (AGP) condition was the first sequential optimality condition whose practical importance was emphasized in the literature by Martínez and Svaiter [192]. The relation of sequential optimality conditions with pointwise optimality conditions under weak constraint qualifications began with [16] and is currently a subject of intense research [12, 13, 14]. In [12], a diagram showing the known connections between different sequential optimality conditions is given. A similar diagram for pointwise conditions is presented in the classical book by Mangasarian [183].

Second-order optimality conditions and the related area of sufficient optimality conditions have not been considered here because they are hardly used in practical constrained optimization algorithms. Textbooks like [40, 83, 154, 182, 213] may be consulted for updated views on that subject. Recent research has been reported in [9, 10, 17, 204].

3.6 ▪ Problems

3.1 Draw Venn diagrams to illustrate the relation between different optimality conditions, different types of minimizers, and constraint qualifications.

3.2 Discuss the strength and usefulness of the following necessary optimality conditions:

(a) x^* is feasible.

(b) $x^* \in \mathbb{R}^n$.

(c) x^* is a global minimizer of the constrained optimization problem.

3.3 In the process of choosing an algorithm for solving your practical optimization problem, you read that Algorithm A generates a primal-dual sequence (x^k, λ^k, μ^k) that, in the case that $\{\lambda^k\}$ and $\{\mu^k\}$ are bounded, converge to KKT points. Would you choose Algorithm A on the basis of this result?

3.4 Under the same assumptions as Theorem 3.1, prove that there exist sequences $\{\lambda^k\} \subseteq \mathbb{R}^m$ and $\{\mu^k\} \subseteq \mathbb{R}^p_+$ such that (3.2) and (3.3) hold, and, in addition,

$$\lim_{k\to\infty} \lambda_i^k h_i(x^k) = 0 \text{ for all } i = 1,\dots,m \text{ and } \lim_{k\to\infty} \mu_j^k g_j(x^k) = 0 \text{ for all } j = 1,\dots,p.$$

Establish the implications between this new optimality condition (called the complementary AKKT (CAKKT) in [18]) and AKKT.

3.5 Explicitly compute sequences $\{x^k\}$ and $\{\lambda^k\}$ that prove that the origin is an AKKT point in the problem

$$\text{Minimize } x_1 \text{ subject to } \|x\|_2^2 = 0.$$

3.6 Prove that KKT implies AKKT and CAKKT.

3.7 Assume that the LICQ constraint qualification holds for $h(x) = 0$ at the feasible point x^*. Prove that x^* does not satisfy LICQ or MFCQ for the duplicated constraints $h(x) = 0$, $h(x) = 0$. However, CPLD holds for these new constraints.

3.8 Assume that LICQ holds at x^* for the constraint $h_1(x) = 0$. Replace this constraint with the two inequalities $h_1(x) \leq 0$ and $-h_1(x) \leq 0$. Show that neither LICQ nor MCFQ holds for the new constraints but CPLD holds.

3.9 Sometimes users are tempted to replace the constraints $h(x) = 0$, $g(x) \leq 0$ with a single constraint $\|h(x)\|_2^2 + \|g(x)_+\|_2^2 = 0$. Discuss this decision from several points of view, including the fulfillment of constraint qualifications.

3.10 Consider the problem

$$\text{Minimize } (x_1 - 1)^2 + x_2^2 \text{ subject to } x_1^2 = 0.$$

Show that $x^* = (0,0)^T$ is the global solution. Analyze the sequence $x^k = (0, 1/k)^T$ and show that this sequence cannot be used to corroborate AKKT. Give a sequence that corroborates that x^* satisfies AKKT. In other words, AKKT expresses a property satisfied by some convergent sequence to the solution but not by all those sequences. Show that a similar (although less convincing) example could be the problem

$$\text{Minimize } x \text{ subject to } x^2 = 0.$$

3.11 Assume that x^* satisfies KKT. Show that for all sequences that converge to x^* there exist multipliers that corroborate AKKT.

3.12 Study the papers [13, 14, 180, 181, 203, 204], where weaker constraint qualifications are introduced, and prove the results of this chapter with the new conditions replacing CPLD.

3.13 In the AKKT condition, replace (3.12) with

$$\mu_i^k = 0 \text{ whenever } g_i(x^k) < 0.$$

Prove that, in this way, we also obtain a necessary optimality condition. Establish the implications between this optimality condition and AKKT.

3.14 Show that, in general, the CPLD constraint qualification does not hold at feasible points of problem (2.1). (RCPLD [13] and CPG do not hold either.) However, every local minimizer of (2.1) satisfies the KKT condition [120]. Does this contradict the fact that CPLD, RCPLD, and CPG are constraint qualifications?

3.15 Prove that the property P + AKKT $\Rightarrow$ KKT does not imply that P is a constraint qualification. Hint: Use the example of minimizing x_2 subject to $x_1 = 0$ and $x_1 x_2 = 0$ at the point $(0,1)^T$. Show that AKKT is satisfied but, obviously, KKT is not. However, show that $(0,1)^T$ satisfies other constraint qualifications.

3.16 Prove the existence and unicity of the projection, (3.28), and (3.29).

3.17 Formulate the pointwise KKT-like optimality condition that corresponds to minimize $f(x)$ subject to $x \in D$, where D is given by (3.26). Discuss constraint qualifications ensuring that local minimizers satisfy such condition.

3.18 Consider the problem of minimizing $f(x)$ subject to $h(x) = 0$. Show that no constraint qualification holds if one replaces $h(x) = 0$ with $h(x)^2 = 0$. (Suggestion: Define $c = \nabla f(x^*)$ and consider the objective function $f(x) = c^T x + ||x - x^*||_2^2$.)

3.19 Given $D \subseteq \mathbb{R}^n$ and $x^* \in D$, the "tangent cone" $T_D(x^*)$ is defined as the set of elements $d \in \mathbb{R}^n$ such that $d = 0$ or there exists a sequence $\{x^k\} \subseteq D$ satisfying $x^k \neq x^*$ for all k and

$$\lim_{k\to\infty} \frac{x^k - x^*}{||x^k - x^*||} = \frac{d}{||d||}.$$

Prove that every local minimizer of f subject to $x \in D$ satisfies $\nabla f(x^*)^T d \geq 0$ for all $d \in T_D(x^*)$.

3.20 Given $F : \mathbb{R}^n \to \mathbb{R}^n$, we say that x^* solves the variational inequality problem (VIP) associated with the set D when $F(x^*)^T d \geq 0$ for all $d \in T_D(x^*)$ (see Shapiro [239]). Define a generalization of KKT for VIP and discuss the relations between solutions of VIP and points that satisfy the defined generalization of KKT for VIP.

Chapter 4

Model Augmented Lagrangian Algorithm

We will consider the optimization problem defined by

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & h(x)=0, \\ & g(x)\leq 0, \\ & x\in\Omega, \end{array} \tag{4.1}$$

where $h:\mathbb{R}^n\to\mathbb{R}^m$, $g:\mathbb{R}^n\to\mathbb{R}^p$, $f:\mathbb{R}^n\to\mathbb{R}$ are continuous and $\Omega\subseteq\mathbb{R}^n$ is closed (not necessarily convex!). In this chapter we do not require the existence of derivatives.

The Lagrangian function $\mathscr{L}$ will be defined by

$$\mathscr{L}(x,\lambda,\mu)=f(x)+\sum_{i=1}^{m}\lambda_i h_i(x)+\sum_{i=1}^{p}\mu_i g_i(x) \tag{4.2}$$

for all $x\in\Omega$, $\lambda\in\mathbb{R}^m$, and $\mu\in\mathbb{R}^p_+$, whereas the Augmented Lagrangian [144, 218, 229] will be given by

$$L_\rho(x,\lambda,\mu)=f(x)+\frac{\rho}{2}\left\{\sum_{i=1}^{m}\left[h_i(x)+\frac{\lambda_i}{\rho}\right]^2+\sum_{i=1}^{p}\left[\max\left(0,g_i(x)+\frac{\mu_i}{\rho}\right)\right]^2\right\} \tag{4.3}$$

for all $\rho>0$, $x\in\Omega$, $\lambda\in\mathbb{R}^m$, and $\mu\in\mathbb{R}^p_+$.

In order to understand the meaning of the definition (4.3), first consider the case in which $\lambda=0$ and $\mu=0$. Then,

$$L_\rho(x,0,0)=f(x)+\frac{\rho}{2}\left(||h(x)||_2^2+||g(x)_+||_2^2\right). \tag{4.4}$$

Therefore, $L_\rho(x,0,0)$ is the "external penalty function" that coincides with $f(x)$ within the feasible set and penalizes the lack of feasibility by means of the term

$$\frac{\rho}{2}\left(||h(x)||_2^2+||g(x)_+||_2^2\right).$$

For big values of the penalty parameter ρ, the penalty term "dominates" $f(x)$ and the level sets of $L_\rho(x,0,0)$ tend to be those of $||h(x)||_2^2+||g(x)_+||_2^2$ in the infeasible region.

Consider the problem

$$\text{Minimize } f(x) \text{ subject to } h(x)=0, \tag{4.5}$$

where

$$f(x) = (x_1 - 6)^2 + x_2^2$$

and

$$h(x) = (x_2 - (x_1/4)^2)^2 + (x_1/4 - 1)^2 - 1,$$

illustrated in Figure 4.1. The level sets of $h(x)^2$ are shown in Figure 4.2(a), while Figures 4.2(b)–4.2(d) show the level sets of $L_\rho(x, 0, 0)$ for $\rho \in \{1, 100, 1{,}000\}$. It is easy to see that, with $\rho = 1{,}000$, the penalty term $\rho h(x)^2$ dominates $f(x)$ in $L_\rho(x, 0, 0)$ and, therefore, the level sets depicted in Figures 4.2(a) and 4.2(d) are very similar (in fact, they are indistinguishable).

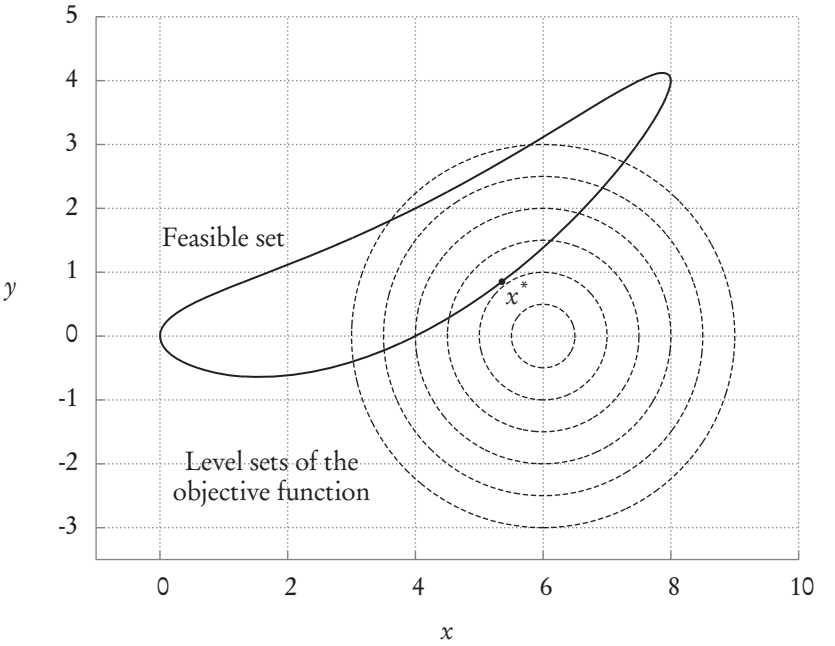

Figure 4.1. *Feasible set and level sets of the objective function of problem (4.5). Solution is given by* $x^* \approx (5.3541, 0.8507)^T$.

This means that, for x infeasible, the external penalty function (4.4) gives little information about the objective function if ρ is very large. The Augmented Lagrangian (4.3) may be seen as the penalized function in which "punishment" of infeasibility occurs, not with respect to the true constraints $h(x) = 0$ and $g(x) \leq 0$ but with respect to the shifted constraints $h(x) + \lambda/\rho = 0$ and $g(x) + \mu/\rho \leq 0$. The reasons for employing shifted constraints, instead of the original ones, will be explained after the definition of the following model algorithm.

4.1 ▪ Main model algorithm and shifts

The following algorithm is a basic Augmented Lagrangian algorithm for solving (4.1). The algorithm proceeds by minimizing the Augmented Lagrangian function at each iteration and updating Lagrange multipliers and penalty parameters between iterations. Its generality will allow us to analyze several particular cases that address the problem (4.1) under different conditions.

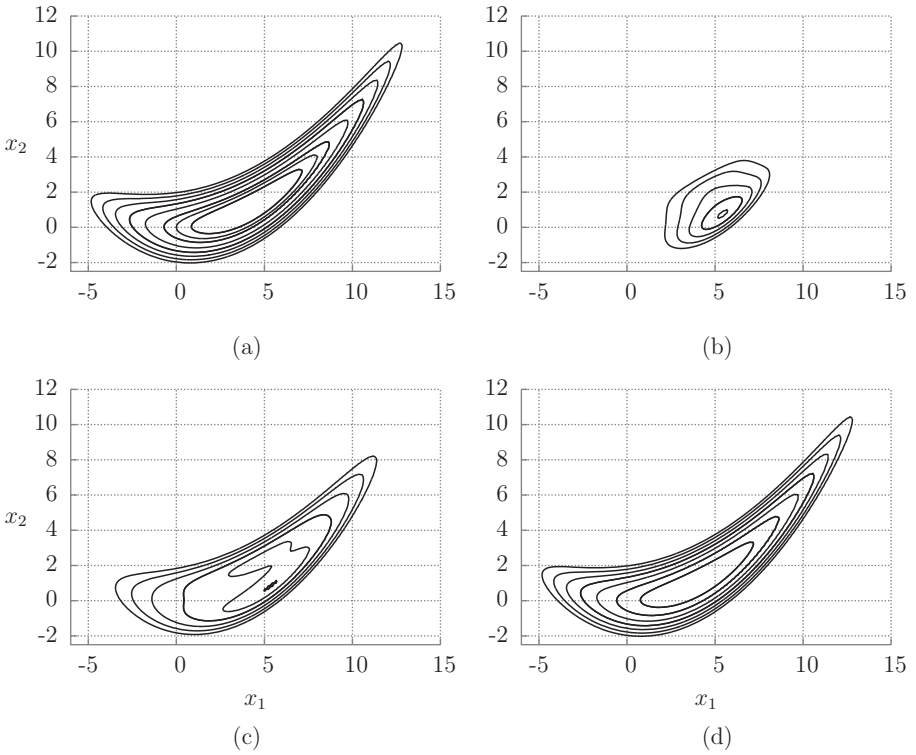

Figure 4.2. *Level sets of (a) $h(x)^2$ and (b)–(d) level sets of $L_\rho(x,0,0)$ for $\rho \in \{1,100,1{,}000\}$, respectively.*

Algorithm 4.1.
Let $\lambda_{\min} < \lambda_{\max}$, $\mu_{\max} > 0$, $\gamma > 1$, and $0 < \tau < 1$. Let $\bar{\lambda}^1 \in [\lambda_{\min}, \lambda_{\max}]^m$, $\bar{\mu}^1 \in [0, \mu_{\max}]^p$, and $\rho_1 > 0$. Initialize $k \leftarrow 1$.

Step 1. Find $x^k \in \mathbb{R}^n$ as an approximate solution of

$$\text{Minimize } L_{\rho_k}(x, \bar{\lambda}^k, \bar{\mu}^k) \text{ subject to } x \in \Omega. \tag{4.6}$$

Step 2. Compute new approximations of the Lagrange multipliers

$$\lambda^{k+1} = \bar{\lambda}^k + \rho_k h(x^k) \tag{4.7}$$

and

$$\mu^{k+1} = \left(\bar{\mu}^k + \rho_k g(x^k)\right)_+. \tag{4.8}$$

Step 3. Define

$$V_i^k = \min\left\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\right\} \text{ for } i = 1, \ldots, p.$$

If $k = 1$ or

$$\max\left\{||h(x^k)||, ||V^k||\right\} \le \tau \max\left\{||h(x^{k-1})||, ||V^{k-1}||\right\}, \tag{4.9}$$

choose $\rho_{k+1} \ge \rho_k$. Otherwise, define $\rho_{k+1} = \gamma \rho_k$.

Step 4. Compute $\bar{\lambda}^{k+1} \in [\lambda_{\min}, \lambda_{\max}]^m$ and $\bar{\mu}_i^{k+1} \in [0, \mu_{\max}]^p$.

Step 5. Set $k \leftarrow k+1$ and go to Step 1.

Note that λ^{k+1} and μ^{k+1} are not used in this algorithm but will be used in more specific versions to define $\bar{\lambda}^{k+1}$ and $\bar{\mu}^{k+1}$, respectively.

Algorithm 4.1 is said to be conceptual because Step 1, which establishes that the iterate x^k should be an "approximate minimizer" of the subproblem (4.6), is defined in a deliberately ambiguous way. (Strictly speaking, any point $x^k \in \mathbb{R}^n$ could be accepted as an "approximate minimizer"!) Constraints defined by $x \in \Omega$ will be said to be "hard," "simple," or "nonrelaxable," following the terminology of Audet and Dennis [24]. Other authors prefer the terms "lower-level constraints" [8] or "subproblem constraints" [48]. The term "hard" only appears contradictory to "simple." These constraints are generally simple in the sense that it is not difficult to satisfy them. However, they are hard (or nonrelaxable) in the sense that it is not admissible not to satisfy them.

The quantities ρ_k are said to be "penalty parameters." Ideally, at each "outer iteration" one solves the subproblem (4.6) and, if enough progress has been obtained in terms of improvement of feasibility and complementarity, the same penalty parameter *may be* used at the next outer iteration ($\rho_{k+1} \geq \rho_k$), while the penalty parameter *must be* increased if progress is not satisfactory.

The quantities $\bar{\lambda}_i^k/\rho_k \in \mathbb{R}$ and $\bar{\mu}_i^k/\rho_k \in \mathbb{R}_+$ can be interpreted as "shifts." In (4.6), one penalizes not merely the violation of infeasibilities (this would be the case if the shifts were null) but the infeasibilities modified by the shifts $\bar{\lambda}_i^k/\rho_k$ and $\bar{\mu}_i^k/\rho_k$. The idea is that, even with a penalty parameter of moderate value, a judicious choice of the shifts makes possible the coincidence, or near coincidence, of the solution to subproblem (4.6) with the desired minimizer of (4.1). See Figure 4.3, where we compare the solution for the problem

$$\text{Minimize } x \text{ subject to } -x \leq 0$$

to the solution for a subproblem with $\rho_k = 1$ and null shift ($\bar{\mu}^k = 0$) and to the solution for a subproblem with $\rho_k = 1$ and the "correct" shift ($\bar{\mu}^k = 1$).

The shifts are corrected according to formulae (4.7) and (4.8). The idea is as follows. Assume that x^k came from solving (4.6) with shifts $\bar{\lambda}_k/\rho_k$ and $\bar{\mu}_k/\rho_k$. After this process, if one verifies that the feasibility of an inequality has been violated by a quantity $g_i(x^k) > 0$, it is reasonable to think that the shift should be increased by a quantity equal to this violation. This leads to the formula $\mu^{k+1}/\rho_k = \bar{\mu}^k/\rho_k + g_i(x^k)$ or, equivalently, $\mu^{k+1} = \bar{\mu}^k + \rho_k g(x^k)$. Moreover, if the infeasibility has not been violated and $-\mu^k/\rho_k < g_i(x^k) \leq 0$, the suggestion is that the shift has been excessively big, and a reduction is necessary to make it possible that $g_i(x) = 0$ at the new iteration, with a possible improvement of the objective function. Again, this suggests the updating rule $\mu^{k+1}/\rho_k = \bar{\mu}^k/\rho_k + g_i(x^k)$. Finally, if $g_i(x^k) < -\mu^k/\rho_k$, we guess that the shift was not necessary and should be null from now on. Similar reasoning may be done with respect to formula (4.7), which updates the shifts corresponding to the equality constraints.

Algorithms are conceivable in which the penalty parameters become fixed and only shift modifications are made. These algorithms can be interpreted, under convexity assumptions, as "proximal point" methods in the dual problem of (4.1), and their properties have been exhaustively studied in textbooks and survey papers (see [39, 151, 229], among others).

In Algorithm 4.1, not only are the shifts updated, but so are the penalty parameters, according to the test (4.9). In (4.9), the progress of two different quantities is considered.

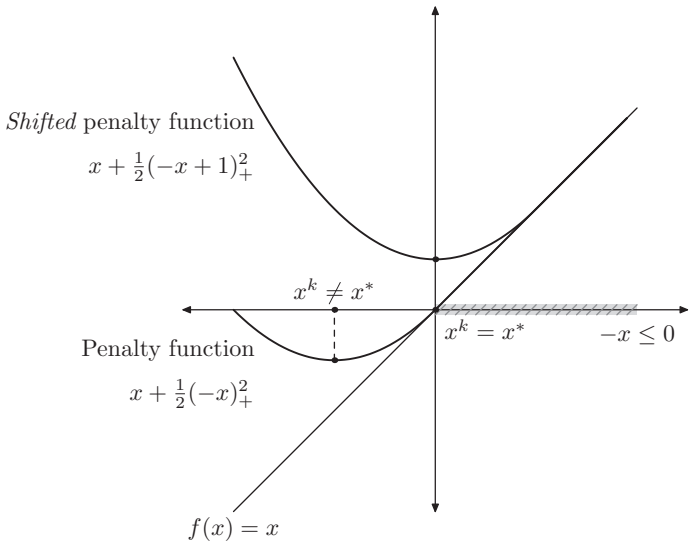

Figure 4.3. *Comparison between the solution to the problem Minimize x subject to* $-x \leq 0$ *and (a) the solution to a subproblem with* $\rho_k = 1$ *and no shift* ($\bar{\mu}^k = 0$) *and (b) the solution to a subproblem with* $\rho_k = 1$ *and the "correct" shift* ($\bar{\mu}^k = 1$).

On the one hand, one needs progress in terms of the feasibility of equality constraints, measured by $||h(x^k)||$. In the absence of improvement of this infeasibility measure, one decides to increase the penalty parameter. On the other hand, through the test (4.9), one also requires reduction of the quantities $\min\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\}$. Note that $\bar{\mu}_i^k/\rho_k$, the shift already employed in (4.6), is nonnegative. Therefore, through consideration of $\min\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\}$, we are implicitly testing the progress in terms of fulfillment of the inequality constraint $g_i(x) \leq 0$. In fact, if $g_i(x^k)$ tends to zero, improvement of $\min\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\}$ very likely occurs, independently of the shifts $\bar{\mu}_i^k/\rho_k$. The interesting question is why we require this improvement even in the case that $g_i(x^k) \ll 0$ and x^k probably converges to a point at which $g_i(x)$ is inactive.

The answer is the following. If $g_i(x^k)$ is "very feasible" and the shift $\bar{\mu}_i^k/\rho_k$ is big, very likely it was the shift that forced $g_i(x^k)$ to be very negative at the solution to (4.6), since the subproblem penalizes deviations of the constraint from the shift, instead of mere infeasibility. However, although we are getting a feasible point and we may get a feasible point in the limit, since the feasible set is unnecessarily being reduced in this case, it is unlikely that an optimum could be obtained in this way. Therefore, we need to decrease the shift by increasing the penalty parameter.

According to the arguments above, it is sensible to choose the new Lagrange multipliers $\bar{\lambda}^{k+1}$ and $\bar{\mu}^{k+1}$ as λ^{k+1} and μ^{k+1}, respectively. In general, this is what is done in practice, but safeguards that guarantee the boundedness of $\{\bar{\lambda}^k\}$ and $\{\bar{\mu}^k\}$ are necessary. Safeguarded boundedness guarantees a crucial commonsense property: The shifts should tend to zero when the penalty parameter tends to infinity. Clearly, if we are led to penalize violations of the constraints with a very large ρ_k, it does not make sense to use shifts bounded away from zero since, in this case, we would be punishing hardly suitable feasible points. So, when ρ_k tends to infinity, common sense dictates that the shifts should tend to zero, and the most straightforward way to guarantee this is to impose bounds on the multipliers. Therefore, we may think of $\bar{\mu}^k$ and $\bar{\lambda}^k$ as being "safeguarded multipliers."

In the Augmented Lagrangian context, some authors prefer, at each outer iteration, to update either the multipliers or the penalty parameters, but not both. Our formulation in Algorithm 4.1 allows this possibility, although in practice we prefer to update penalty parameters and multipliers simultaneously.

The reader should observe that, for the motivation arguments given above, differentiability of the objective function or the constraints has not been invoked. Penalty and Augmented Lagrangian ideas are independent of the degree of smoothness of the functions that define the problem. This characteristic makes possible the application of the Augmented Lagrangian techniques to many nonstandard optimization problems.

4.2 ▪ Multipliers and inactive constraints

Despite the generality of Algorithm 4.1, it is possible to prove a useful property: Inequality multipliers corresponding to constraints that are inactive in the limit are asymptotically null, independently of the feasibility of the limit point. Note that, in the statement of Theorem 4.1, the existence of a limit point x^* is assumed. The existence of limit points in this and several other results should be confirmed by employing, in general, boundedness arguments concerning the feasible set.

Theorem 4.1. *Assume that the sequence* $\{x^k\}$ *is generated by Algorithm 4.1 and* $K \subset_{\infty} \mathbb{N}$ *is such that* $\lim_{k\in K} x^k = x^*$. *Then, for* $k \in K$ *large enough,*

$$\mu_i^{k+1} = 0 \text{ for all } i = 1,\dots,p \text{ such that } g_i(x^*) < 0. \tag{4.10}$$

Proof. By (4.8), $\mu^{k+1} \in \mathbb{R}^p_+$ for all $k \in \mathbb{N}$.

Assume that $g_i(x^*) < 0$ and let $k_1 \in \mathbb{N}$ and $c < 0$ be such that

$$g_i(x^k) < c < 0 \text{ for all } k \in K, k \geq k_1.$$

We consider two cases:

1. The sequence $\{\rho_k\}$ tends to infinity.
2. The sequence $\{\rho_k\}$ is bounded.

In the first case, since $\{\bar{\mu}_i^k\}$ is bounded, there exists $k_2 \geq k_1$ such that, for all $k \in K$, $k \geq k_2$,

$$\bar{\mu}_i^k + \rho_k g_i(x^k) < 0.$$

By (4.8), this implies that

$$\mu_i^{k+1} = 0 \text{ for all } k \in K, k \geq k_2.$$

Consider now the case in which $\{\rho_k\}$ is bounded. Then, (4.9) holds for all k large enough and, consequently,

$$\lim_{k\to\infty} V_i^k = 0.$$

Thus,

$$\lim_{k\to\infty} \left| \min\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\} \right| = 0.$$

Since $g_i(x^k) < c < 0$ for $k \in K$ large enough, we have that

$$\lim_{k \in K} \bar{\mu}_i^k / \rho_k = 0.$$

Thus, since the sequence $\{\rho_k\}$ is bounded,

$$\lim_{k \in K} \bar{\mu}_i^k = 0.$$

Therefore, for $k \in K$ large enough,

$$\bar{\mu}_i^k + \rho_k g_i(x^k) < 0.$$

By the definition (4.8) of μ^{k+1}, this implies that $\mu_i^{k+1} = 0$ for $k \in K$ large enough, as we wanted to prove. □

4.3 ▪ Review and summary

The basic Augmented Lagrangian algorithm is composed of outer iterations, and at each, one minimizes the objective function plus a term that penalizes shifted constraints. The use of shifts has the appeal of avoiding the necessity of increasing the penalty parameter up to values at which the objective function becomes numerically neglected. Both the penalty parameter and the shifts are updated after each outer iteration. In particular, shifts are updated according to commonsense rules whose plausibility does not depend on the differentiability of the problem.

4.4 ▪ Further reading

In the case in which h is a linear mapping, there are no inequality constraints, and Ω represents an n-dimensional box, subproblems generated by an Augmented Lagrangian method based on the Powell–Hestenes–Rockafeller (PHR) [144, 218, 229] Augmented Lagrangian function (4.3) are box-constrained quadratic optimization problems. This fact was exhaustively exploited by Dostál [102] in order to define "optimal" quadratic programming methods. The use of different penalty functions (instead of the quadratic loss) and the generalization of the shifting ideas give rise to many alternative Augmented Lagrangian algorithms [5, 25, 32, 33, 77, 128, 150, 151, 165, 166, 196, 210, 211, 228, 250, 258, 261]. Most of these algorithms can be recommended for particular structures, but, for general problems reported in popular collections, the classical PHR approach seems to be more efficient and robust than nonclassical approaches [44, 104]. In this book, all the theory is dedicated to properties on general (not necessarily convex) problems. When convexity is assumed for the objective function and constraints, profound results can be obtained using the dual equivalence with the so-called proximal point methods. Iusem's survey [151] offers a good overview of this subject. An alternative Augmented Lagrangian approach that deals with nonlinear semidefinite programming was proposed in [167] and gave rise to the PENNON software package [167, 168, 169].

Sometimes, the penalty parameter is increased at the first iterations of the Augmented Lagrangian method, but, at later iterations, smaller penalty parameters are admissible. An extension of the basic Augmented Lagrangian method in which a nonmonotone strategy for penalty parameters is employed may be found in [56]. In cases in which the objective function takes very low values (perhaps going to $-\infty$) at infeasible points, penalty and

Augmented Lagrangian algorithms may be attracted by those points at early iterations and practical convergence could be discouraged. This phenomenon is called "greediness" in [43] and [75], where theoretically justified remedies are suggested. An application of the Augmented Lagrangian philosophy to a relevant family of nonsmooth problems may be found in [42].

4.5 ▪ Problems

4.1 Describe Algorithm 4.1 in terms of "shifts" instead of multipliers. Write explicitly the updating rules for shifts. Replace the boundedness condition on the multipliers with some condition on the shifts guaranteeing that shifts go to zero if multipliers go to infinity.

4.2 Justify the updating formula for the Lagrange multipliers corresponding to equality constraints using commonsense criteria, as we did in the case of inequalities in Section 4.1.

4.3 Analyze Algorithm 4.1 in the case that $\bar{\lambda}^k = 0$ and $\bar{\mu}^k = 0$ for all k.

4.4 Analyze Algorithm 4.1 in the case that at the resolution of the subproblem, one defines x^k to be an arbitrary, perhaps random, point of $\mathbb{R}^n$.

4.5 Analyze Algorithm 4.1 in the case that there are no constraints at all ($m = p = 0$) besides those corresponding to $x \in \Omega$.

4.6 Compare Algorithm 4.1 in the following two situations: when constraints $x \in \Omega$ remain in the lower level and when they are incorporated into the relaxable set $h(x) = 0$ and $g(x) \leq 0$ (if possible).

4.7 Give examples in which constraints $x \in \Omega$ cannot be expressed as systems of equalities and inequalities.

4.8 Assume that you are convinced that (4.7) is the reasonable way to update the equality Lagrange multipliers, but you are not convinced about the plausibility of (4.8). Replace each constraint $g_i(x) \leq 0$ in (4.1) with $g_i(x) + z_i^2 = 0$, where z_i is a slack variable, and reformulate Algorithm 4.1 for the new problem, now without inequality constraints. At the solution x^k of the new formulation of (4.6), observe that it is sensible (why?) to define

$$(z_i^k)^2 = -g_i(x^k) - \bar{\mu}_i^k/\rho_k \text{ if } g_i(x^k) + \bar{\mu}_i^k/\rho_k < 0$$

and

$$(z_i^k)^2 = 0 \text{ if } g_i(x^k) + \bar{\mu}_i^k/\rho_k \geq 0.$$

Deduce that the infeasibility for the constraint $g_i(x) + z_i^2 = 0$, or, equivalently, $g_i(x) \leq 0$, may be defined by

$$\left| \min \left\{ -g_i(x^k), \bar{\mu}_i^k/\rho_k \right\} \right|$$

and that, according to (4.7), the formula for the new multiplier μ^{k+1} should be given by (4.8). In other words, the scheme defined in Algorithm 4.1 with inequality constraints can be deduced from the scheme defined for problems with equality constraints only. See [39].

4.9 Consider the following alternative definition for V^k in Algorithm 4.1:

$$V_i^k = \min\{-g_i(x^k), \mu_i^{k+1}\} \text{ for } i = 1, \ldots, p.$$

Discuss the adequacy of this definition and prove Theorem 4.1 for the modified algorithm.

4.10 Consider the following alternative definition for V^k in Algorithm 4.1:

$$V_i^k = |\mu_i^{k+1} g_i(x^k)| \text{ for } i = 1, \ldots, p, \tag{4.11}$$

and replace the test (4.9) with

$$\max\{\|h(x^k)\|, \|g(x^k)_+\|, \|V^k\|\} \leq \tau \max\{\|h(x^{k-1})\|, \|g(x^{k-1})_+\|, \|V^{k-1}\|\}.$$

Discuss the adequacy of these alternatives and prove Theorem 4.1 for the modified algorithm.

4.11 Define V^k as in (4.11) and

$$W_i^k = |\lambda_i^{k+1} h_i(x^k)| \text{ for } i = 1, \ldots, m.$$

Replace the test (4.9) with

$$\max\{\|h(x^k)\|, \|g(x^k)_+\|, \|V^k\|, \|W^k\|\}$$
$$\leq \tau \max\{\|h(x^{k-1})\|, \|g(x^{k-1})_+\|, \|V^{k-1}\|, \|W^{k-1}\|\}.$$

Discuss the adequacy of these alternatives and prove Theorem 4.1 for the modified algorithm.

4.12 Assume that there exists $c \in \mathbb{R}$ such that at Step 1 of Algorithm 4.1, we have that

$$L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) \leq c$$

for all $k \in \mathbb{N}$. Prove that any limit point x^* of $\{x^k\}$ verifies $h(x^*) = 0$ and $g(x^*) \leq 0$.

4.13 Consider constrained optimization problems that include at least one "semidefinite constraint," which says that a set of variables defines a symmetric positive semidefinite matrix. Consider the possibility of coding that constraint as a set of upper-level constraints of the form $X = MM^T$, where M is an auxiliary matrix. Consider a different possibility: setting positive semidefinitness as a lower-level constraint on which we know how to project (how?). Analyze advantages and disadvantages and repeat a copy of this problem in all chapters of the book. (Specific Augmented Lagrangian methods for this case may be found in Kocvara and Stingl [163] and Stingl [242].)

4.14 Make your choices: Discuss reasonable values for the algorithmic parameters $\lambda_{\min}$, $\lambda_{\max}$, $\mu_{\max}$, ρ_1, γ, and τ. Implement Algorithm 4.1 (in your favorite language) in the case that Ω is a box and using some simple trial and error strategy for the approximate minimization of the Augmented Lagrangian subproblem (4.6). Run your code using simple examples and draw conclusions.

4.15 Try to exploit algorithmically the consequences of these facts:

(a) The original problem that you want to solve is equivalent to the problem in which you add a fixed penalization to the objective function.

(b) In your specific problem a feasible initial point is easily available.

Chapter 5
Global Minimization Approach

In this chapter, the subproblems (4.6) at Step 1 of Algorithm 4.1 will be interpreted in terms of global optimization. Namely, at each outer iteration, we will assume that x^k is an approximate global minimizer of the Augmented Lagrangian on Ω. In principle, the global minimization of the Augmented Lagrangian on Ω could be as difficult as the original problem, since we make no assumptions on the geometry of this set. However, in practice, the set Ω is, in general, simple enough to make global minimization on Ω much easier than on the feasible set of problem (4.1).

The global minimization method for solving (4.1) considered in this chapter will be Algorithm 4.1 with the following algorithmic assumption.

Assumption 5.1. *For all $k \in \mathbb{N}$, we obtain $x^k \in \Omega$ such that*

$$L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) \leq L_{\rho_k}(x, \bar{\lambda}^k, \bar{\mu}^k) + \varepsilon_k \text{ for all } x \in \Omega,$$

where the sequence of tolerances $\{\varepsilon_k\} \subseteq \mathbb{R}_+$ is bounded.

As in Chapter 4, in this chapter we only assume continuity of the objective function and the functions that define the constraints.

5.1 ▪ Feasibility result

Assumption 5.1 says that, at each outer iteration, one finds an approximate global minimizer of the subproblem. In principle, the tolerances ε_k do not need to be small at all. In the following theorem, we prove that, even using possibly big tolerances, we obtain, in the limit, a global minimizer of the infeasibility measure.

Theorem 5.1. *Assume that $\{x^k\}$ is a sequence generated by Algorithm 4.1 under Assumption 5.1. Let x^* be a limit point of $\{x^k\}$. Then, for all $x \in \Omega$, we have that*

$$\|h(x^*)\|_2^2 + \|g(x^*)_+\|_2^2 \leq \|h(x)\|_2^2 + \|g(x)_+\|_2^2.$$

Proof. Since Ω is closed and $x^k \in \Omega$, we have that $x^* \in \Omega$. We consider two cases: $\{\rho_k\}$ bounded and $\rho_k \to \infty$.

If $\{\rho_k\}$ is bounded, there exists k_0 such that $\rho_k = \rho_{k_0}$ for all $k \geq k_0$. Therefore, for all $k \geq k_0$, (4.9) holds. This implies that $\|h(x^k)\| \to 0$ and $\|V^k\| \to 0$, so $g_i(x^k)_+ \to 0$ for all $i = 1, \ldots, p$. Thus, the limit point is feasible.

Now, assume that $\rho_k \to \infty$. Let $K \underset{\infty}{\subset} \mathbb{N}$ be such that

$$\lim_{k\in K} x^k = x^*.$$

Assume by contradiction that there exists $x \in \Omega$ such that

$$\|h(x^*)\|_2^2 + \|g(x^*)_+\|_2^2 > \|h(x)\|_2^2 + \|g(x)_+\|_2^2.$$

By the continuity of h and g, the boundedness of $\{\bar{\lambda}^k\}$ and $\{\bar{\mu}^k\}$, and the fact that ρ_k tends to infinity, there exist $c > 0$ and $k_0 \in \mathbb{N}$ such that for all $k \in K, k \geq k_0$,

$$\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2 > \left\|h(x) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2 + c.$$

Therefore, for all $k \in K, k \geq k_0$,

$$f(x^k) + \frac{\rho_k}{2}\left[\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right]$$

$$> f(x) + \frac{\rho_k}{2}\left[\left\|h(x) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right] + \frac{\rho_k c}{2} + f(x^k) - f(x).$$

Since $\lim_{k\in K} x^k = x^*$, f is continuous, and $\{\varepsilon_k\}$ is bounded, there exists $k_1 \geq k_0$ such that, for $k \in K$, $k \geq k_1$,

$$\frac{\rho_k c}{2} + f(x^k) - f(x) > \varepsilon_k.$$

Therefore,

$$f(x^k) + \frac{\rho_k}{2}\left[\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right]$$

$$> f(x) + \frac{\rho_k}{2}\left[\left\|h(x) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right] + \varepsilon_k$$

for $k \in K, k \geq k_1$. This contradicts Assumption 5.1. □

5.2 ▪ Optimality result

Theorem 5.1 says that Algorithm 4.1, with the iterates defined by Assumption 5.1, finds minimizers of the infeasibility. Therefore, if the original optimization problem is feasible, every limit point of a sequence generated by the algorithm is feasible. Note that we only used boundedness of the sequence of tolerances $\{\varepsilon_k\}$ in the proof of Theorem 5.1. Now, we will see that, assuming that ε_k tends to zero, it is possible to prove that, in the feasible case, the algorithm asymptotically finds global minimizers of (4.1).

Theorem 5.2. *Assume that $\{x^k\}$ is a sequence generated by Algorithm 4.1 under Assumption 5.1 and $\lim_{k\to\infty} \varepsilon_k = 0$. Moreover, assume that, in the case that (4.9) holds, we always choose $\rho_{k+1} = \rho_k$. Let x^* be a limit point of $\{x^k\}$. Suppose that problem (4.1) is feasible. Then, x^* is a global minimizer of (4.1).*

Proof. Let $K \underset{\infty}{\subset} \mathbb{N}$ be such that $\lim_{k\in K} x^k = x^*$. By Theorem 5.1, since the problem is feasible, we have that x^* is feasible. Let $x \in \Omega$ be such that $h(x) = 0$ and $g(x) \leq 0$.

We consider two cases: $\rho_k \to \infty$ and $\{\rho_k\}$ bounded.

Case 1 ($\rho_k \to \infty$). By the definition of the algorithm, we have that

$$\begin{aligned} &f(x^k) + \frac{\rho_k}{2}\left[\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right] \\ &\leq f(x) + \frac{\rho_k}{2}\left[\left\|h(x) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right] + \varepsilon_k \end{aligned} \tag{5.1}$$

for all $k \in \mathbb{N}$.

Since $h(x) = 0$ and $g(x) \leq 0$, we have that

$$\left\|h(x) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 = \left\|\frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 \text{ and } \left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2 \leq \left\|\frac{\bar{\mu}^k}{\rho_k}\right\|_2^2.$$

Therefore, by (5.1),

$$f(x^k) \leq f(x^k) + \frac{\rho_k}{2}\left[\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_k}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_k}\right)_+\right\|_2^2\right] \leq f(x) + \frac{\|\bar{\lambda}^k\|_2^2}{2\rho_k} + \frac{\|\bar{\mu}^k\|_2^2}{2\rho_k} + \varepsilon_k.$$

Taking limits for $k \in K$ and using that $\lim_{k\in K} \|\bar{\lambda}^k\|/\rho_k = \lim_{k\in K} \|\bar{\mu}^k\|/\rho_k = 0$ and $\lim_{k\in K} \varepsilon_k = 0$, by the continuity of f and the convergence of x^k, we get

$$f(x^*) \leq f(x).$$

Since x is an arbitrary feasible point, it turns out that x^* is a global minimizer, as we wanted to prove.

Case 2 ($\{\rho_k\}$ bounded). In this case, there exists $k_0 \in \mathbb{N}$ such that $\rho_k = \rho_{k_0}$ for all $k \geq k_0$. Therefore, by Assumption 5.1,

$$\begin{aligned} &f(x^k) + \frac{\rho_{k_0}}{2}\left[\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_{k_0}}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_{k_0}}\right)_+\right\|_2^2\right] \\ &\leq f(x) + \frac{\rho_{k_0}}{2}\left[\left\|h(x) + \frac{\bar{\lambda}^k}{\rho_{k_0}}\right\|_2^2 + \left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_{k_0}}\right)_+\right\|_2^2\right] + \varepsilon_k \end{aligned}$$

for all $k \geq k_0$. Since $g(x) \leq 0$ and $\bar{\mu}^k/\rho_{k_0} \geq 0$,

$$\left\|\left(g(x) + \frac{\bar{\mu}^k}{\rho_{k_0}}\right)_+\right\|_2^2 \leq \left\|\frac{\bar{\mu}^k}{\rho_{k_0}}\right\|_2^2.$$

Thus, since $h(x) = 0$,

$$f(x^k) + \frac{\rho_{k_0}}{2}\left[\left\|h(x^k) + \frac{\bar{\lambda}^k}{\rho_{k_0}}\right\|_2^2 + \left\|\left(g(x^k) + \frac{\bar{\mu}^k}{\rho_{k_0}}\right)_+\right\|_2^2\right] \leq f(x) + \frac{\rho_{k_0}}{2}\left[\left\|\frac{\bar{\lambda}^k}{\rho_{k_0}}\right\|_2^2 + \left\|\frac{\bar{\mu}^k}{\rho_{k_0}}\right\|_2^2\right] + \varepsilon_k$$

for all $k \geq k_0$. Let $K_1 \underset{\infty}{\subset} K$, $\lambda^* \in R^m$, and $\mu^* \in \mathbb{R}^p$ be such that

$$\lim_{k\in K_1} \bar{\lambda}^k = \lambda^* \text{ and } \lim_{k\in K_1} \bar{\mu}^k = \mu^*.$$

By the feasibility of x^*, taking limits in the inequality above for $k \in K_1$, we get

$$f(x^*)+\frac{\rho_{k_0}}{2}\left[\left\|\frac{\bar{\lambda}^*}{\rho_{k_0}}\right\|_2^2+\left\|\left(g(x^*)+\frac{\bar{\mu}^*}{\rho_{k_0}}\right)_+\right\|_2^2\right]\leq f(x)+\frac{\rho_{k_0}}{2}\left[\left\|\frac{\bar{\lambda}^*}{\rho_{k_0}}\right\|_2^2+\left\|\frac{\bar{\mu}^*}{\rho_{k_0}}\right\|_2^2\right].$$

Therefore,

$$f(x^*)+\frac{\rho_{k_0}}{2}\left\|\left(g(x^*)+\frac{\bar{\mu}^*}{\rho_{k_0}}\right)_+\right\|_2^2\leq f(x)+\frac{\rho_{k_0}}{2}\left\|\frac{\bar{\mu}^*}{\rho_{k_0}}\right\|_2^2.$$

Thus,

$$f(x^*)+\frac{\rho_{k_0}}{2}\sum_{i=1}^{p}\left(g_i(x^*)+\frac{\bar{\mu}_i^*}{\rho_{k_0}}\right)_+^2\leq f(x)+\frac{\rho_{k_0}}{2}\sum_{i=1}^{p}\left(\frac{\bar{\mu}_i^*}{\rho_{k_0}}\right)^2. \tag{5.2}$$

Now, if $g_i(x^*)=0$, since $\bar{\mu}_i^*/\rho_{k_0}\geq 0$, we have that

$$\left(g_i(x^*)+\frac{\bar{\mu}_i^*}{\rho_{k_0}}\right)_+=\frac{\bar{\mu}_i^*}{\rho_{k_0}}.$$

Therefore, by (5.2),

$$f(x^*)+\frac{\rho_{k_0}}{2}\sum_{g_i(x^*)<0}\left(g_i(x^*)+\frac{\bar{\mu}_i^*}{\rho_{k_0}}\right)_+^2\leq f(x)+\frac{\rho_{k_0}}{2}\sum_{g_i(x^*)<0}\left(\frac{\bar{\mu}_i^*}{\rho_{k_0}}\right)^2. \tag{5.3}$$

But, by (4.9), $\lim_{k\to\infty}\max\{g_i(x^k),-\bar{\mu}_i^k/\rho_{k_0}\}=0$. Therefore, if $g_i(x^*)<0$, we necessarily have that $\bar{\mu}_i^*=0$. Therefore, (5.3) implies that $f(x^*)\leq f(x)$. Since x was an arbitrary feasible point, the proof is complete. □

5.3 ▪ Optimality subject to minimal infeasibility

Under an additional assumption, this time on the rule for updating multipliers, a stronger result can be proved that concerns the behavior of the algorithm for infeasible problems. We have already seen that, in this case, the algorithm considered in this chapter converges to global minimizers of the infeasibility, measured by the sum of squares $\|h(x)\|_2^2+\|g(x)_+\|_2^2$. Under Assumption 5.2 below, we will show that limit points minimize the objective function subject to minimal infeasibility.

Assumption 5.2. *For all $k\in\mathbb{N}$, if $\lambda^{k+1}\notin[\lambda_{\min},\lambda_{\max}]^m$ or $\mu^{k+1}\notin[0,\mu_{\max}]^p$, we choose $\bar{\lambda}^{k+1}=0$ and $\bar{\mu}^{k+1}=0$.*

The case in which $\|\lambda^{k+1}\|+\|\mu^{k+1}\|$ is big, contemplated in Assumption 5.2, generally corresponds to situations in which ρ_k is big too. As mentioned in Chapter 4, when infeasibility is severely penalized ($\rho_k\gg 1$), it makes no sense to employ shifts at all, because one could be adding a heavy penalization to the objective function even at reasonably feasible points. This argument, which supports the use of safeguards for the multipliers, can also be used to support the sensibility of the decision made in Assumption 5.2 (see [59]).

Theorem 5.3. *Assume that $\{x^k\}$ is a sequence generated by Algorithm 4.1 under Assumptions 5.1 and 5.2 and that $\lim_{k\to\infty}\varepsilon_k=0$. Moreover, assume that, in the case in which (4.9) holds, we always choose $\rho_{k+1}=\rho_k$. Let x^* be a limit point of $\{x^k\}$. Then,*

$$\|h(x^*)\|_2^2+\|g(x^*)_+\|_2^2\leq\|h(x)\|_2^2+\|g(x)_+\|_2^2 \text{ for all } x\in\Omega$$

and

$$f(x^*) \le f(x) \text{ for all } x \in \Omega \text{ such that } \|h(x)\|_2^2 + \|g(x)_+\|_2^2 = \|h(x^*)\|_2^2 + \|g(x^*)_+\|_2^2.$$

Proof. If $h(x^*) = 0$ and $g(x^*) \le 0$, the first part of the thesis follows immediately and the second part of the thesis follows from Theorem 5.2.

Let us assume from now on that $\|h(x^*)\|_2^2 + \|g(x^*)_+\|_2^2 = c > 0$. This implies by Step 3 that $\lim_{k\to\infty} \rho_k = \infty$. Since by Theorem 5.1 x^* is a global minimizer of $\|h(x)\|_2^2 + \|g(x)_+\|_2^2$, it turns out that, for all $k \in \mathbb{N}$, $\|h(x^k)\|_2^2 + \|g(x^k)_+\|_2^2 \ge c$. By (4.7), (4.8), the boundedness of $\{\bar\lambda^k\}$ and $\{\bar\mu^k\}$, and the fact that ρ_k tends to infinity, we have that, for all k large enough, either $\lambda^{k+1} \notin [\lambda_{\min}, \lambda_{\max}]^m$ or $\mu^{k+1} \notin [0, \mu_{\max}]^p$. Therefore, by Assumption 5.2, there exists $k_0 \in \mathbb{N}$ such that for all $k \ge k_0$ we have that $\bar\lambda^k = 0$ and $\bar\mu^k = 0$.

Let $K \underset{\infty}{\subseteq} \{k_0, k_0+1, k_0+2, \dots\}$ be such that $\lim_{k\in K} x^k = x^*$.

By Assumption 5.1 and the fact that $\|\bar\lambda^k\| = \|\bar\mu^k\| = 0$, we have that, for all $x \in \Omega$,

$$f(x^k) + \frac{\rho_k}{2}[\|h(x^k)\|_2^2 + \|g(x^k)_+\|_2^2] \le f(x) + \frac{\rho_k}{2}[\|h(x)\|_2^2 + \|g(x)_+\|_2^2] + \varepsilon_k \qquad (5.4)$$

for all $k \in K$. In particular, if $x \in \Omega$ is such that $\|h(x)\|_2^2 + \|g(x)_+\|_2^2 = \|h(x^*)\|_2^2 + \|g(x^*)_+\|_2^2$, we have that x is a global minimizer of the infeasibility on Ω. Thus,

$$\frac{\rho_k}{2}[\|h(x^k)\|_2^2 + \|g(x^k)_+\|_2^2] \ge \frac{\rho_k}{2}[\|h(x)\|_2^2 + \|g(x)_+\|_2^2].$$

Therefore, by (5.4) and Assumption 5.1,

$$f(x^k) \le f(x) + \varepsilon_k \text{ for all } k \in K.$$

By the continuity of f, taking limits on both sides of this inequality, we obtain the desired result. □

5.4 ▪ Review and summary

The Augmented Lagrangian paradigm can be used for solving global optimization problems. The only requirement is that we need to use a global optimization procedure for solving the subproblems. Theorem 5.1 indicates that we can always expect to find feasible points (if they exist) if we globally solve the subproblems, even with a loose tolerance. If the original problem is feasible, the global form of the Augmented Lagrangian method finds global solutions. Moreover, with a special safeguard of the Lagrange multipliers, the method finds global minimizers subject to minimal infeasibility.

5.5 ▪ Further reading

Global optimization has many applications in all branches of engineering, sciences, and production. Several textbooks addressing different aspects of global optimization theory and applications are available [28, 117, 123, 148, 240, 247, 251, 259]. Useful review papers have also appeared [118, 212]. In [49], global minimizers of linearly constrained subproblems are computed using the α-BB method [3, 4]. In [227], Augmented Lagrangian box-constrained subproblems are solved employing a stochastic population–based strategy that aims to guarantee global convergence. A variation of the algorithm introduced

in this chapter, with finite termination and finite detection of possible infeasibility, was introduced by Birgin, Martínez, and Prudente [58]. Employing duality arguments, some authors (see, for example, Burachik and Kaya [72] and [124]) transform the original constrained optimization problem into a simpler problem whose variables are Lagrange multipliers and penalty parameters. Applying subgradient techniques in the dual, convergence to global solutions is obtained.

5.6 ▪ Problems

5.1 If the second derivatives of an unconstrained optimization problem evaluated at a global minimizer are very big, a slight perturbation of the global minimizer could represent a large increase in the objective function. In this sense, perhaps, local minimizers with small second derivatives should be preferred over global minimizers with big second derivatives. Reformulate unconstrained minimization problems, taking into account this robustness issue.

5.2 Discuss the following apparent paradox: You need to solve the subproblems only very loosely if you want only to minimize the infeasibility (Theorem 5.1). In particular, it seems that, in order to minimize $||h(x)||_2^2 + ||g(x)_+||_2^2$, you do not need to employ global optimization procedures at all for solving the subproblems. Does this mean that global minimization of $||h(x)||_2^2 + ||g(x)_+||_2^2$ can be achieved without using global optimization? Does this contradict the fact that for obtaining global minimizers without further information one needs to evaluate the function on a dense set?

5.3 Prove Theorem 5.2 without the assumption that $\rho_{k+1} = \rho_k$ when (4.9) holds.

5.4 Note that Assumption 5.2 says that, under some circumstances, it is sensible to eliminate shifts and reduce the algorithm to the penalty method. Suggest alternative tests that could be employed to decide to annihilate $\bar{\lambda}^k$ and $\bar{\mu}^k$.

5.5 Make your choices: Implement Algorithm 4.1 with the assumptions made in this chapter. Consider the possibility of using some heuristic for obtaining an approximate global minimizer of the Augmented Lagrangian at Step 1. Run simple examples and draw conclusions.

5.6 Employ your code to solve problems in which the feasible region is empty. Observe whether it behaves as described by the theorems presented in this chapter.

5.7 Discuss the application of the algorithm and theory presented in this chapter to the capacity expansion planning problem [177] presented in Chapter 2. Note that the binary-variable constraints may be modeled as nonlinear constraints, but this does not prevent the use of mixed-integer techniques in the solution process.

5.8 In terms of detecting infeasibility, the pure penalty method seems to have better convergence properties than the Augmented Lagrangian algorithm (why?). Suggest a safeguarded updating procedure for the multipliers taking advantage of this property.

Chapter 6
General Affordable Algorithms

In the global optimization literature, algorithms that are designed to converge not to global minimizers but to mere stationary points (in fact, not necessarily local minimizers) are known as local algorithms. This denomination could be adopted with a warning that local algorithms are generally guaranteed to converge in some sense to stationary points of the optimization problem, independently of the initial approximation. In this sense, they are said to be globally convergent. Roughly speaking, "local algorithm" is synonymous with the "affordable algorithm" of Chapter 3. In general, global optimization algorithms are not reliable for solving large-scale problems, and, for small to medium-scale problems, they are much slower than local algorithms. On the other hand, global optimization software makes use of local algorithms when associated with branch-and-bound procedures by means of which the search space for a global minimizer is reduced.

The denomination local algorithm does not allude to the concept of local convergence, which is related to the convergence of the whole sequence if one starts close enough to a solution. In fact, local algorithms are usually globally convergent in the sense of stationarity of limit points but are not necessarily locally convergent as they may generate sequences that accumulate in more than one cluster point.

In this chapter, the description of a local algorithm based on the Augmented Lagrangian corresponds to Algorithm 4.1 with a precise interpretation of Step 1, which says that the subproblem solution x^k is approximately a KKT point of the subproblem.

First, we consider lower-level constraints of the following form:

$$\Omega = \{x \in \mathbb{R}^n \mid \underline{h}(x) = 0,\ \underline{g}(x) \leq 0\},$$

where $\underline{h} : \mathbb{R}^n \to \mathbb{R}^{\underline{m}}$, $\underline{g} : \mathbb{R}^n \to \mathbb{R}^{\underline{p}}$, and f, h, g, $\underline{h}$, and $\underline{g}$ admit continuous first derivatives on $\mathbb{R}^n$. Consequently, the constrained optimization problem that we wish to solve is

$$\text{Minimize } f(x) \text{ subject to } h(x) = 0,\ g(x) \leq 0,\ x \in \Omega. \tag{6.1}$$

Note that the simple box constraints $\ell \leq x \leq u$ can be expressed trivially in the form $\underline{g}(x) \leq 0$.

Assumption 6.1 below defines the sense in which the approximate minimization at Step 1 of Algorithm 4.1 should be interpreted in the local minimization context.

Assumption 6.1. *At Step 1 of Algorithm 4.1, we obtain $x^k \in \mathbb{R}^n$ such that there exist $v^k \in \mathbb{R}^{\underline{m}}$ and $w^k \in \mathbb{R}^{\underline{p}}_+$ satisfying*

$$||\nabla L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) + \nabla \underline{h}(x^k)v^k + \nabla \underline{g}(x^k)w^k|| \le \varepsilon_k, \tag{6.2}$$

$$||\underline{h}(x^k)|| \le \varepsilon'_k, \text{ and } ||\min\{-\underline{g}(x^k), w^k\}|| \le \varepsilon'_k, \tag{6.3}$$

where the sequence $\{\varepsilon_k\}$ is bounded and the sequence $\{\varepsilon'_k\}$ tends to zero.

Assumption 6.1 establishes a criterion to measure the degree of feasibility and optimality at the approximate solution of the subproblem

$$\text{Minimize } L_{\rho_k}(x, \bar{\lambda}^k, \bar{\mu}^k) \text{ subject to } x \in \Omega. \tag{6.4}$$

Sometimes it is useful to replace Assumption 6.1 with a condition that involves the projection of the gradient $\nabla L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)$ onto the tangent approximation of the lower feasible set Ω. Namely, the requirement for being an approximate solution of the subproblem is given in that case by

$$||P_k(x^k - \nabla L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)) - x^k|| \le \varepsilon_k, \tag{6.5}$$

$$||\underline{h}(x^k)|| \le \varepsilon'_k, \text{ and } ||\underline{g}(x^k)_+|| \le \varepsilon'_k, \tag{6.6}$$

where the sequence $\{\varepsilon_k\}$ is bounded, the sequence $\{\varepsilon'_k\}$ tends to zero, and P_k represents the Euclidean projection operator onto T_k,

$$T_k = \{x \in \mathbb{R}^n \mid \nabla \underline{h}(x^k)^T(x - x^k) = 0 \text{ and } \underline{g}(x^k)_- + \nabla \underline{g}(x^k)^T(x - x^k) \le 0\}. \tag{6.7}$$

The case in which $\lim_{k\to\infty} \varepsilon_k = 0$ corresponds to the AGP condition introduced by Martínez and Svaiter [192]. The geometrical interpretation for (6.5) is given in Figure 6.1. Note that, in the case that $x^k \in \Omega$ and $\underline{h}$ and $\underline{g}$ are affine functions, the subproblem constraints define a polytope that coincides with T_k. In particular, this is the case when Ω is a box (see Figure 6.2). It can be proved that (6.5), (6.6) imply (6.2), (6.3) (see Problem 6.1).

It is interesting to interpret conditions (6.2) and (6.3) in the case in which the constraints of the subproblem define a box, i.e., in the case in which we have $\underline{m} = 0$, $\underline{p} = 2n$,

$$\underline{g}_i(x) = \ell_i - x_i \text{ and } \underline{g}_{n+i}(x) = x_i - u_i \text{ for all } i = 1, \ldots, n.$$

Methods that solve the subproblem (6.4) when the constraints define a box usually preserve feasibility of all the iterates. Therefore, we will have $\underline{g}(x^k) \le 0$ for all k. Now, let us define, for $i = 1, \ldots, n$,

$$w_i^k = \max\left\{0, \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)\right\} \text{ and } w_{n+i}^k = \max\left\{0, -\frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)\right\}.$$

With these definitions, condition (6.2) is trivially satisfied (even for $\varepsilon_k = 0$). Now, let us define, for $i = 1, \ldots, 2n$,

$$z_i^k = \min\{-\underline{g}_i(x^k), w_i^k\}.$$

By definition, if $\frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) \ge 0$, we have that

$$w_i^k = \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k), \quad w_{n+i}^k = 0,$$

$$z_i^k = \min\left\{x_i^k - \ell_i, \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)\right\}, \quad \text{and} \quad z_{n+i}^k = 0,$$

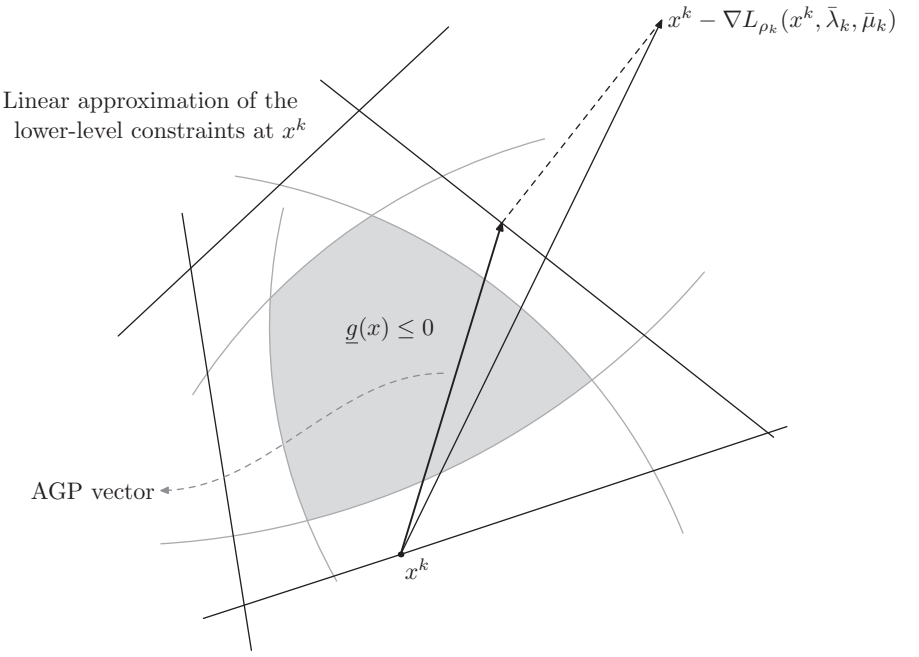

Figure 6.1. *The AGP vector tends to zero if x^k tends to a local minimizer.*

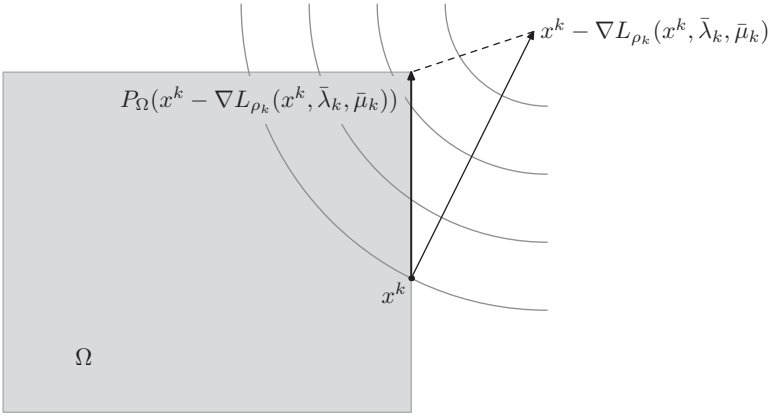

Figure 6.2. *Gradient projection onto a box.*

and, if $\frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) < 0$, we have that

$$w_i^k = 0, \quad w_{n+i}^k = -\frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k),$$

$$z_i^k = 0, \quad \text{and} \quad z_{n+i}^k = \min\left\{u_i - x_i^k, -\frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)\right\}$$

for $i = 1, \ldots, n$.

Therefore, condition (6.3) imposes that $||\hat{z}^k|| \leq \varepsilon'_k$, where $\hat{z}^k \in \mathbb{R}^n$ is defined by

$$\hat{z}_i^k = \begin{cases} \min\left\{x_i^k - \ell_i, \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)\right\} & \text{if } \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) \geq 0, \\ \min\left\{u_i - x_i^k, -\frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)\right\} & \text{if } \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) < 0 \end{cases}$$

for $i = 1, \ldots, n$. Thus,

$$\hat{z}_i^k = \begin{cases} \left|\max\{x_i^k - \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k), \ell_i\} - x_i^k\right| & \text{if } \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) \geq 0, \\ \left|\min\{x_i^k - \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k), u_i\} - x_i^k\right| & \text{if } \frac{\partial}{\partial x_i} L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) < 0 \end{cases}$$

for $i = 1, \ldots, n$. Therefore, $||\hat{z}^k||$ is the norm of $P_\Omega(x^k - \nabla L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k)) - x^k$, where P_Ω represents the projection onto the box Ω. This means that criterion (6.3) coincides with the one ordinarily used for box-constrained minimization, based on projected gradients.

The following theorem plays the role of Theorem 4.1 with respect to the multipliers associated with the subproblem inequality constraints $\underline{g}(x) \leq 0$.

Theorem 6.1. *Assume that the sequence $\{x^k\}$ is generated by Algorithm 4.1 under Assumption 6.1, $\lim_{k\to\infty} \varepsilon_k = 0$, and $K \underset{\infty}{\subset} \mathbb{N}$ is such that $\lim_{k\in K} x^k = x^*$. Then, for $k \in K$ large enough there exists $\tilde{w}_k \in \mathbb{R}^{\underline{p}}_+$ such that*

$$\lim_{k\in K} \left\| \nabla L_{\rho_k}(x^k, \bar{\lambda}^k, \bar{\mu}^k) + \nabla \underline{h}(x^k) v^k + \nabla \underline{g}(x^k) \tilde{w}^k \right\| = 0 \tag{6.8}$$

and

$$\tilde{w}_i^k = 0 \text{ for all } i \in \{1, \ldots, \underline{p}\} \text{ such that } g_i(x^*) < 0. \tag{6.9}$$

Moreover, $\tilde{w}_i^k = w_i^k$ for all $i \in \{1, \ldots, \underline{p}\}$ such that $\underline{g}_i(x^*) \geq 0$.

Proof. Assume that $\underline{g}_i(x^*) < 0$. By (6.3), since $\min\{-\underline{g}_i(x^k), w_i^k\}$ tends to zero, we have that w_i^k tends to zero.

By the continuity of $\nabla \underline{g}_i$, this implies that

$$\lim_{k\in K} w_i^k \nabla \underline{g}_i(x^k) = 0$$

for all $i \in \{1, \ldots, \underline{p}\}$ such that $\underline{g}_i(x^*) < 0$. Therefore, by (6.2) and $\varepsilon_k \to 0$, we have that (6.8) holds by taking

$$\tilde{w}_i^k = 0 \text{ if } \underline{g}_i(x^*) < 0 \text{ and } \tilde{w}_i^k = w_i^k \text{ if } \underline{g}_i(x^*) \geq 0$$

for $i = 1, \ldots, \underline{p}$. This completes the proof. □

In the next theorem, we prove that, when the algorithm analyzed in this chapter admits a feasible limit point, this point satisfies the optimality AKKT condition. In this case, the AKKT condition makes reference to all the constraints of the problem, not only those given by $h(x) = 0$ and $g(x) \leq 0$. More precisely, in the case of the problem of minimizing $f(x)$ subject to $h(x) = 0$, $g(x) \leq 0$, $\underline{h}(x) = 0$, and $\underline{g}(x) \leq 0$, according to

Definition 3.1, we say that x^* satisfies the AKKT condition when there exist sequences $\{x^k\} \subseteq \mathbb{R}^n$, $\{\lambda^k\} \subseteq \mathbb{R}^m$, $\{\mu^k\} \subseteq \mathbb{R}^p_+$, $\{v^k\} \in \mathbb{R}^{\underline{m}}$, and $\{w^k\} \in \mathbb{R}^{\underline{p}}_+$ such that

$$\lim_{k\to\infty} x^k = x^*, \tag{6.10}$$

$$\lim_{k\to\infty} \left\| \nabla f(x^k) + \nabla h(x^k)\lambda^{k+1} + \nabla g(x^k)\mu^{k+1} + \nabla \underline{h}(x^k)v^k + \nabla \underline{g}(x^k)w^k \right\| = 0, \tag{6.11}$$

$$\lim_{k\to\infty} ||\min\{-g(x^k), \mu^{k+1}\}|| = 0, \tag{6.12}$$

and

$$\lim_{k\to\infty} ||\min\{-\underline{g}(x^k), w^k\}|| = 0. \tag{6.13}$$

Of course, we can formally state conditions (6.11) and (6.12) writing λ^k and μ^k instead of λ^{k+1} and μ^{k+1}, respectively. We prefer to write λ^{k+1} and μ^{k+1} here in order to stress the relation with the notation adopted in (4.7) and (4.8).

Theorem 6.2. *Assume that the sequence $\{x^k\}$ is generated by Algorithm 4.1 with Assumption 6.1 for the minimization of $f(x)$ subject to $h(x) = 0, g(x) \leq 0, \underline{h}(x) = 0$, and $\underline{g}(x) \leq 0$ and $K \underset{\infty}{\subseteq} \mathbb{N}$ is such that $\lim_{k\in K} x^k = x^*$ and x^* is feasible. Moreover, assume that the bounded sequence $\{\varepsilon_k\}$ in Assumption 6.1 is such that $\lim_{k\in K} \varepsilon_k = 0$. Then, x^* satisfies the AKKT conditions for the optimization problem.*

Proof. By (6.2) and straightforward calculations using the definitions (4.7) and (4.8) of λ^{k+1} and μ^{k+1}, we have that

$$\lim_{k\in K} \left\| \nabla f(x^k) + \nabla h(x^k)\lambda^{k+1} + \nabla g(x^k)\mu^{k+1} + \nabla \underline{h}(x^k)v^k + \nabla \underline{g}(x^k)w^k \right\| = 0. \tag{6.14}$$

Moreover, by Theorem 4.1, we have that $\lim_{k\in K} ||\min\{-g(x^k), \mu^{k+1}\}|| = 0$. Therefore, by the feasibility of x^* and (6.3), it turns out that x^* is an AKKT point, as we wanted to prove. □

Theorem 6.2 induces a natural stopping criterion for Algorithm 4.1 under Assumption 6.1. Given $\varepsilon > 0$, it is sensible to stop (declaring success) when

$$\left\| \nabla f(x^k) + \nabla h(x^k)\lambda^{k+1} + \nabla g(x^k)\mu^{k+1} + \nabla \underline{h}(x^k)v^k + \nabla \underline{g}(x^k)w^k \right\| \leq \varepsilon, \tag{6.15}$$

$$||h(x^k)|| \leq \varepsilon, ||\min\{-g(x^{k+1}), \mu^{k+1}\}|| \leq \varepsilon, \tag{6.16}$$

$$||\underline{h}(x^k)|| \leq \varepsilon, \text{ and } ||\min\{-\underline{g}(x^{k+1}), w^k\}|| \leq \varepsilon. \tag{6.17}$$

Of course, different tolerances may be used in (6.15), (6.16), and (6.17).

Corollary 6.1. *Under the assumptions of Theorem 6.2, if x^* fulfills the CPLD constraint qualification, then x^* is a KKT point of the problem.*

Proof. The proof is a consequence of Theorems 3.6 and 6.2. □

Constrained optimization algorithms have two goals: finding feasible points and minimizing the objective function subject to feasibility. The behavior of algorithms with respect to feasibility thus demands independent study. Employing global optimization techniques, we saw in Chapter 5 that one necessarily finds global minimizers of the infeasibility, a property that cannot be guaranteed using affordable local optimization procedures. In the next theorem, we prove that, by means of Algorithm 4.1 under Assumption 6.1, we necessarily find stationary points of the sum of squares of infeasibilities. The reader will observe that we do not need $\varepsilon_k \to 0$ for proving this important property.

Theorem 6.3. *Assume that the sequence $\{x^k\}$ is obtained by Algorithm 4.1 under Assumption 6.1. Let x^* be a limit point of $\{x^k\}$. Then, x^* satisfies the AKKT condition of the problem*

$$\text{Minimize } \|h(x)\|_2^2 + \|g(x)_+\|_2^2 \text{ subject to } \underline{h}(x) = 0, \underline{g}(x) \leq 0. \tag{6.18}$$

Proof. Since $\underline{h}$ and $\underline{g}$ are continuous and, by Assumption 6.1, $\lim_{k\to\infty} \varepsilon'_k = 0$, we have that $\underline{h}(x^*) = 0$ and $\underline{g}(x^*) \leq 0$.

If the sequence $\{\rho_k\}$ is bounded, we have by (4.9), that $\lim_{k\to\infty} \|h(x^k)\| = \lim_{k\to\infty} \|g(x^k)_+\| = 0$. Thus, the gradient of the objective function of (6.18) vanishes. This implies that KKT (and, hence, AKKT) holds with null Lagrange multipliers corresponding to the constraints.

Let us consider the case in which ρ_k tends to infinity. Defining

$$\begin{aligned}\delta^k = \nabla f(x^k) + \sum_{i=1}^{m} (\bar{\lambda}_i^k + \rho_k h_i(x^k)) \nabla h_i(x^k) + \sum_{i=1}^{p} \max\{0, \bar{\mu}_i^k + \rho_k g_i(x^k)\} \nabla g_i(x^k) \\ + \sum_{i=1}^{m} v_i^k \nabla \underline{h}_i(x^k) + \sum_{i=1}^{p} w_i^k \nabla \underline{g}_i(x^k),\end{aligned} \tag{6.19}$$

by (6.2) and the fact that $\{\varepsilon_k\}$ is bounded, we have that $\{\|\delta^k\|\}$ is bounded too.

Let $K \underset{\infty}{\subset} \mathbb{N}$ be such that $\lim_{k\in K} x^k = x^*$. By Theorem 6.1, we may assume, without loss of generality, that $w_i^k = 0$ for all $i \in \{1, \ldots, p\}$ such that $\underline{g}_i(x^*) < 0$. Therefore, for all $k \in K$, we have that

$$\begin{aligned}\delta^k = \nabla f(x^k) + \sum_{i=1}^{m} (\bar{\lambda}_i^k + \rho_k h_i(x^k)) \nabla h_i(x^k) + \sum_{i=1}^{p} \max\{0, \bar{\mu}_i^k + \rho_k g_i(x^k)\} \nabla g_i(x^k) \\ + \sum_{i=1}^{m} v_i^k \nabla \underline{h}_i(x^k) + \sum_{\underline{g}_i(x^*)=0} w_i^k \nabla \underline{g}_i(x^k).\end{aligned}$$

Dividing by ρ_k, we obtain

$$\begin{aligned}\frac{\delta^k}{\rho_k} = \frac{1}{\rho_k} \nabla f(x^k) + \sum_{i=1}^{m} \left(\frac{\bar{\lambda}_i^k}{\rho_k} + h_i(x^k) \right) \nabla h_i(x^k) + \sum_{i=1}^{p} \max\left\{0, \frac{\bar{\mu}_i^k}{\rho_k} + g_i(x^k)\right\} \nabla g_i(x^k) \\ + \sum_{i=1}^{m} \frac{v_i^k}{\rho_k} \nabla \underline{h}_i(x^k) + \sum_{\underline{g}_i(x^*)=0} \frac{w_i^k}{\rho_k} \nabla \underline{g}_i(x^k)\end{aligned}$$

and, since $\{\|\delta^k\|\}$ is bounded and ρ_k tends to infinity, we have that $\delta_k/\rho_k \to 0$.

If $g_i(x^*) < 0$, since $\{\bar{\mu}^k\}$ is bounded and ρ_k tends to infinity, we have that $\max\{0, \bar{\mu}_i^k/\rho_k + g_i(x^k)\} = 0$ for $k \in K$ large enough. Therefore, by the boundedness of $\{\nabla f(x^k)\}$ and

$\{\bar{\lambda}^k\}$, $\delta_k/\rho_k \to 0$ implies that

$$\lim_{k\in K}\left\|\sum_{i=1}^{m} h_i(x^k)\nabla h_i(x^k)+\sum_{g_i(x^*)\geq 0}\max\left\{0,\frac{\bar{\mu}_i^k}{\rho_k}+g_i(x^k)\right\}\nabla g_i(x^k)\right.$$
$$\left.+\sum_{i=1}^{\underline{m}}\frac{v_i^k}{\rho_k}\nabla \underline{h}_i(x^k)+\sum_{\underline{g}_j(x^*)=0}\frac{w_j^k}{\rho_k}\nabla \underline{g}_j(x^k)\right\|=0. \tag{6.20}$$

If $g_i(x^*)=0$, we clearly have that, by the boundedness of $\{\bar{\mu}^k\}$ and $\{\nabla g(x^k)\}$,

$$\lim_{k\in K}\max\left\{0,\frac{\bar{\mu}_i^k}{\rho_k}+g_i(x^k)\right\}\nabla g_i(x^k)=0.$$

Then, by (6.20), we have that

$$\lim_{k\in K}\left\|\sum_{i=1}^{m} h_i(x^k)\nabla h_i(x^k)+\sum_{g_i(x^*)>0}\max\left\{0,\frac{\bar{\mu}_i^k}{\rho_k}+g_i(x^k)\right\}\nabla g_i(x^k)\right.$$
$$\left.+\sum_{i=1}^{\underline{m}}\frac{v_i^k}{\rho_k}\nabla \underline{h}_i(x^k)+\sum_{\underline{g}_j(x^*)=0}\frac{w_j^k}{\rho_k}\nabla \underline{g}_j(x^k)\right\|=0.$$

But, if $g_i(x^*)>0$ and $k\in K$ is large enough, we have that

$$\max\left\{0,\frac{\bar{\mu}_i^k}{\rho_k}+g_i(x^k)\right\}=g_i(x^k)+\frac{\bar{\mu}_i^k}{\rho_k}.$$

Thus

$$\lim_{k\in K}\left\|\sum_{i=1}^{m} h_i(x^k)\nabla h_i(x^k)+\sum_{g_i(x^*)>0}\left(\frac{\bar{\mu}_i^k}{\rho_k}+g_i(x^k)\right)\nabla g_i(x^k)\right.$$
$$\left.+\sum_{i=1}^{\underline{m}}\frac{v_i^k}{\rho_k}\nabla \underline{h}_i(x^k)+\sum_{\underline{g}_j(x^*)=0}\frac{w_j^k}{\rho_k}\nabla \underline{g}_j(x^k)\right\|=0.$$

Therefore, since $\bar{\mu}_i^k/\rho_k \to 0$, we have

$$\lim_{k\in K}\left\|\sum_{i=1}^{m} h_i(x^k)\nabla h_i(x^k)+\sum_{g_i(x^*)>0} g_i(x^k)\nabla g_i(x^k)\right.$$
$$\left.+\sum_{i=1}^{\underline{m}}\frac{v_i^k}{\rho_k}\nabla \underline{h}_i(x^k)+\sum_{\underline{g}_j(x^*)=0}\frac{w_j^k}{\rho_k}\nabla \underline{g}_j(x^k)\right\|=0.$$

This obviously implies that

$$\lim_{k\in K}\left\|\sum_{i=1}^{m} h_i(x^k)\nabla h_i(x^k)+\sum_{g_i(x^*)\geq 0} g_i(x^k)\nabla g_i(x^k)\right.$$
$$\left.+\sum_{i=1}^{\underline{m}}\frac{v_i^k}{\rho_k}\nabla \underline{h}_i(x^k)+\sum_{\underline{g}_j(x^*)=0}\frac{w_j^k}{\rho_k}\nabla \underline{g}_j(x^k)\right\|=0. \tag{6.21}$$

But

$$\nabla[||h(x^k)||_2^2+||g(x^k)_+||_2^2]=2\left[\sum_{i=1}^{m}h_i(x^k)\nabla h_i(x^k)+\sum_{g_i(x^*)\geq 0}g_i(x^k)\nabla g_i(x^k)\right].$$

Therefore, by (6.21), the limit point x^* satisfies the AKKT condition of (6.18). □

Corollary 6.2. *Every limit point generated by Algorithm 4.1 under Assumption 6.1 at which the constraints* $\underline{h}(x)=0$, $\underline{g}(x)\leq 0$ *satisfy the CPLD constraint qualification is a KKT point of the problem of minimizing the infeasibility* $||h(x)||_2^2+||g(x)_+||_2^2$ *subject to* $\underline{h}(x)=0$ *and* $\underline{g}(x)\leq 0$.

6.1 ▪ The algorithm with abstract constraints

We will finish this chapter by considering the case in which the lower-level set Ω, instead of being defined by $\underline{h}(x)=0$ and $\underline{g}(x)\leq 0$, is an arbitrary closed and convex set, possibly without an obvious representation in terms of equalities and inequalities. In this case, it may be convenient to define Step 1 of Algorithm 4.1 in a different way than that presented in Assumption 6.1. Assumption 6.2 below gives the appropriate definition in this case.

Assumption 6.2. *At Step 1 of Algorithm 4.1, we obtain* $x^k\in\Omega$ *such that*

$$\left\|P_\Omega(x^k-\nabla L_{\rho_k}(x^k,\bar{\lambda}^k,\bar{\mu}^k))-x^k\right\|\leq\varepsilon_k, \tag{6.22}$$

where the sequence $\{\varepsilon_k\}$ *tends to zero.*

Theorem 6.4. *Assume that the sequence* $\{x^k\}$ *is generated by Algorithm 4.1 under Assumption 6.2 for the minimization of* $f(x)$ *subject to* $h(x)=0$, $g(x)\leq 0$, *and* $x\in\Omega$, *with* Ω *closed and convex. Assume that* $K\underset{\infty}{\subset}\mathbb{N}$ *is such that* $\lim_{k\in K}x^k=x^*$ *and* x^* *is feasible. Then,*

$$\lim_{k\in K}\left\|P_\Omega\left[x^k-\left(\nabla f(x^k)+\nabla h(x^k)\lambda^{k+1}+\nabla g(x^k)\mu^{k+1}\right)\right]-x^k\right\|=0 \tag{6.23}$$

and

$$\lim_{k\in K}\min\{-g_i(x^k),\mu_{i+1}^k\}=0 \text{ for all } i=1,\ldots,p. \tag{6.24}$$

Proof. By (6.22) and straightforward calculations using the definitions of λ^{k+1} and μ^{k+1}, we have that (6.23) holds. Moreover, by Theorem 4.1, we have that

$$\lim_{k\in K}||\min\{-g(x^k),\mu^{k+1}\}||=0.$$

This completes the proof. □

Theorem 6.4 provides another useful stopping criterion. Given $\varepsilon>0$, we stop declaring success when

$$\left\|P_\Omega\left[x^k-\left(\nabla f(x^k)+\nabla h(x^k)\lambda^{k+1}+\nabla g(x^k)\mu^{k+1}\right)\right]-x^k\right\|\leq\varepsilon, \tag{6.25}$$

$$||\min\{-g(x^k),\mu^{k+1}\}||\leq\varepsilon, \tag{6.26}$$

and

$$||h(x^k)|| \leq \varepsilon. \tag{6.27}$$

This is the criterion usually employed in practice when the lower-level constraints have the box form $\ell \leq x \leq u$.

It remains to prove the feasibility result that corresponds to Theorem 6.3 in the case that Ω is an "abstract" closed and convex set. In analogy to Theorem 6.3, Theorem 6.5 below shows that the algorithm makes the best possible work in the process of trying to find feasible points.

Theorem 6.5. *Assume that the sequence $\{x^k\}$ is obtained by Algorithm 4.1 under Assumption 6.2. Assume that x^* is a limit point of $\{x^k\}$. Then, x^* is a stationary point (in the sense of (3.31)) of*

$$\textit{Minimize } ||h(x)||_2^2 + ||g(x)_+||_2^2 \textit{ subject to } x \in \Omega. \tag{6.28}$$

Proof. If the sequence $\{\rho_k\}$ is bounded, the desired result follows as in Theorem 6.3.

Assume that $\rho_k \to \infty$ and let $K \underset{\infty}{\subset} \mathbb{N}$ be such that $\lim_{k\in K} x^k = x^*$. By (6.22), for all $k \in K$, we have that

$$\left\| P_\Omega\left(x^k - \left(\nabla f(x^k) + \rho_k \left[\nabla h(x^k)\left(h(x^k) + \frac{\bar\lambda^k}{\rho_k}\right) + \nabla g(x^k)\left(g(x^k) + \frac{\bar\mu^k}{\rho_k}\right)_+ \right] \right) \right) - x^k \right\| \leq \varepsilon_k. \tag{6.29}$$

Since $\rho_k \to \infty$, we have that $1/\rho_k < 1$ for k large enough. Therefore, by (3.29), (6.29) implies

$$\left\| P_\Omega\left(x^k - \left(\nabla f(x^k)/\rho_k + \nabla h(x^k)\left(h(x^k) + \frac{\bar\lambda^k}{\rho_k}\right) + \nabla g(x^k)\left(g(x^k) + \frac{\bar\mu^k}{\rho_k}\right)_+ \right) \right) - x^k \right\| \leq \varepsilon_k \tag{6.30}$$

for $k \in K$ large enough. Since $\rho_k \to \infty$ and $\{\bar\mu^k\}$ is bounded, we have that for $k \in K$ large enough, $(g_i(x^k) + \bar\mu_i^k/\rho_k)_+ = 0$ whenever $g_i(x^*) < 0$. Therefore, by (6.30),

$$\left\| P_\Omega\left(x^k - \left(\nabla f(x^k)/\rho_k + \nabla h(x^k)\left(h(x^k) + \frac{\bar\lambda^k}{\rho_k}\right) + \sum_{g_i(x^*)\geq 0} \nabla g_i(x^k)\left(g_i(x^k) + \frac{\bar\mu_i^k}{\rho_k}\right)_+ \right) \right) - x^k \right\| \leq \varepsilon_k \tag{6.31}$$

for $k \in K$ large enough. By the boundedness of $\{\bar\lambda^k\}$, the continuity of ∇f, ∇h, ∇g, and P_Ω, and, consequently, the uniform continuity of these functions on a compact set that contains the points x^k for all $k \in K$, since $\{\varepsilon_k\}$ tends to zero, (6.31) implies

$$\lim_{k\in K} \left\| P_\Omega\left(x^k - \left[\nabla h(x^k) h(x^k) + \sum_{g_i(x^*)\geq 0} \nabla g_i(x^k) g_i(x^k)_+ \right] \right) - x^k \right\| = 0. \tag{6.32}$$

This implies the thesis of the theorem. □

6.2 ▪ Review and summary

In this chapter, we considered the model Algorithm 4.1, where the approximate solution of the subproblem is interpreted as the approximate fulfillment of its KKT condition. In this way, we may use a standard affordable solver for solving the subproblems. According to the theoretical results, if the sequence generated by the algorithm converges to a feasible point, this point satisfies the AKKT condition. Moreover, some iterate satisfies, up to any arbitrarily given precision, the KKT condition of the original problem. However, since the original problem may be infeasible, it is useful to show that the limit points of sequences generated by Algorithm 4.1 with the assumptions of this chapter are stationary points of the infeasibility measure (probably local or even global minimizers of infeasibility). Assumptions 6.1 and 6.2 represent different instances of the main algorithm, corresponding to different definitions of the subproblem constraints.

6.3 ▪ Further reading

Using an additional smoothness (generalized Kurdyka–Lojasiewicz) condition, the fulfillment of a stronger sequential optimality condition by the Augmented Lagrangian method was proved by Andreani, Martínez, and Svaiter [18]. The CAKKT defined in [18] states, in addition to the usual AKKT requirements, that the products between multipliers and constraint values tend to zero. CAKKT is strictly stronger than AKKT. Since stopping criteria based on sequential optimality conditions are natural for every constrained optimization algorithm, the question arises of whether other optimization algorithms generate sequences that satisfy AKKT. Counter-examples and results in [15] indicate that for algorithms based on sequential quadratic programming the answer is negative. The behavior of optimization algorithms in situations where Lagrange multipliers do not exist at all is a subject of current research (see [15]).

6.4 ▪ Problems

6.1 Prove that (6.5), (6.6) implies (6.2), (6.3).

6.2 Work on the calculations to prove (6.14) using (6.2) and the definitions (4.7) and (4.8) of λ^{k+1} and μ^{k+1}.

6.3 Formulate Algorithm 4.1 with the assumptions of this chapter for the case in which Ω is a box. Suggest a projected gradient criterion for deciding to stop the iterative subproblem solver.

6.4 Observing that, for proving Theorem 6.3, it is not necessary to assume that $\varepsilon_k \to 0$, define a version of Algorithm 4.1 in which ε_k is a function of the infeasibility measure $||h(x^k)|| + ||g(x^k)_+||$ (see Martínez and Prudente [189]).

6.5 Formulate Algorithm 4.1 with the assumptions of this chapter for the case in which $p = 0$ and $\Omega = \mathbb{R}^n$. Observe that, if $\varepsilon_k = 0$, the stopping condition for the subproblem is a nonlinear system of equations. Formulate Newton's method for this system and identify causes for ill-conditioning of the Newtonian linear system when ρ_k is large. Decompose the system in order to eliminate the ill-conditioning.

6.6 Suppose that your constrained optimization problem is feasible and that the only stationary points of the infeasibility are the feasible points. Assume that the se-

quence $\{x^k\}$ is obtained by Algorithm 4.1 under Assumption 6.1. Assume that $K \underset{\infty}{\subset} \mathbb{N}$ is such that $\lim_{k\in K} x^k = x^*$. Discuss the following arguments:

(a) By Theorem 6.3, x^* feasible.

(b) By (a) and Theorem 4.1, $\lim_{k\in K} \max\{||h(x^k)||, ||V_k||\} = 0$.

(c) As a consequence of (a), (b), and (4.9), $\{\rho_k\}$ remains bounded.

(d) Hence, unboundedness of the penalty parameters occurs only in the case of infeasibility.

Transform these (wrong) arguments in a topic of research.

6.7 Prove (6.23).

6.8 Complete the details of the proof of Theorem 6.5.

6.9 In many nonlinear optimization problems, restoring feasibility is easy because there exists a problem-oriented procedure for finding feasible points efficiently. This possibility can be exploited within the Augmented Lagrangian framework in the choice of the initial point for solving the subproblems. Namely, one can choose that initial approximation as the result of approximately restoring feasibility starting from the Augmented Lagrangian iterate x^{k-1}. In order to take advantage of this procedure, the penalty parameter should be chosen in such a way that the Augmented Lagrangian function decreases at the restored point with respect to its value at x^{k-1}. Define carefully this algorithm and check the convergence theory. (This idea approximates the Augmented Lagrangian framework to the so-called inexact restoration methods and other feasible methods for constrained optimization [1, 53, 111, 129, 160, 173, 186, 188, 198, 199, 200, 230, 231, 232].)

6.10 It is numerically more attractive to solve (4.6) dividing the Augmented Lagrangian by ρ_k, at least when this penalty parameter is large. Why? Formulate the main algorithms in this form and modify the stopping criterion of the subproblems consequently.

6.11 The process of solving (4.6) with a very small tolerance for convergence can be painful. Suggest alternative practical stopping criteria for the subproblems in accordance (or not) with the theory. (Hints: Relative difference between consecutive iterates and lack of progress during some iterations.) Discuss the theoretical and practical impact of the suggested modifications.

6.12 In the context of the problem above, suggest a stopping criterion for the subproblems that depends on the best feasibility-complementarity achieved at previous outer iterations. Note that it may not be worthwhile to solve subproblems with great precision if we are far from optimality.

6.13 In the case of convergence to an infeasible point, the penalty parameter goes to infinity and, consequently, it is harder and harder to solve the subproblems (4.6) with the given stopping criterion. Why? However, probable infeasibility can be detected evaluating optimality conditions of the sum of squares of infeasibities subject to the lower-level constraints. Add this test to Algorithm 4.1. Discuss the possible effect of this modification in numerical tests. Note that you need a test of stationarity for the infeasibility measure with relative big value of the sum of squares (two tolerances are involved).

6.14 Study classical acceleration procedures for sequences in $\mathbb{R}^n$ and apply these procedures to the choice of initial points for solving the Augmented Lagrangian subproblems. Hint: See Brezinski and Zaglia [70] and the DIIS method of Pulay [219].

6.15 Study different ways of exploiting parallelization in Augmented Lagrangian algorithms, for example, using different initial points in parallel at the solution of the subproblems or solving subproblems with different penalty parameters simultaneously.

Chapter 7
Boundedness of the Penalty Parameters

The theoretical results presented up to now are valid even in the case that the Lagrange multipliers are safeguarded by the trivial choice $\lambda_{\min} = \lambda_{\max} = \mu_{\max} = 0$, i.e., $\bar{\lambda}^k = 0$ and $\bar{\mu}^k = 0$ for all k. This case roughly corresponds to the classical external penalty method [88, 109], in which shifts of the constraints are not employed at all. Something better, however, should be expected when we effectively update the shifts employing the classical commonsense strategy $\bar{\lambda}^{k+1} = \bar{\lambda}^k + \rho_k h(x^k)$ and $\bar{\mu}^{k+1} = (\bar{\mu}^k + \rho_k g(x^k))_+$. In this chapter, we will see that, in this case, it is reasonable to expect the boundedness of the sequence of penalty parameters. This is a desirable feature since the difficulty of solving subproblems increases when the penalty parameter grows.

We decided to include in this chapter bounded penalty results that employ minimal advanced knowledge on analysis and perturbation theory. On the one hand, we wish to maintain full readability of this book for scientists outside the field of mathematics. On the other hand, although different results of this type have been published (see [48, 108]) that comprise situations not considered here, none of these results encompasses all the situations addressed by this chapter.

Throughout the chapter, as well as in Chapter 6, we assume that our constrained optimization problem is given by

$$\text{Minimize } f(x) \text{ subject to } h(x) = 0,\ g(x) \leq 0,\ x \in \Omega, \tag{7.1}$$

where $\Omega = \{x \in \mathbb{R}^n \mid \underline{h}(x) = 0,\ \underline{g}(x) \leq 0\}$, $h : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^p$, $\underline{h} : \mathbb{R}^n \to \mathbb{R}^{\underline{m}}$, and $\underline{g} : \mathbb{R}^n \to \mathbb{R}^{\underline{p}}$. We will assume that f, h, g, $\underline{h}$, and $\underline{g}$ admit continuous derivatives for all $x \in \mathbb{R}^n$.

7.1 ▪ Assumptions on the sequence

The results on boundedness of the penalty parameters depend on a set of assumptions about which we can only claim its plausibility, as, in general, there are no theoretical results that rigorously justify their satisfaction. The main one is the following.

Assumption 7.1. *The sequence $\{x^k\}$ generated by Algorithm 4.1 is convergent.*

It is not possible to guarantee theoretically the fulfillment of Assumption 7.1. In fact, the problem (6.1) could have several global minimizers and the algorithm could oscillate between them. There are no practical modifications of the main algorithm that ensure

convergence to only one minimizer. In other words, if one modifies the algorithm trying to guarantee convergence to only one point, one could be deteriorating the algorithm in practice instead of improving it. However, Assumption 7.1 is plausible because, in general, one solves each subproblem by some iterative method that employs, as initial approximation, the solution of the previous subproblem. As a consequence, two consecutive outer iterates tend to be close, the difference between them usually tending to zero, and multiple limit points are quite unusual.

Assumption 7.2. *If x^* is a limit of a sequence $\{x^k\}$ generated by Algorithm 4.1, then x^* is feasible and satisfies the KKT condition of problem (6.1).*

By Assumption 7.2, there exist $\lambda^* \in \mathbb{R}^m$, $\mu^* \in \mathbb{R}^p_+$, $v^* \in \mathbb{R}^{\underline{m}}$, and $w^* \in \mathbb{R}^{\underline{p}}_+$ such that

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* + \nabla g(x^*)\mu^* + \nabla \underline{h}(x^*)v^* + \nabla \underline{g}(x^*)w^* = 0, \tag{7.2}$$

$$h(x^*) = 0,\ g(x^*) \le 0,\ \underline{h}(x^*) = 0,\ \underline{g}(x^*) \le 0, \tag{7.3}$$

$$\mu_i^* = 0 \text{ for all } i \in \{1,\dots,p\} \text{ such that } g_i(x^*) < 0, \tag{7.4}$$

and

$$w_i^* = 0 \text{ for all } i \in \{1,\dots,\underline{p}\} \text{ such that } \underline{g}_i(x^*) < 0. \tag{7.5}$$

The third assumption requires "strict complementarity," i.e., multipliers associated with active inequality constraints should be strictly positive.

Assumption 7.3. *If x^*, μ^*, and w^* satisfy (7.2)–(7.5), we have that*

$$\mu_i^* > 0 \textit{ for all } i \in \{1,\dots,p\} \textit{ such that } g_i(x^*) = 0 \tag{7.6}$$

and

$$w_i^* > 0 \textit{ for all } i \in \{1,\dots,\underline{p}\} \textit{ such that } \underline{g}_i(x^*) = 0. \tag{7.7}$$

Without loss of generality, we assume that there exist indices $1 \le q \le p$ and $1 \le \underline{q} \le \underline{p}$ such that

$$g_i(x^*) = 0 \text{ for all } i \in \{1,\dots,q\} \text{ and } g_i(x^*) < 0 \text{ for all } i \in \{q+1,\dots,p\}, \tag{7.8}$$

and

$$\underline{g}_i(x^*) = 0 \text{ for all } i \in \{1,\dots,\underline{q}\} \text{ and } \underline{g}_i(x^*) < 0 \text{ for all } i \in \{\underline{q}+1,\dots,\underline{p}\}. \tag{7.9}$$

From now on, we denote

$$g^q(x) = \begin{pmatrix} g_1(x) \\ \vdots \\ g_q(x) \end{pmatrix} \text{ and } \underline{g}^{\underline{q}}(x) = \begin{pmatrix} \underline{g}_1(x) \\ \vdots \\ \underline{g}_{\underline{q}}(x) \end{pmatrix}.$$

As a consequence of Assumptions 7.1–7.3, the primal-dual solution $(x^*, \lambda^*, \mu^*, v^*, w^*)$ solves the following nonlinear system of $n + m + q + \underline{m} + \underline{q}$ equations and unknowns:

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(x^*) + \sum_{i=1}^{q} \mu_i^* \nabla g_i(x^*) + \sum_{i=1}^{\underline{m}} v_i^* \nabla \underline{h}_i(x^*) + \sum_{i=1}^{\underline{q}} w_i^* \nabla \underline{g}_i(x^*) = 0, \tag{7.10}$$

$$h(x^*) = 0,\ g^q(x^*) = 0,\ \underline{h}(x^*) = 0,\ \underline{g}^{\underline{q}}(x^*) = 0. \tag{7.11}$$

7.2 ▪ Regularity assumptions

The following assumption condenses regularity properties of the nonlinear system given by (7.10), (7.11). We will assume that the functions f, h, g, $\underline{h}$, and $\underline{g}$ admit continuous second derivatives in a neighborhood of x^*. We will also require the nonsingularity of the Jacobian of the whole system (7.10), (7.11) at this point, which implies the LICQ condition at x^*. By continuity, these regularity properties remain valid in a neighborhood of x^*. A sufficient condition for the nonsingularity of the Jacobian is the LICQ condition combined with the positive definiteness of the Hessian of the Lagrangian on the null-space of the matrix defined by the gradients of the active constraints. However, second-order sufficient conditions will not be postulated in this section, where in fact we do not even assume that x^* is a local minimizer. (In a problem at the end of the chapter the reader will be required to analyze an example where x^* is a feasible *maximizer*.)

In order to understand the essence of the boundedness arguments, consider for a moment the simplification of the problem in which $p = \underline{m} = \underline{p} = 0$ (i.e., there are no inequality constraints or lower-level constraints) and we take $\varepsilon_k = 0$ for all k. Therefore, as in (6.14), by (6.2) and (4.7), and considering $\lambda^k = \bar{\lambda}^k$, we have that

$$\nabla f(x^k) + \nabla h(x^k)\lambda^{k+1} = 0 \text{ and } h(x^k) - \lambda^{k+1}/\rho_k = -\lambda^k/\rho_k.$$

Moreover,

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* = 0 \text{ and } h(x^*) - \lambda^*/\rho_k = -\lambda^*/\rho_k.$$

Therefore, the process of computing (x^k, λ^{k+1}) from (x^{k-1}, λ^k) may be thought of as a fixed-point iteration with the primal-dual solution (x^*, λ^*) being a fixed point for every value of ρ_k. We will see, using typical multivariate Calculus arguments, that this fixed-point iteration is contractive and its contraction rate is proportional to $1/\rho_k$. This means that, for k large enough, we can iterate without modifying the penalty parameter preserving convergence to the solution. The arguments in this chapter confirm this intuitive appeal.

Assumption 7.4. *The Jacobian of the system given by (7.10) and (7.11) is well defined, continuous in a neighborhood of* $(x^*, \lambda^*, \mu^*, v^*, w^*)$, *and nonsingular at* $(x^*, \lambda^*, \mu^*, v^*, w^*)$.

By Assumption 7.4, the second derivatives of all the functions involved in the definition of the problem exist and are continuous in a neighborhood of the primal-dual solution. By the continuity of the matrix inversion, the Jacobian remains nonsingular in some neighborhood of $(x^*, \lambda^*, \mu^*, v^*, w^*)$. In particular, the gradients

$$\nabla h_1(x^*), \ldots, \nabla h_m(x^*), \nabla g_1(x^*), \ldots, \nabla g_q(x^*),$$
$$\nabla \underline{h}_1(x^*), \ldots, \nabla \underline{h}_{\underline{m}}(x^*), \nabla \underline{g}_1(x^*), \ldots, \nabla \underline{g}_{\underline{q}}(x^*)$$

of the active constraints are linearly independent. So, we implicitly assume that the point x^* is regular or that it satisfies the LICQ constraint qualification.

In the rest of this chapter, we will always consider that the sequence $\{x^k\}$ is generated by Algorithm 4.1 with $\varepsilon_k \to 0$ and that Assumption 6.1 holds, with the meaning for λ^k, μ^k, $\bar{\lambda}^k$, and $\bar{\mu}^k$ given in Algorithm 4.1 and v^k and w^k as defined in Assumption 6.1.

The following assumption could seem to be a consequence of Theorem 6.1 and formula (6.9). Theorem 6.1 essentially says that, at the solution of the subproblem, we could assume that $w_i^k = 0$ if $\underline{g}_i(x^*) < 0$. However, rigorously speaking, this is not exactly the

case, because the subproblem solver that gets (6.2) and (6.3) could, perhaps, compute a different sequence $\{w^k\}$ not satisfying that property. An unadvised reader may think it would be enough to redefine $w_i^k = 0$ whenever $\underline{g}_i(x^*) < 0$. In any case, we would still have $w_i^k \to 0$ and this choice of w^k would be seen as corresponding to a different tolerance ε_k in (6.2), which would also tend to zero. However, this would not be completely fair, because the new tolerance ε_k would not be known in advance, so that the modified requirement (6.2) would not be algorithmically correct. For this reason, we decided to state as an assumption that w_i^k should be null for k large enough when $g_i(x^*) < 0$. This is a very plausible assumption for algorithms that solve the subproblems and, when the subproblem solver does not naturally satisfy it, a subtle modification normally ensures its fulfillment.

Assumption 7.5. *If $\underline{g}_i(x^*) < 0$, then, for k large enough, we have that $w_i^k = 0$.*

7.3 ▪ A pause for sensitivity

A desirable characteristic of mathematical models is that their solutions vary continuously (and perhaps smoothly) with respect to variations of their data. In constrained optimization problems like (7.1), the data are parameters implicitly defined in the objective function and the constraints. Representing all the parameters by a single vector $c \in \mathbb{R}^{n_{\text{par}}}$, and considering q and $\bar{q}$ as defined in (7.8) and (7.9), respectively, Assumptions 7.2 and 7.3 state (with some abuse of notation) that there exists $c^* \in \mathbb{R}^{n_{\text{par}}}$ such that

$$\nabla f(x^*,c^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(x^*,c^*) + \sum_{i=1}^{q} \mu_i^* \nabla g_i(x^*,c^*) + \sum_{i=1}^{\underline{m}} v_i^* \nabla \underline{h}_i(x^*,c^*) + \sum_{i=1}^{\underline{q}} w_i^* \nabla \underline{g}_i(x^*,c^*) = 0, \tag{7.12}$$

$$h(x^*,c^*) = 0,\ g^q(x^*,c^*) = 0,\ \underline{h}(x^*,c^*) = 0,\ \underline{g}^{\underline{q}}(x^*,c^*) = 0, \tag{7.13}$$

where $\mu^* > 0$ and $w^* > 0$.

Equations (7.12) and (7.13) form a system of $n + m + q + \underline{m} + \underline{q}$ equations and $n + m + q + \underline{m} + \underline{q} + n_{\text{par}}$ variables (x, λ, μ, v, w, and c). In this context, Assumption 7.4 states that its Jacobian with respect to the variables x, λ, μ, v, and w is nonsingular when computed at $(x^*, \lambda^*, \mu^*, v^*, w^*, c^*)$. Therefore, by the implicit function theorem [20], assuming continuous second derivatives with respect to c as well, we have that the primal-dual solution $(x(c), \lambda(c), \mu(c), v(c), w(c))$ depends continuously on the parameters c. By the strict complementarity condition (Assumption 7.3), this means that, in a neighborhood of the parameters c^*, the KKT condition is satisfied by a slightly perturbed solution whose derivatives with respect to c can be computed using the chain rule [20]. Therefore, our assumptions in this chapter provide straightforward sufficient conditions for stability of the solution.

A particular case of the representation (7.12), (7.13) corresponds to the perturbations given by $h(x,c) = h(x) - c_h$, $g(x,c) = g(x) - c_g$, $\underline{h}(x,c) = \underline{h}(x) - c_{\underline{h}}$, and $\underline{g}(x,c) = \underline{g}(x) - c_{\underline{g}}$. The derivatives of f with respect to c_h, c_g, $c_{\underline{h}}$, and $c_{\underline{g}}$ are interesting because they reflect the variation of the objective function value at the solution with respect to unitary variations of the resources (constraints). The implicit function theorem shows in this case that those variations are exactly the Lagrange multipliers at the solution, a fact from which the usual interpretation of Lagrange multipliers as "prices" of the resources naturally follows.

7.4 ▪ Convergence of the multipliers

Lemma 7.1 below states some relevant properties of the sequence $\{(x^k, \lambda^k, \mu^k, v^k, w^k)\}$ under the assumptions already stated.

Lemma 7.1. *Suppose that Assumptions 7.1–7.5 hold. Then,*

$$\mu_i^k = 0 \text{ for } i = q+1, \ldots, p \text{ and } k \text{ large enough}, \tag{7.14}$$

$$w_i^k = 0 \text{ for } i = \underline{q}+1, \ldots, \underline{p} \text{ and } k \text{ large enough}, \tag{7.15}$$

$$\lim_{k\to\infty} \lambda^k = \lambda^*, \tag{7.16}$$

$$\lim_{k\to\infty} \mu^k = \mu^*, \tag{7.17}$$

$$\lim_{k\to\infty} v^k = v^*, \tag{7.18}$$

and

$$\lim_{k\to\infty} w^k = w^*. \tag{7.19}$$

Proof. By Theorem 4.1, we have that (7.14) takes place. Moreover, (7.15) is a restatement of Assumption 7.5. As a consequence of (7.14) and (7.15), for proving (7.17) and (7.19), we only need to prove that

$$\lim_{k\to\infty} \mu_i^k = \mu_i^* \text{ for } i = 1, \ldots, q \tag{7.20}$$

and

$$\lim_{k\to\infty} w_i^k = w_i^* \text{ for } i = 1, \ldots, \underline{q}. \tag{7.21}$$

As in (6.14), by (6.2), (4.7), (4.8), (7.14), and (7.15), we have that

$$\lim_{k\to\infty} \left\| \nabla f(x^k) + \sum_{i=1}^{m} \lambda_i^{k+1} \nabla h_i(x^k) + \sum_{i=1}^{q} \mu_i^{k+1} \nabla g_i(x^k) + \sum_{i=1}^{\underline{m}} v_i^k \nabla \underline{h}_i(x^k) + \sum_{i=1}^{\underline{q}} w_i^k \nabla \underline{g}_i(x^k) \right\| = 0. \tag{7.22}$$

The sequence $\{(\|\lambda^{k+1}\|, \|\mu^{k+1}\|, \|v^k\|, \|w^k\|)\}$ is bounded. Otherwise, dividing both sides of (7.22) by $\max\{\|\lambda^{k+1}\|, \|\mu^{k+1}\|, \|v^k\|, \|w^k\|\}$ and taking limits for an appropriate subsequence, the term related to $\nabla f(x^k)$ would vanish and the LICQ condition implied by Assumption 7.4 would be violated. Therefore, there exist $K \underset{\infty}{\subseteq} \mathbb{N}$, $\lambda_1, \ldots, \lambda_m$, $\mu_1, \ldots, \mu_q$, $v_1, \ldots, v_{\underline{m}}$, and $w_1, \ldots, w_{\underline{q}}$ such that

$$\lim_{k\in K} \lambda^{k+1} = \lambda, \ \lim_{k\in K} \mu_i^{k+1} = \mu_i \text{ for all } i = 1, \ldots, q,$$

$$\lim_{k\in K} v^k = v, \text{ and } \lim_{k\in K} w_i^k = w_i \text{ for all } i = 1, \ldots, \underline{q}.$$

Therefore, taking limits in (7.22) for $k \in K$, we get

$$\left\| \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) + \sum_{i=1}^{q} \mu_i \nabla g_i(x^*) + \sum_{i=1}^{\underline{m}} v_i \nabla \underline{h}_i(x^*) + \sum_{i=1}^{\underline{q}} w_i \nabla \underline{g}_i(x^*) \right\| = 0. \tag{7.23}$$

By Assumption 7.4, the gradients of the active constraints at x^* are linear independent and, by Assumptions 7.2 and 7.3, (7.10) holds. Therefore, by (7.23), we must have $\lambda = \lambda^*$, $\mu_i = \mu_i^*$ for all $i = 1,\dots,q$, $v = v^*$, and $w_i = w_i^*$ for all $i = 1,\dots,\underline{q}$. Since the same argument is valid for every convergent subsequence, we obtain (7.18) and (7.19). Moreover, analogously, the fact that $\lambda^{k+1} \to \lambda = \lambda^*$ and $\mu^{k+1} \to \mu = \mu^*$ for $k \in K$ also implies (7.16) and (7.17). □

7.5 ▪ Assumption on the true multipliers

Assumption 7.6 below prescribes the natural choice for the safeguarded multipliers. Observe that this is the first time such a specific choice is decided in this book.

Assumption 7.6. *At Step 4 of Algorithm 4.1, if $\lambda^{k+1} \in [\lambda_{\min}, \lambda_{\max}]^m$ and $\mu^{k+1} \in [0, \mu_{\max}]^p$, we choose $\bar{\lambda}^{k+1} = \lambda^{k+1}$ and $\bar{\mu}^{k+1} = \mu^{k+1}$.*

The following assumption establishes that the safeguarding bounds $\lambda_{\min}$, $\lambda_{\max}$, and $\mu_{\max}$ are large enough to contain the true Lagrange multipliers at the solution. If this assumption is not satisfied, the global convergence results of Chapter 6 remain true, but the boundedness of the penalty parameters may no longer hold.

Assumption 7.7. *$\lambda^* \in (\lambda_{\min}, \lambda_{\max})^m$ and $\mu^* \in [0, \mu_{\max})^p$.*

In order to maximize the chance that Assumption 7.7 holds, one may be tempted to use very big values of $\lambda_{\max}$, $\mu_{\max}$, and $-\lambda_{\min}$. However, extreme values of these parameters could lead to extremely big shifts on the Augmented Lagrangian framework, which of course are not recommended from the stability and commonsense points of view. Recall that common sense dictates that, when ρ_k tends to infinity, the shifts should tend to zero. Therefore, the choice of the safeguarding multiplier parameters is a delicate question involving conflicting objectives.

The main consequence of Assumptions 7.6 and 7.7 is that, for k large enough, the safeguarded multipliers are identical to the corresponding multiplier estimates. This is crucial for the local analysis of Algorithm 4.1 and for the boundedness of the penalty parameters.

Lemma 7.2. *Suppose that Assumptions 7.1–7.7 hold. Then, for k large enough, we have that $\bar{\lambda}^k = \lambda^k$ and $\bar{\mu}^k = \mu^k$.*

Proof. By Lemma 7.1, we have that $\lim_{k\to\infty} \lambda^k = \lambda^*$ and $\lim_{k\to\infty} \mu^k = \mu^*$. Then, by Assumption 7.7, we have that $\lambda^{k+1} \in (\lambda_{\min}, \lambda_{\max})^m$ and $\mu^{k+1} \in [0, \mu_{\max})^p$ for k large enough. Thus, by Assumption 7.6, we deduce that $\bar{\lambda}^{k+1} = \lambda^{k+1}$ and $\bar{\mu}^{k+1} = \mu^{k+1}$ for k large enough. □

7.6 ▪ Local reduction of the error

Lemma 7.3. *Suppose that Assumptions 7.1–7.7 hold. Then, there exist $k_0 \in \mathbb{N}$ and $c > 0$ such that, for all $k \geq k_0$,*

$$\begin{aligned} &||x^k - x^*|| + ||\lambda^{k+1} - \lambda^*|| + ||\mu^{k+1} - \mu^*|| + ||v^k - v^*|| + ||w^k - w^*|| \\ &\leq c\Big(\frac{||\lambda^k - \lambda^*||}{\rho_k} + \frac{||\mu^k - \mu^*||}{\rho_k} + ||z(x^k, \lambda^k, \mu_1^k, \dots, \mu_q^k, v^k, w_1^k, \dots, w_{\underline{q}}^k)|| \\ &\qquad + ||\underline{h}(x^k)|| + ||\underline{g}^{\underline{q}}(x^k)|| \Big), \end{aligned} \tag{7.24}$$

where

$$\begin{aligned} &z(x^k,\lambda^{k+1},\mu_1^{k+1},\dots,\mu_q^{k+1},v^k,w_1^k,\dots,w_{\underline{q}}^k)\\ &\equiv \nabla f(x^k)+\nabla h(x^k)\lambda^{k+1}+\sum_{i=1}^{q}\mu_i^{k+1}\nabla g_i(x^k)+\nabla \underline{h}(x^k)v^k+\sum_{i=1}^{\underline{q}}w_i^k\nabla \underline{g}_i(x^k) \end{aligned} \tag{7.25}$$

for all $k\in\mathbb{N}$.

Proof. By (4.7) and Lemma 7.2, we have that

$$h(x^k)-\lambda^{k+1}/\rho_k=-\lambda^k/\rho_k \tag{7.26}$$

for k large enough. By (4.8), we have that $\mu_i^{k+1}=(\bar{\mu}_i^k+\rho_k g_i(x^k))_+$ for all $i=1,\dots,q$. But, by (7.8), (7.6), and Lemma 7.1, $\lim_{k\to\infty}\mu_i^k=\mu_i^*>0$ for all $i=1,\dots,q$. Therefore, for k large enough, we have that $\mu_i^{k+1}=\bar{\mu}_i^k+\rho_k g_i(x^k)$ for all $i=1,\dots,q$. Thus, by Lemma 7.2, for k large enough, $\mu_i^{k+1}=\mu_i^k+\rho_k g_i(x^k)$, or, equivalently,

$$g_i(x^k)-\mu_i^{k+1}/\rho_k=-\mu_i^k/\rho_k \text{ for } i=1,\dots,q. \tag{7.27}$$

Moreover, by (7.2)–(7.5), we have that

$$z(x^*,\lambda^*,\mu_1^*,\dots,\mu_q^*,v^*,w_1^*,\dots,w_{\underline{q}}^*)=0, \tag{7.28}$$

$$h(x^*)-\lambda^*/\rho_k=-\lambda^*/\rho_k, \tag{7.29}$$

$$g_i(x^*)-\mu_i^*/\rho_k=-\mu_i^*/\rho_k \text{ for } i=1,\dots,q, \tag{7.30}$$

$$\underline{h}(x^*)=0, \tag{7.31}$$

and

$$\underline{g^q}(x^*)=0. \tag{7.32}$$

Defining $F_k:\mathbb{R}^{n+m+q+\underline{m}+\underline{q}}\to\mathbb{R}^{n+m+q+\underline{m}+\underline{q}}$ by

$$F_k(x,\lambda,\mu_1,\dots,\mu_q,v,w_1,\dots,w_{\underline{q}})=\begin{pmatrix} z(x,\lambda,\mu_1,\dots,\mu_q,v,w_1,\dots,w_{\underline{q}})\\ h(x)-\lambda/\rho_k\\ g_1(x)-\mu_1/\rho_k\\ \vdots\\ g_q(x)-\mu_q/\rho_k\\ \underline{h}(x)\\ \underline{g^q}(x), \end{pmatrix},$$

by (7.3), (7.8), (7.9), and (7.26)–(7.32), we have that

$$\begin{aligned} &F_k(x^k,\lambda^{k+1},\mu_1^{k+1},\dots,\mu_q^{k+1},v^k,w_1^k,\dots,w_{\underline{q}}^k)-F_k(x^*,\lambda^*,\mu_1^*,\dots,\mu_q^*,v^*,w_1^*,\dots,w_{\underline{q}}^*)\\ &=\begin{pmatrix} z(x^k,\lambda^{k+1},\mu_1^{k+1},\dots,\mu_q^{k+1},v^k,w_1^k,\dots,w_{\underline{q}}^k)\\ (\lambda^*-\lambda^k)/\rho_k\\ (\mu_1^*-\mu_1^k)/\rho_k\\ \vdots\\ (\mu_q^*-\mu_q^k)/\rho_k\\ \underline{h}(x^k)\\ \underline{g^q}(x^k) \end{pmatrix}. \end{aligned} \tag{7.33}$$

By the mean value theorem of integral calculus, we have that

$$\begin{aligned} F_k(x^k,\lambda^{k+1},\mu_1^{k+1},\dots,\mu_q^{k+1},v^k,w_1^k,\dots,w_{\underline{q}}^k)-F_k(x^*,\lambda^*,\mu_1^*,\dots,\mu_q^*,v^*,w_1^*,\dots,w_{\underline{q}}^*) \\ =\left(\int_0^1 F_k'((x^*,\lambda^*,\mu_1^*,\dots,\mu_q^*,v^*,w_1^*,\dots,w_{\underline{q}}^*)+th)dt\right)h, \end{aligned} \tag{7.34}$$

where

$$h=((x^k-x^*)^T,(\lambda^{k+1}-\lambda^*)^T,\mu_1^{k+1}-\mu_1^*,\dots,\mu_q^{k+1}-\mu_q^*,(v^k-v^*)^T,w_1^k-w_1^*,\dots,w_{\underline{q}}^k-w_{\underline{q}}^*)^T.$$

By Assumption 7.4, the average Jacobian in (7.34) is nonsingular for k large enough. Therefore, by (7.33), we have that

$$\begin{aligned} h=&\left(\int_0^1 F_k'((x^*,\lambda^*,\mu_1^*,\dots,\mu_q^*,v^*,w_1^*,\dots,w_{\underline{q}}^*)+th)dt\right)^{-1} \\ &\times\begin{pmatrix} z(x^k,\lambda^{k+1},\mu_1^{k+1},\dots,\mu_q^{k+1},v^k,w_1^k,\dots,w_{\underline{q}}^k) \\ (\lambda^*-\lambda^k)/\rho_k \\ (\mu_1^*-\mu_1^k)/\rho_k \\ \vdots \\ (\mu_q^*-\mu_q^k)/\rho_k \\ \underline{h}(x^k) \\ \underline{g}^{\underline{q}}(x^k) \end{pmatrix} \end{aligned} \tag{7.35}$$

for k large enough. Thus, by the boundedness of the inverse of the Jacobian, the desired result follows applying some norms inequalities to both sides of (7.35). □

Assumption 7.8. *Define, for all* $k\in\mathbb{N}$,

$$E_k=\|h(x^k)\|+\|\min\{-g(x^k),\bar{\mu}^k/\rho_k\}\|.$$

We assume that there exists a sequence $\{\eta_k\}\subseteq\mathbb{R}_{++}$ *such that* $\eta_k\to 0$ *and, for all* $k\in\mathbb{N}$,

$$\|\nabla L_{\rho_k}(x^k,\bar{\lambda}^k,\bar{\mu}^k)+\nabla\underline{h}(x^k)v^k+\nabla\underline{g}(x^k)w^k\|\le\eta_k E_k, \tag{7.36}$$

$$\|\underline{h}(x^k)\|\le\eta_k E_k, \text{ and } \|\min\{-\underline{g}(x^k),w^k\}\|\le\eta_k E_k. \tag{7.37}$$

We may impose the fulfillment of Assumption 7.8 taking $\varepsilon_k\le\eta_k E_k$ and $\varepsilon_k'\le\eta_k E_k$ in (6.2) and (6.3). However, even using an iterative inner algorithm that ensures (6.2) and (6.3), this algorithm may not be able to satisfy (7.36) and (7.37), since the stopping criterion tolerance depends on the iterates, and the increasing precision obtained by the iterative inner algorithm could always be greater than the one imposed by Assumption 7.8 (see [189]). Therefore, the situation in which one cannot satisfy (7.36) and (7.37) for some k should be analyzed. In the following theorem we prove that, when these conditions cannot be achieved, we are able to approximate the solution of (6.1) during the resolution of a single subproblem.

Theorem 7.1. *Assume that, in the process of solving the subproblem at the outer iteration* k, *we compute a sequence* $\{(x^{kj},v^{kj},w^{kj})\}\subseteq\mathbb{R}^n\times\mathbb{R}^{\underline{m}}\times\mathbb{R}^{\underline{p}}$ *such that*

$$\lim_{j\to\infty}\left\|\nabla L_{\rho_k}(x^{kj},\bar{\lambda}^k,\bar{\mu}^k)+\nabla\underline{h}(x^{kj})v^{kj}+\nabla\underline{g}(x^{kj})w^{kj}\right\|=0, \tag{7.38}$$

$$\lim_{j\to\infty} ||\underline{h}(x^{kj})|| = 0, \text{ and } \lim_{j\to\infty} ||\min\{-\underline{g}(x^{kj}), w^{kj}\}|| = 0, \tag{7.39}$$

but, for all $j \in \mathbb{N}$,

$$\begin{gathered}\left\|\nabla L_{\rho_k}(x^{kj}, \bar{\lambda}^k, \bar{\mu}^k) + \nabla \underline{h}(x^{kj})v^{kj} + \nabla \underline{g}(x^{kj})w^{kj}\right\| > \eta_k E_{kj} \text{ or} \\ ||\underline{h}(x^{kj})|| > \eta_k E_{kj} \text{ or } ||\min\{-\underline{g}(x^{kj}), w^{kj}\}|| > \eta_k E_{kj},\end{gathered} \tag{7.40}$$

where $E_{kj} = ||h(x^{kj})|| + ||\min\{-g(x^{kj}), \bar{\mu}^k/\rho_k\}||$. *Then, every limit point* x^* *of the sequence* $\{x^{k1}, x^{k2}, \dots\}$ *is feasible and satisfies the AKKT condition of (6.1).*

Proof. By (7.38)–(7.40), since $\eta_k > 0$, we have that $E_{kj} \to 0$ when $j \to \infty$ and, therefore,

$$\lim_{j\to\infty} ||h(x^{kj})|| = 0 \text{ and } \lim_{j\to\infty} ||\min\{-g(x^{kj}), \bar{\mu}^k/\rho_k\}|| = 0. \tag{7.41}$$

This implies that taking an appropriate subsequence, $\lim_{j\to\infty} ||g(x^{kj})_+|| = 0$ and $\bar{\mu}_i^k/\rho_k = 0$ for all i such that $g_i(x^*) < 0$. Therefore,

$$\lim_{j\to\infty} ||\min\{-g(x^{kj}), \bar{\mu}^k\}|| = 0. \tag{7.42}$$

Now, by (7.38),

$$\begin{aligned}\lim_{j\to\infty} \Big\|&\nabla f(x^{kj}) + \nabla h(x^{kj})(\bar{\lambda}^k + \rho_k h(x^{kj})) + \nabla g(x^{kj})(\bar{\mu}^k + \rho_k g(x^{kj}))_+ \\ &+ \nabla \underline{h}(x^{kj})v^{kj} + \nabla \underline{g}(x^{kj})w^{kj}\Big\| = 0.\end{aligned}$$

Thus, by (7.41),

$$\lim_{j\to\infty} \left\|\nabla f(x^{kj}) + \nabla h(x^{kj})\bar{\lambda}^k + \nabla g(x^{kj})\bar{\mu}^k + \nabla \underline{h}(x^{kj})v^{kj} + \nabla \underline{g}(x^{kj})w^{kj}\right\| = 0. \tag{7.43}$$

Therefore, the thesis follows from (7.43), (7.41), (7.39), and (7.42). □

Lemma 7.4. *Suppose that Assumptions 7.1–7.8 hold. Then, there exists* $k_0 \in \mathbb{N}$ *such that, for all* $k \geq k_0$,

$$||\min\{-g(x^k), \bar{\mu}^k/\rho_k\}|| = ||g^q(x^k)|| \text{ and } ||\min\{-\underline{g}(x^k), w^k\}|| = ||\underline{g}^q(x^k)||.$$

Proof. By Lemma 7.2, we have that $\bar{\mu}^k = \mu^k$ for k large enough. By Lemma 7.1, $\mu_i^k = 0$ if $i > q$ and k large enough. Therefore, for $i > q$ and k large enough, since $-g_i(x^k) > 0$, we have that $\min\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\} = 0$.

By Assumption 7.3 (strict complementarity), (4.8), and the fact that by (7.17) μ^{k+1} tends to μ^*, we have that $\bar{\mu}_i^k + \rho_k\ g_i(x^k) > 0$ if $i \leq q$ and k is large enough. Therefore, $\bar{\mu}_i^k/\rho_k + g_i(x^k) > 0$, and so $-g_i(x^k) < \bar{\mu}_i^k/\rho_k$. Therefore, $\min\{-g_i(x^k), \bar{\mu}_i^k/\rho_k\} = -g_i(x^k)$ if $i \leq q$ and k is large enough.

If $i > \underline{q}$ and k is large enough, we have by (7.15) that $w_i^k = 0$ and $-\underline{g}_i(x^k) > 0$. Therefore, $\min\{-\underline{g}_i(x^k), w_i^k\} = 0$ if $i > \underline{q}$ and k is large enough.

If $i \leq \underline{q}$, we have by Assumption 7.3 and (7.19) that $w_i^k > w_i^*/2 > 0$ for all k large enough. Therefore, since $\underline{g}_i(x^k)$ tends to zero for $i \leq \underline{q}$, we have that $\min\{-\underline{g}_i(x^k), w_i^k\} = -\underline{g}_i(x^k)$ for $i \leq \underline{q}$ and k large enough.

This completes the proof. □

Lemma 7.5. *Suppose that Assumptions 7.1–7.8 hold. Then, there exist $k_0 \in \mathbb{N}$ and $c > 0$ such that, for all $k \geq k_0$,*

$$\begin{aligned} &||x^k - x^*|| + ||\lambda^{k+1} - \lambda^*|| + ||\mu^{k+1} - \mu^*|| + ||v^k - v^*|| + ||w^k - w^*|| \\ &\leq c\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + \eta_k\left(||h(x^k)|| + ||g^q(x^k)||\right)\right). \end{aligned} \tag{7.44}$$

Proof. Let $z^k = z(x^k, \lambda^{k+1}, \mu_1^{k+1}, \ldots, \mu_q^{k+1}, v^k, w_1^k, \ldots, w_{\underline{q}}^k)$ as defined in (7.25). By Lemma 7.3, Lemma 7.4, and Assumption 7.7, we have that, for k large enough,

$$\begin{aligned} &||x^k - x^*|| + ||\lambda^{k+1} - \lambda^*|| + ||\mu^{k+1} - \mu^*|| + ||v^k - v^*|| + ||w^k - w^*|| \\ \leq\ & \bar{c}\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + ||z^k|| + ||\underline{h}(x^k)|| + ||\underline{g}^q(x^k)||\right) \\ \leq\ & \bar{c}\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + ||z^k|| + ||\underline{h}(x^k)|| + ||\min\{-\underline{g}(x^k), w^k\}||\right) \\ \leq\ & \bar{c}\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + 3\eta_k E_k\right) \\ \leq\ & c\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + \eta_k E_k\right) \\ =\ & c\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + \eta_k\left(\left(||h(x^k)|| + ||\min\{-g(x^k), \bar{\mu}^k/\rho_k\}||\right)\right)\right) \\ =\ & c\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + \eta_k\left(\left(||h(x^k)|| + ||g^q(x^k)||\right)\right)\right), \end{aligned}$$

where $c = 3\bar{c}$ and $\bar{c} > 0$ is the constant from Lemma 7.3. □

The following lemma says that, if we suppose that ρ_k tends to infinity, the sequence $\{||h(x^k)|| + ||g^q(x^k)||\}$ converges to zero superlinearly.

Lemma 7.6. *Suppose that Assumptions 7.1–7.8 hold and that ρ_k tends to infinity. Then, there exist $k_0 \in \mathbb{N}$ and $c_1 > 0$ such that*

$$||h(x^k)|| + ||g^q(x^k)|| \leq \frac{c_1}{\rho_k}\left(||h(x^{k-1})|| + ||g^q(x^{k-1})||\right)$$

for all $k \geq k_0$.

Proof. By Lemma 7.5, for $k \geq k_0$,

$$||x^k - x^*|| \leq c\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + \eta_k\left(||h(x^k)|| + ||g^q(x^k)||\right)\right).$$

Thus, by the continuity and boundedness of the derivatives of h and g, there exist $L > 0$ and $k_1 \geq k_0$ such that

$$||h(x^k)|| + ||g^q(x^k)|| \leq cL\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k + \eta_k\left(||h(x^k)|| + ||g^q(x^k)||\right)\right)$$

for all $k \geq k_1$.

Then, by Assumption 7.8 and Lemma 7.4,

$$||h(x^k)|| + ||g^q(x^k)|| \leq 2cL\left(||\lambda^k - \lambda^*||/\rho_k + ||\mu^k - \mu^*||/\rho_k\right) \tag{7.45}$$

for all $k \geq k_1$.

By Lemma 7.5, we also have that there exists $c_2 > 0$ such that for all k large enough,

$$||\lambda^k - \lambda^*|| + ||\mu^k - \mu^*|| \leq c_2\left(\frac{||\lambda^{k-1} - \lambda^*||}{\rho_{k-1}} + \frac{||\mu^{k-1} - \mu^*||}{\rho_{k-1}} + \eta_k\left(||h(x^{k-1})|| + ||g^q(x^{k-1})||\right)\right).$$

Thus,

$$\frac{||\lambda^{k-1}-\lambda^*||}{\rho_{k-1}}+\frac{||\mu^{k-1}-\mu^*||}{\rho_{k-1}}\geq\frac{||\lambda^k-\lambda^*||+||\mu^k-\mu^*||}{c_2}-\eta_k\left(||h(x^{k-1})||+||g^q(x^{k-1})||\right). \tag{7.46}$$

By Lemmas 7.1 and 7.2, for k large enough, we have that

$$\lambda^k=\lambda^{k-1}+\rho_{k-1}h(x^{k-1}) \text{ and } \mu^k=\mu^{k-1}+\rho_{k-1}g_i(x^{k-1}) \text{ for } i=1,\dots,q.$$

Therefore,

$$||h(x^{k-1})||=\frac{||\lambda^k-\lambda^{k-1}||}{\rho_{k-1}} \text{ and } |g_i(x^{k-1})|=\frac{|\mu_i^k-\mu_i^{k-1}|}{\rho_{k-1}} \text{ for } i=1,\dots,q.$$

Thus,

$$||h(x^{k-1})||\geq\frac{||\lambda^{k-1}-\lambda^*||}{\rho_{k-1}}-\frac{||\lambda^k-\lambda^*||}{\rho_{k-1}}$$

and

$$|g_i(x^{k-1})|\geq\frac{|\mu_i^{k-1}-\mu_i^*|}{\rho_{k-1}}-\frac{|\mu_i^k-\mu_i^*|}{\rho_{k-1}} \text{ for } i=1,\dots,q.$$

Since $\mu_i^k=\mu_i^*=0$ for $i=q+1,\dots,p$ and k large enough, the last two inequalities imply that there exists $c_3>0$ such that for all k large enough,

$$\frac{||\lambda^{k-1}-\lambda^*||}{\rho_{k-1}}+\frac{||\mu^{k-1}-\mu^*||}{\rho_{k-1}}\leq c_3\left(||h(x^{k-1})||+||g^q(x^{k-1})||+\frac{||\lambda^k-\lambda^*||}{\rho_{k-1}}+\frac{||\mu^k-\mu^*||}{\rho_{k-1}}\right).$$

Therefore, by (7.46),

$$\begin{aligned}&\frac{||\lambda^k-\lambda^*||+||\mu^k-\mu^*||}{c_2}-\eta_k\left(||h(x^{k-1})||+||g^q(x^{k-1})||\right)\\&\leq c_3\left(||h(x^{k-1})||+||g^q(x^{k-1})||+\frac{||\lambda^k-\lambda^*||}{\rho_{k-1}}+\frac{||\mu^k-\mu^*||}{\rho_{k-1}}\right)\end{aligned}$$

for k large enough. Since $\rho_k\to\infty$, this implies that there exists $c_4>0$ such that for k large enough,

$$||\lambda^k-\lambda^*||+||\mu^k-\mu^*||\leq c_4\left(||h(x^{k-1})||+||g^q(x^{k-1})||\right).$$

Then, by (7.45),

$$||h(x^k)||+||g^q(x^k)||\leq 2cL\left(\frac{||\lambda^k-\lambda^*||}{\rho_k}+\frac{||\mu^k-\mu^*||}{\rho_k}\right)\leq 2cc_4L\frac{(||h(x^{k-1})||+||g^q(x^{k-1})||)}{\rho_k}$$

for k large enough. Defining $c_1=2cc_4L$, we obtain the desired result. □

7.7 ▪ Boundedness theorem

In Algorithm 4.1, when the progress condition (4.9) is satisfied, one chooses $\rho_{k+1} \geq \rho_k$. One of the possibilities (the adequate one if one believes that maintaining a bounded penalty parameter is crucial) is to choose $\rho_{k+1} = \rho_k$ in that case. However, a moderate increase of ρ_k even in that case may be recommended to obtain a high speed of convergence if the conditions of the problem guarantee that big penalty parameters will not cause severe instability problems. We will return to this discussion in a forthcoming chapter. Here, in order to prove the boundedness theorem below, we will state the final assumption of this chapter.

Assumption 7.9. *In Algorithm 4.1, whenever (4.9) is satisfied, we choose* $\rho_{k+1} = \rho_k$.

Theorem 7.2. *Suppose that Algorithm 4.1 under Assumption 6.1 and* $\varepsilon_k \to 0$ *is applied to (6.1) and that Assumptions 7.1–7.9 are fulfilled. Then, there exists* $\bar{\rho} > 0$ *such that* $\rho_k = \bar{\rho}$ *for all* k *large enough.*

Proof. Suppose that $\rho_k \to \infty$. By Lemmas 7.4 and 7.6, there exist $k_0 \in \mathbb{N}$ and $c_1 > 0$ such that, for all $k \geq k_0$,

$$\left\|h(x^k)\right\| + \left\|\min\left\{-g(x^k), \bar{\mu}^k/\rho_k\right\}\right\| \leq \frac{c_1}{\rho_k}\left(\left\|h(x^{k-1})\right\| + \left\|\min\left\{-g(x^{k-1}), \bar{\mu}_{k-1}/\rho_{k-1}\right\}\right\|\right).$$

Let $k_1 \geq k_0$ such that for all $k \geq k_1$, $c_1/\rho_k \leq \tau$. Then, by (4.9) and Assumption 7.9, for all $k \geq k_1$, we have that $\rho_{k+1} = \rho_k$. This contradicts the assumption that $\rho_k \to \infty$. As a consequence, the sequence $\{\rho_k\}$ is bounded and the proof is complete. □

7.8 ▪ Review and summary

If the penalty parameter of a subproblem is very large, the subproblem becomes hard to solve and its approximate solution is not reliable. In this case, if a point y is slightly more feasible than a point x, we generally have that $L_{\rho_k}(x, \bar{\lambda}^k, \bar{\mu}^k) \gg L_{\rho_k}(y, \bar{\lambda}^k, \bar{\mu}^k)$, even when x is a much better approximation to the subproblem's solution than y, and we tend to choose y as an approximate solution of the subproblem, instead of x. Therefore, very large penalty parameters should be avoided as much as possible. In this chapter, we showed that, in general, the Augmented Lagrangian method has the property of converging to the correct solution maintaining bounded penalty parameters. In practice, this generally means that the occurrence of extremely large penalty parameters is a symptom of infeasibility.

7.9 ▪ Further reading

The question about the boundedness of the penalty parameters under the condition that $\rho_{k+1} = \rho_k$ when (4.9) does not hold is closely related to results on the order of convergence when we allow the penalty parameters to grow indefinitely. Birgin, Fernández, and Martínez [48] proved that, for an Augmented Lagrangian algorithm similar (but not identical) to Algorithm 4.1, the results of the present chapter follow by employing a strict Mangasarian–Fromovitz condition, instead of the linear independence of the gradients at the solution, together with a second-order sufficient optimality condition that ensures an error-bound property. In [108], Fernández and Solodov proved that, under a sufficient second-order condition, no constraint qualification is necessary for proving this type of result if one additionally imposes that the initial Lagrange multipliers are close enough to

the true multipliers at the solution. More general results under noncriticality of Lagrange multipliers were obtained in [153]. Izmailov [152] studied the trajectory of the sequence of Lagrange multipliers when the true ones at the solution are not unique. The behavior of Augmented Lagrangian methods in mathematical problems with complementarity constraints was studied in [155].

7.10 ▪ Problems

7.1 Consider the problem

$$\text{Minimize } -(x_1 + x_2) \text{ subject to } x_1 x_2 = 1.$$

Show that $\bar{x} = (1,1)^T$ is a KKT point but is a global maximizer (not a minimizer) of the problem. Show that there exists a sequence that satisfies the assumptions of this chapter and converges to $\bar{x}$. Deduce that the algorithm considered in this chapter may converge to a global maximizer with bounded penalty parameters. Discuss this apparent paradox.

7.2 Work on (7.35) to conclude that it implies the thesis (7.24) in Lemma 7.3.

7.3 Consider Algorithm 4.1 with the following modification: At Step 3 we define $\rho_{k+1} = \gamma\rho_k$ for all $k \in \mathbb{N}$, independently of the fulfillment of (4.9). Verify that the global convergence proofs hold for this modification. Then, mimic the proofs of the present chapter and draw conclusions related to "superlinear" convergence. Does this mean that the modified algorithm should be better than the original one? Discuss.

7.4 Assume that, by chance, you provide a solution to the problem and the true multipliers as initial approximations for the application of the Augmented Lagrangian algorithm. Which is the behavior of the method in this case? What happens if you provide the initial primal solution but not the correct multipliers? What happens if you provide the true multipliers but not the correct primal solution?

7.5 Argue in favor of starting with a big initial penalty parameter ρ_1 (known as a warm start) when you have at hand a solution of a similar problem.

7.6 Show an example in which Algorithm 4.1 generates two subsequences converging to different solutions. Which is the consequence in terms of boundedness of the penalty parameters?

7.7 Consider the Augmented Lagrangian algorithm with subproblems' initial points chosen by a restoration process, as discussed in Problem 6.9. Analyze this algorithm from the point of view of local convergence using the results of the present chapter.

Chapter 8
Solving Unconstrained Subproblems

At each iteration of the Augmented Lagrangian method, we need to minimize the function

$$L_{\rho_k}(x, \bar{\lambda}^k, \bar{\mu}^k)$$

with respect to x on a generally simple set that we call Ω. In this chapter, we consider the less complicated case, in which $\Omega = \mathbb{R}^n$. This means that, at each outer iteration, we need to solve an unconstrained minimization problem. For simplicity, we denote $\rho = \rho_k$, $\lambda = \bar{\lambda}^k$, $\mu = \bar{\mu}^k$, and

$$F(x) = L_\rho(x, \lambda, \mu) \tag{8.1}$$

throughout this chapter.

In principle, we will assume that F has continuous first derivatives for all $x \in \mathbb{R}^n$ without mentioning second derivatives at all. This omission is convenient at the beginning since the Augmented Lagrangian function has second derivative discontinuities when the original optimization problem has inequality constraints $g(x) \leq 0$, independently of the smoothness of $g(x)$.

8.1 ▪ General algorithm

We will define a general algorithm for unconstrained minimization based on line searches. Many effective algorithms for unconstrained minimization have the general form of the algorithm described here. For simplicity (with some abuse of notation) we denote by $\{x^k\}$ the sequence of iterates generated by this and other subproblems' solvers. They must not be confused with the iterates $\{x^k\}$ of the main Augmented Lagrangian algorithm.

Algorithm 8.1.
Let $\theta \in (0, 1)$, $\alpha \in (0, 1/2)$, $\bar{M} \geq 2$, and $\beta > 0$ be algorithmic parameters. Let $x^0 \in \mathbb{R}^n$ be the initial approximation. Given $x^k \in \mathbb{R}^n$, the steps for computing x^{k+1} are the following:

Step 1. If $\|\nabla F(x^k)\| = 0$, finish the execution of the algorithm.

Step 2. Compute $d^k \in \mathbb{R}^n$ such that

$$\nabla F(x^k)^T d^k \leq -\theta \|d^k\|_2 \|\nabla F(x^k)\|_2 \text{ and } \|d^k\| \geq \beta \|\nabla F(x^k)\|. \tag{8.2}$$

Step 3. Compute $t_k > 0$ and $x^{k+1} \in \mathbb{R}^n$ such that

$$F(x^{k+1}) \leq F(x^k + t_k d^k),$$
$$F(x^k + t_k d^k) \leq F(x^k) + \alpha t_k \nabla F(x^k)^T d^k, \tag{8.3}$$

and

$$\left[t_k \geq 1 \right] \text{ or } \left[F(x^k + \bar{t}_k d^k) > F(x^k) + \alpha \bar{t}_k \nabla F(x^k)^T d^k \text{ for some } \bar{t}_k \in [t_k, \bar{M} t_k] \right].$$

Let us explain the reasons that support each step of this algorithm:

1. At Step 1, we establish that, if the gradient at x^k is null, it is not worthwhile to continue the execution of the algorithm. Thus, we accept points where the gradient vanishes and we do not intend to go further in this case. This is not because we are happy with stationary points, but because we do not know how to proceed from that kind of point without using potentially expensive second-order information.

2. If the gradient does not vanish at x^k, we seek a *search direction* d^k for which two conditions are required. The first is that d^k should be a *first-order descent direction*. This means that the directional derivative $\nabla F(x^k)^T d^k$ should be negative. More precisely, the angle between the direction and $-\nabla F(x^k)$ should be smaller than or equal to a fixed angle smaller than $\pi/2$, whose cosine is defined by the algorithmic parameter θ. If $\theta = 1$, one forces the direction to be a multiple of the negative gradient. In general, we are far less exacting, and $\theta = 10^{-6}$ is a traditionally recommended tolerance. The second condition is that the size of d^k should be at least a fixed multiple of $\|\nabla F(x^k)\|$. The constant of proportionality is called β. The reason for this requirement is that we want to accept small directions only if the gradient is small.

3. At Step 3, we require that the final point of our line search, $x^k + t_k d^k$, satisfy the *Armijo condition* (8.3). If we define $\varphi(t) = F(x^k + t d^k)$, the Armijo condition is equivalent to
$$\varphi(t_k) \leq \varphi(0) + \alpha t_k \varphi'(0).$$
In other words, with this condition, we require that $\varphi(t_k)$ stay below the line that passes through $(0, \varphi(0))$ whose slope is $\alpha\varphi'(0)$. In this sense, $F(x^k + t_k d^k)$ should be sufficiently smaller than $F(x^k)$. For this reason, (8.3) is frequently known as a *sufficient descent* condition.

   However, satisfying (8.3) is not enough. We need to guarantee that we are not taking artificially small steps. (A step should be small only if it cannot be much larger.) Consequently, we impose that either $t_k \geq 1$ (a constant different from 1 having the same effect) or a frustrated step $\bar{t}_k$ exists, not much bigger than t_k, for which the Armijo condition did not hold.

4. Finally, we leave the door open to take a point x^{k+1} even better than $x^k + t_k d^k$. For this reason, we impose for x^{k+1} the only requirement that its functional value should not be greater than $F(x^k + t_k d^k)$. Obviously, it is admissible to choose $x^{k+1} = x^k + t_k d^k$.

Step 3 of Algorithm 8.1 may be implemented in many different ways. The most elementary strategy consists of choosing t_k as the first element of the sequence $\{1, 1/2, 1/4, \ldots\}$ satisfying the Armijo condition. It is easy to see that, in this way, we obtain a step as required in the algorithm with $\bar{M} = 2$. The direction d^k also admits many different definitions. The most obvious one is to choose $d^k = -\nabla F(x^k)$ (which satisfies (8.2) for any choice of $\theta \in (0, 1)$ and $0 < \beta \leq 1$). With these choices, we obtain a version of the *steepest descent* method, one of the most popular procedures for unconstrained minimization.

Algorithm 8.1 requires four algorithmic parameters: θ, α, $\bar{M}$, and β. The first three are dimensionless, that is, their values do not depend on the unities in which the problem magnitudes are measured. For example, θ is the cosine of an angle and α is a pure fraction whose traditional value is 10^{-4}. It makes sense to specify recommended values for dimensionless parameters, since the effect of them should not be affected by the scaling of the problem.

The value of $\bar{M}$ is related to the strategy used in the line search to backtrack, when sufficient descent is not verified for some trial step $\bar{t}$. When this happens, we wish to choose a new trial t in the interval $(0, \bar{t})$. However, t should not be excessively close to 0, because, in that case, the evaluation of F at $x^k + td^k$ would add too little information to the knowledge of F. Therefore, line-search methods usually employ safeguards that impose $t \geq (1/\bar{M})\bar{t}$ with $1/\bar{M} \in (0,1)$. Consequently, the third condition of Step 3 is satisfied.

Usual algorithms for computing t_k at Step 3 of Algorithm 8.1 obey the following steps:

Step LS1. Start setting $t_{\text{trial}} \leftarrow 1$.

Step LS2. If t_{trial} satisfies the Armijo condition, find $t_{\text{new}} \in (t_{\text{trial}}, \bar{M}t_{\text{trial}}]$. If t_{new} also satisfies the Armijo condition, set $t_{\text{trial}} \leftarrow t_{\text{new}}$ and repeat Step LS2. Otherwise, define $t_k = t_{\text{trial}}$ and finish the line search.

Step LS3. Find $t_{\text{new}} \in [t_{\text{trial}}/\bar{M}, t_{\text{trial}}/2]$ and set $t_{\text{trial}} \leftarrow t_{\text{new}}$. If t_{trial} satisfies the Armijo condition, define $t_k = t_{\text{trial}}$ and finish the line search. Otherwise, repeat Step LS3.

The value of t_{new} at Step LS3 may be computed as the safeguarded minimizer of a univariate quadratic or cubic interpolating function. At Step LS2, the computation of t_{new} may contemplate extrapolating techniques.

The choice of the parameter β is more tricky because, unlike the others, this parameter is a dimensional parameter and compares magnitudes of different types. For example, suppose that, instead of minimizing the function $F(x)$, we need to minimize the function $\bar{F}(x) = 10F(x)$. Both problems are completely equivalent, and therefore we would like to observe the same behavior of the algorithm in both cases. However, the conditions $\|d^k\| \geq \beta\|\nabla F(x^k)\|$ and $\|d^k\| \geq \beta\|\nabla \bar{F}(x^k)\|$ are not equivalent. In other words, we should use different values of β in those problems. A more careful analysis would reveal that β should be proportional to the norm of the inverse of the Hessian at x^k. Fortunately, the usual procedures used to compute d^k frequently satisfy the condition $\|d^k\| \geq \beta\|\nabla F(x^k)\|$ automatically for an unknown value of β. For this reason, we generally assign a small value to this parameter, trying to accept d^k as frequently as possible.

8.2 ▪ Magic steps and nonmonotone strategies

At Step 3 of Algorithm 8.1, we require that

$$F(x^k + t_k d^k) \leq F(x^k) + \alpha t_k \nabla F(x^k)^T d^k \tag{8.4}$$

and

$$F(x^{k+1}) \leq F(x^k + t_k d^k). \tag{8.5}$$

Clearly, the choice $x^{k+1} = x^k + t_k d^k$ already satisfies (8.5), but several reasons motivate one to try something better. On the one hand, after computing $x^k + t_k d^k$, it could be interesting to test whether some extrapolation of type $x^k + td^k$ with $t > t_k$ could cause

some improvement. Moreover, improvements could come from persevering not only along the computed direction but also along other directions motivated by the specific problem we are trying to solve. Heuristic choices of x^{k+1} satisfying (8.5) are generally called "magic steps" (Conn, Gould, and Toint [83]) and could be crucial for the good practical behavior of the algorithm.

Magic steps may be computed by evoking nonmonotone strategies, watchdog techniques [79], and the spacer step theorem of [182, p. 255]. Sometimes, a sequence of iterates is deemed to converge to the solution of a problem (even very fast!), although without satisfying a monotone decrease of the objective function. This is typical of Newton-like methods for solving subproblems with extreme penalty parameters (and it is related to the phenomenon called the Maratos effect in the optimization literature). In these cases, it is sensible to tolerate some increase in the objective function during some iterations, before returning to rigorous line searches. These ideas may be formalized in the following algorithm.

Algorithm 8.2. Nonmonotone pseudomagic procedure
Given integers $M \geq 0$ and $L \geq 0$ (both around 10), in Algorithm 8.1, after the computation of x^k and before the computation of d^k, set $y^0 = x^k$, $j \leftarrow 0$,

$$F_{\max} = \max\{F(x^k), \ldots, F(x^{\max\{k-M,0\}})\},$$

and execute Steps 1–6 below.

Step 1. If $j < L$, compute y^{j+1} and test whether

$$F(y^{j+1}) \leq F_{\max}. \tag{8.6}$$

If y^{j+1} was computed and (8.6) was satisfied, set $j \leftarrow j+1$ and repeat Step 1.

Step 2. Let $y \in \{y^0, \ldots, y^j\}$ be such that

$$F(y) = \min\{F(y^0), \ldots, F(y^j)\}. \tag{8.7}$$

If $F(y) \geq F(x^k)$, discard y, finish the execution of the present algorithm, and return to the computation of d^k, satisfying (8.2), at Step 2 of Algorithm 8.1.

Step 4. Define $d^k = y - x^k$.

Step 5. If d^k satisfies (8.2) and, in addition,

$$F(y) \leq F(x^k) + \alpha \nabla F(x^k)^T d^k, \tag{8.8}$$

define $x^{k+1} = x^k + d^k = y$ and consider that the kth iteration of Algorithm 8.1 is finished.

Step 6. If d^k does not satisfy (8.2) or does not fulfill (8.8), replace x^k with y and proceed to the computation of d^k at Step 2 of Algorithm 8.1.

Observe that with the inclusion of the nonmonotone magic procedure, the unconstrained Algorithm 8.1 preserves its basic form. The procedure may be considered as an auxiliary device for computing the search direction. If our heuristic for choosing the magic points y^j is good, it could be unnecessary to perform line searches. Note that, after the execution of Algorithm 8.2, we may obtain the next iterate x^{k+1} or we may improve (redefining it) the iterate x^k. In the second case, we should formally "forget" the existence of the previously computed x^k, and we should consider that the new x^k is a result of further improvement in the sense of (8.5) at iteration $k-1$. In the first case, the magic procedure computed, perhaps by chance, a direction d^k that satisfies the basic requirements of Algorithm 8.1. Apparently, there is not a big difference between the possibilities, since in both cases we come up with a better point y. The difference is that,

when (8.2) and (8.8) hold, we avoid the necessity of computing a new direction d^k and we open the possibility that all the iterates could be computed as magic steps. In other words, in both cases the magic step was successful but the status given to the step at Step 5 is more relevant than the one given at Step 6. In any case, Algorithm 8.2 specifies aspects of the implementation of Algorithm 8.1 but does not alter its basic structure.

8.3 ▪ Well-definiteness and global convergence

We are going to show first that Algorithm 8.1 is well defined. This means that if the algorithm does not stop at x^k, it is possible to obtain x^{k+1} in finite time. This is the main theoretical and practical condition that an implementable algorithm must satisfy. Well-defined algorithms that do not satisfy additional global convergence properties may be effective in practice [90, 91, 194, 195] but the well-definiteness property is mandatory.

Theorem 8.1. *Algorithm 8.1 is well defined and stops at x^k if and only if $\nabla F(x^k) = 0$.*

Proof. Assume that $\nabla F(x^k) \neq 0$. By the conditions imposed at Step 2 of the algorithm and the differentiability of F,

$$\lim_{t \to 0} \frac{F(x^k + t d^k) - F(x^k)}{t} = \nabla F(x^k)^T d^k < 0.$$

Thus,

$$\lim_{t \to 0} \frac{F(x^k + t d^k) - F(x^k)}{t \nabla F(x^k)^T d^k} = 1.$$

Since $\alpha < 1$, for t small enough, we have that

$$\frac{F(x^k + t d^k) - F(x^k)}{t \nabla F(x^k)^T d^k} \geq \alpha.$$

Now, as $\nabla F(x^k)^T d^k < 0$, we deduce that

$$F(x^k + t d^k) \leq F(x^k) + \alpha t \nabla F(x^k)^T d^k$$

for $t > 0$ small enough. Thus, choosing t_k as the first element of the sequence $\{\bar{M}^{-\ell}\}_{\ell \in \mathbb{N}_0}$ satisfying the condition above, we have that the requirements of Step 3 of Algorithm 8.1 are fulfilled. □

The following theorem is said to be a global convergence theorem. It establishes that, *independently of the initial point*, the gradient must vanish at every limit point. Global convergence in this sense should not be confused with "convergence to global minimizers." Note that the existence of limit points is assumed, and not guaranteed, at this theorem. A sufficient condition for the existence of the limit points is the boundedness of the generated sequence, which, in turn, holds whenever the level set defined by $F(x^0)$ is bounded.

Theorem 8.2. *If x^* is a limit point of a sequence generated by Algorithm 8.1, we have that $\nabla F(x^*) = 0$.*

Proof. Let $K = \{k_0, k_1, k_2, k_3, \ldots\} \underset{\infty}{\subset} \mathbb{N}$ be such that

$$\lim_{k \in K} x^k = x^*.$$

By the continuity of F,

$$\lim_{k\in K} F(x^k) = F(x^*).$$

By the Armijo condition, since $k_{j+1} \geq k_j + 1$, we have that

$$F(x^{k_{j+1}}) \leq F(x^{k_j+1}) \leq F(x^{k_j}) + \alpha t_{k_j} \nabla F(x^{k_j})^T d^{k_j} < F(x^{k_j})$$

for all $j \in \mathbb{N}$. Then,

$$\lim_{j\to\infty} t_{k_j} \nabla F(x^{k_j})^T d^{k_j} = 0.$$

Therefore, by (8.2),

$$\lim_{j\to\infty} t_{k_j} ||\nabla F(x^{k_j})||_2 ||d^{k_j}||_2 = 0$$

and, by the equivalence of norms in $\mathbb{R}^n$,

$$\lim_{j\to\infty} t_{k_j} ||\nabla F(x^{k_j})|| ||d^{k_j}|| = 0. \tag{8.9}$$

Thus, there exists $K_1 \underset{\infty}{\subset} K$ such that at least one of the two following possibilities is fulfilled:

(a) $$\lim_{k\in K_1} ||\nabla F(x^k)|| = 0,$$

(b)
$$\lim_{k\in K_1} t_k ||d^k|| = 0. \tag{8.10}$$

In case (a), we deduce that $\nabla F(x^*) = 0$ and the thesis is proved.

In case (b), there exists $K_2 \underset{\infty}{\subset} K_1$ such that at least one of the two following possibilities holds:

(c) $$\lim_{k\in K_2} ||d_k|| = 0,$$

(d) $$\lim_{k\in K_2} t_k = 0.$$

In case (c), the conditions (8.2) also imply that $||\nabla F(x^k)|| \to 0$ for $k \in K_2$ and, therefore, $\nabla F(x^*) = 0$.

Let us consider case (d). Without loss of generality, assume that $t_k < 1$ for all $k \in K_2$. Therefore, by Step 3 of the algorithm, for all $k \in K_2$ there exists $\bar{t}_k > 0$ such that

$$F(x^k + \bar{t}^k d^k) > F(x^k) + \alpha \bar{t}_k \nabla F(x^k)^T d^k. \tag{8.11}$$

Moreover, since $\bar{t}_k \leq \bar{M} t_k$, by (8.10), we have that

$$\lim_{k\in K_2} \bar{t}_k ||d_k|| = 0.$$

Thus, defining $s^k = \bar{t}_k d^k$ for all $k \in K_2$, we have that

$$\lim_{k\in K_2} ||s_k|| = 0. \tag{8.12}$$

By (8.11) and the mean value theorem, for all $k \in K_2$ there exists $\xi_k \in [0,1]$ such that

$$\nabla F(x^k + \xi_k s^k)^T s^k = F(x^k + s^k) - F(x^k) > \alpha \nabla F(x^k)^T s^k. \tag{8.13}$$

Let $K_3 \subset_{\infty} K_2$ and $s \in \mathbb{R}^n$ be such that $\lim_{k \in K_3} s^k / \|s^k\| = s$. By (8.12), dividing both sides of the inequality (8.13) by $\|s^k\|$ and taking limits for $k \in K_3$, we obtain

$$\nabla F(x^*)^T s \geq \alpha \nabla F(x^*)^T s.$$

Since $\alpha < 1$ and $\nabla F(x^k)^T s^k < 0$ for all k, this implies that $\nabla F(x^*)^T s = 0$. Since by (8.2),

$$-\frac{\nabla F(x^k)^T s^k}{\|s^k\|_2} \geq \theta \|\nabla F(x^k)\|_2 \tag{8.14}$$

for all $k \in K_2$, taking limits in (8.14) for $k \in K_3$, we obtain that $\nabla F(x^*) = 0$. □

The alert reader should observe that the proof of Theorem 8.2 has two essential arguments. The "argument of success" applies to the case in which the step is bounded away from zero. In this case, the size of the direction should go to zero; otherwise the functional value would tend to minus-infinity. The "argument of failure" applies to the case in which the step tends to zero. In this case, there is a different step that also tends to zero along which the function increased or did not decrease enough. This is possible only if the gradient in the limit is zero. These two arguments appear persistently in every global convergence theorem for continuous optimization algorithms. The reader is invited to find them in other line-search procedures, in trust-region methods, in nonsmooth optimization [69], and in derivative-free minimization algorithms [84].

8.4 ▪ Local convergence

Global convergence in the sense used in the former section is a very welcome property for practical unconstrained minimization algorithms, because it guarantees that convergence to points where the gradient does not vanish cannot occur. Nevertheless, the efficiency of algorithms is also linked to theoretical properties of *local convergence*. These properties say that, when the sequence generated by the algorithm passes close to a minimizer, such proximity is recognized and the sequence converges quickly to the solution. To obtain that property, the distance between x^{k+1} and x^k needs to be small when the gradient $\nabla F(x^k)$ is small. We will formalize this requirement in the following assumption.

Assumption 8.1. *There exists $b > 0$ such that consecutive iterates x^k and x^{k+1} in Algorithm 8.1 satisfy*

$$\|x^{k+1} - x^k\| \leq b \|\nabla F(x^k)\| \tag{8.15}$$

for all $k \in \mathbb{N}$.

This assumption is compatible with line searches that obey the requirements of Step 3 of Algorithm 8.1.

Our strategy to prove local superlinear convergence has three parts. In Theorem 8.3 below, we prove that, if x^* is an isolated limit point, the whole sequence converges to x^*. In Theorem 8.4 below, we prove that, if the initial point is close to a strict local minimizer x^*, the whole sequence converges to x^*. Although analogous, these two theorems are independent (none of them is deduced from the other). However, both show that the convergence of the whole sequence to a single point may be expected in many cases. In other words, convergence of the whole sequence is a *plausible* assumption. Using it, and assuming further that the directions d^k are obtained using a Newtonian philosophy, we will prove superlinear convergence.

A point x^* is said to be *isolated* if there exists $\epsilon > 0$ such that $\nabla F(x) \neq 0$ for all $x \in B(x^*, \epsilon)$ such that $x \neq x^*$.

Theorem 8.3. *Assume that x^* is isolated, the sequence $\{x^k\}$ is generated by Algorithm 8.1 with Assumption 8.1, and $\lim_{k\in K} x^k = x^*$ for some subsequence $K \underset{\infty}{\subset} \mathbb{N}$. Then, $\nabla F(x^*) = 0$ and $\lim_{k\to\infty} x^k = x^*$.*

Proof. The fact that $\nabla F(x^*) = 0$ is a consequence of Theorem 8.2. Since x^* is isolated, there exists $\epsilon > 0$ such that $\nabla F(x) \neq 0$ for all $x \in B(x^*, \epsilon) \setminus \{x^*\}$.

Let us define

$$C = \{x \in \mathbb{R}^n \mid \epsilon/2 \leq ||x - x^*|| \leq \epsilon\}.$$

Clearly, C is compact and does not contain points where the gradient vanishes. Therefore, by Theorem 8.2, C does not contain infinitely many iterates. Let $k_1 \in \mathbb{N}$ be such that $x^k \notin C$ for all $k \geq k_1$.

Define

$$K_1 = \{k \geq k_1 \mid ||x^k - x^*|| \leq \epsilon/2\} \subseteq \mathbb{N}.$$

Note that K_1 is nonempty since, by hypothesis, $\lim_{k\in K} x^k = x^*$.

Since x^* is the unique stationary point in the ball with radius $\epsilon/2$, by Theorem 8.2, we have that $\lim_{k\in K_1} x^k = x^*$ and, by the continuity of ∇F, that $\lim_{k\in K_1} ||\nabla F(x^k)|| = 0$. Then, by Assumption 8.1, $\lim_{k\in K_1} ||x^{k+1} - x^k|| = 0$. This implies that, for all $k \in K_1$ large enough, $||x^{k+1} - x^k|| < \epsilon/2$. Then, since, by the definition of K_1, we have $||x^k - x^*|| \leq \epsilon/2$, the triangle inequality implies that $||x^{k+1} - x^*|| \leq \epsilon$. However, since $x^{k+1} \notin C$, we have that $||x^{k+1} - x^*|| \leq \epsilon/2$. Therefore, we proved that, for all $k \in K_1$ large enough, we have that $k + 1$ also belongs to K_1. This implies that $||x^k - x^*|| \leq \epsilon/2$ for all k large enough. Invoking again Theorem 8.2 and the isolation of x^*, it follows that

$$\lim_{k\to\infty} x^k = x^*,$$

as we wanted to prove. □

Theorem 8.4. *Assume that the isolated point x^* is a strict local minimizer and the sequence $\{x^k\}$ is generated by Algorithm 8.1 with Assumption 8.1. Then, there exists $\delta_1 > 0$ such that, if $||x^{k_0} - x^*|| \leq \delta_1$ for some k_0, we have that $\lim_{k\to\infty} x^k = x^*$.*

Proof. Let $\epsilon > 0$ be such that x^* is a strict global minimizer of f on the ball $B(x^*, \epsilon)$ and assume that this ball does not contain any other point in which the gradient vanishes. By the continuity of ∇F and Assumption 8.1, there exists $\delta \in (0, \epsilon/2)$ such that

$$||x^k - x^*|| \leq \delta \Rightarrow ||x^{k+1} - x^k|| \leq \epsilon/2. \tag{8.16}$$

Let c be the minimum value of $F(x)$ on the set $\{x \in \mathbb{R}^n \mid \delta \leq ||x - x^*|| \leq \epsilon\}$. Let $\delta_1 \in (0, \delta)$ be such that $||x - x^*|| \leq \delta_1 \Rightarrow F(x) < c$.

Let us prove by induction that, if there exists k_0 such that $||x^{k_0} - x^*|| \leq \delta_1$, one obtains $||x^k - x^*|| \leq \delta$ and $F(x^k) < c$ for all $k \geq k_0$. By the definition of δ_1, this is trivially true if $k = k_0$. Now, let us assume it is valid for k and prove it for $k + 1$. Observe that, by (8.16), $||x^k - x^*|| \leq \delta$ implies that $||x^{k+1} - x^*|| \leq \epsilon$. Moreover, $F(x^k) < c$ and the fact that $F(x^{k+1}) < F(x^k)$ imply $F(x^{k+1}) < c$, and this implies that $||x^{k+1} - x^*|| < \delta$.

Therefore, since $\delta < \epsilon/2$ for k large enough, all the elements of the sequence are contained in $B(x^*, \epsilon/2)$. Since x^* is the unique point where the gradient vanishes in this ball, Theorem 8.3 implies that the whole sequence converges to x^* as we wanted to prove. □

8.4.1 ▪ Convergence under Newton-like choices of the search directions

The next algorithm is a particular case of Algorithm 8.1 and defines a general (Newton-like) form in which the direction d^k may be computed. This direction will be the approximate solution of a linear system of the form $B_k d = -\nabla F(x^k)$. The idea is that the gradient $\nabla F(x)$ is well approximated by the linear function $B_k(x-x^k)+\nabla F(x^k)$ in a neighborhood of x^k or, equivalently, that the objective function $F(x)$ is well approximated by the quadratic $(1/2)(x-x^k)^T B_k(x-x^k)+\nabla F(x^k)^T(x-x^k)+F(x^k)$ in such a way that a solution of $B_k(x-x^k)+\nabla F(x^k)=0$ could be a good approximation to the solution of the problem. In fact, this corresponds to the "Newtonian paradigm" for solving many nonlinear mathematical problems: Approximate the original problem by an easy (in some sense, linear) problem using information at x^k and use the solution of the "subproblem" to continue the process. The Newtonian linear system may be solved exactly (discarding rounding errors) and using B_k as the true Hessian at x^k (classical Newton). If the subproblem is solved approximately employing an iterative linear solver but still using the true Hessian, we say we are using the inexact Newton or truncated Newton approach. When B_k is only an approximation of the Hessian at x^k, we talk about quasi-Newton methods (inexact quasi-Newton in the case that the quasi-Newton system is solved only approximately).

In this context, step $t_k = 1$ should be preferred in some sense because it thoroughly corresponds to the Newtonian paradigm. When this step satisfies the Armijo condition, we decide to stop the line search. Global convergence will require, on the other hand, that the matrices B_k be positive definite and that their inverses be bounded.

Algorithm 8.3. This algorithm corresponds to an implementation of Algorithm 8.1 in which the following hold:

(a) The direction d^k is such that

$$||B_k d^k + \nabla F(x^k)|| \le \eta_k ||\nabla F(x^k)||, \tag{8.17}$$

where $B_k \in \mathbb{R}^{n\times n}$ is symmetric and positive definite and $\eta_k \in [0,1)$.

(b) If $F(x^k+d^k) \le F(x^k)+\alpha\nabla F(x^k)^T d^k$, we choose $t_k = 1$ and $x^{k+1} = x^k + d^k$.

In order to say that Algorithm 8.3 is an implementation of Algorithm 8.1, we must show that the direction d^k computed in **(a)** satisfies (8.2). A sufficient condition for the fulfillment of (8.2) is the boundedness of $||B_k||$ and $||B_k^{-1}||$, stated in Assumption 8.2. This claim is proved in Theorem 8.5 below.

Assumption 8.2. *The sets* $\{||B_k||, k\in\mathbb{N}\}$ *and* $\{||B_k^{-1}||, k\in\mathbb{N}\}$ *are bounded.*

Theorem 8.5. *Assume that the sequence* $\{x^k\}$ *is generated by Algorithm 8.3 and that Assumption 8.2 holds. Then, there exist* $\eta_{\max}\in(0,1)$, $\theta\in(0,1)$, *and* $\beta > 0$ *such that, for all* $k\in\mathbb{N}$, *if* $\eta_k \le \eta_{\max}$, *then (8.2) is satisfied.*

Proof. Define $r^k = B_k d^k + \nabla F(x^k)$. Then,

$$\begin{aligned}||\nabla F(x^k)|| &= ||\nabla F(x^k) - r^k + r^k|| \le ||\nabla F(x^k) - r^k|| + ||r^k|| \\ &= ||B^k d^k|| + ||r^k|| \le ||B^k||||d^k|| + ||r^k||.\end{aligned}$$

Therefore,

$$||\nabla F(x^k)|| - ||r^k|| \le ||B^k||||d^k||.$$

Thus, since by (8.17) $||r^k|| \leq \eta_k ||\nabla F(x^k)||$, we deduce that

$$(1-\eta_k)||\nabla F(x^k)|| \leq ||B_k||||d^k||.$$

Consequently,

$$||d^k|| \geq \frac{1-\eta_k}{||B_k||}||\nabla F(x^k)||.$$

Taking $\eta_k \leq 1/2$, we have that

$$||d^k|| \geq \frac{1}{2||B_k||}||\nabla F(x^k)||.$$

Assuming that $||B_k|| \leq c$ for all $k \in \mathbb{N}$, the second requirement of (8.2) is satisfied with $\beta = 1/(2c)$.

Let us prove the fulfillment of the angle condition in (8.2). Since $-\nabla F(x^k) = B_k d^k - r^k$, premultiplying by $(d^k)^T$ yields

$$-(d^k)^T \nabla F(x^k) = (d^k)^T B_k d^k - (d^k)^T r^k.$$

By the spectral decomposition [127] of B_k, we have that

$$\lambda_{\min}(B_k) \leq \frac{(d^k)^T B_k d^k}{||d^k||_2^2} \leq \lambda_{\max}(B_k),$$

where $\lambda_{\min}(B_k)$ and $\lambda_{\max}(B_k)$ represent the smallest and the largest eigenvalues of B_k. Since $||B_k^{-1}||_2 = \lambda_{\max}(B_k^{-1}) = 1/\lambda_{\min}(B_k)$, we have that

$$(d^k)^T B_k d^k \geq \frac{||d^k||_2^2}{||B_k^{-1}||_2}.$$

Therefore,

$$-(d^k)^T \nabla F(x^k) \geq \frac{||d^k||_2^2}{||B_k^{-1}||_2} - (d^k)^T r^k.$$

Thus, by the first part of the proof and the equivalence of norms in $\mathbb{R}^n$, there exists $\beta_2 > 0$ such that

$$-(d^k)^T \nabla F(x^k) \geq \beta_2 \frac{||d^k||_2||\nabla F(x^k)||_2}{||B_k^{-1}||_2} - (d^k)^T r^k$$

for all k such that $\eta_k \leq 1/2$. Assuming that $||B_k^{-1}||_2 \leq c_2$ for all $k \in \mathbb{N}$, this implies that

$$(d^k)^T \nabla F(x^k) \leq -\beta_2 \frac{||d^k||_2||\nabla F(x^k)||_2}{c_2} + (d^k)^T r^k.$$

Now, since $||r^k|| \leq \eta_k ||\nabla F(x^k)||$, by the equivalence of norms in $\mathbb{R}^n$, there exists a constant $c_{\text{norm}} > 0$ such that

$$||r^k||_2 \leq c_{\text{norm}} \eta_k ||\nabla F(x^k)||_2.$$

Hence,

$$(d^k)^T r^k \leq ||d^k||_2||r^k||_2 \leq c_{\text{norm}} \eta_k ||d^k||_2||\nabla F(x^k)||_2$$

and, therefore,

$$\begin{aligned}(d^k)^T\nabla F(x^k) &\le -(\beta_2/c_2)\|d^k\|_2\|\nabla F(x^k)\|_2 + c_{\text{norm}}\eta_k\|d^k\|_2\|\nabla F(x^k)\|_2\\ &= -(\beta_2/c_2 - c_{\text{norm}}\eta_k)\|d^k\|_2\|\nabla F(x^k)\|_2.\end{aligned}$$

Thus if, say, $\eta_k \le \min\{\frac{1}{2}, \frac{1}{2}(\beta_2/(c_2c_{\text{norm}}))\}$, we have that $\bar\theta = \beta_2/c_2 - c_{\text{norm}}\eta_k > 0$ and that

$$(d^k)^T\nabla F(x^k) \le -\bar\theta\|d^k\|_2\|\nabla F(x^k)\|_2.$$

This means that the angle condition of (8.2) is satisfied with $\theta = \bar\theta$. □

The following theorem completes the basic convergence theory of Algorithm 8.1. We will show that under Assumption 8.2, if the sequence generated by Algorithm 8.3 converges to a local minimizer x^* where the Hessian $\nabla^2 F(x^*)$ is positive definite and the matrices B_k are approximations of the Hessians $\nabla^2 F(x^k)$ in the sense of Dennis and Moré [97], the convergence is superlinear and, for k large enough, we have that $t_k = 1$. In other words, for k large enough, we will need only one function evaluation per iteration.

Theorem 8.6. *Assume that the sequence $\{x^k\}$ is generated by Algorithm 8.3 with Assumption 8.2 and $x^k \ne x^*$ for all $k \in \mathbb{N}$, $\lim_{k\to\infty} x^k = x^*$, F admits continuous third derivatives in a neighborhood of x^*, $\nabla^2 F(x^*)$ is positive definite, and the Dennis–Moré condition*

$$\lim_{k\to\infty} \frac{\left\|\left[B_k - \nabla^2 F(x^k)\right]d^k\right\|}{\|d^k\|} = 0$$

and the inexact Newton condition

$$\lim_{k\to\infty} \eta_k = 0$$

are verified. Then, there exists $k_0 \in \mathbb{N}$ such that $t_k = 1$ for all $k \ge k_0$ and the sequence $\{x^k\}$ converges superlinearly to x^*. Moreover, if $B_k = \nabla^2 F(x^k)$ and $\eta_k = 0$ for all $k \in \mathbb{N}$, the convergence is quadratic.

Proof. By Taylor's formula, we have that

$$\begin{aligned}&F(x^k + d^k) - F(x^k) - \alpha\nabla F(x^k)^T d^k\\ &\quad= (1-\alpha)\nabla F(x^k)^T d^k + \tfrac{1}{2}(d^k)^T\nabla^2 F(x^k)d^k + o(\|d^k\|^2)\\ &\quad= (1-\alpha)(d^k)^T\left[\nabla F(x^k) + \nabla^2 F(x^k)d^k\right] + \left(\alpha - \tfrac{1}{2}\right)(d^k)^T\nabla^2 F(x^k)d^k + o(\|d^k\|^2).\end{aligned}$$

Defining $r^k = B_k d^k + \nabla F(x^k)$, by Step 2 of the algorithm and $\eta_k \to 0$, we have that $\|r^k\| = o(\|\nabla F(x^k)\|) = o(\|d^k\|)$. Therefore,

$$\begin{aligned}&F(x^k + d^k) - F(x^k) - \alpha\nabla F(x^k)^T d^k\\ &\quad= (1-\alpha)(d^k)^T r^k + (1-\alpha)(d^k)^T\left[\nabla^2 F(x^k) - B_k\right]d^k + \left(\alpha - \tfrac{1}{2}\right)(d^k)^T\nabla^2 F(x^k)d^k + o(\|d^k\|^2)\\ &\quad= (1-\alpha)(d^k)^T\left[\nabla^2 F(x^k) - B_k\right]d^k + \left(\alpha - \tfrac{1}{2}\right)(d^k)^T\nabla^2 F(x^k)d^k + o(\|d^k\|^2).\end{aligned}$$

Now, by the Dennis–Moré condition, we have that

$$(1-\alpha)(d^k)^T\left[\nabla^2 F(x^k) - B_k\right]d^k = o(\|d^k\|^2),$$

and, therefore,

$$F(x^k + d^k) - F(x^k) - \alpha(d^k)^T\nabla F(x^k) = \left(\alpha - \frac{1}{2}\right)(d^k)^T\nabla^2 F(x^k)d^k + o(\|d^k\|^2). \quad (8.18)$$

Let $\mu > 0$ be a lower bound for the eigenvalues of $\nabla^2 F(x^*)$. Then, there exists k_1 such that $\mu/2$ is a lower bound for the eigenvalues of $\nabla^2 F(x^k)$ for all $k \geq k_1$. Thus, for all $k \geq k_1$, we have that

$$\frac{(d^k)^T \nabla^2 F(x^k) d^k}{||d^k||^2} \geq \mu/2.$$

Since $\alpha < 1/2$, by (8.18), we have that

$$\frac{F(x^k + d^k) - F(x^k) - \alpha (d^k)^T \nabla F(x^k)}{||d^k||^2} \leq \left(\alpha - \frac{1}{2}\right)\frac{\mu}{2} + \frac{o(||d^k||^2)}{||d^k||^2} \tag{8.19}$$

for $k \geq k_1$. But, since $\{||B_k^{-1}||, k \in \mathbb{N}\}$ is bounded, $\nabla F(x^k) \to 0$ (by Theorem 8.2), and $\eta_k \to 0$, $||r_k|| \leq \eta_k ||F(x^k)||$ implies that $||d^k|| \to 0$. Therefore, taking limits in (8.19) for $k \to \infty$, we obtain

$$F(x^k + d^k) - F(x^k) - \alpha \nabla F(x^k)^T d^k \leq 0$$

for k large enough. Then, by the definition of the algorithm, there exists $k_0 \in \mathbb{N}$ such that $t_k = 1$ for all $k \geq k_0$. Thus, the first part of the thesis is proved.

By the first part of the thesis and the definition of Algorithm 8.3, we have that

$$x^{k+1} - x^k = d^k \text{ for all } k \geq k_0. \tag{8.20}$$

Then, by Taylor's formula,

$$\begin{aligned} \nabla F(x^{k+1}) &= \nabla F(x^k) + \nabla^2 F(x^k) d^k + O(||d^k||^2) \\ &= B_k d^k + \nabla F(x^k) + [\nabla^2 F(x^k) - B_k] d^k + O(||d^k||^2) \\ &= r^k + [\nabla^2 F(x^k) - B_k] d^k + O(||d^k||^2). \end{aligned} \tag{8.21}$$

As in the first part of the proof we have that $||r^k|| = o(||d^k||)$. Therefore,

$$\nabla F(x^{k+1}) = [\nabla^2 F(x^k) - B_k] d^k + o(||d^k||).$$

Then, by the Dennis–Moré condition and (8.20),

$$\lim_{k \to \infty} \frac{||\nabla F(x^{k+1})||}{||x^{k+1} - x^k||} = 0.$$

Since, by the mean value theorem of integral calculus, we have that

$$\nabla F(x^{k+1}) - \nabla F(x^*) = \left[\int_0^1 \nabla^2 F(x^* + t(x^{k+1} - x^*) dt\right](x^{k+1} - x^*),$$

then, by the continuity and nonsingularity of the Hessian at x^*, we deduce that

$$\lim_{k \to \infty} \frac{||x^{k+1} - x^*||}{||x^{k+1} - x^k||} = 0.$$

Therefore,

$$\lim_{k \to \infty} \frac{||x^{k+1} - x^*||}{||x^{k+1} - x^*|| + ||x^k - x^*||} = 0.$$

Thus,

$$\lim_{k\to\infty} 1+\frac{||x^k-x^*||}{||x^{k+1}-x^*||}=\infty$$

and, consequently,

$$\lim_{k\to\infty} \frac{||x^{k+1}-x^*||}{||x^k-x^*||}=0. \tag{8.22}$$

Then, the convergence is superlinear, as we wanted to prove.

Finally, let us prove that the convergence is quadratic when $B_k=\nabla^2 F(x^k)$ and $\eta_k=0$ for all $k\in\mathbb{N}$. In this case, by (8.21) we have that

$$||\nabla F(x^{k+1})||=O(||d^k||^2).$$

So, since $t_k=1$ for k large enough, there exists $c>0$ such that

$$||\nabla F(x^{k+1})||\le c||x^{k+1}-x^k||^2 \tag{8.23}$$

for k large enough. By the mean value theorem of integral calculus and the continuity and nonsingularity of the Hessian at x^*, (8.23) implies that there exists $c_1>0$ such that

$$||x^{k+1}-x^*||\le c_1||x^{k+1}-x^k||^2$$

for k large enough. Then,

$$\frac{||x^{k+1}-x^*||}{||x^{k+1}-x^*||+||x^k-x^*||}\le c_1||x^{k+1}-x^k||.$$

Therefore,

$$\frac{||x^k-x^*||}{||x^{k+1}-x^*||}\ge\frac{1}{c_1||x^{k+1}-x^k||}-1=\frac{1-c_1||x^{k+1}-x^k||}{c_1||x^{k+1}-x^k||}.$$

So,

$$\frac{||x^{k+1}-x^*||}{||x^k-x^*||}\le\frac{c_1||x^{k+1}-x^k||}{1-c_1||x^{k+1}-x^k||}.$$

Taking k large enough, since $||x^{k+1}-x^k||\to 0$, we have that

$$\frac{||x^{k+1}-x^*||}{||x^k-x^*||}\le 2c_1||x^{k+1}-x^k||\le 2c_1(||x^{k+1}-x^*||+||x^k-x^*||).$$

But, by (8.22), $||x^{k+1}-x^*||\le||x^k-x^*||$ for k large enough; thus

$$\frac{||x^{k+1}-x^*||}{||x^k-x^*||}\le 4c_1||x^k-x^*||,$$

and the quadratic convergence follows from this inequality. □

Theorem 8.6 assumes the existence and continuity of the second derivatives at x^*. However, we know that the Augmented Lagrangian function $L_\rho(x,\lambda,\mu)$ does not admit second derivatives with respect to x at the points in which $g_i(x)+\mu_i/\rho=0$ for some i. This is not a serious drawback and does not eliminate the explicative power of the theorem because of three main reasons. On the one hand, in many constrained optimiza-

tion problems, the *strict complementarity* property holds, meaning that $\mu_i^* > 0$ whenever $g_i(x^*) = 0$. Coming back to the chapter about boundedness of the penalty parameters, we see that, with additional conditions, this property is inherited by the approximate Lagrange multipliers and by their safeguarded approximations $\bar{\mu}_i$. Since, in addition, the penalty parameter will be bounded, we probably have continuous second derivatives in a neighborhood of the solution. On the other hand, even if the second derivatives are discontinuous at the solution, the gradient of the Augmented Lagrangian is *semismooth* in the sense of Qi and Sun [220]. Roughly speaking, this means that Newton's method and inexact Newton methods (Martínez and Qi [190]) can be defined and have good local convergence properties. Finally, constraints $g_i(x) \leq 0$ may be replaced by $g_i(x) + z_i = 0, z_i \geq 0$, where z_i is a slack variable. With such reformulation, no second derivative discontinuity occurs.

It is important to interpret correctly the results of this section in order to understand the computational behavior in practical situations. The global convergence Theorem 8.2 provides a general frame for the behavior of the algorithms, but the user of computational optimization methods is not expected in practice to observe iterates jumping between different accumulation points of the sequence $\{x^k\}$. The reason is given by the "capture" Theorems 8.3 and 8.4. These theorems say that isolated stationary points are powerful attractors for the algorithms considered here. This means that, in practice, the algorithm produces a sequence $\{x^k\}$ such that $||x^k|| \to \infty$ without limit points or such that it converges to a single point x^*. The possibility $||x^k|| \to \infty$ is discarded if the level sets are bounded; therefore the assumption $x^k \to x^*$ of Theorem 8.6 is not arbitrary. However, this theorem includes additional hypotheses that deserve to be discussed.

The Dennis–Moré condition is one of these hypotheses. It states that

$$\left\| \left[B_k - \nabla^2 F(x^k) \right] d^k \right\| = o(||d^k||).$$

This condition is obviously satisfied if we choose $B_k = \nabla^2 F(x^k)$ and tends to be fulfilled if B_k is close to the true Hessian, in particular if $||B_k - \nabla^2 F(x^k)|| \to 0$. (Consider, for example, the case in which the Hessian is computed using finite differences with discretization steps that tend to zero.) However, the Dennis–Moré condition tends to be fulfilled under much weaker assumptions. Observe first that, by Taylor,

$$\left\| \nabla^2 F(x^k) d^k - \left[\nabla F(x^k + d^k) - \nabla F(x^k) \right] \right\| = o(||d^k||).$$

Therefore, the Dennis–Moré condition will hold whenever

$$\left\| B_k d^k - \left[\nabla F(x^k + d^k) - \nabla F(x^k) \right] \right\| = o(||d^k||).$$

Now, assume that we choose the successive matrices B_k in order to satisfy the *secant equation* (Dennis and Schnabel [98]) given by

$$B_{k+1} d^k = \nabla F(x^k + d^k) - \nabla F(x^k).$$

Under the hypotheses of Theorem 8.6, this condition is asymptotically equivalent to

$$B_{k+1}(x^{k+1} - x^k) = \nabla F(x^{k+1}) - \nabla F(x^k). \tag{8.24}$$

In this case, the Dennis–Moré condition will be satisfied whenever $\lim_{k\to\infty} ||B_{k+1} - B_k|| = 0$. Over several decades, many efforts in numerical optimization have been made to define

methods that enjoy compatibility between the secant equation (8.24) and a low-variation requirement. Those methods, generally called *secant methods*, avoid the computation of second derivatives in the context of unconstrained optimization. The Dennis–Moré condition is the key tool for their convergence analysis.

The second important hypothesis of Theorem 8.6 is $\lim_{k\to\infty} \eta_k = 0$. The strict interpretation of this hypothesis would require the a priori definition of a sequence that converges to zero, as $\{1/k\}$ or $\{1/k^2\}$. A less trivial interpretation comes from considering that η_k is the tolerance for the error in the solution of the linear system $B_k d = -\nabla F(x^k)$, measured by the comparison of the residuals at $d = 0$ and at the computed approximate solution. The smaller this tolerance, the more accurate the linear system solution, and we could expect something close to superlinear convergence. In general, we have two possibilities: (i) we solve the linear system "exactly" or (ii) we solve it only approximately using some iterative method such as conjugate gradients. In the first case, η_k is very small but not exactly zero, because in the computer we work with high precision but not infinite precision. The exact solution of the system is frequently associated with the observance of superlinear convergence. When we solve the system using an iterative method, it is usual to fix a unique value for η_k (small), so that the convergence, although not superlinear, is reasonably fast.

Now, how is superlinear convergence observed in practice? Before answering this question, let us formulate another: Is it observable that, for k large enough, $t_k = 1$, that is, that each iteration asymptotically involves a single function evaluation? The answer to the second question is *yes*. In well-behaved problems, in which the sequence generated by the algorithm converges to a point x^* with positive definite Hessian, we really observe that, near x^*, the first tentative $t = 1$ produces enough decrease and, consequently, $x^{k+1} = x^k + d^k$ if B_k is similar to a true Hessian $\nabla^2 F(x^k)$ and the solution of the linear system is accurate. This behavior is not observed only if the Hessian is almost singular or ill-conditioned or if the third derivatives are dominant, so the Lipschitz constant of the Hessians is very big and the basin of convergence of Newton's method is small. (Unfortunately this may be the situation when F has the form (8.1) and the penalty parameter is big.)

Finally, superlinear convergence means that $\|x^{k+1} - x^*\|/\|x^k - x^*\| \to 0$, a property that, assuming that $\nabla^2 F(x^*)$ is nonsingular and continuous in a neighborhood of x^* (by the mean value theorem of integral calculus), is equivalent to

$$\lim_{k\to\infty} \frac{\|\nabla F(x^{k+1})\|}{\|\nabla F(x^k)\|} = 0. \tag{8.25}$$

Usually, in well-behaved problems, one observes that $\|\nabla F(x^{k+1})\|$ decreases significantly with respect to $\|\nabla F(x^k)\|$, but the user should not expect an academic immaculate convergence to zero of the quotient. If we observe that, at some iteration, the norm of the gradient is, say, one half the norm of the previous one, with some tendency to (usually nonmonotone) decrease, this does not mean that superlinear convergence is being violated. Ultimately, (8.25) is an asymptotic property.

8.5 ▪ Computing search directions

8.5.1 ▪ Newton and stabilized Newton approaches

Newton's method may be applied to minimize the function $F(x)$ given by (8.1). Denoting

$$H(x) = h(x) + \lambda/\rho \text{ and } G(x) = g(x) + \mu/\rho,$$

we have that

$$F(x) = f(x) + \frac{\rho}{2}\left(||H(x)||_2^2 + ||G(x)_+||_2^2\right). \tag{8.26}$$

The function F has continuous first derivatives. The second derivatives are discontinuous, but $\nabla F(x) = 0$ is a semismooth [220] system of equations so that the Newton's approach makes sense and its unitary-step version converges quadratically under local nonsingularity conditions. Inexact Newton methods can also be applied for solving that system [190].

The iterates generated by the algorithm for minimizing $F(x)$ will be denoted, as usual, by x^k. Without loss of generality, let us assume that, given a generic iterate x^k, we have that $G_i(x^k) \geq 0$ for all $i = 1, \ldots, q$ and $G_i(x^k) < 0$ for $i = q+1, \ldots, p$. Consequently, we define

$$\underline{G}(x) = (G_1(x), \ldots, G_q(x))^T.$$

Clearly, Newton's iteration for the minimization of $F(x)$ using the current point x^k coincides with Newton's iteration for the minimization of $f(x) + \frac{\rho}{2}(||H(x)||_2^2 + ||\underline{G}(x)||_2^2)$. With abuse of notation, let us redefine

$$F(x) = f(x) + \frac{\rho}{2}\left(||H(x)||_2^2 + ||\underline{G}(x)||_2^2\right).$$

Therefore,

$$\nabla F(x) = \nabla f(x) + \rho\left(\nabla H(x) H(x) + \nabla \underline{G}(x)\underline{G}(x)\right)$$

and

$$\nabla^2 F(x) = \nabla^2 f(x) + \rho\left(H'(x)^T H'(x) + \underline{G}'(x)^T \underline{G}'(x) + \sum_{i=1}^{m} H_i(x)\nabla^2 H_i(x) + \sum_{i=1}^{q} \underline{G}_i(x)\nabla^2 \underline{G}_i(x)\right).$$

In principle, Newton's iteration requires the solution of the linear system

$$\nabla^2 F(x^k)(x - x^k) = -\nabla F(x^k). \tag{8.27}$$

The value of ρ may be large because it needed to be increased many times after the test (4.9) or, more frequently, because one deliberately decides to start with a big penalty parameter with the aim of getting a solution very fast. This "shortcut" strategy (Fletcher [113]) may be useful when we have a guaranteed good starting point, in addition to a good approximation of the Lagrange multipliers, and we do not want to lose feasibility at the first Augmented Lagrangian iterations. (This is exactly what we wish when dealing with parametric optimization [134, 157].) However, in this case, the naive application of Newton's method may lead to poor results due to the following reasons:

1. Although Newton's direction is generally a descent direction for $F(x)$ (at least after some possible correction on the matrix of the system), the unitary step, which should generate quadratic convergence, may not be accepted by monotone line search procedures unless the current point is very close to the solution.

2. The Newtonian linear system is generally very ill-conditioned if ρ is large.

Both phenomena are associated with the size of the penalty parameter but they are not the same phenomenon. The second one may be overcome by means of a decomposition of the Newtonian linear system. This fact leads some people to argue that there is no real problem with big penalty parameters. However, the first phenomenon persists even if we solve the linear system by means of decomposition techniques, because it is intrinsic

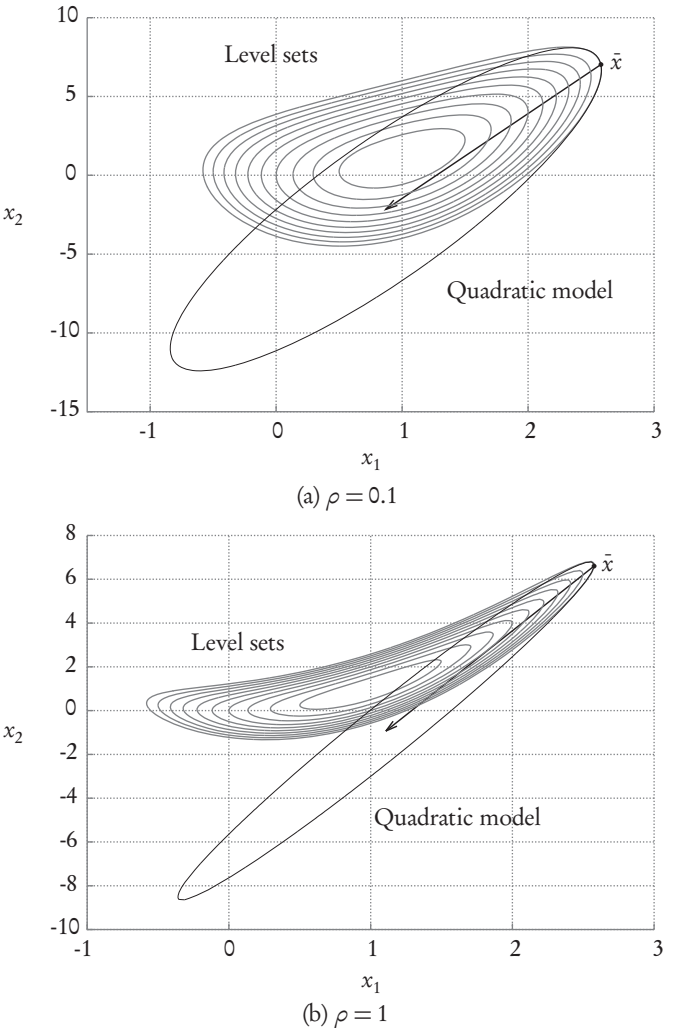

Figure 8.1. *Level sets and Newton's direction for* $F(x) = f(x) + (\rho/2)||H(x)||_2^2$, *where* $f(x) = (1-x_1)^2$ *and* $H(x) = 2(x_2 - x_1^2)$. *(Note that* $F(x)$ *with* $\rho = 100$ *is the Rosenbrock's function.) In (a),* $\rho = 0.1$, *while in (b)* $\rho = 1$. *In both cases,* $\bar{x}$ *is approximately at the same distance to the solution* $x^* = (1,1)^T$. *In the first case, the unitary step along the Newton's direction produces a decrease in* $F(x)$, *while in the second case it does not.*

to the fact that for big values of ρ, functional values of $F(x)$ are dominated by those of $(\rho/2)[||H(x)||_2^2 + ||G(x)_+||_2^2]$. If ρ is large, the level sets of $(\rho/2)[||H(x)||_2^2 + ||G(x)_+||_2^2]$ tend to be parallel surfaces to the feasible set and $F(x)$ tends to produce flat curved valleys as level sets, more or less following the shape of the boundary of the feasible region. In these conditions, the minimizer of the quadratic approximation tends to be outside the level set, as shown in Figure 8.1. Completely overcoming this inconvenience is impossible, but nonmonotone techniques, which tolerate an occasional increase of the functional value, tend to alleviate it.

In any case, it is always convenient to decompose the system in such a way that the computation of Newtonian directions becomes well-conditioned.

The Newtonian linear system may be written as

$$\begin{aligned}&\big[B(x^k)+\rho\big(H'(x^k)^TH'(x^k)+\underline{G}'(x^k)^T\underline{G}'(x^k)\big)\big](x-x^k)\\&\quad=-\big(\nabla f(x^k)+\rho\big(\nabla H(x^k)H(x^k)+\nabla\underline{G}(x^k)\underline{G}(x^k)\big)\big),\end{aligned}\tag{8.28}$$

where

$$B(x^k)=\nabla^2 f(x^k)+\rho\left(\sum_{i=1}^{m}H_i(x^k)\nabla^2H_i(x^k)+\sum_{i=1}^{q}\underline{G}_i(x^k)\nabla^2\underline{G}_i(x^k)\right).\tag{8.29}$$

Note that the matrix of (8.28) should be positive definite. (In particular, its diagonal elements should be positive, a fact that is easy to verify and correct even in the decomposition context that follows.)

The system (8.28) is equivalent to

$$\begin{aligned}&B(x^k)(x-x^k)+\rho\nabla H(x^k)\big(H(x^k)+H'(x^k)(x-x^k)\big)\\&\quad+\rho\nabla\underline{G}(x^k)\big(\underline{G}(x^k)+\underline{G}'(x^k)(x-x^k)\big)=-\nabla f(x^k).\end{aligned}$$

Defining

$$\lambda_{\text{new}}=\rho\big(H(x^k)+H'(x^k)(x-x^k)\big)\ \text{ and }\ \mu_{\text{new}}=\rho\big(\underline{G}(x^k)+\underline{G}'(x^k)(x-x^k)\big),$$

the system becomes

$$B(x^k)(x-x^k)+\nabla H(x^k)\lambda_{\text{new}}+\nabla\underline{G}(x^k)\mu_{\text{new}}=-\nabla f(x^k).$$

In other words, the Newtonian direction $x-x^k$ comes from solving the linear equations

$$\begin{aligned}B(x^k)(x-x^k)+\nabla H(x^k)\lambda_{\text{new}}+\nabla\underline{G}(x^k)\mu_{\text{new}}&=-\nabla f(x^k),\\H'(x^k)(x-x^k)-\lambda_{\text{new}}/\rho&=-H(x^k),\\\underline{G}'(x^k)(x-x^k)-\mu_{\text{new}}/\rho&=-\underline{G}(x^k),\end{aligned}$$

which, in matricial terms, can be written as

$$\begin{pmatrix}B(x^k)&H'(x^k)^T&\underline{G}'(x^k)^T\\H'(x^k)&-I/\rho&0\\\underline{G}'(x^k)&0&-I/\rho\end{pmatrix}\begin{pmatrix}x-x^k\\\lambda_{\text{new}}\\\mu_{\text{new}}\end{pmatrix}=\begin{pmatrix}-\nabla f(x^k)\\-H(x^k)\\-\underline{G}(x^k)\end{pmatrix}.\tag{8.30}$$

Finally, note that $\rho H_i(x^k)=\lambda_i+\rho h_i(x^k)$ and $\rho\underline{G}_i(x^k)=\rho\max\{0,g_i(x^k)+\mu_i/\rho\}=\max\{0,\mu_i+\rho g_i(x^k)\}$. In the case that ρ is big and $h_i(x^k)$ or $g_i(x^k)$ is big too, the corresponding term may be inconveniently large, and so it is recommendable to redefine (8.29) as

$$B(x^k)=\nabla^2 f(x^k)+\sum_{i=1}^{m}\lambda_i^k\nabla^2H_i(x^k)+\sum_{i=1}^{q}\mu_i^k\nabla^2\underline{G}_i(x^k),\tag{8.31}$$

where λ^k and μ^k may be given by the vectors λ_{new} and μ_{new} obtained in the previous Newton's iteration. By (4.7), (4.8), and the results related to the boundedness of the penalty parameter given in Chapter 7, $\rho H(x^k)$ and $\rho\underline{G}(x^k)$ are estimates of the Lagrange multipliers at the solution of the subproblem (4.6). On the other hand, if ρ is very big, and one replaces the matrix $B(x^k)$ given by (8.29) with the one given by (8.31) (with $\lambda^k=\lambda_{\text{new}}$ and $\mu^k=\mu_{\text{new}}$) in (8.30), the linear system (8.30) becomes a stabilized (with ρ) Newtonian linear system for the KKT condition of the original problem (4.1).

The linear system (8.30) (defined with the matrix $B(x^k)$ given by (8.29) or (8.31)) is a linear system with $n + m + q$ equations and unknowns and with no apparent ill-conditioning problems. Note that, when $\rho \to \infty$, the condition number of the matrix tends to the condition number of a matrix where ρ does not appear at all.

It is interesting to analyze the case in which, in the solution of (8.30), we have $(x - x^k, \lambda_{\text{new}}, \mu_{\text{new}}) = (0, \lambda, \mu)$. By the second and third blocks of (8.30), we have, in that case, that $h(x^k) = 0$ and $g_i(x^k) = 0$ for all i such that $g_i(x^k) \geq -\mu_i/\rho$. Therefore, x^k is a feasible point of the original problem (4.1). Moreover, the first block of (8.30) states that the KKT condition of the original problem holds. This means that the distance between the solution of (8.30) and (x^k, λ, μ) provides a sensible measure of optimality.

The system (8.30) needs a possible correction in the case in which the generated direction $d^k \equiv x - x^k$ is not a descent direction for $F(x)$ at x^k. Note that taking a sufficiently big correcting parameter c_k and replacing $B(x^k)$ with $B(x^k) + c_k I$ in (8.30), the angle between d^k and $-\nabla F(x^k)$ can be made as small as desired, so that the condition $\nabla F(x^k)^T d^k \leq -\theta ||\nabla F(x^k)||_2 ||d^k||_2$, required at Step 2 of Algorithm 8.1, is satisfied. Moreover, multiplying d^k by a suitable scalar, the condition $||d^k|| \geq \beta ||\nabla F(x^k)||$ will also be satisfied. After the computation of the (perhaps corrected) direction d^k, the next iterate is computed by means of a line search following the general Algorithm 8.1. The dominance phenomenon of the penalized term with respect to the function $f(x)$ for large values of ρ makes it desirable to employ a nonmonotone decreasing strategy.

Linear algebra

If ρ is not very large, we may try to obtain the Newton direction by directly solving (8.27). Since $\nabla^2 F(x^k)$ is symmetric, we may try to compute its Cholesky factorization. If this factorization can be completed, obtaining $\nabla^2 F(x^k) = LL^T$ with L lower-triangular and nonsingular, the direction obtained solving $LL^T d^k = -\nabla F(x^k)$ is a descent direction along which classical line searches produce sufficient descent. However, directions whose angle with $-\nabla F(x^k)$ is very close to $\pi/2$ should be replaced with more conservative directions that may be obtained by increasing the diagonal of $\nabla^2 F(x^k)$ by a tiny positive scalar. If the Cholesky factorization of the Hessian cannot be completed, several possibilities arise. Using a good estimation of the lowest eigenvalue we may add a suitable positive diagonal matrix to the Hessian by means of which a descent direction is guaranteed after solving the linear system that replaces (8.27). The direction d^k obtained in this way minimizes the quadratic approximation of $F(x)$ on the region defined by $||d||_2 \leq ||d^k||_2$, a property that approximates this strategy to the well-known field of trust-region methods [83].

In the case of a large penalty parameter ρ, the system (8.30) may be solved by means of variations of a factorization proposed by Bunch and Parlett [71] for symmetric matrices. Improvements have been obtained in [23] and any of the successors of the classical Harwell subroutine MA27 (based on [103]), such as MA57, MA86, or MA97, is the rule of choice in this case. The interpretation of diagonal modifications of B_k is the same as above and also connects the strategy to the trust-region framework. Iterative linear solvers may also be used to solve (8.30).

8.5.2 ▪ Truncated Newton approach

In the truncated Newton approach, we consider the quadratic problem

$$\text{Minimize } \frac{1}{2} d^T \nabla^2 F(x^k) d + \nabla F(x^k)^T d \tag{8.32}$$

and we try to find an approximate solution d^k employing the classical conjugate gradient (CG) method of Hestenes and Stiefel [146]. To achieve this goal, we start with the iterate $d = 0$, and, at each CG iteration, we test the conditions (8.2), taking care that the final approximate solution of (8.32) satisfies them. This is not difficult since the first iterate of the CG method is in general a multiple of $-\nabla F(x^k)$, thus satisfying trivially (8.2). As far as (8.2) holds at the CG iterate, we continue until minimizing (8.32) with some required precision.

When the Hessian $\nabla^2 F(x^k)$ is not positive definite, the problem (8.32) may have no solution. In this case, CG is interrupted but the possibility of computing an adequate direction from the point of view of (8.2) is not affected. (In the worst case we may be forced to compute the approximate solution of (8.32) as a multiple of $-\nabla F(x^k)$.)

At each step of the CG method, we need to compute a Hessian-vector product. Sometimes, we do not wish to provide the true Hessian because we cannot compute it or because it is computationally expensive. In this case, each matrix-vector product may be replaced by an incremental quotient, using the approximation

$$\nabla^2 F(x^k)d \approx \frac{1}{t}\left(\nabla F(x^k + td) - \nabla F(x^k)\right)$$

with a small value of t. In this case, each CG iteration involves an additional evaluation of ∇F.

8.5.3 ▪ Quasi Newton and preconditioners

We aim to create an adequate quasi-Newton formula for approximating the Hessian matrix $\nabla^2 F(x)$ in the large-scale case. Recall that this Hessian is well defined whenever $\mu_i + \rho g_i(x) \neq 0$ for all $i = 1, \ldots, p$. By direct calculations, we have that

$$\nabla^2 F(x) = \nabla^2 f(x) + A(x) + C(x),$$

where

$$A(x) = \rho\left(\sum_{i=1}^{m} \nabla h_i(x)\nabla h_i(x)^T + \sum_{i \in I(x,\mu)} \nabla g_i(x)\nabla g_i(x)^T\right),$$

$$C(x) = \sum_{i=1}^{m} [\lambda_i + \rho h_i(x)]\nabla^2 h_i(x) + \sum_{i \in I(x,\mu)} [\mu_i + \rho g_i(x)]\nabla^2 g_i(x),$$

and

$$I(x,\mu) = \{i \in \{1, \ldots, p\} \mid \mu_i + \rho g_i(x) > 0\}.$$

Assume that x^c is the current iterate at Algorithm 8.1 and x^p is the previous one. We want to construct a reasonable and cheap approximation of $\nabla^2 F(x^c)$. Let us define $s = x^c - x^p$ and $y = \nabla F(x^c) - \nabla F(x^p)$. In order to obtain the Hessian approximation, the matrix $A(x^c)$ will be corrected twice. At the first correction, we add a diagonal matrix σI with the objective of guaranteeing nonsingularity. (Note that $A(x)$ is always positive semidefinite.) Following the "spectral approach" [30, 224, 225, 60], we define

$$\sigma_{\text{spec}} = \operatorname{argmin} \|(A(x) + \sigma I)s - y\|^2.$$

This implies that

$$\sigma_{\text{spec}} = \frac{(y - A(x^c)s)^T s}{s^T s}.$$

Using safeguards, we correct σ_{spec} taking

$$\sigma = \max\{\sigma_{\min}, \min\{\sigma_{\max}, \sigma_{\text{spec}}\}\},$$

where $0 < \sigma_{\min} < \sigma_{\max} < +\infty$ are given parameters, and

$$A_+ = A(x^c) + \sigma I.$$

If $s^T y \leq 10^{-8}\|s\|\|y\|$, we define $H = A_+$. Otherwise, we correct A_+ in order to satisfy the secant equation, maintaining its positive definiteness. Since A_+ is positive definite, it is natural to correct this matrix using the famous BFGS formula [98]. So,

$$H = A_+ + \frac{yy^T}{s^T y} - \frac{A_+ ss^T A_+}{s^T A_+ s}. \tag{8.33}$$

In this way, the Hessian approximation H satisfies the secant equation and remains positive definite.

The search direction will be obtained by solving a linear system, whose matrix is H, employing the CG method. A suitable preconditioner for the application of CG may be obtained as follows:

1. Define $D = \text{diag}(A(x^c))$.
2. Compute its associated spectral coefficient
$$\sigma_P = \max\left\{\sigma_{\min}, \min\left\{\sigma_{\max}, \frac{(y - Ds)^T s}{s^T s}\right\}\right\}.$$
3. Compute $D_+ = D + \sigma_P I$.
4. As in the computation of H, if $s^T y \leq 10^{-8}\|s\|\|y\|$, define $H_P = D_+$. Otherwise, define
$$H_P = D_+ + \frac{yy^T}{s^T y} - \frac{D_+ ss^T D_+}{s^T D_+ s}. \tag{8.34}$$
(Note that H_P does not need to be computed explicitly.)

In this process, H_P is the BFGS correction of a positive definite matrix, and, thus, it is positive definite too [98]. The inverse of H_P is given by

$$H_P^{-1} = D_+^{-1} + \frac{(s - D_+^{-1}y)s^T + s(s - D_+^{-1}y)^T}{s^T y} - \frac{(s - D_+^{-1}y)^T yss^T}{(s^T y)^2}. \tag{8.35}$$

The explicit form of H_P^{-1} shows that H_P may be used as preconditioner for the CG method. Moreover, this preconditioner may be used too for the CG iterations in the truncated Newton scheme.

8.6 ▪ Review and summary

In this chapter, we addressed the unconstrained optimization problem with a focus on the case in which the objective function is the Augmented Lagrangian. We concentrated on line-search methods, although a similar chapter could be written along similar lines using the trust-region framework. For a general line-search algorithm that admits intermediate "magic" steps, we proved global and local convergence results.

8.7 ▪ Further reading

The present chapter may be read as a (biased) short course on unconstrained optimization. The classical bibliography on this subject complements this study. The Dennis–Schnabel book [98] covers the essence of Newtonian and quasi-Newton methods and the classical book by Ortega and Rheinboldt [215] is a mandatory reference for local and global properties of Newton and related methods. Trust-region methods, not addressed in this chapter, were exhaustively surveyed in [83]. Generalizations of the Hestenes–Stiefel CG method to the nonquadratic case began with the classical Fletcher–Reeves and Polak–Ribière papers [116, 217] and were followed by many variations. The main drawback of these CG generalizations is that they require rather strict line searches due to their connection with the Hestenes–Stiefel method, in which exact one-dimensional minimization is essential. Modern CG methods try to alleviate this inconvenience as much as possible. The unconstrained CG methods with the best performance according to the available literature are due to Dai and Yuan [93] and Hager and Zhang [138, 139, 140].

If one applies Newton's method to an unconstrained minimization problem and the Hessian is positive definite at every iteration, a reasonable conjecture is that every limit point is stationary. Surprisingly, this conjecture is not true (Mascarenhas [195]), showing that safeguards in terms of the angle between the Newton direction and the gradient are necessary. Related results, concerning the nonconvergence of the BFGS method, were given by Dai [90, 91] and Mascarenhas [194, 195]. For many years, it was believed that limit points generated by the BFGS method, with standard line-search procedures, should be stationary, but the counterexamples exhibited in [90, 91, 194, 195] show that this is not true.

8.8 ▪ Problems

8.1 Prove that, if the search direction is given by $d^k = -H_k \nabla F(x^k)$ and the matrix H_k is symmetric and positive definite, one has that $\nabla F(x^k)^T d^k < 0$ and, consequently, the basic unconstrained optimization algorithm is well defined.

8.2 Find an explicit formula for the global minimizer of a quadratic along a given direction, when such a formula exists.

8.3 Prove that, if the set $\{\|B_k\|, k \in \mathbb{N}\}$ is bounded, the condition $\|d^k\| \geq \beta \|\nabla F(x^k)\|$ is satisfied, as required by Algorithm 8.1. Moreover, prove that if $\eta_k = 0$, then the condition number $\|B_k\|_2 \|B_k^{-1}\|_2$ is smaller than or equal to $1/\theta$ and the angle condition defined in Step 2 also holds.

8.4 Consider $F(x_1, x_2) = x_1^2 + x_2$. Take $x^0 = (-\pi, 0)^T$ as the initial point. Define the search direction d^k as being $d^k = (1,0)^T$ if $x_1^k < 0$ and $d^k = (-1,0)^T$ if $x_1^k > 0$. Assume that t_k is the largest value in $\{1, 1/2, 1/4, \ldots\}$ such that $x^k + t_k d^k$ satisfies the Armijo criterion (8.3) and that $x^{k+1} = x^k + t_k d^k$. Show that the sequence $\{x^k\}$ converges to $x^* = (0,0)^T$ and that $\nabla F(x^*) \neq 0$. Moreover, show that, although the search directions d^k are descent directions, for any $\theta \in (0,1)$ there exists k_0 such that the angle condition in (8.2) does not hold for every $k \geq k_0$.

8.5 Replace condition (8.3) with

$$F(x^{k+1}) \leq F(x^k + t_k d^k) \leq F(x^k) + \alpha t_k \max\{-c, \nabla F(x^k)^T d^k\},$$

where $c > 0$ is a given constant. Interpret geometrically and prove Theorems 8.1 and 8.2 with this modification.

8.6 Prove that, when the condition number of the positive definite Hessian is uniformly bounded by $c > 0$, the cosine of the angle between the negative gradient and the Newton direction is at least $1/c$.

8.7 Prove that, when the objective function is quadratic and one minimizes along the search direction, the corresponding minimizer satisfies the Armijo condition for any $\alpha \leq 1/2$. (This is a strong motivation for using $\alpha < 1/2$. Why?)

8.8 Consider the function $F(x_1, x_2) = (x_1 - 1)^2 + \frac{\rho}{2}(x_2 - x_1^2)^2$. Find $\delta(\rho)$ such that, if $||(x_1, x_2)^T - (1, 1)^T|| \leq \delta(\rho)$, the unitary step along the Newton direction provides descent. Note that $\delta(\rho) \to 0$ when $\rho \to \infty$.

8.9 Prove that the direction d^k obtained solving (8.27) minimizes the quadratic approximation of $F(x)$ on the region defined by $||d||_2 \leq ||d^k||_2$, a property that approximates this strategy to the well-known field of trust-region methods (see the Conn, Gould, and Toint book [83]).

8.10 Consider a matrix B_0 that is symmetric and positive definite. Assume that B_{k+1} must satisfy the secant equation (8.24) and that the rank of $B_{k+1} - B_k$ must be 2. Discover the BFGS formula.

8.11 Consider a matrix H_0 that is symmetric and positive definite. Assume that H_{k+1} must satisfy the secant equation $H_{k+1}(\nabla F(x^{k+1}) - \nabla F(x^k)) = (x^{k+1} - x^k)$ and that the rank of $H_{k+1} - H_k$ must be 2. Discover the so-called DFP (Davidon–Fletcher–Powell) formula.

8.12 Show that H obtained by (8.33) is positive definite.

8.13 Prove that H_P^{-1} defined by (8.35) is the inverse of H_P defined by (8.34).

8.14 The spectral gradient approach comes from approximating the Hessian by a diagonal matrix with identical diagonal elements. The idea of employing more general diagonal approximations has been suggested many times. For general test functions it does not seem to be a good idea, perhaps because the spectral properties of the equally diagonal approach are lost. However, in real-life applications, the presence of separable or almost-separable objective functions is not rare. Practical optimization models should deal with noncorrelated variables as far as possible. The lack of correlation between variables is naturally linked to almost-diagonal Hessians. In this context, the diagonal updating idea may be useful. Define an optimization method based on diagonal updatings of a diagonal approximate Hessian and analyze convergence supported by the theory presented in this chapter.

8.15 Assume that $\nabla F(x^*) = 0$, $\nabla^2 F(x^*)$ is nonsingular, and $\{x^k\}$ converges superlinearly (respectively, quadratically) to x^*. Prove that $F(x^k)$ converges superlinearly (respectively, quadratically) to $F(x^*)$. Show that this result is not true if we replace "superlinearly" with "linearly." Derive consequences for numerical minimization algorithms from this property. (How reliable is to consider that $F(x^k) - F(x^*)$ measures the distance to the solution?)

8.16 Assume that $F(x^k)$ converges to $F(x^*)$ linearly, superlinearly, or quadratically. Imagine that you plot $F(x^k) - F(x^*)$ as a function of k. How does this graphic look in each case? What about the graphic of $\log(F(x^k) - F(x^*))$? Repeat this exercise with $||\nabla F(x^k)||$ instead of $F(x^k) - F(x^*)$.

Chapter 9
Solving Constrained Subproblems

Recall that, at each outer iteration of the Augmented Lagrangian method, we need to minimize the function $L_{\rho_k}(x, \bar{\lambda}^k, \bar{\mu}^k)$ on a lower-level set Ω. The case in which $\Omega = \mathbb{R}^n$ was studied in Chapter 8. Here, we will consider the case in which Ω is closed and convex and the more particular case in which Ω is a box. As in Chapter 8, we will denote $\rho = \rho_k$, $\lambda = \bar{\lambda}^k$, $\mu = \bar{\mu}^k$, and

$$F(x) = L_\rho(x, \lambda, \mu). \tag{9.1}$$

Continuous first derivatives of f, h, and g (and, consequently, of F) will be assumed to exist whenever necessary.

9.1 ▪ Spectral projected gradient

In this section, we consider the case in which the subproblem (4.6) takes the form

$$\text{Minimize } F(x) \text{ subject to } x \in \Omega \tag{9.2}$$

and $\Omega \subseteq \mathbb{R}^n$ is a closed and convex set, which perhaps is not described by a finite number of equalities or inequalities. However, we will consider that computing $P_\Omega(x)$, the Euclidean projection of an arbitrary $x \in \mathbb{R}^n$ onto Ω, is affordable. If in addition n is large, the spectral projected gradient method (SPG) [60, 61, 62, 63] may be the best alternative for solving subproblem (9.2).

The iterates of algorithms for solving (9.2) will be called (with some abuse of notation) x^k for $k = 0, 1, 2, \ldots$. We warn that these iterates should not be confused with the ones that define the Augmented Lagrangian iterations.

Given $x^0 \in \Omega$, $\alpha \in (0, 1/2)$, and $0 < \sigma_{\min} \ll \sigma_{\max}$, the SPG method computes

$$d^k = P_\Omega\left(x^k - \frac{1}{\sigma_k^{SPG}} \nabla F(x^k)\right) - x^k \tag{9.3}$$

and x^{k+1} such that $F(x^{k+1}) \leq F(x^k + t_k d^k)$, where $\sigma_k^{SPG} \in [\sigma_{\min}, \sigma_{\max}]$ and t_k is obtained by means of a backtracking procedure that guarantees the sufficient descent condition

$$F(x^k + t_k d^k) \leq F_k^{\text{ref}} + \alpha t_k \nabla F(x^k)^T d^k. \tag{9.4}$$

The reference value F_k^{ref} is generally chosen as

$$F_k^{\text{ref}} = \max\{f(x^k), f(x^{k-1}), \ldots, f(x^{\max\{0, k-M+1\}})\}$$

(Grippo, Lampariello, and Lucidi [132]) with M around 10. The SPG parameter σ_k^{SPG} is usually defined by

$$\sigma_k^{SPG} = \begin{cases} 1 & \text{if } k = 0, \\ \max\left\{\sigma_{\min}, \min\left\{\dfrac{(s^k)^T y^k}{(s^k)^T s^k}, \sigma_{\max}\right\}\right\} & \text{otherwise,} \end{cases} \tag{9.5}$$

where $s^k = x^k - x^{k-1}$ and $y^k = \nabla F(x^k) - \nabla F(x^{k-1})$. Alternatively, σ_0^{SPG} may be defined as

$$\sigma_0^{SPG} = \max\left\{\sigma_{\min}, \min\left\{\frac{\bar{s}^T \bar{y}}{\bar{s}^T \bar{s}}, \sigma_{\max}\right\}\right\}, \tag{9.6}$$

where $\bar{s} = x^0 - \bar{x}$ and $\bar{y} = \nabla F(x^0) - \nabla F(\bar{x})$, $\bar{x} = x^0 - t_{\text{small}} \nabla F(x^0)$, and t_{small} is a small positive number.

The backtracking procedure for computing t_k is as follows. It begins with the trial $t = 1$ and testing the fulfillment of (9.4) (for $t_k = t$). If (9.4) holds, the backtracking finishes. Otherwise, a new t in the interval $[0.1t, 0.5t]$ is chosen using safeguarded quadratic interpolation and the process is repeated. Since the direction d^k in (9.3) is a descent direction, this loop necessarily finishes with the fulfillment of (9.4) for a sufficiently small t_k.

The SPG method has three main ingredients: projected gradient ideas [38, 126, 174]; the choice of the steplength, motivated by Barzilai and Borwein [30] and elucidated by Raydan [224, 225] for unconstrained problems; and nonmonotone line searches [132]. Because of the simplicity of this method and its capacity to deal with huge problems, it has been used in multiple applications since its introduction by Birgin, Martínez, and Raydan [60, 61]. It can be proved (see [62]) that *every* limit point of a sequence generated by SPG satisfies the optimality condition (3.31), given by $P_\Omega(x^* - \nabla F(x^*)) = x^*$. Here, we will give a simplified and less ambitious proof, where we only show that, if the sequence $\{x^k\}$ is bounded, *at least one* limit point is stationary in the sense of (3.31). In this proof, we will follow the approach of Birgin, Martínez, and Raydan [62].

9.1.1 ▪ Convergence of SPG

Throughout this section, we will denote

$$Q_k(d) = \frac{1}{2}\sigma_k^{SPG} \|d\|_2^2 + \nabla F(x^k)^T d.$$

The global minimizer of $Q_k(d)$ subject to $x^k + d \in \Omega$ is given by (9.3).

Algorithm 9.1. SPG.
Let $\alpha \in (0,1)$, $0 < \sigma_{\min} < \sigma_{\max}$, and M be a positive integer. Let $x^0 \in \Omega$ be an arbitrary initial point. Given $x^k \in \Omega$, the steps to compute x^{k+1} are as follows:

Step 1. Compute $\sigma_k^{SPG} \in [\sigma_{\min}, \sigma_{\max}]$ and d^k as in (9.5) and (9.3), respectively. If $d^k = 0$, stop the execution of the algorithm declaring that x^k is a stationary point.

Step 2. Set $t \leftarrow 1$ and $F_k^{\text{ref}} = \max\{F(x^{k-j+1}) \mid 1 \le j \le \min\{k+1, M\}\}$.

If

$$F(x^k + t d^k) \le F_k^{\text{ref}} + t\alpha \nabla F(x^k)^T d^k, \tag{9.7}$$

set $t_k = t$, choose $x^{k+1} \in \Omega$ such that

$$F(x^{k+1}) \le F(x^k + t_k d^k), \tag{9.8}$$

and finish the iteration. Otherwise, choose $t_{\text{new}} \in [0.1t, 0.5t]$, set $t \leftarrow t_{\text{new}}$ and repeat test (9.7).

The lemma below shows that Algorithm 9.1 is well defined. In particular, it shows that the direction d^k computed as in (9.3) is a descent direction. Then, for completeness, it shows (as already shown in Theorem 8.1) that, if d^k is a descent direction, a steplength t_k that satisfies the Armijo condition can be computed in a finite number of steps.

Lemma 9.1. *Algorithm 9.1 is well defined.*

Proof. If $d^k = 0$ the algorithm stops. Otherwise, we have that $Q_k(d^k) \leq Q_k(0) = 0$, i.e., $\nabla F(x^k)^T d^k \leq -\frac{1}{2}\sigma_k^{SPG}||d^k||_2^2 < 0$, since $d^k \neq 0$ and, by (9.5), $0 < \sigma_{\min} \leq \sigma_k^{SPG}$. Now,

$$\lim_{t\to 0} \frac{F(x^k + t d^k) - F(x^k)}{t} = \nabla F(x^k)^T d^k < 0.$$

Therefore,

$$\lim_{t\to 0} \frac{F(x^k + t d^k) - F(x^k)}{t \nabla F(x^k)^T d^k} = 1$$

and

$$\frac{F(x^k + t d^k) - F(x^k)}{t \nabla F(x^k)^T d^k} > \alpha$$

if t is small enough. Thus, for $t > 0$ small enough,

$$F(x^k + t d^k) < F(x^k) + t\alpha \nabla F(x^k)^T d^k \leq F_k^{\text{ref}} + t\alpha \nabla F(x^k)^T d^k.$$

This completes the proof. □

The lemma below shows that, if Algorithm 9.1 does not generate an infinite sequence of iterates, it stops at a stationary point.

Lemma 9.2. *Assume that the sequence generated by Algorithm 9.1 stops at x^k. Then, x^k is stationary.*

Proof. The proof follows from the characterization given in Lemma 3.2. □

For the remaining results of this section, we assume that the algorithm does not stop. So, infinitely many iterates $\{x^k\}_{k\in\mathbb{N}}$ are generated, and, by (9.7), we have that $F(x^k) \leq F(x^0)$ for all $k \in \mathbb{N}$. In order to ensure the existence of limit points, we state the following assumption.

Assumption 9.1. *The level set $\{x \in \Omega \mid F(x) \leq F(x^0)\}$ is bounded.*

Assumption 9.1 will be supposed to be true all along the present section. Note that Assumption 9.1 holds if Ω is bounded.

The five lemmas below will be used to prove a final theorem that says that the sequence generated by Algorithm 9.1 has at least one limit point that is stationary.

Lemma 9.3. *Assume that $\{x^k\}_{k\in\mathbb{N}}$ is a sequence generated by Algorithm 9.1. Define, for all $j = 1, 2, 3, \ldots$,*

$$V_j = \max\{F(x^{jM-M+1}), F(x^{jM-M+2}), \ldots, F(x^{jM})\}$$

and $\nu(j) \in \{jM - M + 1, jM - M + 2, \ldots, jM\}$ such that $F(x^{\nu(j)}) = V_j$. Then,

$$V_{j+1} \leq V_j + t_{\nu(j+1)-1}\alpha \nabla F(x^{\nu(j+1)-1})^T d^{\nu(j+1)-1} \tag{9.9}$$

for all $j = 1, 2, 3, \ldots$.

Proof. We will prove by induction on ℓ that, for all $\ell = 1,2,\dots,M$ and for all $j = 1,2,3,\dots,$

$$F(x^{jM+\ell}) \le V_j + t_{jM+\ell-1}\alpha\nabla F(x^{jM+\ell-1})^T d^{jM+\ell-1} < V_j. \tag{9.10}$$

By (9.7) and (9.8), we have that, for all $j \in \mathbb{N}$,

$$F(x^{jM+1}) \le V_j + t_{jM}\alpha\nabla F(x^{jM})^T d^{jM} < V_j,$$

so (9.10) holds for $\ell = 1$.

Assume, as the inductive hypothesis, that

$$F(x^{jM+\ell'}) \le V_j + t_{jM+\ell'-1}\alpha\nabla F(x^{jM+\ell'-1})^T d^{jM+\ell'-1} < V_j \tag{9.11}$$

for $\ell' = 1,\dots,\ell$. Now, by (9.7) and (9.8) and the definition of V_j, we have that

$$\begin{aligned} F(x^{jM+\ell+1}) &\le \max_{1\le r\le M}\{F(x^{jM+\ell+1-r}\} + t_{jM+\ell}\alpha\nabla F(x^{jM+\ell})^T d^{jM+\ell} \\ &= \max\{F(x^{(j-1)M+\ell+1}),\dots,F(x^{jM+\ell})\} + t_{jM+\ell}\alpha\nabla F(x^{jM+\ell})^T d^{jM+\ell} \\ &\le \max\{V_j, F(x^{jM+1}),\dots,F(x^{jM+\ell})\} + t_{jM+\ell}\alpha\nabla F(x^{jM+\ell})^T d^{jM+\ell}. \end{aligned}$$

But, by the inductive hypothesis,

$$\max\{F(x^{jM+1}),\dots,F(x^{jM+\ell})\} < V_j,$$

so

$$F(x^{jM+\ell+1}) \le V_j + t_{jM+\ell}\alpha\nabla F(x^{jM+\ell})^T d^{jM+\ell} < V_j.$$

Therefore, the inductive proof is complete, and hence (9.10) is proved. Since $\nu(j+1) = jM+\ell$ for some $\ell \in \{1,\dots,M\}$, this implies the desired result. □

From now on, we define

$$K = \{\nu(1)-1, \nu(2)-1, \nu(3)-1, \dots\}, \tag{9.12}$$

where $\{\nu(j)\}$ is the sequence of indices defined in Lemma 9.3. Note that, when $M = 1$, we have that $K = \{0,1,2,\dots\}$. Clearly,

$$\nu(j) < \nu(j+1) \le \nu(j) + 2M - 1$$

for all $j = 1,2,3,\dots$.

Lemma 9.4. *Let K be given by (9.12). Then,* $\lim_{k\in K} t_k Q_k(d^k) = 0$.

Proof. By (9.9), since F is continuous and bounded below,

$$\lim_{k\in K} t_k \nabla F(x^k)^T d^k = 0. \tag{9.13}$$

But

$$0 > Q_k(d^k) = \frac{1}{2}\sigma_k^{SPG}\|d^k\|_2^2 + \nabla F(x^k)^T d^k \ge \nabla F(x^k)^T d^k \text{ for all } k \in \mathbb{N}.$$

Hence, by (9.13), the desired result is proved. □

Lemma 9.5. *Assume that $K_1 \subseteq_\infty \mathbb{N}$ is a sequence of indices such that*

$$\lim_{k\in K_1} x^k = x^* \in \Omega \text{ and } \lim_{k\in K_1} Q_k(d^k) = 0.$$

Then, x^* is stationary.

Proof. Let $K_2 \subseteq_\infty K_1$ be such that

$$\lim_{k\in K_2} \sigma_k^{SPG} = \sigma > 0.$$

We define

$$Q(d) = \frac{1}{2}\sigma ||d||_2^2 + \nabla F(x^*)^T d \text{ for all } d \in \mathbb{R}^n.$$

Suppose that there exists $\hat{d} \in \mathbb{R}^n$ such that $x^* + \hat{d} \in \Omega$ and

$$Q(\hat{d}) < 0. \tag{9.14}$$

Define

$$\hat{d}^k = x^* + \hat{d} - x^k \text{ for all } k \in K_2.$$

Clearly, $x^k + \hat{d}^k \in \Omega$ for all $k \in K_2$. By continuity, since $\lim_{k\in K_2} x^k = x^*$, we have that

$$\lim_{k\in K_2} Q_k(\hat{d}^k) = Q(\hat{d}) < 0. \tag{9.15}$$

But, by the definition of d^k, we have that $Q_k(d^k) \le Q_k(\hat{d}^k)$. Therefore, by (9.15), $Q_k(d^k) \le Q(\hat{d})/2 < 0$ for $k \in K_2$ large enough. This contradicts the fact that $\lim_{k\in K_2} Q_k(d^k) = 0$. The contradiction came from the assumption that $\hat{d}$ with the property (9.14) exists. Therefore, $Q(d) \ge 0$ for all $d \in \mathbb{R}^n$ such that $x^* + d \in \Omega$. Thus, $\nabla F(x^*)^T d \ge 0$ for all $d \in \mathbb{R}^n$ such that $x^* + d \in \Omega$. So, x^* is stationary. □

Lemma 9.6. *$\{d^k\}_{k\in\mathbb{N}}$ is bounded.*

Proof. For all $k \in \mathbb{N}$,

$$Q_k(d^k) = \frac{1}{2}\sigma_k^{SPG} ||d^k||_2^2 + \nabla F(x^k)^T d^k < 0.$$

Therefore,

$$||d^k||_2^2 < -\frac{2}{\sigma_k^{SPG}} \nabla F(x^k)^T d^k \le \frac{2}{\sigma_{\min}} ||\nabla F(x^k)||_2 ||d^k||_2.$$

Thus,

$$||d^k||_2 \le \frac{2}{\sigma_{\min}} ||\nabla F(x^k)||_2.$$

Since $\{x^k\}_{k\in\mathbb{N}}$ is bounded and F has continuous derivatives, $\{\nabla F(x^k)\}_{k\in\mathbb{N}}$ is bounded. Therefore, the set $\{d^k\}_{k\in\mathbb{N}}$ is bounded. □

Lemma 9.7. *Assume that $K_3 \subseteq_\infty \mathbb{N}$ is a sequence of indices such that*

$$\lim_{k\in K_3} x^k = x^* \in \Omega \text{ and } \lim_{k\in K_3} t_k = 0.$$

Then,

$$\lim_{k\in K_3} Q_k(d^k) = 0, \tag{9.16}$$

and hence x^* is stationary.

Proof. Suppose that (9.16) is not true. Then, for some infinite set of indices $K_4 \underset{\infty}{\subset} K_3$, $Q_k(d^k)$ is bounded away from zero.

On the other hand, since $t_k \to 0$, by the definition of Algorithm 9.1, for $k \in K_4$ large enough, there exists $t_k' \geq t_k$ such that $\lim_{k\in K_4} t_k' = 0$, and (9.7) does not hold for $t = t_k'$. So,

$$F(x^k + t_k' d^k) > \max\{F(x^{k-j+1}) \mid 1 \leq j \leq \min\{k+1, M\}\} + t_k' \alpha \nabla F(x^k)^T d^k,$$

and hence

$$F(x^k + t_k' d^k) > F(x^k) + \alpha t_k' \nabla F(x^k)^T d^k$$

for all $k \in K_4$. Therefore,

$$\frac{F(x^k + t_k' d^k) - F(x^k)}{t_k'} > \alpha \nabla F(x^k)^T d^k$$

for all $k \in K_4$. By the mean-value theorem, there exists $\xi_k \in [0,1]$ such that

$$\nabla F(x^k + \xi_k t_k' d^k)^T d^k > \alpha \nabla F(x^k)^T d_k \tag{9.17}$$

for all $k \in K_4$. Since, by Lemma 9.6, the set $\{d^k\}_{k\in K_4}$ is bounded, there exists a sequence of indices $K_5 \underset{\infty}{\subset} K_4$ such that $\lim_{k\in K_5} d^k = d$ and $\lim_{k\in K_5} \sigma_k^{SPG} = \sigma$ for some $d \in \mathbb{R}^n$ and some $\sigma > 0$. Taking limits for $k \in K_5$ in both sides of (9.17), we obtain $\nabla F(x^*)^T d \geq \alpha \nabla F(x^*)^T d$. This implies that $\nabla F(x^*)^T d \geq 0$. So,

$$\frac{1}{2}\sigma \|d\|_2^2 + \nabla F(x^*)^T d \geq 0.$$

Therefore, since $Q_k(d^k) < 0$ for all k,

$$\lim_{k\in K_5} \frac{1}{2}\sigma_k^{SPG} \|d^k\|_2^2 + \nabla F(x^k)^T d^k = 0.$$

Thus, $\lim_{k\in K_5} Q_k(d^k) = 0$. This contradicts the assumption that $Q_k(d^k)$ is bounded away from zero for $k \in K_4$. Therefore, (9.16) is true. Thus, the hypothesis of Lemma 9.5 holds, with K_3 replacing K_1, and, therefore, by Lemma 9.5, x^* is stationary. □

Theorem 9.1. *Every limit point of $\{x^k\}_{k\in K}$ is stationary.*

Proof. Let $K_6 \underset{\infty}{\subset} K$ be such that $\lim_{k\in K_6} x^k = x^*$. By Lemma 9.4, we have that

$$\lim_{k\in K_6} t_k Q_k(d^k) = 0.$$

Now we have two possibilities: (a) $\lim_{k\in K_6} Q_k(d^k) = 0$ or (b) there exists $K_7 \underset{\infty}{\subset} K_6$ such that $Q_k(d^k) \leq c < 0$ for all $k \in K_7$. In case (a), by Lemma 9.5, x^* is stationary. In case (b), $t_k \to 0$ for $k \in K_7$. By Lemma 9.7, this implies that x^* is stationary. □

Theorem 9.1 shows that every limit point of $\{x^k\}_{k\in K}$ is stationary. If $M = 1$, since in this case we have that $K = \mathbb{N}$, this means that every limit point is stationary. Even when $M > 1$, it is also true that every limit point is stationary. See [62] for details.

9.1.2 ▪ SPG and magic steps

As discussed in the case of unconstrained subproblems, the requirement (9.8) allows one to employ a big variety of heuristic procedures, accelerations, and "magic" ideas that may improve the behavior of SPG. In many cases, the efficiency of an algorithm depends on such heuristics, although its global convergence is guaranteed by an underlying algorithm like SPG. Assuming that one believes in a standard heuristic procedure, we may formalize its interlacing with SPG in the following way.

Algorithm 9.2. SPG with magic steps.
Let $\theta_{\text{progress}} \in [0, 1)$ and the parameters $\alpha \in (0, 1)$ and $0 < \sigma_{\min} < \sigma_{\max}$ that are necessary to execute SPG. Let $x^0 \in \Omega$ be an arbitrary initial point. Set $\text{ItType}_0 = \text{MAGIC}$. Given $x^k \in \Omega$ and the iteration type ItType_k, in order to compute x^{k+1}, proceed as follows:

Step 1. If $\text{ItType}_k = \text{MAGIC}$, execute Steps 1.1–1.3 below. Otherwise, go to Step 2.

Step 1.1. Compute $y \in \Omega$ by means of a heuristic procedure.

Step 1.2. If $F(y) > F(x^k)$, discard y and go to Step 2.

Step 1.3. Set $x^{k+1} = y$. If

$$\|P_\Omega(x^{k+1} - \nabla F(x^{k+1})) - x^{k+1}\| \leq \theta_{\text{progress}} \|P_\Omega(x^k - \nabla F(x^k)) - x^k\|, \tag{9.18}$$

set $\text{ItType}_{k+1} = \text{MAGIC}$. Otherwise, set $\text{ItType}_{k+1} = \text{SPG}$. In any case, finish the kth iteration.

Step 2. Compute x^{k+1} using SPG (with $M = 1$) and set $\text{ItType}_{k+1} = \text{MAGIC}$.

If in employing Algorithm 9.2, infinitely many SPG iterations are performed, the SPG convergence theory guarantees that every limit point of the corresponding subsequence is stationary. The other possibility is that, for all k large enough, the inequality (9.18) holds. In this case, by the continuity of the projection, every limit point is stationary. Algorithm 9.2 also defines a watchdog strategy in the sense of [79] and can be employed as an alternative to the Grippo, Lampariello, and Lucidi [132] nonmonotone strategy in the implementation of SPG. For example, SPG may be executed using $M = 1$ but establishing that the heuristic y is also obtained by means of some SPG iterations without requiring descent.

9.2 ▪ Active set methods

In this section, we consider the case in which the lower-set Ω is a box, given by

$$\Omega = \{x \in \mathbb{R}^n \mid \ell \leq x \leq u\}.$$

The vectors ℓ and u in $\mathbb{R}^n$, with $\ell < u$, are the lower and upper bounds of Ω, respectively.

It will be useful to consider that Ω is the union of disjoint *faces*. Given a set of indices $I \subseteq \{1,\ldots,2n\}$, we denote by $\mathscr{F}_I$ the set of points $x \in \Omega$ such that

1. $x_i = \ell_i$ if $i \in I$,
2. $x_i = u_i$ if $n+i \in I$,
3. $\ell_i < x_i < u_i$ if $i \notin I$ and $n+i \notin I$.

For example, $\mathscr{F}_\emptyset$ is the topological interior of Ω, $\mathscr{F}_{\{1,\ldots,n\}}$ is the set whose unique element is $(\ell_1,\ldots,\ell_n)$, and so on. Clearly, $\cup\mathscr{F}_I = \Omega$ and, if $I \neq J$, we have that $\mathscr{F}_I \cap \mathscr{F}_J = \emptyset$. A face is also characterized by its *free* variables and *fixed* variables. The free variables at the face $\mathscr{F}_I$ are those variables x_i such that $i \notin I$ and $n+i \notin I$. The variables x_i such that $i \in I$ are said to be *fixed at the lower bound* and the variables x_i such that $n+i \in I$ are said to be *fixed at the upper bound*. Given $\bar{x} \in \Omega$, we also denote by $\mathscr{F}_{I(\bar{x})}$ the face to which $\bar{x}$ belongs, i.e., $I(\bar{x}) = \{i \mid \bar{x}_i = \ell_i\} \cup \{n+i \mid \bar{x}_i = u_i\}$. We denote by $\bar{\mathscr{F}}_I$ the closure of $\mathscr{F}_I$. For example, $\bar{\mathscr{F}}_\emptyset = \Omega$. Since boxes are simple sets, reasonable algorithms for minimizing onto boxes generate points x^k that belong to the box for all k. Therefore, each iterate x^k belongs to one (and only one) face of Ω. Many algorithms are based on the *principles of active constraints* that we state below.

9.2.1 ▪ Strong principle of active constraints

Assume that our goal is to find a global minimizer of the continuous function $F(x)$ subject to $x \in \Omega$. Let $x^0 \in \Omega$ and assume that the sequence $\{x^k\}$, generated starting from x^0, obeys the following axioms:

A1. If x^k is a global minimizer of F on Ω, the sequence *stops* at x^k. Otherwise, the point x^{k+1} satisfies $F(x^{k+1}) < F(x^k)$.

A2. If x^k is not a global minimizer of F on $\mathscr{F}_{I(x^k)}$, the point x^{k+1} is a global minimizer of F on $\mathscr{F}_{I(x^k)}$ or belongs to the boundary of $\mathscr{F}_{I(x^k)}$, i.e., belongs to $\bar{\mathscr{F}}_{I(x^k)} \setminus \mathscr{F}_{I(x^k)}$.

We say that an algorithm whose iterates x^k obey the axioms A1 and A2 follows the *strong principle of active constraints*. The philosophy behind this principle is that a face must be explored until a global minimizer on that face is found or until a point on its boundary is reached (and, in consequence, x^{k+1} belongs to a "boundary" face).

The iterations of an algorithm of this type may be of three types:

Internal iterations: These are the iterations in which x^k is not a global minimizer of F on $\mathscr{F}_{I(x^k)}$ and $x^{k+1} \in \mathscr{F}_{I(x^k)}$ is a global minimizer of F on $\mathscr{F}_{I(x^k)}$.

Boundary iterations: These are the iterations in which x^k is not a global minimizer of F on $\mathscr{F}_{I(x^k)}$ and x^{k+1} belongs to the boundary of $\mathscr{F}_{I(x^k)}$.

Leaving-face iterations: These are the iterations in which x^k *is* a global minimizer of F on $\mathscr{F}_{I(x^k)}$ (and hence on its closure) but, because it is not a global minimizer on Ω, we have that $F(x^{k+1}) < F(x^k)$ and $x^{k+1} \notin \bar{\mathscr{F}}_{I(x^k)}$.

In the following theorem, we prove that, if an algorithm is able to obey the strong principle of active constraints, it finds a global minimizer in a finite number of steps.

Theorem 9.2. *Assume that the sequence $\{x^k\}$ was generated by an algorithm that obeys the strong principle of active constraints. Then, the sequence is finite and there exists k such that x^k is a global minimizer of F onto Ω.*

Proof. Assume that the sequence $\{x^k\}$ has infinitely many elements. Since the number of faces is finite and $F(x^{k+1}) < F(x^k)$ whenever x^k is not a global solution of the problem, the number of internal iterations and the number of leaving-face iterations are finite. This means that there exists k_0 such that, for all $k \geq k_0$, all the iterations are of boundary type. But, at each boundary iteration, the number of free variables strictly decreases. Therefore, there cannot be infinitely many boundary iterations either. Thus, the sequence necessarily stops at some x^k, which must be a global minimizer of F onto Ω. □

9.2.2 ▪ Practical principle of active constraints

The strong principle of active constraints indicates an algorithmic direction but does not generate affordable methods, at least for problems with a large number of variables. The reason is that we are almost never able to meet, in finite time, a global minimizer of F on the face to which x^k belongs.

Nevertheless, the principle of staying in a face while good progress is obtained at every internal iteration is valid. The sensible way to stay in the face to which the iterate x^k belongs comes from evoking the unconstrained problem whose variables are the free variables at the face and to apply an unconstrained minimization iteration starting from x^k. The unconstrained algorithm could converge to a stationary point with respect to the free variables (a point where the derivatives with respect to the free variables vanish) or could hit the boundary of the face, stopping at a face of lower dimension. On the other hand, at each iteration, it will be necessary to decide whether it is worthwhile to continue in the same face (perhaps hitting the boundary) or if it is convenient to abandon it, in order to explore a face with additional free variables. We now give a reasonable criterion for such a decision.

Decision based on the continuous projected gradient

The decision of persisting on a face or changing it may be taken using the components of the *continuous projected gradient*.

Given $x^k \in \Omega$, consider the problem

$$\text{Minimize } \nabla F(x^k)^T(x - x^k) + \frac{1}{2}||x - x^k||_2^2 \text{ subject to } x \in \Omega. \tag{9.19}$$

It is easy to see that this problem is equivalent to

$$\text{Minimize } ||x^k - \nabla F(x^k) - x||_2^2 \text{ subject to } x \in \Omega. \tag{9.20}$$

The solution $\bar{x}$ of (9.20) is, by definition, the projection of $x^k - \nabla F(x^k)$ onto Ω. By direct inspection, we see that this solution is given by

$$\bar{x}_i = \max\left\{\ell_i, \min\left\{x_i^k - \frac{\partial F}{\partial x_i}(x^k), u_i\right\}\right\}, i = 1, \ldots, n.$$

We denote $\bar{x} = P_\Omega(x^k - \nabla F(x^k))$. It is easy to see that $\bar{x}$ is the unique stationary point of (9.19) and that $\bar{x}$ depends continuously on x^k.

On the other hand, if x^k were a KKT point of the problem of minimizing $F(x)$ onto Ω, it would also be a KKT point of (9.19), and vice versa. This fact would be equivalent

to saying that $\bar{x} = x^k$. These considerations lead us to define the continuous projected gradient $g_P(x^k)$ as

$$g_P(x^k) = P_\Omega(x^k - \nabla F(x^k)) - x^k$$

and to define the degree of stationarity of x^k as the norm of this vector.

Moreover, $g_P(x^k)$ may be decomposed in a unique way as

$$g_P(x^k) = g_I(x^k) + \Big[\, g_P(x^k) - g_I(x^k) \Big],$$

where $g_I(x^k)$ belongs to the subspace associated with $\mathscr{F}_{I(x^k)}$ and $g_P(x^k) - g_I(x^k)$ belongs to its orthogonal complement. In other words, $g_I(x^k)$ is identical to $g_P(x^k)$ for the free variables while the remaining coordinates are null. Of course $g_I(x^k)$, which will be called the *internal gradient*, also depends continuously on x^k.

If $g_I(x^k) = 0$, we have that x^k is a stationary point of $F(x)$ restricted to $x \in \mathscr{F}_{I(x^k)}$. In this case, nothing else can be expected from an unconstrained algorithm that uses gradients, and so the sensible recommendation is to abandon the face (unless, of course, $g_P(x^k) = 0$, in which case the problem of minimizing F on the box should be considered solved). The same recommendation should be made if the norm of $g_I(x^k)$ is small with respect to the norm of $g_P(x^k)$. Summing up, in a practical algorithm, the face $\mathscr{F}_I(x^k)$ should be abandoned when

$$\|g_I(x^k)\| \leq \eta \|g_P(x^k)\|,$$

where $\eta \in (0,1)$ is an algorithmic parameter.

9.2.3 ▪ Practical scheme for the active set strategy

In [8] and [51, 52, 19], box-constrained minimization problems are solved by using unconstrained methods within the faces and abandoning the faces, when necessary, using monotone SPG iterations.

Monotone SPG iterations

As described in Section 9.1, an SPG iteration, which requires constants $\alpha \in (0, 1/2)$ and $0 < \sigma_{\min} \ll \sigma_{\max}$ defined independently of k, computes the (scaled) projected gradient direction d^k and the SPG parameter σ_k^{SPG} given by (9.3) and (9.5), respectively. Given the direction d^k, a monotone SPG iteration requires a step t_k satisfying the Armijo criterion

$$F(x^k + t_k d^k) \leq F(x^k) + \alpha t_k \nabla F(x^k)^T d^k. \tag{9.21}$$

Note that it corresponds to satisfying (9.4) with $F_k^{\text{ref}} \equiv F(x^k)$, i.e., with $M = 1$. As already described, the backtracking procedure for computing t_k begins with $t = 1$ and computes a new $t \in [0.1t, 0.5t]$ while (9.21) is not satisfied. When a value of t that satisfies (9.21) is found, we define $t_k = t$ and x^{k+1} such that $F(x^{k+1}) \leq F(x^k + t_k d^k)$.

Lemma 9.8. *The monotone SPG iteration is well defined.*

Proof. See Lemma 9.1. □

Lemma 9.9. *Assume that there exists an infinite sequence of indices $K = \{k_1, k_2, \dots\}$ such that, for all $k \in K$, x^{k+1} is obtained by means of a monotone SPG iteration. Then, every limit point of the sequence $\{x^k\}_{k \in K}$ is stationary for the box-constrained optimization problem.*

Proof. This lemma is a particular case of Theorem 9.1 with $M = 1$. (Note that iterates x^{k_j} with $j > 1$ in the sequence $\{x^k\}_{k \in K}$ can be seen as the result of some magic steps computed after the iterate $x^{k_{j-1}+1}$ that was computed by means of a monotone SPG iteration.) □

Internal iterations

Following the active set strategy, we minimize functions on a box by combining iterations that do not modify the fixed variables of the current iterate (perhaps hitting the boundary of the current face) with monotone SPG iterations that abandon the current face when the corresponding test indicates this decision.

The internal iterations only modify the free variables, in such a way that they can be considered as unconstrained iterations with respect to those variables. The objective function remains to be F but modifying the fixed variables is forbidden. Infinitely many consecutive iterations of such an algorithm should lead to a point at which the internal gradient vanishes. This property is formalized in the following assumption.

Assumption 9.2. *If $x^k, x^{k+1}, \cdots \in \mathscr{F}_I$ is a sequence of infinitely many iterations obtained by an unconstrained internal algorithm, then*

$$\lim_{k\to\infty} \frac{\partial F}{\partial x_j}(x^k) = 0$$

for all the free variables x_j. (This implies that $g_I(x^{k+\ell}) \to 0$ when $\ell \to \infty$.)

The results of Chapter 8 indicate that Assumption 9.2 is easy to verify if we use a reasonable unconstrained algorithm within the faces. There is subtlety in this statement due to the existence of bounds for the free variables. In fact, the unconstrained algorithm may indicate as a new iterate a point that does not fulfill the constraints for the free variables. In this case, one should be able to reject that iterate returning to the interior of the face or be able to hit the boundary providing a decrease of the objective function.

Convergence of the active set strategy

The following algorithm condenses the main characteristics of a suitable active set strategy for the box-constrained minimization problems that must be solved in the Augmented Lagrangian context.

Algorithm 9.3. Active set strategy.
Let $\eta \in (0, 1)$ be the algorithmic parameter that provides the criterion for abandoning the faces. (Typically one takes $\eta = 0.1$ in practical calculations.) Let $x^0 \in \Omega$ be the initial point. If $x^k \in \Omega$ is a typical iterate, the steps for obtaining $x^{k+1} \in \Omega$ or interrupting the execution of the algorithm are the following.

Step 1. If $g_P(x^k) = 0$, stop. (The point x^k is stationary for minimizing F on Ω.)

Step 2. If $||g_I(x^k)|| \le \eta ||g_P(x^k)||$, obtain x^{k+1} using a monotone SPG iteration.

Step 3. If $||g_I(x^k)|| > \eta ||g_P(x^k)||$, obtain $x^{k+1} \in \mathscr{F}_{I(x^k)}$ such that $F(x^{k+1}) < F(x^k)$ using an unconstrained internal algorithm or obtain x^{k+1} in the boundary of $\mathscr{F}_{I(x^k)}$ such that $F(x^{k+1}) < F(x^k)$.

Theorem 9.3. *Let $\{x^k\}$ be generated by Algorithm 9.3. Then, this sequence stops at a stationary point x^k or admits a stationary limit point.*

Proof. Assume that the sequence does not stop and thus generates infinitely many iterations. If infinitely many iterations are of SPG type, the thesis follows from Lemma 9.9.

Assume that only a finite number of iterations are of type SPG. Therefore, for k large enough, x^{k+1} belongs to the same face as x^k or belongs to a face of lower dimension.

Then, there exist k_0 and a face $\mathscr{F}_{I(x^{k_0})}$ such that for all $k \geq k_0$ all the iterates belong to $\mathscr{F}_{I(x^{k_0})}$. By Assumption 9.2, this implies that $g_I(x^k) \to 0$. But, by Step 2, we have that $\|g_I(x^k)\| > \eta \|g_P(x^k)\|$ for all $k \geq k_0$. Therefore, $g_P(x^k) \to 0$. By the continuity of g_P, this implies that $g_P(x)$ vanishes at every limit point. □

Theorem 9.3 has the merit of showing that, ultimately, the active set algorithm finds stationary points. However, it is rather disappointing that the gradient SPG iterations are of overwhelming importance in the proof. It would be better if we were able to show that only a finite number of SPG iterations would be needed at each execution of the algorithm, so that convergence should rest on the properties of the internal algorithm, which, as we showed in Chapter 8, may be quite strong. In Theorem 9.4 below, we will give a sufficient condition to ensure that, from some iteration on, all the iterations are internal.

We say that a stationary point is *dual-degenerate* if there exists $i \in \{1, \ldots, n\}$ such that $i \in I(x^*)$ or $n + i \in I(x^*)$, but

$$\frac{\partial F}{\partial x_i}(x^*) = 0.$$

A *dual-nondegenerate* point is a point that is not dual-degenerate. In other words, at dual-nondegenerate points, only derivatives with respect to free variables can vanish.

Theorem 9.4. *Assume that all the stationary points of the box-constrained problem are dual-nondegenerate. Then, the number of SPG iterations in Algorithm 9.3 is finite.*

Proof. Let K be the set of indices such that x^{k+1} is obtained by means of monotone SPG iterations for all $k \in K$. Suppose that K has infinitely many terms. By Lemma 9.9, there exists $K_1 \underset{\infty}{\subseteq} K$ such that

$$\lim_{k \in K_1} x^k = x^*$$

and x^* is stationary. Without loss of generality, let us assume, by contradiction, that there exist $K_2 \underset{\infty}{\subseteq} K_1$ and $i \in \{1, \ldots, n\}$ such that $x_i^k = \ell_i$ and $x_i^{k+1} > \ell_i$ for all $k \in K_2$. Hence, we have that $x_i^* = \ell_i$. However, since x^* is dual-nondegenerate,

$$\frac{\partial F}{\partial x_i}(x^*) > 0.$$

By the continuity of ∇F, we deduce that

$$\frac{\partial F}{\partial x_i}(x^k) > 0$$

for all $k \in K_2$ large enough. Therefore, for those indices k,

$$[g_P(x^k)]_i = 0.$$

This implies that the constraint $x_i = \ell_i$ could not be abandoned at iteration k. □

9.2.4 ▪ Active set stabilized Newton

Consider again the case in which $\Omega = \{x \in \mathbb{R}^n \mid \ell \leq x \leq u\}$. The active set philosophy described in the previous section allows one to use any unconstrained minimization method for minimizing $L_\rho(x, \lambda, \mu)$ within a particular face. Newton's method is one of these possibilities. Writing, as before, $F(x) = L_\rho(x, \lambda, \mu)$, and, in addition,

$$H(x) = h(x) + \lambda/\rho \text{ and } G(x) = g(x) + \mu/\rho,$$

we have that $F(x)$ has the form (8.26). Moreover, the function that we need to minimize within the current face also has the form (8.26) with different values for the number of (free) variables. If we consider, with some abuse of notation, that $F : \mathbb{R}^n \to \mathbb{R}$, $H : \mathbb{R}^n \to \mathbb{R}^m$, and $G : \mathbb{R}^n \to \mathbb{R}^p$ (even in the case in which x is constrained to some face and hence the number of variables is less than n), then the stabilizing techniques explained in Section 8.5.1 may be applied.

9.3 ▪ Interior stabilized Newton

Unfortunately, there is no unanimous opinion with respect to the best strategy for solving box-constrained subproblems. Interior-point strategies are the main competitors of active set ones. Assume, as before, that we wish to minimize $F(x)$ subject to $\ell \leq x \leq u$, where F is given by (8.26). The interior-point (or barrier) idea consists of considering the barrier function

$$F_\nu(x) = F(x) - \nu\left(\sum_{i=1}^{n} \log(x_i - \ell_i) + \sum_{i=1}^{n} \log(u_i - x_i)\right) \tag{9.22}$$

for a small $\nu > 0$ that tends to zero. The application of Newton's method to the minimization of $F_\nu(x)$ should take into account possible instabilities due not only to big values of ρ but also to small values of ν. As a consequence, the basic Newtonian linear systems give rise to stabilized decoupled systems as in (8.30).

9.4 ▪ Review and summary

The reason the Augmented Lagrangian method is useful for solving large-scale optimization problems is that well-established large-scale methods for solving the subproblems are available. The most usual case is when the nonrelaxable constraints define a box. In this chapter, we described, in a self-contained way, an active set framework based on unconstrained methods and projected gradients for solving the Augmented Lagrangian subproblems. Chapters 8 and 9 may be used as an independent (biased) short course on unconstrained, bound-constrained, and convex-constrained optimization.

9.5 ▪ Further reading

In 1988, Barzilai and Borwein published a new gradient algorithm (the BB method) for minimizing convex quadratics. Given $q : \mathbb{R}^n \to \mathbb{R}$ defined by $q(x) = \frac{1}{2}x^T Ax + b^T x$, where A is symmetric and positive definite, the BB method computes

$$x^{k+1} = x^k - \alpha_k \nabla q(x^k), \tag{9.23}$$

where $\alpha_0 > 0$ is arbitrary and, for all $k = 0, 1, 2, \ldots$,

$$\alpha_{k+1} = \frac{\nabla q(x^k)^T \nabla q(x^k)}{\nabla q(x^k)^T A \nabla q(x^k)}. \tag{9.24}$$

Formula (9.24) used in the BB method for defining the step at iteration $k+1$ was used in the classical Cauchy steepest descent method for defining the step at iteration k [78] and could also be derived from the scaling strategy of Oren [214]. Raydan [224] proved the convergence of the BB method for general strictly convex quadratic functions. The possibility of obtaining superlinear convergence for arbitrary n was discarded by Fletcher [114]. However, the conditions were given for the implementation of the BB method for general unconstrained minimization with the help of a nonmonotone procedure. Raydan [225] defined this method in 1997 using the Grippo–Lampariello–Lucidi strategy [132]. He proved global convergence and exhibited numerical experiments that showed that the method was more efficient than classical CG methods for minimizing general functions. These nice comparative numerical results were possible because although the CG method of Hestenes and Stiefel continued to be the rule of choice for solving many convex quadratic problems, its efficiency was hardly inherited by generalizations for minimizing general functions. Therefore, a wide space existed for variations of the Barzilai–Borwein idea [64]. The SPG method of Birgin, Martínez, and Raydan [60, 61, 62] combines Barzilai–Borwein (spectral) nonmonotone ideas with classical projected gradient strategies [40, 126, 174]. SPG is applicable to convex-constrained problems in which the projection onto the feasible set is easy to compute. Since its appearance, the method has been intensively used in applications, including optics [7, 26, 45, 46, 80, 89, 208, 209, 222, 252, 253, 254], support vector machines [86, 92, 238], optimal control [47], topology optimization [246], compressive sensing [36, 37, 110, 179], geophysics [31, 41, 87, 95, 260], image restoration [35, 68, 135], and atmospheric sciences [156, 207]. Moreover, it has been the object of several spectral-parameter modifications, alternative nonmonotone strategies have been suggested, convergence and stability properties have been elucidated, and it has been combined with other algorithms for different optimization problems.

Fletcher [115] introduced a limited memory steepest descent method that generalizes the spectral gradient method by using a few additional vectors of storage and obviously can be extended to convex-constrained minimization. On the CG box-constrained side, the CG active set method of Hager and Zhang [139] seems to be the best known alternative. Spectral residual methods that extend spectral gradient ideas to nonlinear systems of equations were introduced by La Cruz, Martínez, and Raydan [171, 172].

9.6 ▪ Problems

9.1 In the discussion of SPG, we observed that a theorem exists that says every limit point satisfies the optimality condition. For simplicity, we included a shorter theorem that guarantees that if limit points exist, at least one of them satisfies the optimality condition. Do you think that the results are equally relevant from the practical point of view? Give arguments supporting positive and negative answers to this question.

9.2 Define variations of the algorithms presented in this chapter in which the initial step size at the current iteration depends on the successful step size at the previous one. Furthermore, try to exploit "regularities" observed at different iterations regarding step size acceptance.

9.3 Consider the following alternative to Algorithm 9.3. At each ordinary iteration, we define as free variables those variables that verify $\ell_i < x_i^k < u_i$ (as always) plus those that verify $x_i = \ell_i$ with $\partial F/\partial x_i < 0$ and those that verify $x_i = u_i$ with $\partial F/\partial x_i > 0$. We compute a descent direction d^k for F in the face defined by

these free variables. For all i such that $x_i^k = \ell_i$ with $d_i^k < 0$ and for all i such that $x_i^k = u_i$ with $d_i^k > 0$, we redefine $d_i^k = 0$. Prove that d^k is a descent direction. We redefine d^k again, taking the maximal step such that $x^k + t d^k$ is feasible. Prove that $t > 0$. If the descent direction computed so far is smaller than a fixed small multiple of the projected gradient, we discard this direction and we proceed to a projected gradient iteration. Otherwise, we execute a sufficient descent line search. Prove convergence, suggest different choices for d^k, and write a code.

9.4 In the case of minimization with box constraints, consider the stopping criterion in which one finishes the execution when no progress is obtained along coordinate variations with small relative tolerances. Analyze this stopping criterion in connection with subproblems of Augmented Lagrangian methods.

9.5 Generalize the active set strategies to the case of general linear (equality and inequality) constraints. Moreover, extend the generalization to general nonlinear constraints, pointing out the difficulties of implementation.

9.6 Try different strategies for the choice of steplength in projected gradient methods, including random choices. Compare with the one-dimensional exact minimization strategy in the case of convex quadratics. Develop an "artificial intelligence" strategy that chooses the steplength according to the performance of different strategies at previous iterations.

9.7 Analyze the strategy that consists of leaving the current face following gradient components orthogonal to the current face (called "chopped gradient" by Friedlander and Martínez [121] and "proportional gradient" by Dostál [101]). Show that, in the case of a convex quadratic objective function, it is possible to define a leaving criterion that guarantees returning to the current face a bounded number of times, which leads to complexity conclusions.

9.8 Write subroutines implementing the active set stabilized Newton method and the interior stabilized Newton method and compare.

9.9 Box-constrained optimization problems can be reformulated as unconstrained optimization problems by means of a nonlinear change of variables. To fix ideas, consider the change of variables $x_i = y_i^2$ to eliminate a constraint of the form $x_i \geq 0$. The inconvenience of this approach is that derivatives with respect to null variables are null, and so algorithms tend to stay artificially close to the boundary. A possible remedy for this drawback is to consider second derivatives information [9]. Discuss.

Chapter 10
First Approach to Algencan

Algencan is a Fortran subroutine for solving constrained optimization problems using the Augmented Lagrangian techniques described in this book. Understanding the Augmented Lagrangian principles will help you to make a good use of Algencan. (Analogous statements are valid for every numerical software.) Algencan involves many parameters with crucial influence in the algorithmic behavior. Most of them assume predetermined default values and you do not need to think about them in a first approach. Eventually, we will explain how to modify these parameters too.

Trained users should be able to obtain much better results using constrained optimization software than will untrained users. The software is your car, but you are the driver. A good use of Algencan usually overtakes a naive use to several orders of magnitude. Clever exploitation of the software capabilities may transform failures into remarkable successes.

Correct choices of parameters make a big difference. It is common that users change initial points after a presumed failure but it is less frequent to change algorithmic parameters when, for example, the local minimizer obtained is not satisfactory for practical purposes. We strongly encourage algorithmic parameter variations with at least the same intensity that we encourage changing initial approximations. Families of problems (perhaps a family to which your problem belongs) may be satisfactorily solved employing some set of algorithmic parameters that could not be adequate for other families.

Throughout this chapter, you will see that, to use Algencan, you need to code subroutines that compute functions and derivatives. In fact, you can avoid writing codes for computing derivatives (in which case Algencan will employ finite differences) but this is not recommended in terms of efficiency.[1]

10.1 ▪ Problem definition

In this chapter the notation is slightly different from that used in previous chapters. In the theoretical chapters we employed a notation that favors understanding mathematical definitions and proofs, whereas here the notation is more compatible with existing codes and subroutines. For example, instead of using $h(x) = 0$ and $g(x) \leq 0$ for equality and inequality constraints, respectively, we use $c_j(x)$ with $j \in E$ for functions that define equality constraints and $c_j(x)$ with $j \in I$ for inequality constraints. Equality constraints are assumed to be of the form $c_j(x) = 0$, while inequality constraints are assumed to be

[1] Algencan enjoys the benefits of first- and second-order derivatives computed by automatic differentiation tools through its AMPL and CUTEst interfaces.

of the form $c_j(x) \leq 0$. An inequality constraint of the form $c_j^\ell \leq c_j(x) \leq c_j^u$ must be rewritten as (a) $c_j(x) - c_j^u \leq 0$ and $c_j^\ell - c_j(x) \leq 0$, or (b) $c_j(x) - s = 0$ and $c_j^\ell \leq s \leq c_j^u$, where s is a new auxiliary variable. The general form of the problems tackled by Algencan is then given by

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & c_j(x) = 0, \; j \in E, \\ & c_j(x) \leq 0, \; j \in I, \\ & \ell \leq x \leq u, \end{array} \tag{10.1}$$

where the sets of indices E and I are such that $E \cap I = \emptyset$, $E \cup I = \{1, 2, \ldots, m\}$, and f and $c_1, \ldots, c_m$ are real valued functions in the n-dimensional space. The vectors ℓ and u (lower and upper bounds of the variables, respectively) are Algencan parameters, as well as the number of variables n, the number of constraints m, and the sets E and I. Functions f and c_j, $j = 1, \ldots, m$, and, optionally, their first and second derivatives, must be coded by the user. Note that the definition (10.1) involves only the case in which the lower-level set Ω is defined by a box $\ell \leq x \leq u$.

For further reference, we restate the definition of the Lagrangian and Augmented Lagrangian functions given in (4.2) and (4.3), respectively, using the notation that will be adopted from this chapter on. The Lagrangian function can be restated as

$$\mathscr{L}(x, \lambda) = f(x) + \sum_{j \in E \cup I} \lambda_j c_j(x), \tag{10.2}$$

while the Augmented Lagrangian function can be restated as

$$L_\rho(x, \lambda) = f(x) + \sum_{j \in E \cup I} \mathscr{P}_\rho(c_j(x), \lambda_j), \tag{10.3}$$

where

$$\mathscr{P}_\rho(c_j(x), \lambda_j) = \begin{cases} c_j(x)\left(\lambda_j + \frac{1}{2}\rho c_j(x)\right) & \text{if } j \in E \text{ or } \lambda_j + \rho c_j(x) > 0, \\ -\frac{1}{2}\lambda_j^2/\rho & \text{otherwise.} \end{cases}$$

Note that, from now on, Lagrange multipliers will be denoted by $\lambda \in \mathbb{R}^m$, independently of being associated with equality or inequality constraints.

10.2 ▪ Parameters of the subroutine Algencan

We assume that the user is reasonably familiar with Fortran. The subroutine Algencan may be called from any main code or any other subroutine provided (coded) by the user. You can use Algencan to solve a single problem or a sequence of problems that may appear in your application. As a Fortran subroutine, Algencan has a rather large list of parameters. As in the case of any other optimization code, the behavior of Algencan depends on the choice of the parameters and the correct coding of the user-provided subroutines. You must provide all the input parameters, the names and meaning of which will be given below.

The prototype of Algencan is as follows:

```
subroutine algencan(fsub,gsub,hsub,csub,jacsub,hcsub,fcsub,gjacsub, &
     gjacpsub,hlsub,hlpsub,jcnnzmax,hnnzmax,epsfeas,epsopt,efstain, &
     eostain,efacc,eoacc,outputfnm,specfnm,nvparam,vparam,n,x,l,u,  &
     m,lambda,equatn,linear,coded,checkder,f,cnorm,snorm,nlpsupn,   &
     inform)
```

Therefore, the program that calls Algencan must declare the specific type of each parameter (double precision, integer, logical, or character) and must include a statement of the following form:

```
call algencan(myevalf,myevalg,myevalh,myevalc,myevaljac,myevalhc,   &
     myevalfc,myevalgjac,myevalgjacp,myevalhl,myevalhlp,jcnnzmax,   &
     hnnzmax,epsfeas,epsopt,efstain,eostain,efacc,eoacc,outputfnm,  &
     specfnm,nvparam,vparam,n,x,l,u,m,lambda,equatn,linear,coded,   &
     checkder,f,cnorm,snorm,nlpsupn,inform)
```

We usually talk about input and output parameters, independently of the existence of an ontological difference between them in the programming language at hand. (The difference exists in Fortran 90, but it does not exist in Fortran 77.) In the calling sequence, `myevalf`, `myevalg`, `myevalh`, `myevalc`, `myevaljac`, `myevalhc`, `myevalfc`, `myevalgjac`, `myevalgjacp`, `myevalhl`, `myevalhlp`, `jcnnzmax`, `hnnzmax`, `epsfeas`, `epsopt`, `efstain`, `eostain`, `efacc`, `eoacc`, `outputfnm`, `specfnm`, `nvparam`, `vparam`, `n`, `l`, `u`, `m`, `equatn`, `linear`, `coded`, and `checkder` are input parameters. Parameters `f`, `cnorm`, `snorm`, `nlpsupn`, and `inform` are output parameters. The remaining ones, `x` and `lambda`, are, at the same time, input and output parameters.

10.2.1 ▪ Brief description of the input parameters

The subroutines denominated in the calling program as `myevalf`, `myevalg`, `myevalh`, `myevalc`, `myevaljac`, `myevalhc`, `myevalfc`, `myevalgjac`, `myevalgjacp`, `myevalhl`, and `myevalhlp` might be coded by the user to represent the functions that define the problem. The parameters `jcnnzmax`, `hnnzmax`, `epsfeas`, `epsopt`, `efstain`, `eostain`, `efacc`, `eoacc`, `outputfnm`, `specfnm`, `nvparam`, `vparam`, `n`, `l`, `u`, `m`, `equatn`, `linear`, `coded`, and `checkder` are input parameters with different degrees of relevance.

The parameters related to the description of the problem are the number of variables `n`, the number of constraints `m`, the vector of lower bounds `l` and the vector of upper bounds `u`, a logical vector `equatn` that defines which constraints are equalities and which are inequalities, and a logical vector `linear` that says whether each constraint is linear. The logical vector `coded` indicates which functional subroutines were effectively coded by the user. Finally, `checkder` is a logical variable by means of which you may express your desire for checking the correctness of coded derivatives. `jcnnzmax` must store an upper bound on the number of nonnull elements in (more precisely, the number of triplets used to store the sparse representation of) the sparse Jacobian of the constraints. `hnnzmax` must be an upper bound for the sum of the number of (triplets required to represent the) nonnull elements in the lower triangles of the Hessian of the objective function, the Hessians of the constraints, and the matrix $\sum_{j=1}^{m} \nabla c_j(x) \nabla c_j(x)^T$. The meaning of parameter `hnnzmax` may vary depending on the subroutines coded by the user and the method used to solve the Augmented Lagrangian subproblems.

The parameters `epsfeas` and `epsopt` should be small positive numbers used by Algencan to stop the execution declaring feasibility and optimality, respectively. Parameters `efstain` and `eostain` are related to feasibility and optimality tolerances, respectively, and are used by Algencan to stop the execution declaring that an infeasible stationary point of the sum of the squared infeasibilities was found. Their values should depend on whether the user is interested in stopping the execution of Algencan at this type of point. Parameters `efacc` and `eoacc` are feasibility and optimality levels, below which a Newton-based acceleration process is launched.

Parameter `ouputfnm` is a string that may contain the name of an output file. Declaring `outputfnm = ''` in the calling program indicates that no output file is required. `nvparam`, `vparam`, and `specfnm` are parameters related to the setting of additional (or implicit) input parameters of Algencan.

10.2.2 ▪ Tolerances to declare convergence: `epsfeas` and `espsopt`

The input parameters `epsfeas` and `epsopt` are double precision values related to the Algencan main stopping criterion and correspond to the *feasibility tolerance* $\varepsilon_{\text{feas}}$ and to the *optimality tolerance* ε_{opt}, respectively. Roughly speaking, Algencan declares convergence when feasibility has been obtained with tolerance $\varepsilon_{\text{feas}}$ and optimality has been obtained with tolerance ε_{opt}. It is highly recommended that after returning from Algencan, you test your own criteria of feasibility and optimality, since you may not be happy with the ones adopted by Algencan. Below we give a more detailed description of the meaning of $\varepsilon_{\text{feas}}$ and ε_{opt}. You may skip this explanation when first reading this chapter.

Scaling and stopping

Algencan considers a scaled version of problem (10.1) given by

$$\begin{array}{ll} \text{Minimize} & w_f f(x) \\ \text{subject to} & w_{c_j} c_j(x) = 0,\ j \in E, \\ & w_{c_j} c_j(x) \leq 0,\ j \in I, \\ & \ell \leq x \leq u, \end{array} \tag{10.4}$$

where w_f and w_{c_j}, $j = 1, \ldots, m$, are scaling factors such that $0 < w_f \leq 1$ and $0 < w_{c_j} \leq 1$, $j = 1, \ldots, m$. By default, the scaling factors are computed as

$$\begin{aligned} w_f &= 1/\max(1, \|\nabla f(x^0)\|_\infty), \\ w_{c_j} &= 1/\max(1, \|\nabla c_j(x^0)\|_\infty), \quad j = 1, \ldots, m, \end{aligned} \tag{10.5}$$

where x^0 is the initial estimation to the solution given by the user. Instructions on how to modify these scaling factors adopted by Algencan will be given in Section 12.5.

Let $(x^k, \bar{\lambda}^k)$ be an iterate of primal and dual variables and, following (4.7) and (4.8) applied to the scaled problem (10.4), define

$$\lambda_j^{k+1} = \begin{cases} \bar{\lambda}_j^k + \rho_k w_{c_j} c_j(x^k) & \text{if } j \in E \text{ or } \bar{\lambda}_j^k + \rho_k w_{c_j} c_j(x^k) \geq 0, \\ 0 & \text{otherwise} \end{cases}$$

for $j = 1, \ldots, m$. The pair (x^k, λ^{k+1}) is considered an approximate solution to (10.1) when

$$\left\| P_{[\ell,u]}\left(x^k - \left[w_f \nabla f(x^k) + \sum_{j=1}^{m} \lambda_j^{k+1} w_{c_j} \nabla c_j(x^k) \right] \right) - x^k \right\|_\infty \leq \varepsilon_{\text{opt}}, \tag{10.6}$$

$$\max\left\{ \max_{j \in E}\{|w_{c_j} c_j(x^k)|\}, \max_{j \in I}\{|\min\{-w_{c_j} c_j(x^k), \lambda_j^{k+1}\}|\} \right\} \leq \varepsilon_{\text{feas}}, \tag{10.7}$$

and

$$\max\left\{ \max_{j \in E}\{|c_j(x^k)|\}, \max_{j \in I}\{c_j(x^k)_+\} \right\} \leq \varepsilon_{\text{feas}}, \tag{10.8}$$

where in (10.6), $P_{[\ell,u]}$ represents the Euclidean projection operator onto the box $\{x \in \mathbb{R}^n \mid \ell \leq x \leq u\}$. Conditions (10.6), (10.7) say that (x^k, λ^{k+1}) is an approximate stationary pair of the *scaled* problem (10.4), while (10.8) says that x^k also satisfies the required feasibility tolerance for the original *unscaled* problem (10.1).[2] Fulfillment of the stopping criterion (10.6)–(10.8) is reported by Algencan by returning `inform = 0`.

10.2.3 ▪ Tolerances to declare convergence to an infeasible point: `efstain` and `eostain`

In addition to the main stopping criterion described in the previous subsection, there is another stopping criterion related to the convergence to an infeasible point. This stopping criterion considers two additional double precision parameters, named `efstain` and `eostain`, that are associated with the tolerances $\varepsilon_{\text{fstain}}$ and $\varepsilon_{\text{ostain}}$, respectively. Algencan considers that an infeasible stationary point of the infeasibility measure

$$\Phi(x) = \frac{1}{2}\left(\sum_{j\in E} w_{c_j}^2 c_j(x)^2 + \sum_{j\in I} w_{c_j}^2 c_j(x)_+^2\right) \tag{10.9}$$

was found if the current point x^k is such that $\ell \leq x^k \leq u$,

$$\max\left\{\max_{j\in E}\{|w_{c_j} c_j(x^k)|\}, \max_{j\in I}\{[w_{c_j} c_j(x^k)]_+\}\right\} > \varepsilon_{\text{fstain}}, \tag{10.10}$$

and

$$\left\| P_{[\ell,u]}\left(x^k - \left[\sum_{j\in E} w_{c_j}^2 c_j(x^k)\nabla c_j(x^k) + \sum_{j\in I} w_{c_j}^2 c_j(x^k)_+ \nabla c_j(x^k)\right]\right) - x^k \right\|_\infty \leq \varepsilon_{\text{ostain}}. \tag{10.11}$$

Whether this stopping criterion should be enabled or inhibited is not a simple question. On the one hand, Algencan may be able to "visit" an iterate x^k that satisfies (10.10), (10.11) but could abandon this point and finally converge to a point that satisfies the main stopping criterion (10.6)–(10.8) associated with success. On the other hand, the previous described situation may be very time-consuming and painful, and the user may prefer to rapidly stop the optimization process if (10.10), (10.11) is satisfied, perhaps modifying the initial guess or some algorithmic parameter, and starting the optimization process all over again.

Possible values for $\varepsilon_{\text{fstain}}$ and $\varepsilon_{\text{ostain}}$, in order to enable the stopping criterion (10.10), (10.11), would be $\sqrt{\varepsilon_{\text{feas}}}$ and $\varepsilon_{\text{opt}}^{1.5}$, respectively. If an iterate x^k satisfying (10.10), (10.11) is found, Algencan stops with the diagnostic parameter `inform = 1`.

To inhibit the stopping criterion (10.10), (10.11), it would be enough to set $\varepsilon_{\text{ostain}}$ to any negative value and to leave $\varepsilon_{\text{fstain}}$ undefined. In this scenario, conditions (10.10), (10.11) are never satisfied and convergence to an infeasible point may be perceived by the occurrence of a very large value of the penalty parameter or by the exhaustion of the maximum number of iterations of Algencan.

10.2.4 ▪ Thresholds to launch the acceleration process: `efacc` and `eoacc`

The input double precision parameters `efacc` and `eoacc` correspond to feasibility and optimality tolerances $\varepsilon_{\text{facc}}$ and $\varepsilon_{\text{oacc}}$, respectively, embedded in a criterion used to launch

[2] A similar stopping criterion (similar in that it considers a scaled problem but enforces the feasibility tolerance to be satisfied independently of the scaling) is also considered by the interior-points method Ipopt [255].

the so-called acceleration process. As suggested by its name, the acceleration process aims to accelerate the convergence of Algencan. It is automatically activated when some criterion determines that the current point is close enough to a solution. Namely, the main criterion to launch the acceleration process (there are other additional criteria too) is fulfilled when a pair $(x^k, \bar{\lambda}^k)$ satisfies (10.6)–(10.7) with $\varepsilon_{\text{feas}}$ and ε_{opt} replaced by $\max\{\sqrt{\varepsilon_{\text{feas}}}, \varepsilon_{\text{facc}}\}$ and $\max\{\sqrt{\varepsilon_{\text{opt}}}, \varepsilon_{\text{oacc}}\}$, respectively. Starting at the iteration in which this criterion is satisfied, a KKT system of the original *unscaled* problem (10.1) is built and an attempt to solve it by Newton's method is made. On success, the whole optimization method stops. Otherwise, the attempt is ignored and the Augmented Lagrangian execution continues.

Large positive values of `efacc` and `eoacc` should be used to launch the acceleration process at the early stages of Algencan. Null values may be used to launch the acceleration process when Algencan reaches half the required precision, i.e., when (10.6)–(10.7) is satisfied with $\varepsilon_{\text{feas}}$ and ε_{opt} replaced by $\sqrt{\varepsilon_{\text{feas}}}$ and $\sqrt{\varepsilon_{\text{opt}}}$, respectively. To inhibit the usage of the acceleration process, parameters `efacc` and `eoacc` should both be set to any negative value.

Note that the acceleration process requires first-order and second-order derivatives to be provided by the user plus requires the availability of a linear system solver. (Algencan may use the Fortran 95 subroutines MA57, MA86, or MA97 from the HSL Mathematical Software Library [149].) A description of the acceleration process will be given in Section 12.10 and can also be found in [54]. Below, we give a brief description in order to state the stopping criterion that is satisfied when the last iterate of Algencan is found in the acceleration process. You may skip this explanation when first reading this chapter.

Acceleration process and stopping

The acceleration process deals with the only-equalities reformulation of the *unscaled* original problem (10.1) given by

$$\begin{array}{lllll} \text{Minimize} & f(x) & & & \\ \text{subject to} & c_j(x) & = 0, & j \in E, & \\ & c_j(x) + 1/2\, s_j^2 & = 0, & j \in I, & \\ & \ell_i - x_i + 1/2\, (s_i^\ell)^2 & = 0, & i = 1, \ldots, n, & \\ & x_i - u_i + 1/2\, (s_i^u)^2 & = 0, & i = 1, \ldots, n, & \end{array} \tag{10.12}$$

whose KKT system is given by

$$\nabla f(x) + \sum_{j \in E \cup I} \lambda_j \nabla c_j(x) - \lambda^\ell + \lambda^u = 0, \tag{10.13}$$

$$c_j(x) = 0,\ j \in E, \tag{10.14}$$

$$c_j(x) + 1/2\, s_j^2 = 0,\ j \in I, \tag{10.15}$$

$$\ell_i - x_i + 1/2\, (s_i^\ell)^2 = 0,\ i = 1, \ldots, n, \tag{10.16}$$

$$x_i - u_i + 1/2\, (s_i^u)^2 = 0,\ i = 1, \ldots, n, \tag{10.17}$$

$$\lambda_j s_j = 0,\ j \in I, \tag{10.18}$$

$$\lambda_i^\ell s_i^\ell = 0,\ i = 1, \ldots, n, \tag{10.19}$$

$$\lambda_i^u s_i^u = 0,\ i = 1, \ldots, n. \tag{10.20}$$

The acceleration process consists of solving the nonlinear system (10.13)–(10.20) by Newton's method iteratively. Based on simple strategies to identify active bound constraints and nonactive inequality constraints, reduced Newtonian linear systems (with a possible correction of its inertia) are solved at each step.

On success, the acceleration process returns $(\tilde{x}^k, \tilde{\lambda}^k)$ such that $\ell \leq \tilde{x}^k \leq u$, $\tilde{\lambda}^k_j \geq 0$ for all $j \in I$, and

$$\left\| P_{[\ell,u]}\left(\tilde{x}^k - \left[\nabla f(\tilde{x}^k) + \sum_{j=1}^{m} \tilde{\lambda}^k_j \nabla c_j(\tilde{x}^k) \right] \right) - \tilde{x}^k \right\|_\infty \leq \varepsilon_{\text{opt}}, \tag{10.21}$$

$$\max\left\{ \max_{j\in E}\left\{ \left| c_j(\tilde{x}^k) \right| \right\}, \max_{j\in I}\left\{ \left| \min\left\{ -c_j(\tilde{x}^k), \tilde{\lambda}^k_j \right\} \right| \right\} \right\} \leq \varepsilon_{\text{feas}}. \tag{10.22}$$

This means that $(\tilde{x}^k, \tilde{\lambda}^k)$ is an approximate stationary pair of the *unscaled* original problem (10.1). In this case, Algencan stops.

10.2.5 ▪ Output filename

Algencan displays on the screen information related to the parameters being used, the problem being solved, the progress of the optimization process, and some final simple statistics. The level of the detail of the displayed information depends on a couple of parameters that will be described in Section 12.2. A copy of the information displayed on the screen may be sent to an output file if desired. The parameter `outputfnm` is a string that may contain the name of the output file (restricted to a length of 80 characters), as, for example, `myalgencan.out`. If the output file is not desired, `outputfnm` should be set to an empty string, i.e., `outputfnm = ''`.

10.2.6 ▪ Setting of the additional or implicit parameters

A rather large number of additional input parameters have default values set by Algencan. The default value of any of these additional parameters (also called "implicit"), which are not part of the Algencan calling sequence, may be modified with two alternatives: (a) the specification file (whose name is given by parameter `specfnm`) or (b) the array of parameters (driven by parameters `nvparam` and `vparam`).

The (80-character-long) string parameter `specfnm` corresponds to the name of the so-called specification file. In a first run of Algencan, parameter `specfnm` may be set to an empty string, i.e., `specfnm = ''`. The other option for setting Algencan implicit parameters is to use the array of (80-character-long) strings named `vparam`. The integer parameter `nvparam` corresponds, as suggested by its name, to the number of entries of `vparam`. In a first run of Algencan, it would be enough to set `nvparam = 0`.

In either case, the modification of a default value of an implicit parameter is done by passing a string to Algencan. The string must contain a *keyword*, sometimes (but not always) followed by an additional value that may be an integer number, a real number, or a string. For example, Algencan may save the final approximation to the solution

(primal and dual variables) into a file, action that *is not* done by default. In order to save the final approximation to the solution into a file named `mysolution.txt`, the string `SOLUTION-FILENAME mysolution.txt` should be passed to Algencan. This can be done in two different ways:

1. In the calling subroutine, before calling Algencan, set

```
nvparam = 1
vparam(1) = 'SOLUTION-FILENAME mysolution.txt'
```

2. In the calling subroutine, before calling Algencan, set the name of the specification file to, e.g., myalgencan.dat, with the command

```
specfnm = 'myalgencan.dat'
```

   Then, create a text file in the current folder named `myalgencan.dat` and containing a line with the sentence

```
SOLUTION-FILENAME mysolution.txt
```

These methods are equivalent and their usage is described in Section 12.1.

As can be seen in the example above, modifying the default value of an implicit parameter requires the usage of a keyword that may or may not be followed by some value (integer, real, or string), depending on the parameter whose value is being set. For completeness, Table 10.1 lists all possible keywords. If a keyword in the table appears followed by a *D*, an *I*, or an *S*, it requires an extra real, integer, or string value, respectively. If the letter is lowercase, the additional value is optional.

Table 10.1. *Keywords that may be used to set Algencan's additional or implicit parameters.*

Keyword	Additional value
SKIP-ACCELERATION-PROCESS	
LINEAR-SYSTEMS-SOLVER-IN-ACCELERATION-PROCESS	*S*
TRUST-REGIONS-INNER-SOLVER	*s*
LINEAR-SYSTEMS-SOLVER-IN-TRUST-REGIONS	*S*
NEWTON-LINE-SEARCH-INNER-SOLVER	*s*
LINEAR-SYSTEMS-SOLVER-IN-NEWTON-LINE-SEARCH	*S*
TRUNCATED-NEWTON-LINE-SEARCH-INNER-SOLVER	*s*
MATRIX-VECTOR-PRODUCT-IN-TRUNCATED-NEWTON-LS	*S*
FIXED-VARIABLES-REMOVAL-AVOIDED	
ADD-SLACKS	
OBJECTIVE-AND-CONSTRAINTS-SCALING-AVOIDED	
IGNORE-OBJECTIVE-FUNCTION	
ITERATIONS-OUTPUT-DETAIL	*I*
NUMBER-OF-ARRAYS-COMPONENTS-IN-OUTPUT	*I*
SOLUTION-FILENAME	*S*
ACCELERATION-PROCESS-ITERATIONS-LIMIT	*I*
INNER-ITERATIONS-LIMIT	*I*
OUTER-ITERATIONS-LIMIT	*I*
PENALTY-PARAMETER-INITIAL-VALUE	*D*
LARGEST-PENALTY-PARAMETER-ALLOWED	*D*

10.2.7 ▪ Variables, bounds, and constraints

The input parameters `n`, `l`, `u`, `m`, `equatn`, and `linear` are part of the problem description: `n` and `m` are integer variables, `l` and `u` are `n`-dimensional double precision arrays, and `equatn` and `linear` are `m`-dimensional logical arrays. Their meanings follow:

`n`: number of variables n.

`l`: lower bound ℓ (ℓ_i may be equal to $-\infty$ if x_i has no lower bound).

`u`: upper bound u (u_i may be equal to $+\infty$ if x_i has no upper bound).

`m`: number of constraints m.

`equatn`: `equatn(j)` must be true if the jth constraint is an equality and false if it is an inequality.

`linear`: `linear(j)` must be true if $c_j(x)$ is linear and false otherwise.

In the description of the bounds on the variables, you should set `l(i) = -1.0d+20` if $\ell_i = -\infty$ and set `u(i) = 1.0d+20` if $u_i = +\infty$. Any value smaller than -10^{20} for a lower bound or greater than 10^{20} for an upper bound would also be acceptable. These values indicate that the corresponding bound does not exist and that it should be ignored. Any other value within the open interval $(-10^{20}, 10^{20})$ is considered by Algencan as a bound constraint to be satisfied.

10.2.8 ▪ Initial approximation and solution

The parameters `x` (an n-dimensional double precision array) and `lambda` (an m-dimensional double precision array) are input/output parameters. On input, they represent the initial estimation of the primal and dual (Lagrange multipliers) variables. On output, they are the final estimations computed by Algencan. If you do not have reliable initial estimates of the Lagrange multipliers, set `lambda = 0.0d0`.

10.2.9 ▪ Checking derivatives

`checkder` is a logical input parameter that should be used to indicate whether the user would like to check the coded subroutines that compute derivatives against simple finite differences approximations. Since computing derivatives by hand is prone to error, it is recommended to set `checkder = .true.` in the first attempt to solve a problem using Algencan. In this case, previously to the optimization process, each coded subroutine that computes derivatives will be called to compute the derivatives at a random perturbation of the initial guess `x` and its output will be compared against the same derivatives computed by finite differences. Both results (analytic values given by the subroutines coded by the user and finite differences approximations) will be displayed on the screen, relative and absolute errors will be shown, and it will be left to the user's discretion whether derivatives' subroutines coded by the user appear to be delivering the correct result. If derivatives appear to be wrong, the execution should be interrupted and the code corrected, compiled, and rerun. If, after a few iterations of this correction-testing phase, derivatives appear to be correct, `checkder` should be set to false.

Note that calls to the user-provided subroutines made during this checking phase *are* taken into account in the final counting of calls to each user-provided subroutine.

Skipping this derivatives-checking phase may seem to be a time-saving shortcut to zippy users, but in the authors' experience it happens to be a big headache in most cases.

10.2.10 ▪ Subroutines coded by the user

The input parameters `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, `hcsub`, `fcsub`, `gjacsub`, `gjacpsub`, `hlsub`, and `hlpsub` correspond to the names of the subroutines that can

be coded by the user to describe the problem functions and, optionally, their derivatives. They must be declared as external in the calling program or subroutine. The input logical array `coded`, with at least 11 elements, must indicate whether each subroutine was coded. Although redundant, this array aids the robust and appropriate usage of Algencan. Table 10.2 shows the correspondence between the entries of the array `coded` and the subroutines that the user may potentially provide.

Table 10.2. *Correspondence between the entries of array* `coded` *and the potentially user-supplied subroutines.*

`coded`	Name	Subroutine description
`coded(1)`	`fsub`	Objective function
`coded(2)`	`gsub`	Gradient of the objective function
`coded(3)`	`hsub`	Sparse Hessian of the objective function
`coded(4)`	`csub`	Individually computed constraints
`coded(5)`	`jacsub`	Sparse gradient of an individual constraint
`coded(6)`	`hcsub`	Sparse Hessian of an individual constraint
`coded(7)`	`fcsub`	Objective function and all constraints
`coded(8)`	`gjacsub`	Gradient of the objective function and sparse Jacobian of the constraints
`coded(9)`	`gjacpsub`	Gradient of the objective function and product of the Jacobian (or its transpose) by a given vector
`coded(10)`	`hlsub`	Sparse Hessian of the Lagrangian
`coded(11)`	`hlpsub`	Product of the Hessian of the Lagrangian by a given vector

Based on the array `coded` (i.e., based on the subroutines coded by the user to define the problem), different algorithmic options within Algencan may be available. For now, let us say that the only mandatory subroutines are `fsub` and `csub`, to compute the objective function and the constraints, respectively, or alternatively, `fcsub`, to compute both together. If the problem has no constraints (other than the bound constraints), then coding `csub` is not mandatory and coding `fsub` is the natural choice. An adequate choice of the subroutines that should be coded to represent a problem at hand is the subject of Chapter 11.

We consider now the subroutines related to the problem evaluation listed in Table 10.2. The prototypes of the subroutines are as follows:

```
subroutine fsub(n,x,f,flag)
subroutine gsub(n,x,g,flag)
subroutine hsub(n,x,hrow,hcol,hval,hnnz,lim,lmem,flag)
subroutine csub(n,x,ind,cind,flag)
subroutine jacsub(n,x,ind,jcvar,jcval,jcnnz,lim,lmem,flag)
subroutine hcsub(n,x,ind,hcrow,hccol,hcval,hcnnz,lim,lmem,flag)
subroutine fcsub(n,x,f,m,c,flag)
subroutine gjacsub(n,x,g,m,jcfun,jcvar,jcval,jcnnz,lim,lmem,flag)
subroutine gjacpsub(n,x,g,m,p,q,work,gotj,flag)
subroutine hlsub(n,x,m,lambda,sf,sc,hlrow,hlcol,hlval,hlnnz,lim,lmem,flag)
subroutine hlpsub(n,x,m,lambda,sf,sc,p,hp,gothl,flag)
```

Subroutines `fsub`, `gsub`, and `hsub` should compute the objective function $f(x)$, its gradient $\nabla f(x)$, and its Hessian $\nabla^2 f(x)$, respectively. Subroutines `csub`, `jacsub`, and `hcsub` should compute, given a constraint index `ind`, $c_{\text{ind}}(x)$, $\nabla c_{\text{ind}}(x)$, and $\nabla^2 c_{\text{ind}}(x)$,

respectively. Subroutine `fcsub` should compute the objective function $f(x)$ and all constraints $c(x)$, while subroutine `gjacsub` should compute the gradient of the objective function $\nabla f(x)$ and the Jacobian $J(x)$ of the constraints, defined as

$$J(x) = \begin{pmatrix} \nabla c_1(x)^T \\ \vdots \\ \nabla c_m(x)^T \end{pmatrix}. \tag{10.23}$$

Subroutine `gjacpsub` should compute the gradient of the objective function $\nabla f(x)$ and the product of the Jacobian of the constraints $J(x)$, or its transpose, by a given vector, depending on the value of parameter `work`. Finally, subroutine `hlsub` should compute the *scaled* Hessian of the Lagrangian given by

$$\nabla^2 \mathscr{L}(x,\lambda) = w_f \nabla^2 f(x) + \sum_{j \in E \cup I} \lambda_j w_{c_j} \nabla^2 c_j(x), \tag{10.24}$$

and subroutine `hlpsub` should compute the product of the *scaled* Hessian of the Lagrangian by a given vector.

The integer parameter `n` and the double precision n-dimensional array parameter `x` are input parameters for the subroutines, while the integer parameter `flag` is an output parameter in all cases. In the subroutines coded by the user, the output parameter `flag` must be used to indicate whether an exception occurred during the computation. For example, if the objective function calculation (or any other) requires a division to be done and the denominator is null, or if a square root should be taken and the radicand is negative, parameter `flag` must be set to any nonnull value. In this way, on return Algencan will know that the required quantity was not properly computed and a warning message will be shown to the user. Depending on the situation, the optimization process may be interrupted. If everything went well in the computations, parameter `flag` must be set to zero.

The Jacobian and Hessians must be stored in the well-known coordinate scheme (see, for example, [94]). In the coordinate scheme, a sparse matrix A is specified as its set of entries in the form of an unordered set of triplets (a_{ij}, i, j). The set of triplets is held in one double precision array `aval` and two integer arrays `arow` and `acol`. The number of entries of these three arrays should be `annz`. When a Hessian is required, only the lower triangle is required and any computed value above the diagonal will be ignored. For the Jacobian and Hessians, if more than one triplet is present for the same element of the matrix, the sum of the values of the duplicated triplets is considered as the element value. No order is required.

Subroutines that compute sparse matrices (Jacobian and Hessians) have an input integer parameter named `lim` and an output logical parameter named `lmem`. `lim` corresponds to the dimension of the arrays that play the role of `arow`, `acol`, and `aval` and should be used to avoid accessing elements out of the arrays' range. If the arrays' dimension is not enough to save the matrix being computed, parameter `lmem` (meaning "lack of memory") must be set to true. Otherwise, it must be set to false.

Summing up, we have the following:

`fsub` computes `f` $= f(x)$.

`gsub` computes the dense n-dimensional vector `g` $= \nabla f(x)$.

hsub computes the lower triangle of the sparse Hessian $\nabla^2 f(x)$. An element h_{ij} with $i \geq j$ must be stored as `hrow(k)` $= i$, `hcol(k)` $= j$, and `hval(k)` $= h_{ij}$ for some `k` between 1 and `hnnz`.

csub computes `cind` $= c_{\text{ind}}(x)$.

jacsub computes the sparse gradient $\nabla c_{\text{ind}}(x)$. An element $[\nabla c_{\text{ind}}(x)]_i$ must be stored as `jcvar(k)` $= i$ and `jcval(k)` $= [\nabla c_{\text{ind}}(x)]_i$ for some `k` between 1 and `jcnnz`.

hcsub computes the lower triangle of the sparse Hessian $\nabla^2 c_{\text{ind}}(x)$. The storage scheme is identical to the one used by `hsub` but using `hcrow`, `hccol`, `hcval`, and `hcnnz`.

fcsub computes `f` $= f(x)$ and the m-dimensional array `c` such that `c(j)` $= c_j(x)$ for $j = 1, \ldots, m$.

gjacsub computes the gradient of the objective function `g` $= \nabla f(x)$ and the Jacobian of the constraints (10.23). An element $[\nabla c_j(x)]_i$ must be saved as `jcfun(k)` $= j$, `jcvar(k)` $= i$, and `jcval(k)` $= [\nabla c_j(x)]_i$ for some `k` between 1 and `jcnnz`.

gjacpsub computes the gradient of the objective function `g` $= \nabla f(x)$. It also computes the product $q = J(x)^T p$ when `work` is equal to T or t and computes the product $p = J(x)q$ otherwise. Note that q refers to `q` and p refers to `p`; i.e., `p` and `q` play the role of input or output parameters depending on the value of `work`. Moreover, note that, by the role they play, it is clear that `p` is an m-dimensional double precision array, while `q` is an n-dimensional double precision array. `gotj` is a logical input/output parameter that is set to false by the calling subroutine every time subroutine `gjacpsub` is called by the first time with a point `x`. So, assume that your subroutine `gjacpsub` computes the Jacobian matrix (10.23) at `x`, saves it, sets `gotj = .true.`, and computes the required matrix-vector product. If, in a forthcoming call to `gjacpsub`, `gotj` is still true, then you can use the stored Jacobian matrix instead of computing it again. Moreover, in this case, if the gradient of the objective function was also stored, it can be copied into the output parameter `g` without recomputing it.

hlsub computes the lower triangle of the sparse scaled Hessian of the Lagrangian (10.24) with $x =$ `x`, $\lambda =$ `lambda`, $w_f =$ `sf`, and $w_{c_j} =$ `sc(j)` for $j = 1, \ldots, m$. The storage scheme is identical to the one used by `hsub` and `hcsub` but using `hlrow`, `hlcol`, `hlval`, and `hlnnz`.

hlpsub computes `hp` $= \nabla^2 \mathscr{L}(x, \lambda)p$, where $\nabla^2 \mathscr{L}(x, \lambda)$ is the scaled Hessian of the Lagrangian given by (10.24) and $p =$ `p`. `gothl` is a logical input/output parameter that is set to false by the calling subroutine every time subroutine `hlpsub` is called with a new set of parameters `(x,lambda,sf,sc)`. So, assume that your subroutine `hlpsub` computes the Hessian matrix (10.24) and saves it, sets `gothl = .true.`, and computes the required matrix-vector product. If, in a forthcoming call to `hlpsub`, `gothl` is still true, then you can use the stored Hessian matrix instead of computing it again.

Not-coded (empty-body) subroutines

If a subroutine of the Algencan calling sequence (also listed in Table 10.2) is not coded by the user, the corresponding parameter in the calling sequence may point to a dummy subroutine, since it will never be called by Algencan. However, for example, even in the case of setting `coded(7) = .false.`, it would be a conservative choice to code subroutine

fcsub with an empty body as follows:

```
subroutine fcsub(n,x,f,m,c,flag)
  implicit none
  integer, intent(in) :: m,n
  integer, intent(out) :: flag
  real(kind=8), intent(in) :: x(n)
  real(kind=8), intent(out) :: c(m),f
end subroutine fcsub
```

This is because it is not rare for a user to choose some subroutines to code and to set array coded incorrectly. If this is the case, an uncoded subroutine may be called by Algencan. To rapidly detect this case and correct the coding problem, it is recommended to include the statement flag = -1 in all empty-body unused subroutines. Namely,

```
subroutine fcsub(n,x,f,m,c,flag)
  implicit none
  integer, intent(in) :: m,n
  integer, intent(out) :: flag
  real(kind=8), intent(in) :: x(n)
  real(kind=8), intent(out) :: c(m),f
  flag = - 1
end subroutine fcsub
```

10.2.11 ▪ Maximum sizes jcnnzmax and hnnzmax

Integer input parameters jcnnzmax and hnnzmax are used by Algencan to allocate the arrays that save the sparse Jacobian and the (lower triangle of the) "Hessians," respectively. Therefore, the user must set these parameters in the main code with upper bounds on the number of nonnull elements in the Jacobian and the Hessian matrices, respectively.

In most cases, a nonnull element a_{ij} of a matrix A (Jacobian or Hessian) is represented with a single triplet (a_{ij}, i, j), where i is the row index, j is the column index, and a_{ij} is the value of the element. In these cases, the number of triplets that are necessary to describe the whole matrix is equal to the number of nonnull elements. However, in some cases, it is easier to represent an element a_{ij} of a matrix by two triplets (b_1, i, j) and (b_2, i, j) in such a way that $a_{ij} = b_1 + b_2$. Algencan allows the user to proceed in that way and even more than two triplets can be used to represent only one element. However, it must be noted that, in those cases, jcnnzmax and hnnzmax must be upper bounds on the number of *triplets* and not on the number of *nonnull elements* of the corresponding matrices. Strictly speaking jcnnzmax and hnnzmax should be *upper bounds on the number of triplets that one uses to describe the nonnull elements of the corresponding matrices*. However, in order to avoid the pedantism of this definition, we will refer to these parameters, when this does not lead to confusion, as upper bounds on the number of nonzero elements of the corresponding matrices.

Independently of having coded jacsub, gjacsub, or none of them (in which case the Jacobian of the constraints is approximated by finite differences), jcnnzmax should be enough to hold the whole sparse Jacobian of the constraints. The number of nonnull elements in the Jacobian matrix may be difficult to compute when the Jacobian is not being coded. In this case, $n \times m$ is a suitable and conservative upper bound. If this amount of memory is unaffordable, a smaller quantity may be found by trial and error. jcnnzmax may be set to zero if the user codes gjacpsub.

If the user has coded subroutines hsub and hcsub, then hnnzmax should be enough to save, at the same time, the (lower triangle of the) Hessian matrix of the objective function and the (lower triangles of the) m Hessians of the constraints. On the other hand,

if the user coded subroutine `hlsub`, then `hnnzmax` should be enough to save (the lower triangle of) the Hessian of the Lagrangian matrix.

Independently of the coded subroutines (`hsub` and `hcsub`, or `hlsub`), and in addition to the mentioned Hessians, if the method used to solve the Augmented Lagrangian subproblems is the Euclidean trust-region method (see Section 12.8), `hnnzmax` must be such that there exists enough space to save also the lower triangle of the (symmetric) matrix $\sum_{j=1}^{m} \nabla c_j(x) \nabla c_j(x)^T$. If Newton's or the truncated Newton's method is selected to solve the Augmented Lagrangian subproblems, no additional space needs to be considered when setting the value of parameter `hnnzmax`.

An upper bound on the number of (triplets required to represent the) nonnull elements of the lower triangle of matrix $\sum_{j=1}^{m} \nabla c_j(x) \nabla c_j(x)^T$ can be easily computed as $\frac{1}{2} \sum_{j=1}^{m} \texttt{jcnnzmax}_j(\texttt{jcnnzmax}_j + 1)$, where $\texttt{jcnnzmax}_j$ is the number of (triplets used to represent the) nonnull elements of the jth row of the Jacobian (gradient of the jth constraint).

If second-order derivatives were not coded by the user, `hnnzmax` may be set to zero. If the user did code `hlpsub`, `hnnzmax` may be set to zero too.

10.2.12 ▪ Output parameters `nlpsupn`, `snorm`, and `cnorm`

On output, if the final iterate is the result of an Augmented Lagrangian iteration, then Algencan returns $\texttt{x} = x^k$, $\texttt{lambda} = \lambda^{k+1}$, $\texttt{f} = f(x^k)$, `nlpsupn` equal to the left-hand side of (10.6), `snorm` equal to the left-hand side of (10.7), and `cnorm` equal to the left-hand side of (10.8), i.e.,

$$\begin{aligned} \texttt{nlpsupn} &= \left\| P_{[\ell,u]}\left(x^k - \left[w_f \nabla f(x^k) + \sum_{j=1}^{m} \lambda_j^{k+1} w_{c_j} \nabla c_j(x^k) \right] \right) - x^k \right\|_\infty, \\ \texttt{snorm} &= \max\left\{ \max_{j \in E}\{|w_{c_j} c_j(x^k)|\}, \max_{j \in I}\{|\min\{-w_{c_j} c_j(x^k), \lambda_j^{k+1}\}|\} \right\}, \\ \texttt{cnorm} &= \max\left\{ \max_{j \in E}\{|c_j(x^k)|\}, \max_{j \in I}\{c_j(x^k)_+\} \right\}. \end{aligned} \tag{10.25}$$

If the final iterate is the result of a successful acceleration process, Algencan returns $\texttt{x} = \tilde{x}^k$, $\texttt{lambda} = \tilde{\lambda}^k$, $\texttt{f} = f(\tilde{x}^k)$, `nlpsupn` equal to the left-hand side of (10.21), `snorm` equal to the left-hand side of (10.22), and `cnorm` equal to the left-hand side of (10.8) evaluated at $\tilde{x}^k$ and $\tilde{\lambda}^k$, i.e.,

$$\begin{aligned} \texttt{nlpsupn} &= \left\| P_{[\ell,u]}\left(\tilde{x}^k - \left[\nabla f(\tilde{x}^k) + \sum_{j=1}^{m} \tilde{\lambda}_j^k \nabla c_j(\tilde{x}^k) \right] \right) - \tilde{x}^k \right\|_\infty, \\ \texttt{snorm} &= \max\left\{ \max_{j \in E}\{|c_j(\tilde{x}^k)|\}, \max_{j \in I}\{|\min\{-c_j(\tilde{x}^k), \tilde{\lambda}_j^k\}|\} \right\}, \\ \texttt{cnorm} &= \max\left\{ \max_{j \in E}\{|c_j(\tilde{x}^k)|\}, \max_{j \in I}\{c_j(\tilde{x}^k)_+\} \right\}. \end{aligned} \tag{10.26}$$

The arrays `x` and `lambda` must be declared double precision with at least n and m positions, respectively, in the code that calls the subroutine Algencan. The scalars `f`, `nlpsupn`, `cnorm`, and `snorm` must be double precision scalars in the calling code.

10.2.13 ▪ Diagnostic parameter `inform`

The value of the output integer parameter `inform` is equal to `0` when Algencan stops satisfying the stopping criterion (10.6)–(10.8) or the stricter stopping criterion (10.21), (10.22), both related to success.

The returned value `inform` $= 1$ suggests that x^k is infeasible and stationary for the infeasibility measure (10.9) subject to $\ell \leq x \leq u$. Therefore, Algencan returns `inform` equal to 1 when a point $x^k \in [\ell, u]$ that satisfies (10.10), (11.11) is found.

The constant $\rho_{\max}$ is an additional parameter of Algencan whose default value is 10^{20}. If, at iteration k, x^k was computed considering $\rho_k > \rho_{\max}$, Algencan stops returning `inform = 2`. Algencan returns `inform = 3` when x^k was computed and $k \geq k_{\max}$, where $k_{\max}$ is an implicit parameter of Algencan whose default value is 100. Both stopping criteria may also suggest that an infeasible point x^k that is stationary for the infeasibility measure (10.9) subject to $\ell \leq x \leq u$ was found.

The value of $k_{\max}$ may be modified with the keyword OUTER-ITERATIONS-LIMIT (see Section 12.3), while the value of $\rho_{\max}$ may be modified with the keyword LARGEST-PENALTY-PARAMETER-ALLOWED (see Section 12.4).

10.3 ▪ A simple example

Consider the problem of finding the rectangle (centered at the origin) with smallest area within which p circles with radii $r_i = i$ can be packed without overlapping. The problem can be modeled as

$$\begin{array}{lllllll} \text{Minimize} & wh & & & & & \\ \text{subject to} & (c_i^x - c_{i'}^x)^2 + (c_i^y - c_{i'}^y)^2 & \geq & (r_i + r_{i'})^2, & i' > i, & & \\ & -w/2 + r_i & \leq & c_i^x & \leq & w/2 - r_i & \text{for all } i, \\ & -h/2 + r_i & \leq & c_i^y & \leq & h/2 - r_i & \text{for all } i, \\ & w, h \geq 0. & & & & & \end{array} \tag{10.27}$$

The variables of the problem are the centers of the circles (c_i^x, c_i^y), $i = 1, \ldots, p$, the width w, and the height h of the rectangle. The objective function to be minimized is the area of the rectangle. The first set of constraints models the nonoverlapping between the circles, requesting the necessary minimum distance between their centers (distance is squared to preserve differentiability), and the remaining linear constraints say that the circles must be placed within the rectangle.

We will illustrate the usage of Algencan to solve this problem coding subroutines `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, and `hcsub`. Other reasonable choices would have been to code (i) `fcsub`, `gjacsub`, and `hlsub` or (ii) `fcsub`, `gjacpsub`, and `hlpsub`. Our choice of subroutines to be coded is the simplest one and hence the most appropriate for our first example. Objective function, constraints, gradients, and Hessians are all coded in separately. Case (i) would be appropriate if, for some reason, coding all the constraints at once or coding the whole Jacobian of the constraints at once brings some cost savings. The price to be paid is to code the sparse Hessian of the Lagrangian, which requires coding the sum of the sparse Hessians of the constraints plus the Hessian of the objective function. Naive approaches to this task may lead to time-consuming subroutines. Case (ii) would be recommended when the problem is such that computing individual gradients and Hessians, or computing the Jacobian of the constraints or the Hessian of the Lagrangian, is a very time-consuming task but efficient algorithms exist to compute

the product of the Jacobian of the constraints or the Hessian of the Lagrangian by a given vector. An example of this situation will be given in Chapter 11.

We start by coding the definition of the problem and setting the Algencan parameters in a main program named `algencanma`, as shown in Listing 10.1. Let $p > 0$ be the number of circles to be packed. The problem has $n = 2p+2$ variables and $m = p(p-1)/2+4p$ constraints. Variables are given by $x = (c_1^x, c_1^y, c_2^x, c_2^y, \ldots, c_p^x, c_p^y, w, h)^T \in \mathbb{R}^n$. All the constraints are inequalities. The first $p(p-1)/2$ constraints correspond to the nonoverlapping nonlinear constraints. The remaining constraints are linear. (See Listing 10.1.) The other subroutines are self-explanatory and are shown in Listings 10.2–10.7. Subroutines that correspond to `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, and `hcsub` were named `myevalf`, `myevalg`, `myevalh`, `myevalc`, `myevaljac`, and `myevalhc`, respectively, in the present example.

Subroutines `myevalf` and `myevalg` are very simple and require no further explanations. Subroutine `myevalh` is also very simple and its only highlight is that only the lower triangle of the Hessian matrix of the objective function is being computed. The lower triangle of the matrix has a single element $h_{n,n-1} = 1$. In subroutine `myevalc` (as well as in `myevaljac` and `myevalhc`), the first $p(p-1)/2$ constraints correspond to the nonoverlapping nonlinear constraints. The p constraints that follow correspond to $-w/2 + r_i - c_i^x \leq 0$, $i = 1, \ldots, p$. Then, p constraints correspond to $-w/2 + r_i + c_i^x \leq 0$, $i = 1, \ldots, p$. The constraints $-h/2 + r_i - c_i^y \leq 0$, $i = 1, \ldots, p$, follow and, finally, $-h/2 + r_i + c_i^y \leq 0$, $i = 1, \ldots, p$. Subroutines `myevaljac` and `myevalhc` compute the sparse gradient and the lower triangle of the sparse Hessian, respectively, of the `ind`th constraint. In these three subroutines related to the constraints, a subroutine named `pair` is being used. It is assumed that, given $1 \leq k \leq p(p-1)/2$, the subroutine implements a bijection that returns a pair (i, j) satisfying $1 \leq i < j \leq p$. (See Problem 10.1.) The external subroutine `drand` is the random number generator of Schrage [235].

Adding the remaining empty-body subroutines `myevalfc`, `myevalgjac`, `myevalgjacp`, `myevalhl`, and `myevalhlp`, we solved an instance of problem (10.27) with $p = 3$. Figure 10.1 shows Algencan's solution, and Figure 10.2 shows a graphical representation of the solution found.

10.3.1 ▪ Output analysis

We now analyze the output given by Algencan (Figure 10.1). The first lines correspond to a banner that indicates the version of Algencan being used, which is Algencan 3.0.0 in this example. Since Algencan is a live code, this version number is essential when reporting the performance of Algencan. Then, a sentence appears indicating that no additional-parameters default values are being modified through the array of parameters `vparam`, and another sentence indicates that the specification file is not being used either. The following phrase shows the HSL subroutines to deal with linear systems that were compiled together with Algencan and, therefore, are available for use. In this example, subroutines MA57, MA86, and MA97 for solving linear systems are present.

Then comes the so-called preamble, which indicates the values of some parameters being used by Algencan that strongly affect its performance. `firstde` stands for "first derivatives" and its possible values are true and false. In the same way, `seconde` stands for "second derivatives," `truehpr` stands for "true Hessian-vector product" (where by Hessian we mean Hessian of the Augmented Lagrangian function), and their possible values are true and false. In our example, since we code first and second derivatives, all those parameters are true. At this point, we skip all other parameters reported in the preamble, with the exception of `epsfeas`, `epsopt`, `efstain`, `eostain`, `efacc`, `eoacc`,

Listing 10.1. *Main program.*

```
program algencanma
  implicit none
  ! LOCAL SCALARS
  logical :: checkder
  integer :: hnnzmax,i,inform,jcnnzmax,m,n,npairs,nvparam
  real(kind=8) :: cnorm,efacc,efstain,eoacc,eostain,epsfeas,epsopt,f,nlpsupn,seed,snorm
  ! LOCAL ARRAYS
  character(len=80) :: specfnm,outputfnm,vparam(10)
  logical :: coded(11)
  logical, pointer :: equatn(:),linear(:)
  real(kind=8), pointer :: l(:),lambda(:),u(:),x(:)
  ! COMMON SCALARS
  integer :: p
  ! COMMON BLOCKS
  common /pdata/ p
  ! FUCNTIONS
  real(kind=8) :: drand
  ! EXTERNAL SUBROUTINES
  external :: myevalf,myevalg,myevalh,myevalc,myevaljac,myevalhc,myevalfc, &
       myevalgjac,myevalgjacp,myevalhl,myevalhlp
  ! Problem data
  p = 3
  npairs = p * ( p - 1 ) / 2
  ! Number of variables
  n = 2 * p + 2
  ! Set lower bounds, upper bounds, and initial guess
  allocate(x(n),l(n),u(n))
  l(1:2*p) = - 1.0d+20
  u(1:2*p) =   1.0d+20
  l(2*p+1:n) = 0.0d0
  u(2*p+1:n) = 1.0d+20
  seed = 654321.0d0
  do i = 1,2*p
     x(i) = - 5.0d0 + 10.0d0 * drand(seed)
  end do
  x(2*p+1:n) = 10.0d0
  ! Constraints
  m = npairs + 4 * p
  allocate(equatn(m),linear(m),lambda(m))
  equatn(1:m) = .false.
  linear(1:npairs) = .false.
  linear(npairs+1:m) = .true.
  lambda(1:m) = 0.0d0
  ! Coded subroutines
  coded(1:6)  = .true.  ! fsub,gsub,hsub,csub,jacsub,hcsub
  coded(7:11) = .false. ! fcsub,gjacsub,gjacpsub,hlsub,hlpsub
  ! Upper bounds on the number of sparse-matrices non-null elements
  jcnnzmax = 4 * npairs + 2 * ( 4 * p )
  hnnzmax  = 1 + 6 * npairs + 10 * npairs + 3 * ( 4 * p )
  ! Checking derivatives?
  checkder = .false.
  ! Parameters setting
  epsfeas   =   1.0d-08
  epsopt    =   1.0d-08
  efstain   =   1.0d+20
  eostain   = - 1.0d+20
  efacc     = - 1.0d+20
  eoacc     = - 1.0d+20
  outputfnm = ''
  specfnm   = ''
  nvparam   = 0
  ! Optimize
  call algencan(myevalf,myevalg,myevalh,myevalc,myevaljac,myevalhc, &
       myevalfc,myevalgjac,myevalgjacp,myevalhl,myevalhlp,jcnnzmax, &
       hnnzmax,epsfeas,epsopt,efstain,eostain,efacc,eoacc,outputfnm,&
       specfnm,nvparam,vparam,n,x,l,u,m,lambda,equatn,linear,coded, &
       checkder,f,cnorm,snorm,nlpsupn,inform)
  deallocate(x,l,u,lambda,equatn,linear)
  stop
end program algencanma
```

`specfnm` (specification filename), and `outputfnm` (output filename), that display the values that we set in the main program.

After the preamble, Algencan shows the number of variables, equality constraints, inequality constraints, and bound constraints. The displayed figures can be easily checked

Listing 10.2. *Subroutine evalf.*

```
subroutine myevalf(n,x,f,flag)
  implicit none
  ! SCALAR ARGUMENTS
  integer, intent(in) :: n
  integer, intent(out) :: flag
  real(kind=8), intent(out) :: f
  ! ARRAY ARGUMENTS
  real(kind=8), intent(in) :: x(n)
 ! Compute objective function
  flag = 0
  f = x(n-1) * x(n)
end subroutine myevalf
```

Listing 10.3. *Subroutine evalg.*

```
subroutine myevalg(n,x,g,flag)
  implicit none
  ! SCALAR ARGUMENTS
  integer, intent(in) :: n
  integer, intent(out) :: flag
  ! ARRAY ARGUMENTS
  real(kind=8), intent(in) :: x(n)
  real(kind=8), intent(out) :: g(n)
  ! Compute gradient of the objective function
  flag = 0
  g(1:n-2) = 0.0d0
  g(n-1)   = x(n)
  g(n)     = x(n-1)
end subroutine myevalg
```

Listing 10.4. *Subroutine evalh.*

```
subroutine myevalh(n,x,hrow,hcol,hval,hnnz,lim,lmem,flag)
  implicit none
  ! SCALAR ARGUMENTS
  logical, intent(out) :: lmem
  integer, intent(in) :: lim,n
  integer, intent(out) :: flag,hnnz
  ! ARRAY ARGUMENTS
  integer, intent(out) :: hcol(lim),hrow(lim)
  real(kind=8), intent(in)  :: x(n)
  real(kind=8), intent(out) :: hval(lim)
  ! Compute (lower triangle of the) Hessian of the objective function
  flag = 0
  lmem = .false.
  hnnz = 1
  if ( hnnz .gt. lim ) then
     lmem = .true.
     return
  end if
  hrow(1) = n
  hcol(1) = n - 1
  hval(1) = 1.0d0
end subroutine myevalh
```

against the model of problem (10.27) for the particular case $p = 3$. Then, Algencan says that "There are no fixed variables to be removed." In fact, Algencan removes from the problem variables x_i such that $\ell_i = u_i$. This kind of variable should not be part of any problem, but sometimes its existence simplifies the modeling process. To maintain fixed variables in the problem, without a negative impact in the problem resolution, Algencan allows the user to include such artificial variables that are then removed and do not take part in the optimization process. Besides the number of removed fixed variables, the output shows the scaling factors for the objective function and the constraints, automatically computed by Algencan. It is easy to see in the main program `algencanma`

Listing 10.5. *Subroutine evalc.*

```
subroutine myevalc(n,x,ind,c,flag)
  implicit none
  ! SCALAR ARGUMENTS
  integer, intent(in) :: ind,n
  integer, intent(out) :: flag
  real(kind=8), intent(out) :: c
  ! ARRAY ARGUMENTS
  real(kind=8), intent(in) :: x(n)
  ! COMMON SCALARS
  integer :: p
  ! LOCAL SCALARS
  integer :: i,j
  ! COMMON BLOCKS
  common /pdata/ p
  ! Compute ind-th constraint
  flag = 0
  if ( 1 .le. ind .and. ind .le. p * ( p - 1 ) / 2 ) then
     call pair(p,ind,i,j)
     c = ( dble(i) + dble(j) ) ** 2 - &
           ( x(2*i-1) - x(2*j-1) ) ** 2 - ( x(2*i) - x(2*j) ) ** 2
  else if ( ind .le. p * ( p - 1 ) / 2 + p ) then
     i = ind - p * ( p - 1 ) / 2
     c = - 0.5d0 * x(n-1) + dble(i) - x(2*i-1)
  else if ( ind .le. p * ( p - 1 ) / 2 + 2 * p ) then
     i = ind - p * ( p - 1 ) / 2 - p
     c = - 0.5d0 * x(n-1) + dble(i) + x(2*i-1)
  else if ( ind .le. p * ( p - 1 ) / 2 + 3 * p ) then
     i = ind - p * ( p - 1 ) / 2 - 2 * p
     c = - 0.5d0 * x(n) + dble(i) - x(2*i)
  else
     i = ind - p * ( p - 1 ) / 2 - 3 * p
     c = - 0.5d0 * x(n) + dble(i) + x(2*i)
  end if
end subroutine myevalc
```

(Listing 10.1) that at the initial guess x^0, we have $w = h = 10$. This means that $\nabla f(x^0) = (0,\ldots,0,10,10)^T$, $||\nabla f(x^0)||_\infty = 10$, and $w_f = 1/\max(1,||\nabla f(x^0)||_\infty) = 0.1$. The displayed smallest constraints' scale factor corresponds to $\min_{j=1,\ldots,m}\{w_{c_j}\}$. (Its actual value cannot be computed here without showing the values of the first $n-2$ components of the initial guess x^0.)

The output described so far corresponds to a preprocessing phase. Then, the optimization subroutine is effectively called. It starts by displaying the actual number of variables (discarding removed variables) and constraints (equalities plus inequalities) of the problem to be optimized. Considering the default level of detail in Algencan's output (which corresponds to 10 and can be modified with the keyword ITERATIONS-OUTPUT-DETAIL followed by an integer value between 0 and 99, as will be explained in Section 12.2), Algencan prints a single line per iteration, starting with "iteration 0," that corresponds to the initial guess x^0. For each iteration, the line displays (from left to right) (a) the iteration number k, (b) the penalty parameter ρ_k, (c) the (unscaled) objective function value $f(x^k)$, (d) the (unscaled) infeasibility measure `cnorm` as defined in (10.25), (e) the scaled objective function $w_f f(x^k)$, (f) the scaled infeasibility measure given by

$$\max\{\max_{j\in E}\{|w_{c_j} c_j(x^k)|\}, \max_{j\in I}\{[w_{c_j} c_j(x^k)]_+\}\},$$

(g) the scaled infeasibility-complementarity measure `snorm` as defined in (10.25), (h) the sup-norm of the scaled projected gradient of the Lagrangian `nlpsupn` as defined in (10.25), (i) the sup-norm of the projected gradient of the infeasibility measure (10.9) given by the left-hand side of (10.11), and (j) the accumulated total number of inner iterations needed to solve the subproblems. The letter glued to the number of inner iterations corresponds to the termination criterion satisfied by the inner solver. C means convergence, M means

Listing 10.6. *Subroutine evaljac.*

```
subroutine myevaljac(n,x,ind,jcvar,jcval,jcnnz,lim,lmem,flag)
  implicit none
  ! SCALAR ARGUMENTS
  logical, intent(out) :: lmem
  integer, intent(in) :: ind,lim,n
  integer, intent(out) :: flag,jcnnz
  ! ARRAY ARGUMENTS
  integer, intent(out) :: jcvar(lim)
  real(kind=8), intent(in) :: x(n)
  real(kind=8), intent(out) :: jcval(lim)
  ! COMMON SCALARS
  integer :: p
  ! LOCAL SCALARS
  integer :: i,j
  ! COMMON BLOCKS
  common /pdata/ p
  ! Compute gradient of the ind-th constraint
  flag = 0
  lmem = .false.
  if ( 1 .le. ind .and. ind .le. p * ( p - 1 ) / 2 ) then
     call pair(p,ind,i,j)
     jcnnz = 4
     if ( jcnnz .gt. lim ) then
        lmem = .true.
        return
     end if
     jcvar(1) = 2 * i - 1
     jcval(1) = - 2.0d0 * ( x(2*i-1) - x(2*j-1) )
     jcvar(2) = 2 * j - 1
     jcval(2) =   2.0d0 * ( x(2*i-1) - x(2*j-1) )
     jcvar(3) = 2 * i
     jcval(3) = - 2.0d0 * ( x(2*i) - x(2*j) )
     jcvar(4) = 2 * j
     jcval(4) =   2.0d0 * ( x(2*i) - x(2*j) )
  else if ( ind .le. p * ( p - 1 ) / 2 + p ) then
     i = ind - p * ( p - 1 ) / 2
     jcnnz = 2
     if ( jcnnz .gt. lim ) then
        lmem = .true.
        return
     end if
     jcvar(1) = n - 1
     jcval(1) = - 0.5d0
     jcvar(2) = 2 * i - 1
     jcval(2) = - 1.0d0
  else if ( ind .le. p * ( p - 1 ) / 2 + 2 * p ) then
     i = ind - p * ( p - 1 ) / 2 - p
     jcnnz = 2
     if ( jcnnz .gt. lim ) then
        lmem = .true.
        return
     end if
     jcvar(1) = n - 1
     jcval(1) = - 0.5d0
     jcvar(2) = 2 * i - 1
     jcval(2) =   1.0d0
  else if ( ind .le. p * ( p - 1 ) / 2 + 3 * p ) then
     i = ind - p * ( p - 1 ) / 2 - 2 * p
     jcnnz = 2
     if ( jcnnz .gt. lim ) then
        lmem = .true.
        return
     end if
     jcvar(1) = n
     jcval(1) = - 0.5d0
     jcvar(2) = 2 * i
     jcval(2) = - 1.0d0
  else
     i = ind - p * ( p - 1 ) / 2 - 3 * p
     jcnnz = 2
     if ( jcnnz .gt. lim ) then
        lmem = .true.
        return
     end if
     jcvar(1) = n
     jcval(1) = - 0.5d0
     jcvar(2) = 2 * i
     jcval(2) =   1.0d0
  end if
end subroutine myevaljac
```

Listing 10.7. *Subroutine evalhc.*

```
subroutine myevalhc(n,x,ind,hcrow,hccol,hcval,hcnnz,lim,lmem,flag)
  implicit none
  ! SCALAR ARGUMENTS
  logical, intent(out) :: lmem
  integer, intent(in) :: ind,lim,n
  integer, intent(out) :: flag,hcnnz
  ! ARRAY ARGUMENTS
  integer, intent(out) :: hccol(lim),hcrow(lim)
  real(kind=8), intent(in) :: x(n)
  real(kind=8), intent(out) :: hcval(lim)
  ! COMMON SCALARS
  integer :: p
  ! LOCAL SCALARS
  integer :: i,j
  ! COMMON BLOCKS
  common /pdata/ p
  ! Compute lower-triangle of the ind-th constraint's Hessian
  flag = 0
  lmem = .false.
  if ( 1 .le. ind .and. ind .le. p * ( p - 1 ) / 2 ) then
     call pair(p,ind,i,j)
     hcnnz = 6
     if ( hcnnz .gt. lim ) then
        lmem = .true.
        return
     end if
     hcrow(1) = 2 * i - 1
     hccol(1) = 2 * i - 1
     hcval(1) = - 2.0d0
     hcrow(2) = 2 * i
     hccol(2) = 2 * i
     hcval(2) = - 2.0d0
     hcrow(3) = 2 * j - 1
     hccol(3) = 2 * j - 1
     hcval(3) = - 2.0d0
     hcrow(4) = 2 * j
     hccol(4) = 2 * j
     hcval(4) = - 2.0d0
     hcrow(5) = 2 * j - 1
     hccol(5) = 2 * i - 1
     hcval(5) =   2.0d0
     hcrow(6) = 2 * j
     hccol(6) = 2 * i
     hcval(6) =   2.0d0
  else
     hcnnz = 0
  end if
end subroutine myevalhc
```

maximum number of iterations reached, P means lack of progress, and U means that the subproblem seems to be unbounded from below. The last two columns correspond to the number of times the acceleration process was launched and the accumulated number of Newtonian iterations that were used in those trials, respectively. Their exact meaning will be clarified later.

The output shows that, after five outer iterations, the stopping criterion (10.6)–(10.8) is satisfied and Algencan stops. In fact, the sentence "Flag of ALGENCAN: Solution was found." indicates that the stopping criterion (10.6)–(10.8) was satisfied. The total CPU time in seconds is displayed and the total number of calls to each subroutine coded by the user is shown.

10.4 ▪ Installing and running Algencan

This section contains simple instructions for installing Algencan on your computer and running the code for the first time. The instructions below may be outdated when you read this book. Updated instructions may be found in the README file that comes with the Algencan distribution that can be downloaded from the TANGO Project Web

```
================================================================================
This is ALGENCAN 3.0.0.
ALGENCAN, an Augmented Lagrangian method for nonlinear programming, is part of
the TANGO Project: Trustable Algorithms for Nonlinear General Optimization.
See http://www.siam.org/books/fa10 for details.
================================================================================

There are no strings to be processed in the array of parameters.

The specification file is not being used.

Available HSL subroutines = MA57 MA86 MA97

ALGENCAN PARAMETERS:

firstde                =                    T
seconde                =                    T
truehpr                =                    T
hptype in TN           =               TRUEHP
lsslvr in TR           =            MA57/MC64
lsslvr in NW           =            MA57/MC64
lsslvr in ACCPROC      =            MA57/MC64
innslvr                =                   TR
accproc                =                    T
rmfixv                 =                    T
slacks                 =                    F
scale                  =                    T
epsfeas                =           1.0000D-08
epsopt                 =           1.0000D-08
efstain                =           1.0000D+20
eostain                =          -1.0000D+20
efacc                  =          -1.0000D+20
eoacc                  =          -1.0000D+20
iprint                 =                   10
ncomp                  =                    6

Specification filename =                   ''
Output filename        =                   ''
Solution filename      =                   ''

Number of variables               :       8
Number of equality constraints    :       0
Number of inequality constraints  :      15
Number of bound constraints       :       2

There are no fixed variables to be removed.

Objective function scale factor   : 1.0D-01
Smallest constraints scale factor : 7.5D-02

Entry to ALGENCAN.
Number of variables  :       8
Number of constraints:      15

out penalt  objective infeas  scaled    scaled infeas norm   |Grad| inner Newton
ite         function  ibilty  obj-funct infeas +compl graLag infeas totit forKKT
  0         1.000D+02 9.D+00  1.000D+01 1.D+00 1.D+00 1.D+00 1.D+00    0   0   0
  1 8.D+01  8.356D+01 4.D-02  8.356D+00 4.D-02 4.D-02 3.D+00 4.D-02   10C  0   0
  2 8.D+01  5.923D+01 1.D-02  5.923D+00 1.D-02 1.D-02 3.D-07 9.D-03   28C  0   0
  3 8.D+01  5.939D+01 1.D-04  5.939D+00 3.D-05 3.D-05 1.D-10 2.D-05   33C  0   0
  4 8.D+01  5.939D+01 1.D-07  5.939D+00 1.D-07 1.D-07 1.D-11 9.D-08   35C  0   0
  5 8.D+01  5.939D+01 2.D-09  5.939D+00 4.D-10 5.D-10 2.D-14 3.D-10   37C  0   0

Flag of ALGENCAN: Solution was found.

User-provided subroutines calls counters:

Subroutine fsub     (coded=T):      117
Subroutine gsub     (coded=T):       63
Subroutine hsub     (coded=T):       37
Subroutine csub     (coded=T):     1916 (     127 calls per constraint in avg)
Subroutine jacsub   (coded=T):      332 (      22 calls per constraint in avg)
Subroutine hcsub    (coded=T):      184 (      12 calls per constraint in avg)
Subroutine fcsub    (coded=F):        0
Subroutine gjacsub  (coded=F):        0
Subroutine gjacpsub (coded=F):        0
Subroutine hlsub    (coded=F):        0
Subroutine hlpsub   (coded=F):        0


Total CPU time in seconds:     0.00
```

Figure 10.1. *Algencan's output when solving an instance of problem (10.27) with* $p = 3$.

site. Moreover, several Web addresses pointing to third-party software and compilers are provided; these URLs should still be valid at the time you read this book.

In order to run Algencan, calling it from a Fortran code, follow the instructions below. It is assumed that you use a standard Linux environment, that basic commands such as `tar` and `make` are installed, and that `gfortran` (the Fortran compiler from GNU) is also installed. Instructions are divided into two main steps: (i) building the Algencan library and (ii) compiling your own code that calls Algencan. The Algencan library needs to be built only once.

The instructions below are a simple and arbitrary suggestion, since Algencan can also be used in Windows and Mac OS platforms as well and with a variety of Fortran compilers. Algencan can also be used from a C/C++ calling code or in connection with the modeling languages AMPL [119] and CUTEst [131].

10.4.1 ▪ Installing Algencan: Building the library

Step 1: Download the Algencan "tarball" file from the TANGO Project Web site, the link to which can be found at

http://www.siam.org/books/fa10

and save it into the folder where you would like to install Algencan. For example, assume that the name of this folder is `/home/myusername/`.

Step 2: Go to the folder `/home/myusername/` and uncompress the tarball file by typing

```
tar -zxvf algencan-3.0.0.tgz
```

We assume the name of the downloaded file is `algencan-3.0.0.tgz`, which is the name of the file associated with the current version of Algencan. Note that as a consequence of this step, a folder called `algencan-3.0.0` has been created within `/home/myusername/`.

Step 3: Optionally (highly recommended), place the Fortran 95 version of subroutine MA57 from HSL into the folder

`/home/myusername/algencan-3.0.0/sources/hsl/`

that was automatically created in Step 2. Subroutine MA57 must be contained in a file named `hsl_ma57d.f90`. Additionally, files named `hsl_zd11d.f90`, `ma57ad.f`, `mc21ad.f`, `mc34ad.f`, `mc47ad.f`, `mc59ad.f`, `mc64ad.f`, `mc71ad.f`, and `fakemetis.f` from HSL (which correspond to the MA57 dependencies from HSL) and files named `dgemm.f`, `dgemv.f`, `dtpmv.f`, `dtpsv.f`, `idamax.f`, `lsame.f`, and `xerbla.f` from BLAS (which correspond to the MA57 dependencies from BLAS) must be placed within the same folder too. How to obtain those files and other options, related to the usage of subroutines MA86 and MA97 from HSL and the usage of the BLAS and LAPACK libraries, is detailed below.

Step 4: Go to folder `/home/myusername/algencan-3.0.0/` and type

```
make
```

If everything went well, the Algencan's library file `libalgencan.a` has been created within the folder `/home/myusername/algencan-3.0.0/lib/`. If the optional Step 3 has been followed, an HSL-related library file `libhsl.a` has been created (within the same folder) too.

10.4.2 ▪ Compiling your own code that calls Algencan

Step 5: Set an environment variable with the complete path to the Algencan folder. For example, if Algencan is installed at `/home/myusername/`, type

```
export ALGENCAN=/home/myusername/algencan-3.0.0
```

Step 6: Go to the folder where your main program and problem subroutines are. Assume that the file `myprob.f90` contains the source code of the main program and subroutines that are needed to run Algencan. Then, you can obtain the executable file `myprob` by typing

```
gfortran -O3 myprob.f90 -L$ALGENCAN/lib -lalgencan -lhsl -o myprob
```

if you followed Step 3 or by typing

```
gfortran -O3 myprob.f90 -L$ALGENCAN/lib -lalgencan -o myprob
```

if you skipped Step 3.

Step 7: Type

```
./myprob
```

to run and see the output in the screen.

10.4.3 ▪ Installation and compilation remarks

Algencan is an open source software. Therefore, the tarball file downloaded in Step 1 of the instructions above includes all the source files of Algencan, as well as the source files of *all* the examples described in the present and forthcoming chapters. Compiled versions for different platforms are *not* distributed.

The statement in Step 2 uncompresses the tarball file and creates the folders structure of Algencan. This means that in the place where the tarball file is uncompressed, the Algencan main folder, named `algencan-3.0.0` for the current version, is created. Within this main folder there are four files and three subfolders. The files are (a) the license file `license.txt`, (b) the `README` file with a few instructions to compile Algencan as well as the different Algencan's interfaces with AMPL, C/C++, and CUTEst, (c) a file named `WHATSNEW` that describes the main features of each release of Algencan, and (d) the main `Makefile` file that is used in the compilation process. As the README file explains, a few variables with paths to third-party codes may need to be edited within the file `Makefile` in order to compile the Algencan interfaces. The three subfolders are (a) `sources`, where the Algencan source files are, as well as the interfaces' source files, and several examples of usage of Algencan, (b) `bin`, and (c) `lib`. These two last subfolders are empty and receive the interfaces' executable files and the Algencan lib file, respectively, after the respective compilation processes.

Remarks on the possibilities related to the usage of the HSL Mathematical Software Library (in Step 3 of the compilation process) will be given below.

In Step 4, the Algencan library file is created and saved within folder `lib`. In Step 5, an environment variable with the full path to Algencan should be set. This variable is used in Step 6 to indicate to the compiler where the Algencan library is located (using the flag `-L`). Several examples in Fortran 77 and 90 are provided within folders `sources/examples/f77/` and `sources/examples/f90/`, respectively. Steps 6–8 may be repeated several times to test the different provided examples and/or to solve your own problems.

10.4.4 ▪ Usage of HSL subroutines in Algencan

Step 3, which is optional but highly recommended, is the step where the HSL linear algebra subroutines are made available to Algencan. Algencan is ready to solve linear systems using subroutines MA57, MA86, and/or MA97 from HSL. The more the better, since different subroutines may be used in different places of Algencan, depending on the dimension of the linear system that may need to be solved at each time. Moreover, Algencan allows the user to determine which linear system solver should be used each time.

Use of the mentioned HSL subroutines is not mandatory but is highly recommended and has a strong influence on Algencan's performance. This does not mean that the HSL linear algebra subroutines are much more powerful than other linear algebra subroutines included in Algencan. This means that Algencan does not include any linear algebra subroutine and that if the HSL subroutines are not available, only a few basic subalgorithms will be available within Algencan. These subalgorithms, which do not depend on linear algebra subroutines and are matrix-free, may be adequate only for huge-scale problems.

When running Algencan, the output's preamble shows in a dedicated line which linear algebra subroutines are available for solving linear systems. Checking the content of this line is crucial for verifying whether the HSL subroutines are being used by Algencan. If the user places the HSL subroutines in the wrong place, with the wrong filenames, or with missing dependencies, Algencan might not consider them.

Each HSL [149] subroutine (among MA57, MA86, and/or MA97) has its own dependencies from HSL, BLAS [66], and LAPACK [6]. If the BLAS or LAPACK library already exists on your computer, the respective dependencies may be omitted, while the corresponding flags (`-lblas` and/or `-llapack`) must be added to the compilation/linking command in Step 6. Depending on the installation of the libraries, the complete path to them may need to be indicated with the flag `-L` (as is done with the path to the Algencan library). HSL subroutines may be downloaded from http://www.hsl.rl.ac.uk/, and BLAS and LAPACK subroutines may be downloaded from http://www.netlib.org/blas/ and http://www.netlib.org/lapack/, respectively. When this book was written, HSL subroutines were available at no cost for academic research and teaching, while BLAS and LAPACK packages could be freely downloaded. In any case, it is the users' responsibility to check the third-party software licensing terms and conditions.

Each dependency must be placed in a single file. The relation between the subroutine and the filename is immediate and, for completeness, the dependencies of each linear system solver HSL subroutine that may be used by Algencan follow.

Subroutine MA57 should be saved with a file named `hsl_ma57d.f90`. Its dependencies from HSL are `hsl_zd11d.f90`, `ma57ad.f`, `mc21ad.f`, `mc34ad.f`, `mc47ad.f`, `mc59ad.f`, `mc64ad.f`, and `mc71ad.f`. Its dependencies from BLAS are `dgemm.f`, `dgemv.f`, `dtpmv.f`, `dtpsv.f`, `idamax.f`, `lsame.f`, and `xerbla.f`. Subroutine MA57 may be used in connection with Metis [159]. If Metis is not used, file `fakemetis.f` (with the empty subroutine metis_nodend) must be included.

Subroutine MA86 should be saved with a file named `hsl_ma86d.f90`. Its dependencies from HSL are `hsl_mc34d.f90`, `hsl_mc68i.f90`, `hsl_mc69d.f90`, `hsl_mc78i.f90`, `hsl_zb01i.f90`, `mc21ad.f`, `mc64ad.f`, and `mc77ad.f`. Its dependencies from BLAS are `daxpy.f`, `dcopy.f`, `dgemm.f`, `dgemv.f`, `dswap.f`, `dtrsm.f`, `dtrsv.f`, `lsame.f`, and `xerbla.f`. Subroutine MA86 has no dependencies from LAPACK. Subroutine MA86 may be used in connection with Metis. If Metis is not used, file `fakemetis.f` (with the empty subroutine metis_nodend) must be included.

Subroutine MA97 should be saved with a file named `hsl_ma97d.f90`. Its dependencies from HSL are `hsl_mc34d.f90`, `hsl_mc64d.f90`, `hsl_mc68i.f90`,

hsl_mc69d.f90, hsl_mc78i.f90, hsl_mc80d.f90, hsl_zb01i.f90, hsl_zd11d.f90, mc21ad.f, mc30ad.f, mc64ad.f, and mc77ad.f. Its dependencies from BLAS are daxpy.f, ddot.f, dgemm.f, dgemv.f, dnrm2.f, dscal.f, dswap.f, dsyrk.f, dtrmm.f, dtrmv.f, dtrsm.f, dtrsv.f, lsame.f, and xerbla.f. Its dependencies from LAPACK are disnan.f, dlaisnan.f, dpotf2.f, dpotrf.f, ieeeck.f, ilaenv.f, and iparmq.f. As well as subroutine MA57 and MA86, Subroutine MA97 may be used in connection with Metis. If Metis is not used, file fakemetis.f (with the empty subroutine metis_nodend) must be included.

Note that subroutines MA86 and MA97 use OpenMP to be able to run in parallel. Therefore, in order to take advantage of the parallelism provided by them, OpenMP must be installed on your computer. Note that, today, OpenMP and GFortran are both provided by GCC (the GNU Compiler Collection). This means that if you have GCC installed on your computer, then you simply need to add the flag -fopenmp within the file Makefile in the Algencan main folder /home/myusername/algencan-3.0.0/ in order to use OpenMP. This flag should be added if at least one subroutine between MA86 and MA97 is present (since MA57 does not use parallelism). Refer to the OpenMP or HSL documentation to see how to set the desired number of threads. Documentation related to the GCC implementation of OpenMP can be found at http://gcc.gnu.org/onlinedocs/libgomp/.

10.5 ▪ Common questions

1. I do not want to use the automatic scaling because I feel that I have a better one for my problem. How should I proceed? Do I need to code subroutines with scale as well?

   It may be the case that the user considers his or her problem well scaled, or that the user has the ability to redefine the original problem in such a way that it is well scaled. If this is the case, the automatic scaling provided by default by Algencan should be inhibited in order to have $w_f \equiv 1$ and $w_{c_j} \equiv 1$ for all $j \in E \cup I$ in (10.4). This makes the scaled problem (10.4), which is the one solved by Algencan, to coincide with the original problem (10.1). The use of the default scaling factors (10.5) computed by Algencan can be inhibited with the keyword OBJECTIVE-AND-CONSTRAINTS-SCALING-AVOIDED. For details, refer to Section 12.5.

2. Does Algencan use proprietary subroutines? If so, which are the legal procedures that I must follow?

   Algencan may optionally use some linear algebra subroutines from the HSL Mathematical Software Library, namely, MA57, MA86, or MA97, for solving linear systems. Of course, dependencies (from HSL, BLAS, and LAPACK) of those subroutines would also be required. It is the user's responsibility to check the license terms of third-party software. (When this book was written, HSL subroutines were available free of charge and on demand for academic research and teaching.)

3. Can I call Algencan from a program written in a language other than Fortran?

   Yes. Algencan is coded in Fortran but has interfaces to be called from C/C++ programs or to be used in connection with the modeling languages AMPL and CUTEst. The README file that comes with the Algencan distribution includes examples and instruction on how to interface Algencan with the mentioned languages.

4. Which are the recommended Fortran compilers and compilation options?

   Algencan source code is divided into several source files and folders. The Algencan distribution includes a Makefile that should work well in the current common platforms running Unix/Linux, Mac OS, or Windows with MinGW or Cygwin. The current version of Algencan (3.0.0) works well with the current version of GFortran (which is part of GCC version 4.6.3) (see http://gcc.gnu.org/fortran/). Other Fortran compilers were not tested but are expected to work well.

10.6 ▪ Review and summary

We described a Fortran subroutine for minimizing a smooth function with smooth equality and inequality constraints and bounds on the variables. The subroutine is an implementation of Algorithm 4.1 with an approximate KKT condition for the solution of subproblems. The subproblems are solved using the active set strategies with projected gradients described in Chapter 9. A first, very simple example was presented, a few algorithmic parameters were discussed, and the output was described. Presentation of the most adequate way to code a problem and a discussion of the available algorithmic choices were delayed to forthcoming chapters.

10.7 ▪ Further reading

The Fortran 90 implementation of the simple example presented in Section 10.3 can be found in the file `chap10-ex1.f90` within folder `sources/examples/f90/` of the Algencan distribution. In the same folder, file `chap10-simplest-example.f90` presents an additional simpler example, as well as file `toyprob.f90`. The behavior of Algencan in a trivial infeasible problem (described in Section 10.2.3) can be observed in the problem given in file `infeas.f90`, located in the same folder.

10.8 ▪ Problems

10.1 Listing 10.8 shows a simple version of subroutine `pair`, used in the example of Section 10.3. This solution has time complexity $O(p)$. The same goal can be easily achieved with time complexity $O(1)$ by saving the pairs in a $2 \times p(p-1)/2$ matrix in an initialization phase (like within the main program `algencanma` of Listing 10.1) and then accessing this matrix in constant time. However, there exists an $O(1)$ version of subroutine `pair` that requires no initialization and no auxiliary arrays. Code it.

Listing 10.8. *Subroutine pair.*

```
subroutine pair(p,ind,i,j)
  implicit none
  ! SCALAR ARGUMENTS
  integer, intent(in) :: p,ind
  integer, intent(out) :: i,j
  ! LOCAL SCALARS
  integer :: k
  i = 1
  k = ind
  do while ( k .gt. p - i )
     k = k - ( p - i )
     i = i + 1
  end do
  j = i + k
end subroutine pair
```

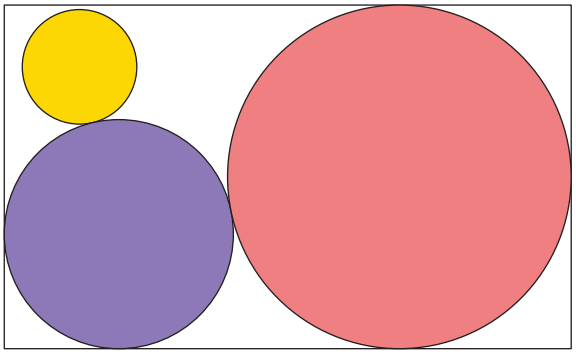

Figure 10.2. *Graphical representation of the solution obtained by Algencan.*

10.2 With the single executable statement `flag = -1`, code subroutines `myevalfc`, `myevalgjac`, `myevalgjacp`, `myevalhl`, and `myevalhlp`. Save all of them in a file named myfirstexample.f, together with the main program `algencanma` and subroutines `myevalf`, `myevalg`, `myevalh`, `myevalc`, `myevaljac`, and `myevalhc`, provided in Listings 10.1–10.7, plus subroutine `pair` from Problem 10.1. Solve problem (10.27) for different values of p. Analyze the output identifying the main elements of the model Algorithm 4.1.

10.3 Add a mistake to any of the subroutines that computes derivatives, set `checkder` equal to true in the main program `algencanma`, and analyze the way Algencan compares coded derivatives with finite differences approximations.

10.4 (a) Code subroutines `fcsub`, `gjacsub`, and `hlsub` for problem (10.27). (b) Code subroutines `fcsub`, `gjacpsub`, and `hlpsub` for problem (10.27). (c) Solve problem (10.27) using Algencan with the provided subroutines and with the subroutines from parts (a) and (b). Compare the results. In particular, check the number of calls to each user-provided subroutine.

10.5 (a) Solve problem (10.27), discarding the subroutines that compute second derivatives. (b) Do the same, discarding subroutines that compute first derivatives. (c) Compare the results.

10.6 In order to analyze the obtained solutions, code your own subroutine to generate a graphical representation of the solution, as depicted in Figure 10.2.

10.7 Consider the infeasible problem of minimizing $x_1 + x_2$ subject to $x_1 + x_2 \leq -2$ and $x_1 + x_2 \geq 2$. To check the behavior of Algencan when dealing with an infeasible problem, solve it twice, inhibiting and enabling the stopping criterion described in Section 10.2.3. Compare the results.

Chapter 11
Adequate Choice of Subroutines

In Chapter 10, a first approach for solving a problem using Algencan was given. Several alternatives for coding a problem were presented, and the simplest one (Choice S1 below) was illustrated with an example. In the present chapter, alternatives for coding a problem will be presented and described. The appropriate choice for each type of problem will be analyzed.

The three choices for coding a problem that we wish to analyze are

S1: `fsub`, `csub`, `gsub`, `jacsub`, `hsub`, and `hcsub`;

S2: `fcsub`, `gjacsub`, and `hlsub`;

S3: `fcsub`, `gjacpsub`, and `hlpsub`.

For the quantities that each subroutine above must compute, see Table 10.2. Other odd combinations (different from Choices S1–S3 above) may also be possible but they will not be discussed here.

11.1 ▪ Separate evaluation of objective function and constraints

Choice S1 appears to be the most simple one. Besides its simplicity, there is another reason for using Choice S1. According to (10.3), the Augmented Lagrangian function associated with problem (10.4) is given by

$$L_\rho(x^k, \bar{\lambda}^k) = w_f f(x^k) + \sum_{j \in E \cup I} \mathscr{P}_\rho(w_{c_j} c_j(x^k), \bar{\lambda}_j^k), \tag{11.1}$$

where

$$\mathscr{P}_\rho(w_{c_j} c_j(x^k), \bar{\lambda}_j^k) = \begin{cases} w_{c_j} c_j(x^k)\left(\bar{\lambda}_j^k + \frac{1}{2}\rho\, w_{c_j} c_j(x^k)\right) & \text{if } j \in E \cup I_k, \\ -\frac{1}{2}(\bar{\lambda}_j^k)^2/\rho & \text{otherwise,} \end{cases}$$

and

$$I_k \equiv I_\rho(x^k, \bar{\lambda}^k) = \left\{ j \in I \mid \bar{\lambda}_j^k + \rho\, w_{c_j} c_j(x^k) > 0 \right\}.$$

Differentiating (11.1) with respect to its first argument, we have that the gradient of the Augmented Lagrangian is given by

$$\nabla L_\rho(x^k, \bar{\lambda}^k) = w_f \nabla f(x^k) + \sum_{j \in E \cup I_k} \left[\bar{\lambda}_j^k + \rho\, w_{c_j} c_j(x^k) \right] w_{c_j} \nabla c_j(x^k) \tag{11.2}$$

and that the Hessian of the Augmented Lagrangian is given by

$$\nabla^2 L_\rho(x^k,\bar\lambda^k) = w_f \nabla^2 f(x^k) + \sum_{j\in E\cup I_k} \left\{ \left[\bar\lambda_j^k + \rho\, w_{c_j} c_j(x^k) \right] w_{c_j} \nabla^2 c_j(x^k) + \rho\, w_{c_j}^2 \nabla c_j(x^k) \nabla c_j(x^k)^T \right\}. \tag{11.3}$$

It is worth noting that (11.2) coincides with the gradient of the Lagrangian function of problem (10.4) evaluated at $(x^k,\hat\lambda^k)$ with

$$\hat\lambda_j^k = \begin{cases} \bar\lambda_j^k + \rho\, w_{c_j} c_j(x^k), & j \in E, \\ \max\left\{0, \bar\lambda_j^k + \rho\, w_{c_j} c_j(x^k)\right\}, & j \in I, \end{cases}$$

i.e., $\nabla L_\rho(x^k,\bar\lambda^k) = \nabla \mathscr{L}(x^k,\hat\lambda^k)$. When the gradient of the Augmented Lagrangian needs to be evaluated at the current point $(x^k,\bar\lambda^k)$, the gradient of the Lagrangian is computed (using the user-supplied subroutines) at $(x^k,\hat\lambda^k)$. Therefore, under Choice S1, subroutine `jacsub` is called to compute only the gradient of constraints j with $j \in E\cup I_k$ (instead of $j \in E\cup I$).

Similarly, the Hessian of the Augmented Lagrangian (11.3) coincides with the Hessian of the Lagrangian (10.24) of problem (10.4) evaluated at $(x^k,\hat\lambda^k)$ plus a first-order term involving the gradients of the constraints $j \in E\cup I_k$, i.e.,

$$\nabla^2 L_\rho(x^k,\bar\lambda^k) = \nabla^2 \mathscr{L}(x^k,\hat\lambda^k) + \sum_{j\in E\cup I_k} \rho\, w_{c_j}^2 \nabla c_j(x^k) \nabla c_j(x^k)^T.$$

Thus, in order to compute $\nabla^2 L_\rho(x^k,\bar\lambda^k)$, user-supplied subroutines `jacsub` and `hcsub` are called to compute, respectively, the gradient and the Hessian of constraints j with $j \in E\cup I_k$ only (instead of $j \in E\cup I$). This could represent a great advantage in the extreme case of a problem with a huge number of inequality constraints that are "sufficiently" strictly satisfied (i.e., "sufficiently" nonactive) at the solution.

Consider the problem given by

$$\begin{array}{ll} \text{Minimize} & x_1 + x_2 \\ \text{subject to} & x_1 - 3x_2 \le 1, \\ & -x_1 + x_2 \le 1, \\ & \{x_1 - (1-r)\cos(\alpha)\}^2 + \{x_2 - (1-r)\sin(\alpha)\}^2 \le r^2,\ \alpha \in [0,\pi/2], \end{array} \tag{11.4}$$

illustrated in Figure 11.1. To model the feasible region (light gray region in the figure), the frontier of the intersection of the unitary-radius small circle with the positive orthant is approximated by a huge number of larger identical circles with radii $r \equiv 5$. As written, the problem has an infinite number of constraints, but any arbitrary huge number can be considered discretizing $\alpha \in [0,\pi/2]$. In Figure 11.1, we considered $\alpha \in \{0°, 1°, \ldots, 90°\}$. Clearly, the solution is given by $x^* = (-2,-1)^T$ with $\lambda_1^* = 1$, $\lambda_2^* = 2$, and $\lambda_j^* = 0$ for all $j \ne 1,2$, and only the two linear constraints are active at the solution.

Starting at $x^0 = (0,0)^T$, Algencan solves the problem using two outer iterations and a total of four inner iterations. Coding the problem with Choice S1, the objective function, its gradient, and its Hessian are evaluated 19, 13, and 4 times, respectively. The two first constraints are evaluated 21 times, while the remaining constraints are evaluated 20 times. The gradient and the (null) Hessian of the first constraint are evaluated 10 and 3 times, respectively, while these figures are 9 and 2, respectively, for the second constraint. For all

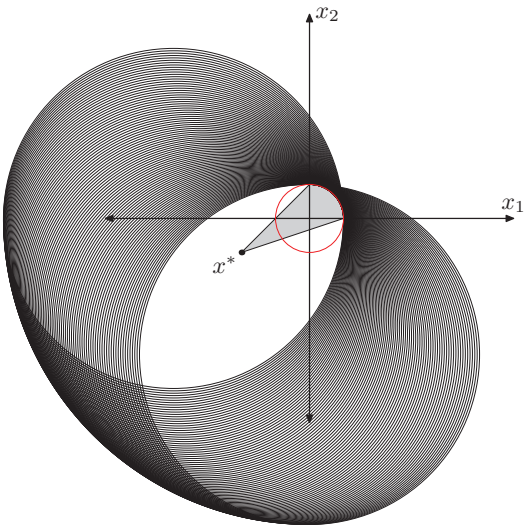

Figure 11.1. *Graphical representation of a simple problem with a potentially huge number of nonactive inequality constraints at the solution.*

the other constraints, the gradient is evaluated only once (in the preoptimization phase that computes the scaling factors) and the Hessian is never evaluated. Coding the problem with Choice S2, the objective function and the constraints are evaluated 22 times, the gradient of the objective function and the Jacobian of the constraints are evaluated 13 times, and the Hessian of the Lagrangian is evaluated 4 times. As suggested, coding constraints (and their derivatives) individually appears to be the most adequate choice in this kind of problem.

The careful reader may have noticed in the paragraph above that, under Choice S1, even the constraints were evaluated a different number of times. (The first two constraints were evaluated 21 times while the remaining constraints were evaluated 20 times.) On the other hand, all explanations included in the present section until now pointed to a possible different number of evaluations of the (first- and second-order) derivatives of the constraints, but not the constraints themselves. Therefore, an additional explanation is in order.

For different reasons, given a point $(\hat{x}, \hat{\lambda})$ at which the Augmented Lagrangian function of (10.4) and its gradient were evaluated, Algencan may need to evaluate the gradient of the Augmented Lagrangian at a point $(\hat{x}^{\text{near}}, \hat{\lambda})$ with $\hat{x}^{\text{near}}$ near $\hat{x}$. The gradient $\nabla L_\rho(\hat{x}^{\text{near}}, \hat{\lambda})$ may be used to compute the initial spectral steplength σ_0^{SPG} in (9.6) or, as suggested in Section 8.5.2 in the context of truncated Newton methods, to approximate by an incremental quotient the product $\nabla^2 L_\rho(\hat{x}, \hat{\lambda})\, d$, i.e.,

$$\nabla^2 L_\rho(\hat{x}, \hat{\lambda})\, d \approx \frac{1}{t}\Big[\nabla L_\rho(\hat{x}^{\text{near}}, \hat{\lambda}) - \nabla L_\rho(\hat{x}, \hat{\lambda})\Big],$$

where $\hat{x}^{\text{near}} = \hat{x} + t\, d$ and $t > 0$ is a small step. In any case, to deal with the discontinuity of the second-order derivatives of the Augmented Lagrangian function, the gradient $\nabla L_\rho(\hat{x}^{\text{near}}, \hat{\lambda})$ *is not* the gradient of L_ρ evaluated at $(\hat{x}^{\text{near}}, \hat{\lambda})$ (there is an abuse of notation

in the present paragraph) but is the gradient of

$$w_f f(x) + \sum_{E \cup I_\rho(\hat{x},\hat{\lambda})} w_{c_j} c_j(x) \left(\lambda_j + \frac{1}{2} \rho \, w_{c_j} c_j(x) \right), \tag{11.5}$$

i.e.,

$$\nabla L_\rho(\hat{x}^{\text{near}}, \hat{\lambda}) \equiv w_f \nabla f(\hat{x}^{\text{near}}) + \sum_{E \cup I_\rho(\hat{x},\hat{\lambda})} \left[\hat{\lambda}_j + \rho \, w_{c_j} c_j(\hat{x}^{\text{near}}) \right] w_{c_j} \nabla c_j(\hat{x}^{\text{near}}). \tag{11.6}$$

We stress that, in (11.6), the summation is in $E \cup I_\rho(\hat{x}, \hat{\lambda})$ instead of $E \cup I_\rho(\hat{x}^{\text{near}}, \hat{\lambda})$ in such a way that the contribution to $\nabla L_\rho(\hat{x}^{\text{near}}, \hat{\lambda})$ is given by exactly the same constraints that contributed to the computation of $\nabla L_\rho(\hat{x}, \hat{\lambda})$. This means that the computation of (11.6) involves computing only a subset of the constraints (and their gradients).

The paragraph above explains why Algencan requires from the user the Jacobian of the constraints or the individual gradients of the constraints, instead of allowing the user to compute the gradient of the Lagrangian (as may be done with the Hessian of the Lagrangian). This is because approximating or computing the gradients of the constraints *individually* is necessary to deal with the discontinuity of the second-order derivatives of the Augmented Lagrangian function.

11.2 ▪ Combined evaluation of objective function and constraints

In Choice S1, no interaction between the functions that define the problem must be considered by the user. By no interaction we mean that, for example, sparse gradients of the constraints are coded individually, as well as Hessians of the constraints. This means that the user does not need to take care of adding the sparse Hessians of the objective function and the constraints to build up the scaled Hessian of the Lagrangian (see (10.24)). In this case, the task of building the sparse scaled Hessian of the Lagrangian is performed by Algencan. Assume that the (lower triangle of the) Hessian of the objective function has $\texttt{hnnz}_0$ elements and that the (lower triangle of the) Hessian of the jth constraint has $\texttt{hnnz}_j$ elements ($j = 1, \ldots, m$), and let $q \equiv \sum_{j=0}^{m} \texttt{hnnz}_j$. Giving priority to the time complexity over the usage of space (memory), by saving all the $m + 1$ sparse Hessians and then computing their sparse weighted sum, Algencan computes, calling the user supplied subroutines `hsub` and `hcsub`, the Hessian of the Lagrangian with time and space complexities $O(q)$. If q is too large, the $O(q)$ space requirement may be prohibited and the intervention of the user may be necessary. In this case, Choice S2 may be preferred, since it allows users themselves to code the Hessian of the Lagrangian within subroutine `hlsub`.

When coding your Hessian of the Lagrangian, you may like to start by computing the Hessian of the objective function and then to compute the Hessians of the constraints one by one and perform the weighted sum of the just computed jth Hessian with the already computed partial weighted sum of the previously computed Hessians. Let q_j be the number of elements of the (lower triangle of the) matrix given by the sum of the (lower triangle of the) Hessian of the objective function and the (lower triangle of the) Hessians of constraints from 1 to $j-1$. In this case, the space complexity would be kept at $O(\max_{j=1,\ldots,m}\{q_j + \texttt{hnnz}_j\})$. Since $q_j \leq \sum_{k=1}^{j-1} \texttt{hnnz}_k$ for all j, we have that $q_j + \texttt{hnnz}_j \leq \sum_{k=1}^{j} \texttt{hnnz}_k$ for all j and, hence, $\max_{j=1,\ldots,m}\{q_j + \texttt{hnnz}_j\} \leq \sum_{k=0}^{m} \texttt{hnnz}_k = q$. The price to be paid for the memory savings is an increase in the time complexity,

given by $O(\sum_{j=0}^{m} q_j + \texttt{hnnz}_j)$, i.e., a time complexity increase of $O(\sum_{j=0}^{m} q_j)$. Of course, complexities may differ if the user is ready to take advantage of known particular Hessian structures. See Problem 11.1.

Another case in which the evaluation of all constraints at once may be preferred is the case in which constraints are given by $c(x) \equiv Ax - b$. If A is a dense matrix saved column-wise (like in Fortran), then computing $Ax - b$ all at once (as the combination of the columns of A minus array b) is more efficient than computing constraints $c_k(x) \equiv a_k^T x - b_k$, where a_k^T is the kth row of A, one by one. The same is true if A is a sparse matrix given by a set of triplets with no specific order.

11.3 ▪ Providing matrix-vector products

Choices S1 and S2 may differ in memory requirements for computing the Hessian of the Lagrangian, but they share a possible bottleneck regarding time and memory requirements: they both imply in potentially computing and saving the whole Jacobian of the constraints. The full Jacobian of the constraints is explicitly computed by the user within subroutine `gjacsub` in Choice S2, while it is internally saved by Algencan with repeated calls to subroutine `jacsub` in Choice S1. If saving the full Jacobian requires a prohibited amount of memory, or if computing the full Jacobian is very time-consuming, while computing its product by a given array is not, Choice S3 must be considered.

Consider a problem with linear constraints of the form $Ax = b$ and let $a_k^T \in \mathbb{R}^n$ be the kth row of matrix $A \in \mathbb{R}^{m \times n}$. Choice S1 requires the computation of each inner product $a_k^T x - b_k$ individually, in order to compute the kth constraint. This is clearly inconvenient when coding the problem in a column-oriented language like Fortran that saves matrices columnwise (in contrast to a language like C that saves matrices rowwise). Therefore, in this case, it would be much more natural to opt for Choice S2 and code the matrix-vector product $Ax - b$ within subroutine `gjacsub`. The same reasoning applies if A is a sparse matrix whose elements are saved in triplets of the form (a_{ij}, i, j) with no specific order. In this case, computing the inner product involved in a single constraint may require visiting all matrix elements or sorting them in a preprocessing stage. (Sorting the matrix elements in place; i.e., without using extra storage, may be a nontrivial task for an ordinary user.) On the other hand, computing the whole matrix-vector product at once is simple and cheap. In any case, we are assuming that matrix A is explicitly stored or at least that its elements are known individually.

Assume now that matrix A is not given explicitly but there is a subroutine that cheaply computes the product of matrix A by a given vector. This is the case in compressive sensing applications (see, for example, [36, 110]), where matrix A represents the product of two matrices named Φ and Ψ^T. The matrix $\Psi \in \mathbb{R}^{n \times n}$ is such that a signal $t \in \mathbb{R}^n$ has a sparse representation in the space generated by the columns of Ψ^T; i.e., there exists $s \in \mathbb{R}^n$ with a few nonnull elements such that $t = \Psi^T s$. Matrix $\Phi \in \mathbb{R}^{m \times n}$ represents the task of performing $m < n$ observations $b \in \mathbb{R}^m$ that are linear combinations of the elements of the n-dimensional signal t, i.e., $b = \Phi t$. The problem consists of finding a vector $s \in \mathbb{R}^n$ with the smallest possible number of nonnull elements such that $\Phi\Psi^T s = b$. In other words, knowing that a signal $t \in \mathbb{R}^n$ is sparse in the space generated by the columns of Ψ^T, the problem consists in recovering it from a reduced number of samples $b \in \mathbb{R}^m$. Matrices Φ and Ψ are implicitly defined sparse matrices such that computing a single matrix-vector product is very cheap, but $A = \Phi\Psi^T$ may be dense and computing it may be very expensive. In this case, S3 is the most adequate choice and the Jacobian-vector product should be coded by the user within subroutine `gjacpsub`.

In compressive sensing, the basis pursuit problem [81] consists of minimizing $\|s\|_1$ subject to $\Phi\Psi^T s = b$. By the change of variables $s = u - v$ with $u \geq 0$ and $v \geq 0$, this problem can be reformulated as the linear programming problem given by

$$\text{Minimize } \sum_{i=1}^{n} u_i + v_i \text{ subject to } \Phi\Psi^T(u-v) = b,\ u \geq 0,\ v \geq 0. \tag{11.7}$$

Consider the "original" signal $t \in \mathbb{R}^n$ with $n = 256 \times 256 = 65{,}536$ pixels (each one representing a gray scale between 0 and 1), depicted in Figure 11.2. Let Ψ be the $n \times n$ (orthogonal) Haar transform matrix (see, for example, [112, Chapter 6]) and let $\Phi \in \mathbb{R}^{m \times n}$ be the (full rank) highly sparse binary permuted block diagonal measurement matrix (with $L = 2$ block diagonal matrices; see [142] for details). The sparse representation of t in the space generated by the columns of Ψ^T is given by $s = \Psi t$ and it has only $k = 6{,}391$ nonnull elements, which corresponds to a density of $k/n \times 100\% \approx 9.75\%$. Consider eight different linear programming problems of the form (11.7) with $m = \alpha k$ and $\alpha \in \{1.5, 2.0, 2.5, \ldots, 5.0\}$. The problems have $2n = 131{,}072$ variables and m equality constraints plus the bound constraints. Starting from the least-squares solution $(u^0, v^0) = ([s^0]_+, [-s^0]_+)$, with $s^0 = \Theta^T(\Theta\Theta^T)^{-1}b$ and $\Theta = \Phi\Psi^T$, and coding the problem with Choice S3, Algencan solves the eight problems using much less time than the time needed to perform the strongly unrecommended task of computing the matrix Θ only once. Figure 11.3 illustrates the solutions found by considering different numbers of measurements m. See [59] for details.

Figure 11.2. *Phantom "original" image with $n = 256 \times 256 = 65{,}536$ pixels.*

As mentioned in Chapter 10, Algencan deals with the scaled version (10.4) of the original problem (10.1). Scaling factors w_f for the objective function and w_{c_j}, $j = 1, \ldots, m$, for the constraints are computed automatically by Algencan following (10.5). This means that, for scaling the constraints, Algencan needs to compute the gradient of each constraint individually. In the basis pursuit problem above, this coincides with the expensive task of computing matrix Θ. When the user chooses S3, this task is performed by (the highly unrecommended task of) doing m independent calls to subroutine `gjacpsub` to obtain the products of the transpose of the Jacobian matrix by the m canonical vectors in $\mathbb{R}^m$. To avoid this very undesired situation, when using Choice S3, you must take care of scaling the problem by yourself and must ask Algencan to skip the automatic scaling of the problem. (To inhibit Algencan's automatic scaling, use the keyword OBJECTIVE-AND-CONSTRAINTS-SCALING-AVOIDED; see Section 12.5 for details.)

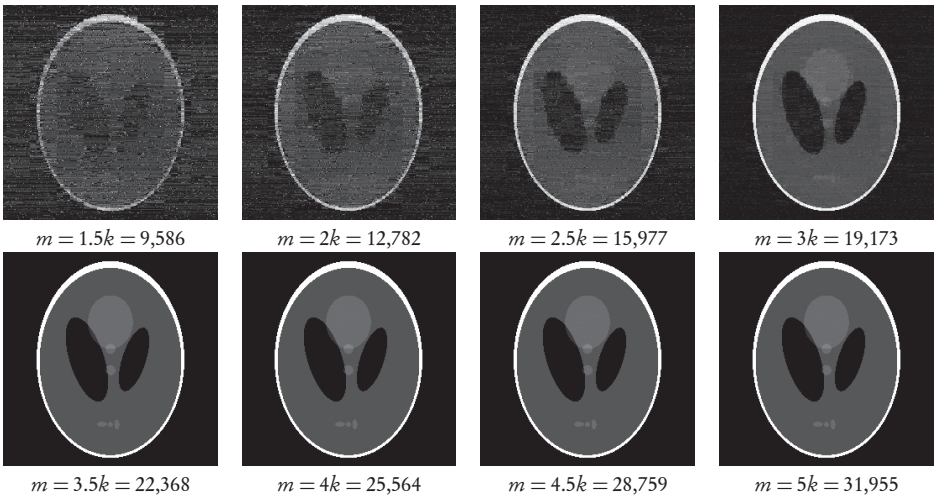

Figure 11.3. *Recovered images obtained as solutions of the linear programming problem with different numbers of constraints (observations).*

11.4 ▪ Available derivatives and algorithmic options

In the subsections above, we discussed which would be the most adequate option among Choices S1–S3 for coding a given problem, as if they were equivalent alternatives for describing a problem to be solved by Algencan. They are not. Algencan deals with the Augmented Lagrangian function, and Choices S1–S3 *do not* provide the same information on the Augmented Lagrangian function to Algencan. On the one hand, Choices S1 and S2 provide the Jacobian of the constraints and the Hessian of the Lagrangian, and in this sense Choices S1 and S2 are equivalent, since they provide enough information to compute the Hessian matrix of the Augmented Lagrangian function. On the other hand, Choice S3 provides partial information on the Jacobian of the constraints and the Hessian of the Lagrangian by providing the value of the product of any of them by a given vector. Of course, this information could be used to compute the full matrices, but this is out of question since it is in opposition to the motivation for choosing S3. Therefore, Choice S3 allows Algencan to compute the product of the Hessian of the Augmented Lagrangian function by a given array, but not the Hessian itself. This means that Choices S1 and S2 make it possible to apply Newtonian approaches to the Augmented Lagrangian subproblems (see Sections 8.5.1 and 12.8), while Choice S3 is a more restrictive choice that makes possible the application of a truncated Newton approach, like the one described in Sections 8.5.2 and 12.8.

Independently of choosing from among Choices S1–S3, there exists the option of providing second-order derivatives or not (i.e., coding `hsub` and `hcsub` in Choice S1, coding `hlsub` in Choice S2, or coding `hlpsub` in Choice S3). If second-order derivatives are not provided, only a subset of the algorithmic possibilities will be available (the ones that do not require second derivatives), namely, a truncated Newton approach with incremental quotients to approximate the Hessian-vector products, like that described in Sections 8.5.2 and 12.8.

If second derivatives are provided, first-order derivatives must be provided too. Otherwise, coded second derivatives will be ignored. If second derivatives are not provided, you also have the option of providing first-order derivatives or not (i.e., coding `gsub`

and `jacsub` in Choice S1, coding `gjacsub` in Choice S2, or coding `gjacpsub` in Choice S3). If first-order derivatives are not provided by the user, they will be approximated by finite differences, implying a potentially time-consuming huge number of objective function and constraints evaluations. Moreover, in this case, only the truncated Newton approach will be available. We arrive to the same situation if none among the linear system solvers from HSL that can be used in connection with Algencan (namely, MA57, MA86, and/or MA97) is made available by the user (see Sections 10.4.1 and 10.4.4).

11.5 ▪ Common questions

1. I feel that only Choice S1 is necessary since common information between subroutines can be passed by means of common statements. Am I right?

   No. At a given point, Algencan may need to evaluate the whole set of gradients of the constraints, or a subset of them. If the user chose S1, only the required gradients will be computed. If the user chose S2 or S3, the whole set of gradients will be computed, only the required subset will be used, and the remaining computed values will be discarded, representing a waste of time. In a more drastic way, the same reasoning applies to the Hessians of the constraints.

   Moreover, since only a subset of the gradients of the constraints may be needed, the problem remains of deciding (within the user-coded subroutine that receives a single index `ind` $\in \{1,2,\ldots,m\}$) when to compute and save all the gradients and when to use a saved value. A solution would be to store the point at which gradients were computed and saved. However, comparing the given point with the stored one, in order to know whether the saved gradient is the required one, would imply computing the (sparse) Jacobian of the constraint in the potentially unaffordable time complexity $O(nm)$.

2. In which way does Algencan take advantage of the presence of linear constraints? It seems to me that linear constraints are artificially evaluated at each call of `csub`.

   Computing a linear constraint involves an inner product. Saving the linear constraint coefficients and computing the inner product within Algencan or asking the user to perform the task involves the same number of operations. Algencan asks the user to perform the task, ignoring the fact that the constraint is linear. The same applies to the gradient of a linear constraint, and even to the (null) Hessian of a linear constraint. Algencan obtains the value of a linear constraint, its gradient, or even its (null) Hessian calling the corresponding user-supplied subroutine. Moreover, counters of number of constraints evaluations, gradients of constraints evaluations, and Hessians of constraints evaluations are increased every time the corresponding user-provided subroutine is called (including the case of linear constraints).

   At this point, you may wonder about the purpose of the logical array `linear`, the input parameter of Algencan that says whether each constraint is linear. Linearity of the constraints is important for computing an approximation of the product of the Hessian of the Augmented Lagrangian by a given vector using incremental quotients and also for computing an approximation of the Hessian of the Augmented Lagrangian (when second-order information is not provided by the user). Other than that, linear constraints receive no special treatment within Algencan.

11.6 ▪ Review and summary

Given a constrained optimization problem with the structure considered in Algencan, one has different alternatives for coding the subroutines that evaluate functions and derivatives. In some cases, the lack of availability of derivatives imposes restrictions on the available options. In other cases, in order to improve efficiency, one should be careful when choosing the appropriate coded set of subroutines. A crucial element to be taken into account is the presence of expensive quantities that need to be computed and are present in different functions or derivatives calculations. In this chapter, we analyzed different coding options and presented illustrative examples.

11.7 ▪ Further reading

The Fortran 90 implementation of problem (11.4) is available in the Algencan distribution. Choice S1 corresponds to file `chap11-ex1a.f90`, while Choice S2 corresponds to file `chap11-ex1b.f90`, both within folder `sources/examples/f90/`. Code and data required to tackle the compressive sensing problem (11.7) and to reproduce the result presented in Section 11.3 is also available. The corresponding file is `chap11-ex2.f90` and can also be found within folder `sources/examples/f90/`. In addition, files `toyprob.f90`, `toyprob2.f90`, and `toyprob3.f90` present a very simple example coded considering Choices S1–S3, respectively.

11.8 ▪ Problems

11.1 Code and solve problem (11.4) with Choices S1 and S2. Compare the results with those reported in Section 11.2. Pay attention to the values attributed to the integer input parameters `jcnnzmax` and `hnnzmax` in Choices S1 and S2. If N is the number of considered angles $\alpha \in [0, \pi/2]$, and hence the number of constraints is $m = 2 + N$, verify that the value of those parameters should be such that `jcnnzmax` $\geq 2m$ and `hnnzmax` $\geq 2N+3m$ in Choice S1 and `jcnnzmax` $\geq 2m$ and `hnnzmax` $\geq 2+3m$ (which does not depend on N) in Choice S2.

11.2 Code and solve problem (11.4) considering Choice S3. Note that, in this case, you can have `jcnnzmax` $= 0$ and `hnnzmax` $= 0$. Analyze the results.

11.3 Code and solve the compressive sensing problem (11.7). Analyze the results.

Chapter 12

Making a Good Choice of Algorithmic Options and Parameters

Recall that Algencan is a subroutine that can be called one or many times from a main program for solving one or many different optimization problems. Before each call to Algencan, in your main program you must set the parameters that you judge to be appropriate for the problem that will be solved in that case. Since the names of the functional subroutines are Algencan parameters, you may code the subroutines for different problems using different names at each call.

Explicit parameters (those in the Algencan calling sequence) were described in Chapter 10. They define the problem and state a few algorithmic options (mainly tolerances). However, Algencan possesses other parameters, called implicit or additional parameters, which take default values that, for theoretical or empirical reasons, we judge to be appropriate for most problems. Therefore, users who don't wish to make decisions about parameter values need not do so. However, default values of implicit parameters are in many cases arbitrary and there are ways in which you can override them if you find it necessary. Sophisticated users of Algencan usually feel the need to change one or several default choices in order to optimize the use of Algencan for their specific problems.

12.1 ▪ Alternatives for setting additional parameters

The additional or implicit parameters receive default values that can be overridden by the user through the usage of keywords (listed in Section 10.2.6). The keywords may be placed in a specification file (inspired by the specification file used by the software Lancelot [82, Chapter 4]) or in the array of parameters `vparam` (an explicit parameter of Algencan).

The array of parameters `vparam` and the specification file are both optional; i.e., you do not need to use them if you do not want to modify any implicit parameter. If you do not want to use the array of parameters `vparam`, you should set `nvparam = 0` before calling Algencan (see Listing 10.1), indicating that the array of parameters `vparam` has no keywords. Otherwise, you should set `nvparam` to be the number of keywords in the array `vparam`. If you do not want to use the specification file, you should set `specfnm = ''` before calling Algencan (see Listing 10.1). If you want to use the specification file, its name must be set in the calling code. For example, if the name of the specification file is `myalgencan.dat`, you should set `specfnm = 'myalgencan.dat'` before calling Algencan. The specification file must be a text file and it must be located in the current folder, containing a keyword per line.

Keywords in array `vparam` (in case there is any) are processed first and then keywords in the specification file (in case it exists) are processed. Keywords are *not* case sensitive. Any line of the specification file or entry in the array of parameters `vparam` starting with # or * is considered a comment and is ignored by the parser. If, due to a typo, a keyword is not recognized by the parser, a warning message is printed and the corresponding keyword is ignored. The same happens if a keyword's mandatory additional value (integer, real, or string) is missing.

12.2 ▪ Output level of detail and output files

The main parameters that control the Algencan output are named `iprint` and `ncomp`. `iprint` assumes values between 1 and 99. The left digit (assumed to be 0 if `iprint` is smaller than 10) controls the level of detail of the outer (Augmented Lagrangian) iterations, while the right digit controls the level of detail of the output related to the inner iterations (iterations of the solver devoted to solving the Augmented Lagrangian subproblems). The greater the digits' value, the larger the amount of information printed. The default value of `iprint` is 10, meaning that Algencan prints a single line per outer iteration and prints no information related to the inner iterations. Larger values are recommended when debugging or analyzing the performance of Algencan on a specific problem. The parameter `ncomp` should assume values between 0 and $\max\{n, m\}$ and says how many entries of an array should be shown in the output. Its default value is 6. The keywords related to `iprint` and `ncomp` are ITERATIONS-OUTPUT-DETAIL and NUMBER-OF-ARRAYS-COMPONENTS-IN-OUTPUT, respectively, and both must be followed by a nonnegative integer value.

It is important to note, as described above, that by default, Algencan prints no information related to the inner iterations. If the original problem is unconstrained or bound constrained, all iterations are "inner" iterations, since the subproblems' solver is directly applied to the original problem. In this case, by default, Algencan does not report any information regarding the progress of the optimization process. Hence, for unconstrained or bound-constrained problems, it is recommended to set the value of `iprint` to any value with the less significant digit being at least 1.

The existence of an output file, which reproduces the output that Algencan shows on the screen, was mentioned in Chapter 10, since its name is a parameter (string with no more than 80 characters) of the Algencan's calling sequence. Before calling Algencan, you should set `outputfnm = ''` to avoid this file and `outputfnm = 'myalgencan.out'` if you want an output file named `myalgencan.out`.

There is also the possibility of obtaining an output file containing the final estimation of the solution (primal and dual variables) delivered by Algencan. By default, this file is not created, and the keyword SOLUTION-FILENAME followed by a string (with no more than 80 characters) should be used to set the name of the file. For example, use

```
SOLUTION-FILENAME mysolution.txt
```

to obtain, in the current folder, a file named `mysolution.txt` containing the approximation to the solution obtained by Algencan.

12.3 ▪ Iteration limits

There are three different types of iterations whose limits may be set by the user: (i) the number of (outer) iterations of the Augmented Lagrangian method, (ii) the number of (inner) iterations that the subproblems' solver may use to solve each subproblem, and (iii) the number of iterations of each trial of the acceleration process.

The default value for the maximum number of outer iterations is 100 and it can be modified with the keyword OUTER-ITERATIONS-LIMIT, followed by the desired integer number.

The default value for the maximum number of inner iterations is 1,000 for constrained problems and a huge number for unconstrained and bound-constrained problems. It can be modified with the keyword INNER-ITERATIONS-LIMIT, followed by the desired integer number. In the case of constrained problems, independently of the maximum allowed number of inner iterations, a maximum of 10 iteration is imposed in the first subproblem. This is because it is assumed that the initial estimation of the dual variables (Lagrange multipliers) may be rough, and, therefore, wasting a lot of time in this first subproblem may not be profitable.

Finally, the default value for the maximum number of iterations in each trial of the acceleration process is 10 and it can be modified with the keyword ACCEL-ITERATIONS-LIMIT, followed by the desired integer number.

Recall that if, when running the acceleration process, a solution to the original problem is not reached, everything that was done in the acceleration process is discarded and the Augmented Lagrangian method proceeds as if the acceleration process had never happened. Thus, if, when following the progress of the acceleration process, you observe that a solution with the desired precision was almost found, but the acceleration process stopped attaining its maximum allowed number of iterations, this is the case of increasing this limit using the keyword ACCEL-ITERATIONS-LIMIT.

12.4 ▪ Penalty parameter: Initial value and limit

The default value of ρ_1, the penalty parameter associated with the first Augmented Lagrangian subproblem, is given by

$$\rho_1 = \max\left\{10^{-8}, \min\left\{10\frac{\max\{1, |w_f f(x^0)|\}}{\max\{1, |\Phi(x^0)|\}}, 10^8\right\}\right\}, \tag{12.1}$$

where $\Phi(x)$ is the infeasibility measure given by (10.9) and x^0 is the initial estimation of the solution given by the user. The motivation for this formula follows. For the particular case $\lambda = 0$, the Augmented Lagrangian function associated with the scaled problem (10.4) reduces to

$$L_\rho(x, 0) = w_f f(x) + \rho\, \Phi(x).$$

So, if $\Phi(x) \neq 0$, the value of ρ that keeps the Augmented Lagrangian well balanced is given by $\rho = |w_f f(x)|/\Phi(x)$. In (12.1), a safeguarded value that gives one more order of magnitude to the feasibility term than to the objective function is considered.

The default choice for ρ_1 described above is arbitrary and depends on the choice of x^0. The user may set the value of ρ_1 with the keyword PENALTY-PARAMETER-INITIAL-VALUE, followed by a positive real number. Starting with a large value may be a desirable choice when one has at hand good initial approximations for the primal and dual variables (warm start) or when x^0 is feasible or almost feasible and loosing feasibility is undesired.

The Augmented Lagrangian method stops when, at iteration k, the iterate x^k was computed as the solution of a subproblem with an associated penalty parameter $\rho_k > \rho_{\max}$. This is because, for large values of ρ_k, stability and numerical issues make it very unlikely to obtain a relatively small required optimality tolerance in the resolution of a subproblem and, in consequence, of the original (scaled) problem. On the other hand, it is assumed that, if the penalty parameter is large, then the original problem is infeasible or the required feasibility tolerance was already achieved. The default value of $\rho_{\max}$ is 10^{20}

and the user can modify this value with the keyword LARGEST-PENALTY-PARAMETER-ALLOWED, followed by a real number.

12.5 ▪ Scaling objective function and constraints

In Section 10.2.2, when describing the main stopping criterion of Algencan, we mentioned that Algencan solves a scaled version of the original problem (10.1) given by

$$\begin{array}{ll} \text{Minimize} & w_f f(x) \\ \text{subject to} & w_{c_j} c_j(x) = 0,\ j \in E, \\ & w_{c_j} c_j(x) \leq 0,\ j \in I, \\ & \ell \leq x \leq u, \end{array}$$

where w_f and w_{c_j}, $j = 1, \ldots, m$, are scaling factors such that $0 < w_f \leq 1$ and $0 < w_{c_j} \leq 1$, $j = 1, \ldots, m$.

By default, the scaling factors are computed as

$$\begin{array}{rcl} w_f & = & 1/\max(1, \|\nabla f(x^0)\|_\infty), \\ w_{c_j} & = & 1/\max(1, \|\nabla c_j(x^0)\|_\infty), \quad j = 1, \ldots, m, \end{array} \tag{12.2}$$

where x^0 is the initial estimation to the solution given by the user.

Algencan displays on the screen the scaling factor for the objective function and the smallest scaling factor for the constraints as follows:

```
Objective function scale factor   : 1.0D-01
Smallest constraints scale factor : 5.2D-02
```

(See Figure 10.1.) Note that the objective function and the constraints are subject to scaling, but no scaling on the variables is done. Moreover, scaling factors, which depend on the initial guess x^0, are rather arbitrary and may be inadequate for the problem at hand. Therefore, you are encouraged to scale your problem and to inhibit the use of these default scaling factors.

By default, the problem at hand *is* scaled by Algencan. To inhibit this default scaling, you should use the keyword OBJECTIVE-AND-CONSTRAINTS-SCALING-AVOIDED. The keyword does not require any additional value.

12.6 ▪ Removing fixed variables

Algencan has no difficulty dealing with a problem that includes variables x_i with $\ell_i = u_i$ in its definition. However, since the optimal value of this kind of "fixed" variable is already known (the only feasible value is $x_i = \ell_i = u_i$), it makes no sense to consider these variables in the optimization process. Hence, by default, Algencan checks the original formulation of the problem looking for fixed variables, eliminates them from the problem formulation, and solves a potentially smaller problem.

On the one hand, Algencan prints (see Figure 10.1) the number of variables, equality constraints, inequality constraints, and bound constraints of the original formulation of the problem,

```
Number of variables                :        8
Number of equality constraints     :        0
Number of inequality constraints   :       15
Number of bound constraints        :        2
```

and, on the other hand, it also prints the number of removed fixed variables,

```
Number of removed fixed variables :       N
```

where N is the number of removed fixed variables, or it prints

```
There are no fixed variables to be removed.
```

if there are no fixed variables to be removed. The actual number of variables of the problem effectively solved is shown by the optimization subroutine as

```
Entry to ALGENCAN.
Number of variables  :       8
Number of constraints:      15
```

This number corresponds to the number of variables of the original problem minus the number of removed variables plus the number of possibly added slack variables (as will be explained below).

The value of the fixed variables is internally saved by Algencan. Every time a subroutine provided by the user is called, the current point is rearranged and completed with the values of the fixed variables, the user-provided subroutine is called, and then the current point is restored. Arrays (current point, gradient, projected gradient, search directions, etc.) displayed on the screen correspond to the components associated with the nonfixed variables that are in fact being considered as variables by Algencan.

Since there seems to be no reason to avoid this potential reduction in the size of the problem effectively solved by Algencan, the elimination of fixed variables is applied by default. However, if, for some reason, you prefer to inhibit this feature, you should use the keyword FIXED-VARIABLES-REMOVAL-AVOIDED. The keyword requires no value.

12.7 ▪ Adding slack variables

Algencan deals with problems of the form (10.1) that may include inequality constraints in their formulation. Moreover, as pointed out in Section 11.1, Algencan may take advantage of the existence of a large number of inequality constraints that are sufficiently satisfied at the solution found. Note that, in principle, there seems to be no reason to convert an inequality constraint into an equality constraint adding a nonnegative slack variable. However, there is a special case in which adding slack variables for that purpose may be convenient. This is the case of quadratic programming problems. Quadratic programming problems with only equality constraints have a quadratic Augmented Lagrangian function, i.e., the Augmented Lagrangian subproblems consist of minimizing a quadratic function subject to bound constraints. Of course, the inconvenience in adding slack variables is an increase in the number of variables in the problem. However, the simplicity of the subproblems seems to compensate for this in the case of quadratic programming problems.

No input parameter of Algencan indicates whether the objective function is quadratic. Therefore, although it is able to detect that all constraints are linear, Algencan is not able to identify that the original problem is a quadratic programming problem. For this reason, by default, Algencan *does not* add slack variables to convert inequalities into equalities. To request the addition of nonnegative slacks for converting inequalities into equalities, the user should use the keyword ADD-SLACKS. The keyword requires no extra numeric value.

The addition of slack variables is recommended for quadratic programming problems.

12.8 ▪ Solving unconstrained and bound-constrained (sub)problems

Gencan [52, 9, 19, 51, 55] is an active set method with projected gradients for bound-constrained minimization that implements the ideas described in Chapters 8 and 9. It was born as a first-order matrix-free method (still available as an option), using conjugate gradients and the incremental quotients approach described in Section 8.5.2. Through the years, it has incorporated, in chronological order, (i) second-order derivatives (still using conjugate gradients to solve the Newtonian systems), (ii) a direct linear systems solver (as described in Section 8.5.1), and (iii) the approximation to the Augmented Lagrangian Hessian matrix described in Section 8.5.3 and its associated preconditioner to be used in connection with conjugate gradients. Moreover, Gencan also has a trust-region-based approach (as an alternative to the Newton line search approach described in Section 8.5.1) that may be adequate for small and medium-sized problems.

A detailed evaluation of solvers that apply to bound-constrained minimization, including Gencan, can be found in [50].

Gencan is the method used by Algencan to solve the bound-constrained Augmented Lagrangian subproblems. The automatic choice of the strategy effectively employed by Gencan to solve a subproblem depends on a few factors, mainly, (a) whether the components needed to compute the Hessian of the Augmented Lagrangian are available, (b) whether a linear systems solver is available, and (c) the number of variables of the problem. The rest of this subsection analyzes each possibility in detail. Since default decisions are arbitrary in many cases, testing all possibilities may have a great impact on the performance of Algencan.

If the Hessian of the Augmented Lagrangian function can be computed, if a linear systems solver is available, and if the number of variables n is not greater than 500, then a classical Euclidean trust-region method is used within the faces. The arbitrary limit in the number of variables is related to the fact that the Hessian of the Augmented Lagrangian must be actually build up and that it may be factorized more than once per iteration. If the Hessian of the Augmented Lagrangian function can be computed, a linear systems solver is available, but the number of variables n is greater than 500, the line search method based on the Newton's direction described in Section 8.5.1 is used. This option requires a single factorization per iteration and, as described below, requires computation of the Hessian of the Lagrangian and the Jacobian of the constraints (but it does not require building up the Hessian of the Augmented Lagrangian).

In the paragraph above, a Euclidean trust-region method and a Newton line-search method were mentioned. It was highlighted that the former may need more than one factorization of the Hessian matrix of the Augmented Lagrangian, while the latter requires a single matrix factorization. An explanation of the matrices effectively being factorized is in order. For the sole purpose of simplifying the presentation, consider a problem with only equality constraints of the form

$$\text{Min } f(x) \text{ subject to } c(x) = 0.$$

The Hessian of the Augmented Lagrangian is given by

$$\nabla^2 L_\rho(x,\lambda) = \nabla^2 \mathscr{L}(x,\hat{\lambda}) + \rho\, c'(x)^T c'(x), \tag{12.3}$$

where $\hat{\lambda} \equiv \lambda + \rho\, c(x)$, $\nabla^2 \mathscr{L}(x,\hat{\lambda})$ is the Hessian of the Lagrangian evaluated at $(x,\hat{\lambda})$, and $c'(x) = (\nabla c_1(x), \ldots, \nabla c_m(x))^T$ is the Jacobian of the constraints.

In the trust-region approach, the matrix (12.3) must be computed and factorized. In the Newton line-search approach, the Newtonian linear system

$$\nabla^2 L_\rho(x,\lambda)d = -\nabla L_\rho(x,\lambda) \tag{12.4}$$

is decomposed as

$$\begin{pmatrix} \nabla^2 \mathscr{L}(x,\hat{\lambda}) & c'(x)^T \\ c'(x) & -\rho I \end{pmatrix}\begin{pmatrix} d \\ w \end{pmatrix} = \begin{pmatrix} -\nabla L_\rho(x,\lambda) \\ 0 \end{pmatrix}, \tag{12.5}$$

where w is an auxiliary variable to be discarded. Apart from conditioning issues (see, for example, [191] and the references therein), computing (12.3) requires computing $c'(x)^T c'(x)$, which may be dense or may have a dense factorization, even if $c'(x)$ is sparse, while the coefficients' matrix in (12.5) preserves the sparsity pattern of the involved matrices.

As a third option, if the coefficients' matrix in (12.5) has a dense factorization, or if the Hessian of the Augmented Lagrangian has a convenient eigenvalue distribution, the truncated Newton approach described in Section 8.5.2 may be used, in which case the linear system (12.4) is solved by conjugate gradients. In this case, the coefficients matrix (12.3) does not need to be computed explicitly, since only its product by an arbitrary vector p is needed. This product may be computed as (i) the product of the Hessian of the Lagrangian by p plus (ii) the product $\rho\, c'(x)^T \tilde{p}$, where $\tilde{p} = c'(x)p$. This is the way this product is computed when the user provides subroutines `hsub` plus `hcsub`, or `hlsub`, to compute Hessians and provides `jacsub` or `gjacsub` to compute the Jacobian of the constraints. The same product may be computed if the user provides `hlpsub` plus `gjacpsub`. In the latter case, since full Hessians and Jacobian are not provided, trust-region and Newton line-search methods are not admissible choices.

If the Hessian of the Augmented Lagrangian function cannot be computed or if a linear system solver is not available, the truncated Newton approach described in Section 8.5.2 is the only choice. In the truncated Newton approach, a Newtonian linear system is solved by conjugate gradients. Only the product of the coefficients matrix (of the Newtonian linear system) by a given vector is required and the three possibilities are (a) the product with the true Hessian of the Augmented Lagrangian can be computed, (b) the product with the Hessian approximation described in Section 8.5.3 is considered, and (c) the matrix-vector product is approximated by incremental quotients as described in Section 8.5.2. In cases (a) and (b), the preconditioner described in Section 8.5.3 is applied. Case (a) requires availability of the information needed to compute the Hessian of the Augmented Lagrangian or at least the true matrix-vector product. Case (b) requires the full Jacobian of the constraints. Case (c) has no requirements. In case (c), if first-order information is available, it will be used to make an extra evaluation of the gradient of the Augmented Lagrangian. Otherwise, it will be approximated by finite differences.

Algencan reports in its preamble, with the value of "parameter" `innslvr`, which one among the Euclidean trust-region approach (`TR`), the line-search Newton method (`NW`), and the line search truncated Newton method (`TN`) is employed. Independently of the selected inner solver, it also shows which would be the type of matrix-vector product if the truncated Newton approach were used (parameter `hptype`) and which would be the linear system solver subroutine (and its scaling choice) if the Euclidean trust-region or the Newton line search strategy were used (parameters `lsslvr in TR` and `lsslvr in NW`, respectively). For parameter `hptype`, `TRUEHP` stands for case (a) in the paragraph above, `HAPPRO` stands for case (b), and `INCQUO` stands for case (c).

The default arbitrary Algencan choices for the subproblems' solver were described above. However, if the requisites are given, any of the alternatives may be selected by the user employing implicit parameters. The user is encouraged to test them, since this choice may have a big influence on Algencan's performance. To pick out the classical Euclidean trust-region method, use the keyword TRUST-REGIONS-INNER-SOLVER, to choose Newton's method with line search, use the keyword NEWTON-LINE-SEARCH-INNER-SOLVER, and for the truncated Newton approach, use TRUNCATED-NEWTON-LINE-SEARCH-INNER-SOLVER. These keywords optionally may be followed by an additional value (string).

Keyword TRUNCATED-NEWTON-LINE-SEARCH-INNER-SOLVER may be followed by a string representing the desired type of matrix-vector product. Options are TRUE-HESSIAN (or TRUEHP), HESSIAN-APPROXIMATION (or HAPPRO), and INCREMENTAL-QUOTIENTS (or INCQUO), representing cases (a)–(c) above, respectively. This means that, for example, with the keyword

```
TRUNCATED-NEWTON-LINE-SEARCH-INNER-SOLVER INCREMENTAL-QUOTIENTS
```

the user will be choosing the truncated Newton approach with incremental quotients to approximate the Hessian-vector products, as described in Section 8.5.2. In this case, Algencan reduces to a first-order matrix-free method. However, under some circumstances, Algencan may need to perform a truncated Newton iteration, even if the Newton line-search inner solver was selected (by the user or by default). An example of this situation is if for some unspecified reason (such as lack of memory) the linear system solver subroutine fails to solve the Newtonian system. To deal with this situation, there is a way to choose the matrix-vector product that should be used in case a truncated Newton iteration is performed, but without selecting the truncated Newton approach as the subproblems' solver choice. The keyword for selecting the matrix-vector product of the truncated Newton approach is MATRIX-VECTOR-PRODUCT-IN-TRUNCATED-NEWTON-LS. It must be followed by a string, and the options are the same as those that are optional for the keyword TRUNCATED-NEWTON-LINE-SEARCH-INNER-SOLVER.

Following the spirit of the above paragraph, there are also keywords to indicate which linear system solver subroutine, and which choice for the scaling of the linear system, should be used in a Euclidean trust-region iteration and in a Newton line-search iteration. These keywords are LINEAR-SYSTEMS-SOLVER-IN-TRUST-REGION and LINEAR-SYSTEMS-SOLVER-IN-NEWTON-LINE-SEARCH, respectively. It is important to stress that these keywords can be used to select the linear system solver subroutine that may be used in each case (and that can be a different one), but no selection regarding the subproblems' solver is being done. These two keywords must be followed by one or two strings. The first string is mandatory and must be used to select the linear system solver subroutine. The second string is optional and may be used to determine the scaling choice. For the trust-region method, the only choice of linear systems solver is MA57, while the choices for the scaling method are MC64 and NONE. For the Newton line-search method, the choices for the linear systems solver subroutine are MA57, MA86, and MA97. For subroutine MA57, scaling choices are MC64 and NONE. For subroutine MA86, scaling choices are MC64, MC77, and NONE. For subroutine MA97, choices are MC64, MC77, MC30, and NONE.

The following lines, extracted from Algencan's preamble, exemplifies how Algencan reports the choices described in the present section (see Figure 10.1):

```
firstde                =                    T
seconde                =                    T
truehpr                =                    T
hptype in TN           =              TRUEHP
```

```
lsslvr in TR          =          MA57/MC64
lsslvr in NW          =          MA97/MC64
innslvr               =                NW
```

The logical value (`T` stands for true and `F` stands for false) that follows `firstde`, `seconde`, and `truehp` simply says whether, using the user-provided subroutines that describes the problem at hand, first-order derivatives (gradient of the objective function and gradients of the constraints), second-order derivatives (Hessian of the Augmented Lagrangian), and the true product of the Hessian of the Augmented Lagrangian by a given vector can be computed. Those values are very important for understanding whether the requirements for a specific subproblems' solver are satisfied. The remaining parameters show the following:

`hptype in TN`: the type of matrix-vector product that would be used if a truncated Newton iteration is performed.

`lsslvr in TR`: the linear systems solver subroutine (and the scaling choice) that would be used if a Euclidean trust-region iteration were done.

`lsslvr in NW`: the linear systems solver subroutine (and the scaling choice) that would be used if an iteration of the Newton line-search approach were performed.

`innslvr`: the type of method that will be used to solve the Augmented Lagrangian subproblems (this choice may correspond to a default choice or to a choice made by the user).

It is extremely important to check the values of the parameters described above in order to confirm whether the selections made by the user were in fact implemented and, if they were not, which requirements are missing.

12.9 ▪ How to solve feasibility problems

In this section, we consider the case in which the problem to be solved has a constant or null objective function, i.e., we are in the presence of a "feasibility problem." To fix notation, assume the problem is given by finding $x \in \mathbb{R}^n$ such that

$$\bar{c}_j(x) = 0,\ j \in E,\ \bar{c}_j(x) \le 0,\ j \in I,\ \ell \le x \le u. \tag{12.6}$$

There are two basic ways to fit this problem in the format of problem (10.1). The first choice consists of defining $f(x) \equiv 0$ and $c(x) \equiv \bar{c}(x)$. The second choice involves defining

$$f(x) \equiv \sum_{j \in E} \bar{c}_j(x)^2 + \sum_{j \in I} \bar{c}_j(x)_+^2 \tag{12.7}$$

and $c(x) \equiv 0$. We now discuss some properties of each choice, starting with the second one.

The second choice consists of modeling the feasibility problem (12.6) as a bound-constrained problem. In this case, the Augmented Lagrangian method does not apply and Gencan, the method used by Algencan to solve the Augmented Lagrangian subproblems, is directly applied to the original bound-constrained problem. For a discussion related to the method used for solving bound-constrained subproblems, see Section 12.8. At this point, it is important to note that since the problem is modeled as a problem without

constraints (other than the bound constraints) the automatic scaling approach described in Section 12.5 is not applicable. Moreover, the natural choice for coding an unconstrained or bound-constrained problem is Choice S1 (see Chapter 11) and the user should be ready to code subroutines `fsub`, `gsub`, and `hsub` to compute the objective function (12.7), its gradient, and its Hessian, respectively. Among these, the most annoying task would be to write a piece of code (within subroutine `hsub`) to efficiently compute the sparse Hessian of f, as defined in (12.7), that implies in computing the sum of the sparse Hessians of $\bar{c}_j$ for all $j \in E \cup I$. Last but not least, the main stopping criterion (10.6)–(10.8) reduces, in this case, to

$$\begin{aligned}&\|P_{[\ell,u]}\big(x^k - \nabla f(x^k)\big) - x^k\|_\infty \\ &= \|P_{[\ell,u]}\Big(x^k - \Big[\textstyle\sum_{j\in E} \bar{c}_j(x^k)\nabla \bar{c}_j(x^k) + \sum_{j\in I} \bar{c}_j(x^k)_+ \nabla \bar{c}_j(x^k)\Big]\Big) - x^k\|_\infty \le \varepsilon_{\text{opt}},\end{aligned}$$

where $P_{[\ell,u]}$ represents the Euclidean projection operator onto the box $\{x \in \mathbb{R}^n \mid \ell \le x \le u\}$. Although reasonable, the stopping criterion above does not allow the user to control the desired level of feasibility at the solution.

The reasons described in the paragraph above give a hint about the convenience of the first choice, i.e., to define $f(x) \equiv 0$ and $c(x) \equiv \bar{c}(x)$. However, since there is no parameter telling Algencan that the objective function is of a particular type, if nothing else is done the Augmented Lagrangian approach would be applied to the problem. In principle, there are no contraindications for doing that. However, the role of the Lagrange multipliers and the penalty parameter is not clear in this case. The keyword IGNORE-OBJECTIVE-FUNCTION says to Algencan that, as suggested, the objective function must be ignored. The keyword has no additional value. We now describe how Algencan proceeds in this case.

In the case in which the user requires the objective function to be ignored, Algencan solves, using Gencan, the bound-constrained problem given by minimizing the Augmented Lagrangian function $L_\rho(x, \lambda)$ defined in (10.3) subject to $\ell \le x \le u$, setting $\lambda = 0$, and $\rho = 1$. The scaling factor w_f for the objective function is defined as $w_f = 0$, while the scaling factors w_{c_j} for the constraints are the defaults given by (12.2). In this case, $L_\rho(x, \lambda)$ coincides with

$$\frac{1}{2}\left(\sum_{j\in E}(w_{c_j} c_j(x))^2 + \sum_{j\in I}(w_{c_j} c_j(x)_+)^2\right) = \frac{1}{2}\left(\sum_{j\in E}(w_{c_j} \bar{c}_j(x))^2 + \sum_{j\in I}(w_{c_j} \bar{c}_j(x)_+)^2\right),$$

i.e., coincides with the squared sum of the infeasibilities. The highlights of this approach are as follows:

1. Automatic scaling is done (recall that it can be easily inhibited with the keyword OBJECTIVE-AND-CONSTRAINTS-SCALING-AVOIDED; see Section 12.5).

2. The user may code subroutines `csub`, `jacsub`, and `hcsub` (included in the so-called Choice S1 in the context of choosing the adequate subroutines to code a problem; see Chapter 11). In these subroutines, constraints, their gradients, and Hessians are coded individually, and the sum of the sparse Hessians of the constraints is done internally by Algencan.

3. The stopping criterion is solely based on the (unscaled) feasibility and is given by

$$\begin{aligned}&\max\left\{\max_{j\in E}\left\{|c_j(x^k)|\right\}, \max_{j\in I}\left\{c_j(x^k)_+\right\}\right\} \\ &= \max\left\{\max_{j\in E}\left\{|\bar{c}_j(x^k)|\right\}, \max_{j\in I}\left\{\bar{c}_j(x^k)_+\right\}\right\} \le \varepsilon_{\text{feas}}.\end{aligned}$$

Since, a priori, only feasibility matters, the natural stopping criterion given by

$$\left\| P_{[\ell,u]}\left(x^k - \left[\sum_{j\in E} w_{c_j}^2 \bar{c}_j(x^k)\nabla \bar{c}_j(x^k) + \sum_{j\in I} w_{c_j}^2 \bar{c}_j(x^k)_+ \nabla \bar{c}_j(x^k) \right] \right) - x^k \right\|_\infty \leq \varepsilon_{\text{opt}}$$

is ignored. This may be a drawback in the case of an infeasible feasibility problem, in which Gencan may show a slow and painful performance until stopped by an alternative stopping criterion based on lack of progress.

4. For the reasons mentioned in the item above, the Algencan criterion for stopping at an infeasible point that is stationary for the infeasibility measure (10.9) is verified at each iteration of the subproblems' solver when the objective function is being ignored. Therefore, the alternative stopping criterion

$$\max\left\{ \max_{j\in E}\{|w_{c_j}\bar{c}_j(x^k)|\}, \max_{j\in I}\{[w_{c_j}\bar{c}_j(x^k)]_+\} \right\} > \varepsilon_{\text{fstain}}$$

and

$$\left\| P_{[\ell,u]}\left(x^k - \left[\sum_{j\in E} w_{c_j}^2 \bar{c}_j(x^k)\nabla \bar{c}_j(x^k) + \sum_{j\in I} w_{c_j}^2 \bar{c}_j(x^k)_+ \nabla \bar{c}_j(x^k) \right] \right) - x^k \right\|_\infty \leq \varepsilon_{\text{ostain}}$$

may also be considered. As mentioned in Section 10.2.3, possible values for $\varepsilon_{\text{fstain}}$ and $\varepsilon_{\text{ostain}}$ would be $\sqrt{\varepsilon_{\text{feas}}}$ and $\varepsilon_{\text{opt}}^{1.5}$, respectively.

Summing up, using the keyword IGNORE-OBJECTIVE-FUNCTION is recommended when the problem at hand is a feasibility problem.

12.10 ▪ Acceleration process

The acceleration process is a strategy applied between the Augmented Lagrangian iterations. It aims to solve the KKT system (10.13)–(10.20) of the original (unscaled) problem (10.1) by Newton's method iteratively. If Newton's method finds a solution to the original problem, Algencan stops. (The stopping criterion that may be satisfied by the acceleration process is described in Section 10.2.4.) If the Newton's method "fails" trying to solve the nonlinear system (10.13)–(10.20), the acceleration process is abandoned and Algencan continues with the next Augmented Lagrangian iteration as if the acceleration process had never happened.

The usage of the explicit parameters `efacc` and `eoacc` to determine when the acceleration process should be launched is described in Section 10.2.4. The acceleration process requires first-order (full Jacobian) and second-order derivatives to be provided by the user, plus requires the availability of a linear system solver (namely, subroutine MA57, MA86, or MA97 from HSL).

If the requirements are given, the acceleration process is enabled by default. To inhibit its usage, the keyword SKIP-ACCELERATION-PROCESS may be used. It requires no additional value. The keyword LINEAR-SYSTEMS-SOLVER-IN-ACCELERATION-PROCESS may be used to choose the linear systems solver subroutine that should be applied in the acceleration process to solve the Newtonian linear system. The keyword must be followed by a string MA57, MA86, or MA97. Optionally, the string that determines the linear systems solver may be followed by another string to determine the scaling method. Choices associated with subroutine MA57 are MC64 and NONE. Choices associated with sub-

routine MA86 are MC64, MC77, and NONE. Choices associated with MA97 are MC64, MC77, MC30, and NONE.

The following two lines, extracted from Algencan's preamble, exemplifies how Algencan reports whether the acceleration process will be considered or skipped (`accproc` equal to `T` or `F`, respectively) and which would be the linear system solver used in the acceleration process (see Figure 10.1):

```
lsslvr in ACCPROC      =            MA97/MC64
accproc                =                   T
```

It is extremely important to check the values of the parameters described above in order to confirm whether the selections made by the user were in fact implemented and, if they were not, which requirements are missing.

A short description of the acceleration process follows.

At this point, it is assumed that $\ell_i < u_i$ for all i. Variables x_i such that $\ell_i = u_i$ are automatically removed from the problem formulation (10.1) by Algencan (see Section 12.6) and, consequently, they are not present in (10.12) or in (10.13)–(10.20).

The KKT system (10.13)–(10.20) has $5n + m + |I|$ variables and equations. At the beginning of each Newton's step, it is first determined which subset of x-variables and constraints of type (10.15)–(10.17) will be considered in the step. This means that reduced Newtonian linear systems are solved at each step. A maximum of 10 iterations is considered by default. Refer to Section 12.3 for how to modify this value.

Let $\bar{x}$ be the current iterate in the Newton's iterative process. Considered x-variables will be those such that $\ell_i < \bar{x}_i < u_i$. (The others will be set as $\bar{x}_i \leftarrow P_{[\ell_i,u_i]}(\bar{x}_i)$.) Let I_L and I_U be the sets of indices of the x-variables fixed in their lower and upper bounds, respectively, and define $\bar{n} = |I_L| + |I_U|$. This decision eliminates $\bar{n}$ x-variables and $\bar{n}$ equations from (10.13), $2\bar{n}$ equations and slack variables from (10.16)–(10.17), and $2\bar{n}$ equations and Lagrange multipliers from the complementarity conditions (10.19)–(10.20).

Regarding the equations, the current Newton step will consider equations related to equality constraints of the original problem (10.1), equations related to inequality constraints of the original problem (10.1) that are nearly active or that were considered in the previous Newton's step, and nearly active bound constraints of variables that are not fixed at their bounds. Namely, the considered equations are (a) all $(n - \bar{n})$ equations in (10.13) with $i \notin I_L \cup I_U$, (b) all equations in (10.14) that correspond to original equality constraints, (c) equations in (10.15) that were considered in the previous step or such that $c_j(\bar{x}) \geq -\sqrt{\varepsilon_{\text{feas}}}$, (d) equations in (10.16) associated with bound constraints such that $\ell_i < \bar{x}_i \leq \ell_i + \sqrt{\varepsilon_{\text{feas}}}$, and (e) equations in (10.17) associated with bound constraints such that $u_i - \sqrt{\varepsilon_{\text{feas}}} \leq \bar{x}_i < u_i$. Each eliminated (or unconsidered) constraint also eliminates a slack variable, a Lagrange multiplier, and a complementarity constraint. Eliminated equations can be trivially satisfied by setting the slack variables and the Lagrange multipliers appropriately.

On success, Newton's method returns $(\tilde{x}^k, \tilde{s}, \tilde{s}^\ell, \tilde{s}^u, \tilde{\lambda}^k, \tilde{\lambda}^\ell, \tilde{\lambda}^u)$ such that

$$
\begin{aligned}
\|\nabla f(\tilde{x}^k) + \textstyle\sum_{j \in E \cup I} \tilde{\lambda}^k_j \nabla c_j(\tilde{x}^k) - \tilde{\lambda}^\ell + \tilde{\lambda}^u\|_\infty &\leq \varepsilon_{\text{opt}}, \\
\|c_j(\tilde{x}^k)\|_\infty &\leq \varepsilon_{\text{feas}}, \quad j \in E, \\
\|c_j(\tilde{x}^k) + 1/2\, \tilde{s}_j^2\|_\infty &\leq \varepsilon_{\text{feas}}, \quad j \in I, \\
\|\ell_i - \tilde{x}^k_i + 1/2\, (\tilde{s}^\ell_i)^2\|_\infty &\leq \varepsilon_{\text{feas}}, \quad i = 1, \ldots, n, \\
\|\tilde{x}^k_i - u_i + 1/2\, (\tilde{s}^u_i)^2\|_\infty &\leq \varepsilon_{\text{feas}}, \quad i = 1, \ldots, n, \\
\tilde{\lambda}^\ell_i &= 0 \quad \text{for all } i \in \{1, \ldots, n\} \text{ such that } \tilde{x}^k_i > \ell_i, \\
\tilde{\lambda}^u_i &= 0 \quad \text{for all } i \in \{1, \ldots, n\} \text{ such that } \tilde{x}^k_i < u_i, \\
\tilde{\lambda}_j &= 0 \quad \text{for all } j \in I \text{ such that } c_j(\tilde{x}^k) < -\varepsilon_{\text{feas}},
\end{aligned}
$$

plus $\ell \leq \tilde{x}^k \leq u$. Thus, $(\tilde{x}^k, \tilde{\lambda}^k)$ satisfies

$$\left\| P_{[\ell,u]}\left(\tilde{x}^k - \left[\nabla f(\tilde{x}^k) + \sum_{j\in E\cup I} \tilde{\lambda}_j^k \nabla c_j(\tilde{x}^k)\right]\right) - \tilde{x}^k \right\|_\infty \leq \varepsilon_{\text{opt}},$$
$$\max\left\{\max_{j\in E}\left\{\left|c_j(\tilde{x}^k)\right|\right\}, \max_{j\in I}\left\{\left|\min\left\{-c_j(\tilde{x}^k), \tilde{\lambda}_j^k\right\}\right|\right\}\right\} \leq \varepsilon_{\text{feas}}.$$

If, in addition, $\tilde{\lambda}_j^k \geq 0$ for all $j \in I$, then $(\tilde{x}^k, \tilde{\lambda}^k)$ is an approximate stationary pair of the *unscaled* original problem (10.1). In this case, the acceleration process is considered successful and Algencan stops.

Otherwise (that is, there exists $j \in I$ such that $\tilde{\lambda}_j^k < 0$), the following least-squares problem, in which $\tilde{x}^k$ is a constant, is considered:

$$\begin{array}{ll} \text{Min}_{(\lambda,\lambda^\ell,\lambda^u)} & \|\nabla f(\tilde{x}^k) + \sum_{j\in E\cup I} \lambda_j \nabla c_j(\tilde{x}^k) - \lambda^\ell + \lambda^u\|_2^2 \\ \text{subject to} & \lambda_i^\ell = 0 \quad \text{for all } i \in \{1,\ldots,n\} \text{ such that } \tilde{x}_i^k > \ell_i, \\ & \lambda_i^u = 0 \quad \text{for all } i \in \{1,\ldots,n\} \text{ such that } \tilde{x}_i^k < u_i, \\ & \lambda_j = 0 \quad \text{for all } j \in I \text{ such that } c_j(\tilde{x}^k) < -\varepsilon_{\text{feas}}, \\ & \lambda_j \geq 0 \quad \text{for all } j \in I. \end{array} \tag{12.8}$$

Gencan, the same method that is applied to the Augmented Lagrangian subproblems, is used to solve problem (12.8). Since, in the best case, only first-order derivatives of the objective function of problem (12.8) are available, the strategy used by Gencan is the truncated Newton approach with incremental quotients described in Section 8.5.2. If a feasible point $(\hat{\lambda}^k, \hat{\lambda}^\ell, \hat{\lambda}^u)$ of (12.8) such that

$$\|\nabla f(\tilde{x}^k) + \sum_{j\in E\cup I} \hat{\lambda}_j^k \nabla c_j(\tilde{x}^k) - \hat{\lambda}^\ell + \hat{\lambda}^u\|_\infty \leq \varepsilon_{\text{opt}}$$

is found, then the pair $(\tilde{x}^k, \hat{\lambda}^k)$ satisfies

$$\left\| P_{[\ell,u]}\left(\tilde{x}^k - \left[\nabla f(\tilde{x}^k) + \sum_{j\in E\cup I} \hat{\lambda}_j^k \nabla c_j(\tilde{x}^k)\right]\right) - \tilde{x}^k \right\|_\infty \leq \varepsilon_{\text{opt}},$$
$$\max\left\{\max_{j\in E}\left\{\left|c_j(\tilde{x}^k)\right|\right\}, \max_{j\in I}\left\{\left|\min\left\{-c_j(\tilde{x}^k), \hat{\lambda}_j^k\right\}\right|\right\}\right\} \leq \varepsilon_{\text{feas}}$$

with $\ell \leq \tilde{x}^k \leq u$ and $\hat{\lambda}^k \geq 0$, it is an approximate stationary pair of the *unscaled* original problem (10.1), the acceleration process is considered successful, and Algencan stops. Otherwise, the acceleration process is abandoned and Algencan continues with the next Augmented Lagrangian iteration.

12.11 ▪ Review and summary

Algencan is a software designed to solve constrained optimization problems with equality and inequality constraints and bounds. It is based on the Augmented Lagrangian approach and subproblems are solved using several alternatives. Most of its additional parameters and algorithmic choices were discussed in this chapter.

Summing up, the scaling of the objective function and the constraints used by default by Algencan is arbitrary, and users are encouraged to scale the problem themselves. First- and second-order derivatives should be coded whenever possible to enable several

choices for the subproblems' solver and the usage of the acceleration process. The keyword IGNORE-OBJECTIVE-FUNCTION should be used to solve feasibility problems and the keyword ADD-SLACKS should be used to solve quadratic programming problems. These options *must be set by the user*, since Algencan has no way to know whether the objective function is nonlinear, quadratic, or even constant.

12.12 ▪ Problems

12.1 The possibilities for problems in this chapter are enormous. Code your own problem and check all possibilities described in the present chapter. Compare the results.

12.2 Some authors prefer to update the Lagrange multipliers only when the penalty parameter increasing test (4.9) indicates that the penalty parameter should not be increased. Discuss and design experiments to evaluate this alternative numerically.

12.3 Design experiments to test different problem-dependent alternatives for stopping the solution process of the subproblems. Present your results in the form of performance profiles [100].

12.4 Test Algencan for solving large-scale linear programming problems.

12.5 Run Algencan (a) coding second-order derivatives and (b) approximating second-order derivatives with finite differences. Evaluate both possibilities in a set of problems and derive practical conclusions.

12.6 Run Algencan approximating first-order derivatives with finite differences. Derive conclusions. Compare with the results obtained in Problem 12.5.

12.7 Write a code to solve a set of problems with Algencan choosing randomly the explicit and the implicit parameters. Indicate the best combination of parameters.

12.8 Express Problem 12.7 in the form of a derivative-free optimization problem.

Chapter 13
Practical Examples

In this chapter, we illustrate the use of Algencan in four practical applications. In all cases, Algencan 3.0.0 with the (arbitrarily chosen) HSL MA57 subroutine was considered. Algencan was compiled with GNU Fortran (GFortran) included in GCC version 4.6.3. All the experiments were executed on a 2.4-GHz Intel Core 2 Quad Q6600 with 4.0 GB of RAM memory running the GNU/Linux operating system.

13.1 ▪ Packing molecules

Molecular dynamics is a powerful technique for comprehension at the molecular level of a great variety of chemical processes. With the enhancement of computational resources, very complex systems can be studied. The simulations need starting points that must have adequate energy requirements. However, if the starting configuration has close atoms, the temperature scaling is disrupted by excessive potentials that accelerate molecules over the accepted velocities for almost any reasonable integration time step. In fact, the starting coordinates must be reliable in that they must not exhibit overlapping or close atoms, so that temperature scaling can be performed with reasonable time steps for a relatively fast energy equilibration of the system.

In [187, 193], the problem of finding the initial positions of the molecules is represented as a "packing problem." The goal is to place known objects in a finite domain in such a way that the distance between any pair of points of objects is larger than a threshold tolerance. In our case, objects are molecules and points are atoms. Following this idea, an optimization problem is defined. The mathematical (optimization) problem consists of the minimization of a function of (generally) many variables, subject to constraints that define the region in which the molecules should be placed.

Let us call $nmol$ the total number of molecules that we want to place in a region $\mathscr{R}$ of the three-dimensional space. For each $i = 1, \ldots, nmol$, let $natom(i)$ be the number of atoms of the ith molecule. Each molecule is represented by the orthogonal coordinates of its atoms. To facilitate the visualization, assume that the origin is the barycenter of all the molecules. For all $i = 1, \ldots, nmol$, $j = 1, \ldots, natom(i)$, let

$$A(i,j) = (a_1^{ij}, a_2^{ij}, a_3^{ij})$$

be the coordinates of the jth atom in the ith molecule.

Suppose that one rotates the ith molecule sequentially around the axes x_1, x_2, and x_3, where $\gamma^i = (\gamma_1^i, \gamma_2^i, \gamma_3^i)$ are the angles that define such rotations. Moreover, suppose that

after these rotations, the whole molecule is displaced so that its barycenter, instead of the origin, becomes $t^i = (t_1^i, t_2^i, t_3^i)$. These movements transform the atom of coordinates $A(i,j)$ in a displaced atom of coordinates

$$P(i,j) = (p_1^{ij}, p_2^{ij}, p_3^{ij}).$$

Observe that $P(i,j)$, $j = 1,\ldots,natom(i)$, is a function of (t^i, γ^i), the relation being

$$P(i,j) = t^i + R(\gamma^i)A(i,j),\ j = 1,\ldots,natom(i),$$

where

$$R(\gamma^i) = \begin{pmatrix} c_1^i c_2^i c_3^i - s_1^i s_3^i & s_1^i c_2^i c_3^i + c_1^i s_3^i & -s_2^i c_3^i \\ -c_1^i c_2^i s_3^i - s_1^i c_3^i & -s_1^i c_2^i s_3^i + c_1^i c_3^i & -s_2^i s_3^i \\ c_1^i s_2^i & s_1^i s_2^i & c_2^i \end{pmatrix}, \tag{13.1}$$

in which $s_k^i \equiv \sin\gamma_k^i$ and $c_k^i \equiv \cos\gamma_k^i$ for $k = 1,2,3$.

In [187, 193], the objective is to find angles γ_i and displacements t_i, $i = 1,\ldots,nmol$, in such a way that whenever $i \neq i'$,

$$\|P(i,j) - P(i',j')\|_2^2 \geq d^2 \tag{13.2}$$

for all $j = 1,\ldots,natom(i)$, $j' = 1,\ldots,natom(i')$, where $d > 0$ is the required minimum distance, and

$$P(i,j) \in \mathscr{R} \tag{13.3}$$

for all $i = 1,\ldots,nmol$, $j = 1,\ldots,natom(i)$. In other words, the rotated and displaced molecules must remain in the specified region and the distance between any pair of atoms must not be less than d. Note that region $\mathscr{R}$ does not need to be convex and that it could be replaced by a region $\mathscr{R}^{ij}$ for each atom of each molecule (as required in most real cases).

The objective (13.2) leads us to define the following merit function f:

$$\begin{aligned} &f(t_1,\ldots,t_{nmol},\gamma_1,\ldots,\gamma_{nmol}) \\ &= \sum_{i=1}^{nmol} \sum_{j=1}^{natom(i)} \sum_{i'=i+1}^{nmol} \sum_{j'=1}^{natom(i')} \max\left\{0, d^2 - \left\|P(i,j) - P(i',j')\right\|_2^2\right\}^2. \end{aligned} \tag{13.4}$$

Note that $f(t_1,\ldots,t_{nmol},\gamma_1,\ldots,\gamma_{nmol})$ is nonnegative for all angles and displacements. Moreover, f vanishes if and only if the objective (13.2) is fulfilled. This means that if we find displacements and angles where $f = 0$, the atoms of the resulting molecules are sufficiently separated. This leads us to define the following minimization problem:

$$\text{Minimize } f(t_1,\ldots,t_{nmol},\gamma_1,\ldots,\gamma_{nmol}) \tag{13.5}$$

subject to (13.3) for all $i = 1,\ldots,nmol$, $j = 1,\ldots,natom(i)$.

The objective function f is continuous and differentiable, although their second derivatives are discontinuous. The number of variables is $6 \times nmol$ (three angles and a displacement per molecule). The analytical expression of f is cumbersome, since it involves consecutive rotations and its first derivatives are not very easy to code. However, optimization experience leads us to pay the cost of writing a code for computing derivatives with the expectation that algorithms that take advantage of first-order information are profitable, especially when the number of variables is large. Having a code that computes f and its gradient, we are prepared to solve (13.5) using constrained optimization techniques.

13.1.1 ▪ Analyzing a simplified version

In order to focus in the application of Algencan to the described molecules packing problem, we consider a simplified (less realistic) instance in which molecules are all identical and they are composed of a single atom. Note that the single-atom feature eliminates the rotation angles as variables and the variables of the problem become the translations only. If, in addition, we set $nmol = N$, $d = 2r$, and $\mathscr{R} = \{x \in \mathbb{R}^3 \mid ||x||_2 \leq R\}$, the problem can be seen as the problem of placing N identical spheres with radius r within a sphere with radius R centered at the origin. This is one of the many variants of the well-known packing problems described in, for example, [65].

Summing up, the problem considered in this section is given by

$$\text{Minimize } f(x) \text{ subject to } c(x) \leq 0, \tag{13.6}$$

where $x = (t_1, \ldots, t_N) \in \mathbb{R}^{3N}$,

$$f(t_1, \ldots, t_N) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \max\{0, (2r)^2 - ||t_i - t_j||_2^2\}^2, \tag{13.7}$$

and

$$c_i(x) = ||t_i||_2^2 - (R-r)^2, \; i = 1, \ldots, N. \tag{13.8}$$

A key point that strongly affects the difficulty of an instance of a packing problem is its density (occupied fraction), which in the case of problem (13.6)–(13.8) is given by $N(r/R)^3$. Considering that each atom is a sphere with a radius of 1Å, the volume occupied by the atoms in liquid water is roughly 30% of the total volume. This density was used in the illustrative examples of this section.

The treatment of the packing problem that we present below is valid as an illustration of the techniques applied to solve the much more complex molecules packing problem described in the previous subsection. Numerical examples intend to resemble the techniques implemented in the software Packmol [187, 193]. If the focus were classical packing problems by themselves, many other techniques, such as those based on lattices [85], might be considered.

The problem has $n = 3N$ variables and $m = N$ inequality constraints. Its has no equality constraints and no bounds on the variables. We coded subroutines `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, and `hcsub`. The last three subroutines compute the constraints and their first- and second-order derivatives, respectively, and they are very easy to code. Subroutine `fsub` computes the objective function. It is also easy to code and its naive implementation requires $O(N^2)$ operations to evaluate f at a given point. (This feature will be addressed below.) Subroutine `gsub` is also easy to code, while coding subroutine `hsub` is rather boring and prone to error. They both share with `fsub` the $O(N^2)$ time complexity to evaluate the gradient and the Hessian of the objective function, respectively, at a given point. The six subroutines are part of the Algencan distribution (file `chap13-packmol-dense.f90` within folder `sources/examples/f90/`). As a starting guess, we consider $x^0 = (t_1^0, \ldots, t_N^0)$ with uniformly distributed random $t_i^0 \in [-R, R]^3$.

Before considering a particular instance, we need to analyze (a) the number of nonnull elements in the Jacobian and (b) the number of memory positions required to compute the lower triangle of the Hessian of the Augmented Lagrangian function of problem (13.6)–(13.8). These are the values that must be given to Algencan's parameters `jcnnzmax` and `hnnzmax`, respectively, when the user opts for choice S1 (see Chapter 11) and codes subroutines `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, and `hcsub`. The number of positions in

(b) is the sum of (b1) the number of nonnull elements in the lower triangle of the Hessian of the objective function, (b2) the sum of the number of nonnull elements in the lower triangle of the Hessian matrix of each constraint, and (b3) the number of nonnull elements in the lower triangle of the matrix given by the transpose of the Jacobian times the Jacobian. An upper bound on this last number is given by $\frac{1}{2}\sum_{j=1}^{m} \texttt{jcnnzmax}_j(\texttt{jcnnzmax}_j + 1)$, where $\texttt{jcnnzmax}_j$ is the number of nonnull elements of the jth row of the Jacobian (i.e., the number of nonnull elements in the gradient of the jth constraint). Including the quantity described in (b3) in the value of `hnnzmax` is a conservative approach, since it is required only if the method used to solve the Augmented Lagrangian subproblems is the Euclidean trust-region method (which is not the usual choice in large-scale problems; see Sections 10.2.11 and 12.8). Anyway, including this quantity is costless in this case because the transpose of the Jacobian times the Jacobian is a very sparse matrix for the problem at hand.

Looking at the nonlinear interaction of the variables in the constraints (13.8), it is easy to see that the gradient of each constraint has no more than three nonnull elements and that the (diagonal and constant) Hessian matrix of each constraint has exactly three nonnull elements. This means that (i) the Jacobian matrix has no more than $3N$ nonnull entries and we must set the Algencan's parameter $\texttt{jcnnzmax} = 3N$, (ii) the sum of the number of nonnull elements in the lower triangle of the Hessian matrix of each constraint is $\texttt{hnnzmax}_{b2} = 3N$, and (iii) the number of nonnull elements in the lower triangle of the matrix given by the transpose of the Jacobian times the Jacobian is $\texttt{hnnzmax}_{b3} = 6N$. The objective function (13.7) has a potential nonlinear interaction between every pair of variables, meaning that its Hessian matrix is potentially dense and its lower triangle may have up to $\texttt{hnnzmax}_{b1} = 3N(3N+1)/2$ nonnull elements. Therefore, we must set the Algencan parameter `hnnzmax` as $\texttt{hnnzmax}_{b1}$ plus $\texttt{hnnzmax}_{b2}$ plus $\texttt{hnnzmax}_{b3}$, i.e., $\texttt{hnnzmax} = 3N(3N+1)/2+3N+6N$. This $O(N^2)$ memory requirement, which together with the $O(N^2)$ time complexity for evaluating the objective function may prevent the application of Algencan to instances with large N, will be tackled soon.

13.1.2 ▪ Tuning problem's subroutines and Algencan parameters

As a starting example, we consider the application of Algencan to an instance of problem (13.6)–(13.8) with $r = 1$, $N = 1{,}000$, and $R = 15$ (density ≈ 0.3). We set $\texttt{epsfeas} = \texttt{epsopt} = 10^{-8}$, $\texttt{efstain} = \sqrt{\texttt{epsfeas}}$, $\texttt{eostain} = \texttt{epsopt}^{1.5}$, $\texttt{efacc} = \sqrt{\texttt{epsfeas}}$, and $\texttt{eoacc} = \sqrt{\texttt{epsopt}}$. We also set `outputfnm = ''`, `specfnm = ''`, and `nvparam = 0`. A solution was found as a result of the second acceleration process, using 3 outer iterations (65 inner iterations, 398 calls to `fsub`, 118 calls to `gsub`, 74 calls to `hsub`, 429 calls to `csub` per constraint in average, 25 calls to `jacsub` per constraint in average, and 8 calls to `hcsub` per constraint in average) and 3.53 seconds of CPU time.

Since we are seeking applications with much larger values of N, we decided to tackle the $O(N^2)$ time and memory requirements mentioned above. The idea is very simple: at a solution, each sphere "touches" no more than other 12 identical spheres (the kissing number in $\mathbb{R}^3$ [85]). This means that, near a solution, most of the spheres pairs are such that spheres are far from each other and do not contribute to the sum in (13.7). The strategy to compute (13.7) efficiently is based on a partitioning of the three-dimensional space into cubes of side $2r$ and consists of (a) assigning each t_i to a cube and (b) computing the terms in the summation in (13.7) associated with every pair (i, j) such that t_i and t_j belong to the same cube or to neighbor cubes. Remaining pairs do not contribute to the sum in (13.7) and can be ignored. See [65] for details. This idea is based on efficient

algorithms developed to reduce the asymptotic computational complexity of the N-body problem [147].

We implemented this idea (which involves modifications in subroutines `fsub`, `gsub`, and `hsub`) and solved the same instance again. The modified subroutines are part of the Algencan distribution (file `chap13-packmol-sparse.f90` within folder `sources/examples/f90/`). From now on, we will call "dense" and "sparse" implementation of problem (13.6)–(13.8) the implementation that computes all terms in (13.7) and the implementation that computes only a few terms, respectively. Since the implementations perform floating point operations in different orders, slightly different results may be expected. In fact, Algencan found a solution as a result of the first acceleration process, using 2 outer iterations (61 inner iterations, 377 calls to `fsub`, 106 calls to `gsub`, 67 calls to `hsub`, 407 calls to `csub` per constraint in average, 26 calls to `jacsub` per constraint in average, and 10 calls to `hcsub` per constraint in average). The highlight is that the number of computed terms in the summation in (13.7) went from $N(N-1)/2 = 499{,}500$ to 6,775 in average (computed over all points at which the objective function was evaluated during the problem resolution). The same reduction applies to the evaluation of the gradient and the Hessian of the objective function. Assuming that computing the objective function and its derivatives is the dominant task, a similar reduction (99%) would also be expected in the elapsed CPU time. This was *not* the case. The elapsed CPU time was 1.45 seconds.

In fact, in the dense implementation of problem (13.6)–(13.8), the computation of the objective function and its derivatives represents 70% of the computational effort (2.46 seconds of CPU time, over a total of 3.53 seconds). The actual reduction, estimated in 99% considering the reduction in the number of computed terms in the summation in (13.7), was in fact 92%, due to the overheads associated with the sparse implementation of (13.7). This explains the overall reduction of 59%, going from the 3.53 seconds of CPU time of the dense implementation to the 1.45 seconds of the sparse implementation.

If we now profile the sparse implementation of problem (13.6)–(13.8), the most expensive task happens to be the factorization of matrices, consuming 74% of the computational effort. Hence, the natural question is, are the matrices of problem (13.6)–(13.8) (Hessian of the Augmented Lagrangian and Jacobian of the KKT system) such that they favor the application of iterative linear systems solvers instead of direct methods (i.e., matrices factorizations)?

Since first- and second-order derivatives were coded, the MA57 subroutine from HSL is available for solving linear systems, and the instance at hand has more than 500 variables, the active set strategy used to solve the Augmented Lagrangian subproblems employs a line-search Newton's method within the faces (see Sections 8.5.1 and 12.8). This is one of the places where linear systems are being solved. The other place is within the acceleration process (see Section 12.10) to solve the KKT system by Newton's method. The alternative that avoids the usage of a direct linear systems solver in the first case is to use a line-search truncated Newton approach, which solves Newtonian systems by conjugate gradients (see Sections 8.5.2 and 12.8). This selection of the inner-to-the-faces method can be achieved with the keyword TRUNCATED-NEWTON-LINE-SEARCH-INNER-SOLVER. Regarding the second place where linear systems are being solved, since the current implementation of Algencan does not consider the possibility of applying the acceleration process in connection with an iterative linear systems solver, the only choice is to inhibit the application of the acceleration process with the keyword SKIP-ACCELERATION-PROCESS.

We modified these two parameters and applied Algencan once again to the same instance. Algencan found a solution using 6 outer iterations (47 inner iterations, 95 calls to

fsub, 67 calls to gsub, 47 calls to hsub, 95 calls to csub per constraint in average, 8 calls to jacsub per constraint in average, and 6 calls to hcsub per constraint in average) and 0.16 seconds of CPU time. In this run of Algencan, approximately 22 times faster than our first trial, the most expensive tasks were the computation of the Hessian of the Augmented Lagrangian, its products by a given vector (main task of the CG method), the linear algebra of the CG method itself, and the evaluation of the objective function and its derivatives.

After having modified the evaluation of the objective function and its derivatives, and having tuned a few parameters of Algencan, it appears that we are now ready to address larger instances of problem (13.6)–(13.8).

13.1.3 ▪ Solving large instances

In the present section, we are illustrating the application of Algencan to a simple packing problem that mimics a real problem in the field of molecular dynamics. This is why we mentioned in the previous subsection that we were interested in instances of problem (13.6)–(13.8) with a "density" of approximately 0.3. For the same reason, we close the section by showing the performance of Algencan in instances with $10^5 \leq N \leq 10^6$, which is of the same order of magnitude of the largest real applications reported in the literature (see [193]).

Before running Algencan (on large-scale instances) with the parameters chosen in the previous subsection, there is a single parameter to adjust: the number hnnzmax of memory positions required to store the lower triangle of the Hessian of the Augmented Lagrangian function. In fact, when a method different from the Euclidean trust-region method is used as part of the Augmented Lagrangian subproblems' solver, hnnzmax must be an upper bound on the number of triplets needed to store the Hessian matrix of the Lagrangian function (instead of the Hessian matrix of the *Augmented* Lagrangian function). (See Section 10.2.11.) Since this is the case of the present numerical experiments, in which we chose a truncated Newton line-search strategy, the value of hnnzmax may be modified. Moreover, the motivation to compute a new value for hnnzmax is that the value computed in the previous subsections (hnnzmax $= 3N(3N+1)/2+3N+6N$) is not suitable for large values of N. Since the transpose of the Jacobian times the Jacobian does not need to be stored any more, $6N$ memory positions can be disregarded and the new value for hnnzmax might be $3N(3N+1)/2+3N$, which is also *not suitable* for large values of N. However, in practice, a much smaller amount of memory is required. Therefore, based on the sparsity of the Hessian of the objective function near a solution discussed above, in the next experiment we heuristically set hnnzmax $= 100N$.

Table 13.1 shows the results and Figure 13.1 illustrates the solutions found. In the table, outit is the number of outer iterations, innit is the total number of inner iterations, fcnt, gcnt, and hcnt are, respectively, the number of calls to subroutines fsub, gsub, and hsub. Finally, ccnt, jcnt, and hccnt are the average number of calls to subroutines csub, jacsub, and hcsub per constraint, and Time is the CPU time in seconds.

Table 13.1. *Computational effort measures of the application of Algencan to large-scale instances of the packing problem (13.6)–(13.8).*

N	R	outit	innit	fcnt	gcnt	hcnt	ccnt	jcnt	hccnt	Time
10^5	70	6	138	532	158	138	532	11	8	92.56
5×10^5	120	6	262	1,000	282	262	1,000	20	17	1,340.11
10^6	150	6	375	1,561	395	375	1,561	24	21	4,506.39

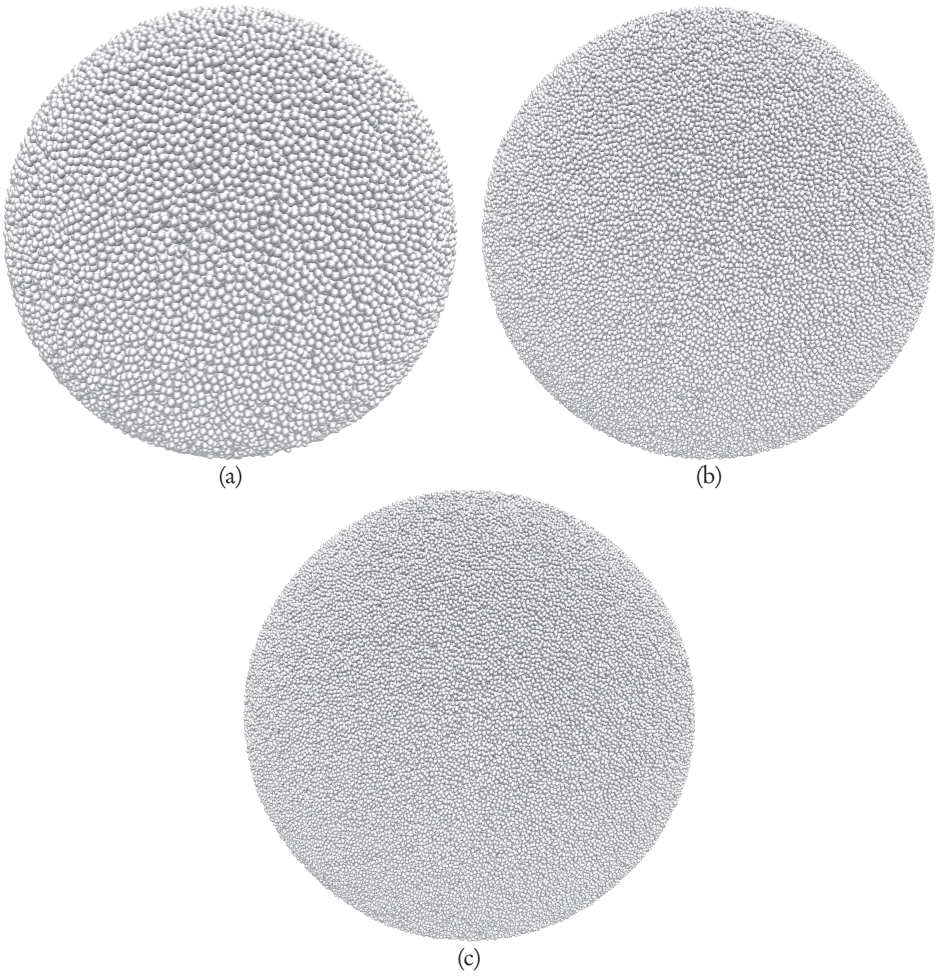

Figure 13.1. *Graphical representation of the solution found to three instances of the unitary-radius (i.e., $r = 1$) packing problem (13.6)–(13.8) with (a) $N = 100{,}000$ and $R = 70$, (b) $N = 500{,}000$ and $R = 120$, and (c) $N = 1{,}000{,}000$ and $R = 150$.*

13.2 ▪ Drawing proportional maps

A sketch of a map of the Americas is presented in Figure 13.2. The sketch was drawn by joining with segments 132 arbitrarily selected points in the borders of 17 regions of a regular map. Most of the regions are associated with countries (from south to north): Argentina, Chile, Uruguay, Brazil, Paraguay, Bolivia, Peru, Ecuador, Colombia, Venezuela, the Guianas (Guyana, Suriname, and French Guiana), Central America (Panama, Costa Rica, Nicaragua, Honduras, El Salvador, Guatemala, and Belize), Mexico, Cuba, Canada, and Alaska. It is a fact that no map can maintain proportionality between areas and distances simultaneously. Here, we wish to redraw the map of the Americas keeping the proportionality among the area of the regions. Moreover, the redrawn map should be as similar as possible to the regular map [185]. As a different project, we wish to draw maps in which the size of each region is proportional to the population of the region or to its gross domestic product (GDP).

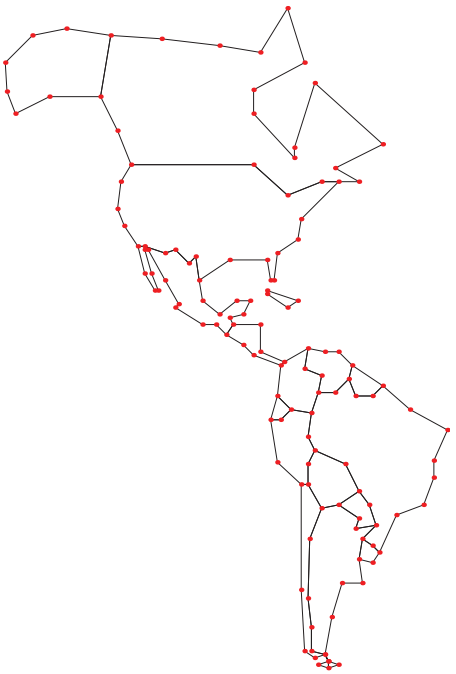

Figure 13.2. *Sketch of a regular map of the Americas. Points were arbitrarily selected in the borders of some regions and borders were approximated by joining the selected points with segments.*

Let $n_r = 17$ be the number of considered regions and let $n_p = 132$ be the number of arbitrarily selected points $\bar{p}_1, \ldots, \bar{p}_{n_p}$ (illustrated in Figure 13.2). For each region j, let o_j be the number of points in its border and let $\gamma_{1j}, \ldots, \gamma_{o_j j}$ be the indices of its vertices (regions are in fact polygons) numbered counterclockwise. The area $\bar{\alpha}_j$ of the polygon that represents region j can be computed (see, for example, [67]) as

$$\bar{\alpha}_j = \sum_{i=1}^{o_j} (\bar{p}^x_{\gamma_{ij}} \bar{p}^y_{\gamma_{i\oplus 1,j}} - \bar{p}^x_{\gamma_{i\oplus 1,j}} \bar{p}^y_{\gamma_{ij}}),$$

where $\bar{p}_i = (\bar{p}^x_i, \bar{p}^y_i)^T$ for $i = 1, \ldots, n_p$, and for each region j, $\gamma_{i\oplus 1,j}$ represents the index of its $(i+1)$th vertex if $i < o_j$ and $\gamma_{o_j \oplus 1,j} \equiv \gamma_{1,j}$. This means that the area of each region j in the considered regular map is approximately $\bar{\alpha}_j$ and the total area of the considered regular map of the Americas is given by $\bar{\alpha} = \sum_{j=1}^{n_r} \bar{\alpha}_j$. Let $\bar{\beta}_j$ be the target area of region j for $j = 1, \ldots, n_r$, and let $\bar{\beta} = \sum_{j=1}^{n_r} \bar{\beta}_j$. These target areas may represent the real territorial areas of the regions, or their population, or the GDP.

13.2.1 ▪ Problem definition: A first approach

We are ready to define our optimization problem, whose variables will be the "new" locations $p_1, \ldots, p_{n_p} \in \mathbb{R}^2$ of the given points $\bar{p}_1, \ldots, \bar{p}_{n_p}$. Constraints regarding the propor-

tionality among the regions will be given by

$$\frac{1}{2}\sum_{i=1}^{o_j}(p^x_{\gamma_{ij}}p^y_{\gamma_{i\oplus 1,j}}-p^x_{\gamma_{i\oplus 1,j}}p^y_{\gamma_{ij}})=\beta_j(\bar{\alpha}/\bar{\beta}),\ j=1,\ldots,n_r, \tag{13.9}$$

where the scaling has the purpose of obtaining a new map of the same size of the considered regular map. The objective function, which represents the desire to obtain a map similar to the considered regular map, may be given by

$$\frac{1}{2}\sum_{j=1}^{n_p}||p_j-\bar{p}_j||^2. \tag{13.10}$$

Hence, the problem consists of minimizing (13.10) subject to (13.9). The problem has $n=2n_p$ variables, $m=n_r$ (equality) constraints, and no bound constraints.

First- and second-order derivatives are easy to code by hand and since there is no relation (common expressions) between the constraints, we opted for coding subroutines `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, and `hcsub`. The six subroutines are part of the Algencan distribution (file `chap13-america-areas.f90` within folder `sources/examples/f90/`). As a starting guess, we consider the natural choice $p_i=\bar{p}_i$ for $i=1,\ldots,n_p$.

A relevant comment concerns the computation of the gradient and Hessian of each constraint in (13.9). Constraint j involves points (variables) $p_{\gamma_1,j},\ldots,p_{\gamma_{o_j},j}\in\mathbb{R}^2$. Each point appears twice (interacting with the previous and the next vertex of the polygon that represents the jth region), and since the constraint is a sum, it is natural to think of each partial derivative as the sum of the partial derivatives of the two terms where each variable appears. Since subroutines `jacsub` and `hcsub` must compute sparse gradients and Hessians, respectively, this reasoning leads us to conclude that the user should implement (within those subroutines) the sum of these particular sparse arrays and matrices. Another option is to use the possibility of representing an element of a sparse array or matrix as the sum of several triplets in the sparse structure built up by the user-provided subroutine. By this, we mean that if, for example, the sparse Hessian of constraint j computed by subroutine `hcsub` contains two different entries k_1 and k_2 saying

```
hcrow(k1) = r, hccol(k1) = c, hcval(k1) = v,
```

and

```
hcrow(k2) = r, hccol(k2) = c, hcval(k2) = w,
```

Algencan understands this as $[\nabla^2 c_j]_{rc}=v+w$. This interpretation eliminates the requirement of computing the sum of sparse arrays and/or matrices within the user-provided subroutine, simplifying the user coding task. The price to be paid is that Algencan might need to save and manipulate sparse structures with a larger number of elements. This may be undesirable in critical cases, but in the present problem, in which the number of variables and constraints is small, it appears to be a very convenient choice.

It is not hard to see that the number of memory positions needed to save the Jacobian of the constraints is `jcnnzmax` $=\sum_{j=1}^{n_r}4o_j$. The number `hnnzmax` of memory positions needed to save the Hessian of the Augmented Lagrangian is given by `hnnzmax1` $=2n_p$ for the (diagonal) Hessian of the objective function plus `hnnzmax2` $=\sum_{j=1}^{n_r}2o_j$ to save, simultaneously, the lower triangles of the Hessians of all constraints plus `hnnzmax3` $=$

$\sum_{j=1}^{n_r}(4o_j)(4o_j+1)/2$ to save the lower triangle of $c'(x)^T c'(x) = \sum_{j=1}^{n_r} \nabla c_j(x) \nabla c_j(x)^T$, i.e., $\texttt{hnnzmax} = 2\,n_p + \sum_{j=1}^{n_r} 2o_j + \sum_{j=1}^{n_r}(4o_j)(4o_j+1)/2$.

The amount of memory `hnnzmax` described in the paragraph above is the one required by Algencan to compute the Hessian matrix of the Augmented Lagrangian (see, for example, (12.3)). This matrix needs to be computed and factorized if the trust-region approach is used for solving the Augmented Lagrangian subproblems (which is the default choice for small problems with available second-order derivatives and an available linear systems solver). On the other hand, if the Newton line-search approach is used to solve the Augmented Lagrangian subproblems, the matrix to be computed and factorized is the one in (12.5). In this case, the quantity `hnnzmax3` is not needed and can be dismissed in the computation of `hnnzmax`. The same remark applies if a truncated Newton approach is used to solve the Augmented Lagrangian subproblems (see Section 12.8 for details). In any case, since the value of `hnnzmax` must be an upper bound on the required number of memory positions, the one computed above may be used in combination with any algorithmic possibility for solving the Augmented Lagrangian subproblems, except in large-scale cases in which sharp upper bounds on the memory requirements may be necessary.

Setting the target values $\bar{\beta}_j$ as the real territorial areas of the 17 considered regions, we are ready to solve the problem with Algencan. (Constants that define the problem can be found in the source file `chap13-america-areas.f90` within folder `sources/examples/f90/`, that accompanies the Algencan distribution. The constants are in fact defined in the module `modamerica.f90`, within the same folder.) We set $\texttt{epsfeas} = \texttt{epsopt} = 10^{-8}$, $\texttt{efstain} = \sqrt{\texttt{epsfeas}}$, $\texttt{eostain} = \texttt{epsopt}^{1.5}$, $\texttt{efacc} = \sqrt{\texttt{epsfeas}}$, and $\texttt{eoacc} = \sqrt{\texttt{epsopt}}$. We also set `outputfnm = ''`, `specfnm = ''`, and `nvparam = 0`. A solution was found as a result of the first acceleration process, using 8 outer iterations (19 inner iterations, 43 calls to `fsub`, 48 calls to `gsub`, 21 calls to `hsub`, 51 calls to `csub` per constraint in average, 48 calls to `jacsub` per constraint in average, and 21 calls to `hcsub` per constraint in average) and 0.05 seconds of CPU time. Figures 13.3(a) and 13.3(b) show, respectively, the considered regular map of the Americas and the map redrawn with sizes proportional to the real area of each region.

Note that the map in Figure 13.3(b) is indeed a map (no crossing segments). We see this as a bonus because there is no constraint in the model requesting the solution to be associated with the picture of a map. This bonus comes from the fact that the regular map is similar to the redrawn map, and using it as a starting point, the final solution turns out to be associated with a map . If the target values are replaced by the population or the GDP of each region, the regular map is far from a solution and the solution of the optimization problem described above is not a map anymore. For those cases a new model is needed.

13.2.2 ▪ Dealing with population and GDP proportional maps

The inconvenience of the model presented in the previous subsection is that the minimization of the objective function (13.10) is not enough to obtain a solution associated with a map (i.e., a set of points that after being joined with segments has no crossing segments). Since adding a small number of constraints to represent exactly this requirement may be hard and adding simple sufficient constraints (like each point p_i to be in a vicinity of $\bar{p}_i$) can make the problem infeasible, we will try with a different objective function. The idea is to minimize the distance of each redrawn region to a scaled, slightly rotated and translated rendering of the region in the regular map.

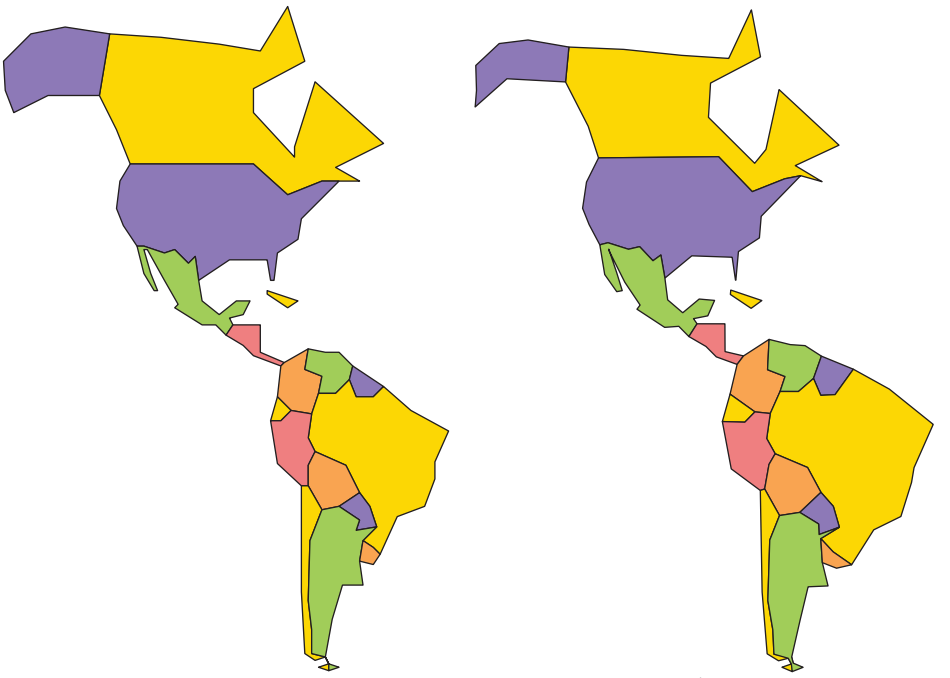

Figure 13.3. *(a) Regular map of the Americas. (b) Map redrawn with sizes proportional to the real area of each region.*

The new model can be written as

$$\begin{aligned}
\text{Minimize} \quad & \frac{1}{2}\sum_{j}^{n}\sum_{i=1}^{o_j}\|p_{\gamma_{ij}} - (t_j + \bar{c}_j + R_j D_j(\bar{p}_{\gamma_{ij}} - \bar{c}_j))\|^2 \\
\text{subject to} \quad & \frac{1}{2}\sum_{i=1}^{o_j}(p^x_{\gamma_{ij}} p^y_{\gamma_{i\oplus 1,j}} - p^x_{\gamma_{i\oplus 1,j}} p^y_{\gamma_{ij}}) = \beta_j(\bar{\alpha}/\bar{\beta}), \quad j = 1,\ldots,n, \\
& -0.1 \leq \theta_j \leq 0.1, \qquad j = 1,\ldots,n, \\
& 0 \leq d^j_k \leq 2, \qquad j = 1,\ldots,n, k = 1,2, \\
& -0.1|\bar{c}^j_k| \leq t^j_k \leq 0.1|\bar{c}^j_k|, \qquad j = 1,\ldots,n, k = 1,2,
\end{aligned} \tag{13.11}$$

where $\bar{c}_j = (1/o_j)\sum_{i=1}^{o_j} \bar{p}_{\gamma_{ij}}$ is the constant "center of mass" of each region j,

$$R_j = \begin{pmatrix} \cos(\theta_j) & -\sin(\theta_j) \\ \sin(\theta_j) & \cos(\theta_j) \end{pmatrix},$$

$D_j = \text{diag}(d^j_1, d^j_2)$, and $t_j \in \mathbb{R}^2$ are a variable (counterclockwise) rotation matrix, a variable deformation, and a variable translation, respectively, for each region j. The objective function in (13.11) says that a redrawn region resembles its usual picture if it is similar to a scaled, slightly rotated and/or translated version of its usual picture. The bound constraints in (13.11) are arbitrary and express our idea of "slightly" rotated and translated.

Problem (13.11) has $n = 2n_p + 5n_r$ variables and $m = n_r$ constraints. Second-order derivatives of the objective function are a little bit harder to code (with respect to the

second derivatives of the objective function of the previous model), but it is our experience that their availability, in general, improves the behavior of Algencan. Note that problems presented up to now in this chapter were solved with the acceleration process, which can be used only if second derivatives are available. The value of `jcnnzmax` is the same as in the previous subsection (since the constraints, other than the bound constraints, are the same), i.e., $\texttt{jcnnzmax} = \sum_{j=1}^{n_r} 4o_j$. The upper bound `hnnzmax` on the number of memory positions required to compute the Hessian matrix of the Augmented Lagrangian is given by $\texttt{hnnzmax} = \texttt{hnnzmax1} + \texttt{hnnzmax2} + \texttt{hnnzmax3}$. Values of `hnnzmax2` and `hnnzmax3` also depend only on the constraints and remain unchanged, i.e., $\texttt{hnnzmax2} = \sum_{j=1}^{n_r} 2o_j$ and $\texttt{hnnzmax3} = \sum_{j=1}^{n_r} (4o_j)(4o_j + 1)/2$. The value of `hnnzmax1` is given by $\texttt{hnnzmax1} = \sum_{j=1}^{n_r} 24o_j$.

Setting the target values $\bar{\beta}_j$ as the GDP of each region, we are ready to solve the problem with Algencan. The corresponding file that accompanies the Algencan distribution is `chap13-america-pop-gdp.f90` (within folder `sources/examples/f90/`). Setting the Algencan parameters with the values of the previous subsection, Algencan finds a solution as a result of the acceleration process after 17 outer iterations. Figure 13.4(a) shows the solution. In fact, Algencan reaches the required precisions to launch the acceleration process after the tenth outer iteration. However, although the required feasibility and optimality tolerances are almost obtained in every acceleration process, they are strictly satisfied only after the seventh trial. As a whole, Algencan uses 3.90 seconds of CPU time, 17 outer iterations, 839 inner iterations, 1,700 calls to `fsub` subroutine, 1,151 calls to `gsub` subroutine, 911 calls to `hsub` subroutine, 1,890 calls to subroutine `csub` per constraint in average, 1,151 calls to subroutine `jacsub` per constraint in average, and 911 calls to subroutine `hcsub` per constraint in average.

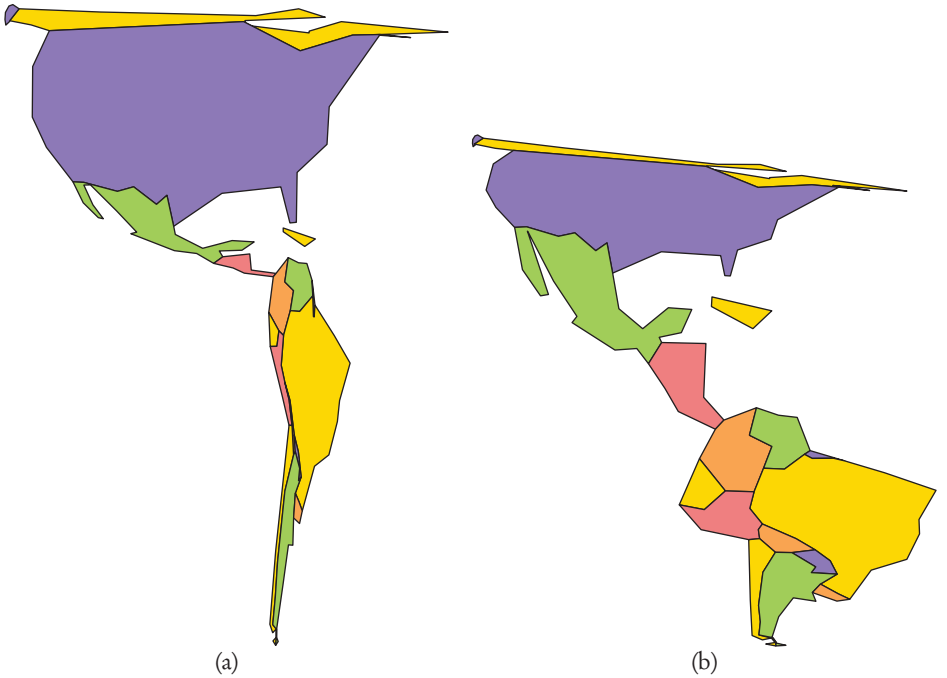

Figure 13.4. *(a) Map redrawn with sizes proportional to the GDP of each region and (b) map redrawn with sizes proportional to the population of each region.*

The behavior described in the paragraph above suggests that increasing the maximum number of iterations of the acceleration process (whose value is 10 by default) may improve the performance of Algencan. Setting this value to 100 with the help of the keyword ACCELERATION-PROCESS-ITERATIONS-LIMIT, we run Algencan again. This time, the same solution was found in the first acceleration process, using 55 iterations (of the Newton's method applied to the KKT system). As a whole, Algencan used 2.97 seconds of CPU time, 10 outer iterations, 693 inner iterations, 1,345 calls to `fsub` subroutine, 928 calls to `gsub` subroutine, 742 calls to `hsub` subroutine, 1,502 calls to subroutine `csub` per constraint in average, 928 calls to subroutine `jacsub` per constraint in average, and 742 calls to subroutine `hcsub` per constraint in average.

Only as an illustrative example of the expected difference in the performance of Algencan when second derivatives are not provided, we solved the same problems without considering the coded Hessians. A solution was found after 23 outer iterations. The solver of the Augmented Lagrangian subproblems achieved its maximum number of iterations when solving the subproblems of the second and third outer iterations. It also stopped by lack of progress when solving the subproblems of outer iterations 11 to 17. As a whole, Algencan used 29,984 calls to `fsub`, 6,918 calls to `gsub`, 30,074 calls to `csub` in average, 6,918 calls to `jacsub` in average, and 17.11 seconds of CPU time (four times the time required when using second-order derivatives). Of course, sometimes, second derivatives may not be available and running Algencan without them is the only possible choice. In such a situation, modifying the relatively strict values of parameters `epsfeas` and `epsopt` used in the present example may be a reasonable course of action.

To end this section, we show in Figure 13.4(b) the result when the target values $\bar{\beta}_j$ are the population of each region. Algencan used 13 outer iterations (429 inner iterations, 1,001 calls to `fsub` subroutine, 621 calls to `gsub` subroutine, 472 calls to `hsub` subroutine, 1,110 calls to subroutine `csub` per constraint in average, 621 calls to subroutine `jacsub` per constraint in average, and 472 calls to subroutine `hcsub` per constraint in average) and 2.38 seconds of CPU time.

13.3 ▪ Optimal control

Optimal control deals with the problem of finding a control law for a given system such that some functional cost is minimized. The problem includes state and control variables. The state variables (which describe the physical system) are solutions of a differential system of equations and the control variables should be chosen in order to optimize some criterion defined by the cost. The differential equations are known as dynamic constraints. The system may also include "path constraints" and boundary conditions. Many relevant engineering problems may be described by the control framework. Typical examples involve minimizing traveling times or fuel consumption of vehicles, from ordinary cars to rockets and satellites.

In this section, we consider control problems of the form

$$\begin{aligned} \text{Minimize} \quad & \int_{t_0}^{t_f} f_0(s(t), u(t))\, dt \\ \text{subject to} \quad & \dot{s}(t) = F(s(t), u(t)), \\ & s(t_0) = s_0, \end{aligned} \tag{13.12}$$

where the state variable is $s(t) \in \mathbb{R}^{n_s}$, $\dot{s} = ds/dt$, the control variable is $u(t) \in \mathbb{R}^{n_u}$, t varies between t_0 and t_f, $f_0 : \mathbb{R}^{n_s} \times \mathbb{R}^{n_u} \to \mathbb{R}$, and $F : \mathbb{R}^{n_s} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_s}$. The initial state is given by $s_0 \in \mathbb{R}^{n_s}$.

13.3.1 ▪ Discretization of the optimal control problem

Frequently, shooting techniques [21, 243] are employed for solving control problems. Unfortunately, shooting procedures are prone to severe ill-conditioning and usually require very good initial approximations [161]. A popular approach to overcoming these inconveniences consists of discretizing the domain $[t_0, t_f]$ with discretization of the derivatives of $s(t)$ and the conversion of the problem into a finite-dimensional optimization problem with as many dynamic constraints as discretization points [137].

We subdivide the time domain $[t_0, t_f]$ into N intervals with equidistant points $t_i = t_{i-1} + \Delta t$ or, equivalently, $t_i = t_0 + i\ \Delta t$, $i = 0, \ldots, N$, where $\Delta t = (t_f - t_0)/N$ and, hence, $t_N = t_f$. Considering the Euler discretization scheme $s_{i+1} = s_i + \Delta t F(s_i, u_i)$ and approximating the integral in the objective function of (13.12) by its Riemann sum, we arrive at the discretized optimal control problem

$$\begin{aligned} &\text{Minimize} \quad \Delta t \sum_{i=0}^{N-1} f_0(s_i, u_i) \\ &\text{subject to} \quad s_{i+1} = s_i + \Delta t F(s_i, u_i), i = 0, \ldots, N-1, \end{aligned}$$

where s_0 is given, the variables s_i approximate the states $s(t_i)$ for $i = 1, \ldots, N$, and the variables u_i approximate the controls $u(t_i)$ for $i = 0, \ldots, N-1$. The number of variables is $n = (n_s + n_u)N$ and the number of (equality) constraints is $m = n_s N$. Higher-order discretization schemes, such as the ones in the Runge–Kutta family of methods, can also be used.

13.3.2 ▪ Van der Pol system with unbounded control

As a simple example, we consider the van der Pol system with unbounded control (see, for example, [160]) given by

$$\begin{aligned} \dot{x}(t) &= y(t), \\ \dot{y}(t) &= -x(t) - (x(t)^2 - 1)y(t) + u(t), \end{aligned}$$

where $t \in [0, 1]$, $s(t) \equiv (x(t), y(t))^T \in \mathbb{R}^2$ describes the state of the system and $u(t) \in \mathbb{R}$ is the control. The aim is to minimize

$$\frac{1}{2} \int_0^1 (x(t)^2 + y(t)^2 + u(t)^2)\, dt.$$

The initial condition is given by $s_0 = (-2, 4)^T$.

The discretized optimal control problem (as described in the previous subsection) is given by

$$\begin{aligned} &\text{Minimize} \quad \frac{1}{2}\Delta t \sum_{i=0}^{N-1} (x_i^2 + y_i^2 + u_i^2) \\ &\text{subject to} \quad x_{i+1} = x_i + \Delta t\ y_i, \qquad\qquad i = 0, \ldots, N-1, \\ &\qquad\qquad y_{i+1} = y_i + \Delta t\ (-x_i - (x_i^2 - 1)y_i + u_i), \quad i = 0, \ldots, N-1, \end{aligned} \tag{13.13}$$

where $x_0 = -2$ and $y_0 = 4$ are constants. This problem has $n = 3N$ variables ($x_i, y_i, i = 1, \ldots, N$ and $u_i, i = 0, \ldots, N-1$) and $m = 2N$ constraints.

An equivalent formulation of (13.13) that is usually considered in practice is given by

$$\begin{array}{lll} \text{Minimize} & z_N & \\ \text{subject to} & x_{i+1} = x_i + \Delta t\, y_i, & i = 0,\ldots,N-1, \\ & y_{i+1} = y_i + \Delta t\, (-x_i - (x_i^2 - 1)y_i + u_i), & i = 0,\ldots,N-1, \\ & z_{i+1} = z_i + \Delta t\, (x_i^2 + y_i^2 + u_i^2)/2, & i = 0,\ldots,N-1, \end{array} \tag{13.14}$$

where $x_0 = -2$, $y_0 = 4$, and $z_0 = 0$ are constants. Problem (13.14) has $n = 4N$ variables $(x_i, y_i, z_i, i = 1,\ldots,N$, and $u_i, i = 0,\ldots,N-1)$ and $m = 3N$ constraints.

Modern optimization methods for solving problems of this type may be found in [160] and [27]. Moreover, computationally more challenging versions of this problem can be considered by imposing constraints on the state and control variables.

13.3.3 ▪ A first run

To illustrate the usage of Algencan, we coded subroutines `fsub`, `gsub`, `hsub`, `csub`, `jacsub`, and `hcsub` for problems (13.13) and (13.14). Coded subroutines are part of the Algencan distribution and can be found as files `chap13-control.f90` and `chap13-control2.f90` (within folder `sources/examples/f90/`) for problems (13.13) and (13.14), respectively. In both cases first and second derivatives are very simple. Computing the values of parameters `jcnnzmax` and `hnnzmax` is also simple and possible values are `jcnnzmax` $= 7N$ and `hnnzmax` $= 21N$ for problem (13.13) and `jcnnzmax` $= 12N$ and `hnnzmax` $= 37N$ for problem (13.14). Setting the remaining parameters of Algencan as `epsfeas` $=$ `epsopt` $= 10^{-8}$, `efstain` $= \sqrt{\texttt{epsfeas}}$, `eostain` $=$ `epsopt`$^{1.5}$, `efacc` $= \sqrt{\texttt{epsfeas}}$, and `eoacc` $= \sqrt{\texttt{epsopt}}$, `outputfnm = ''`, `specfnm = ''`, and `nvparam` $= 0$, we are ready to run our first examples.

In a first set of numerical experiments, we considered $N \in \{10, 100, 1{,}000, 2{,}000, 3{,}000\}$. Table 13.2 shows some figures. In all cases Algencan found a solution with the required precision, and solutions were found as the result of the acceleration process. The last column in the table corresponds to the objective function value at the solution. The values of `cnorm`, `snorm`, and `nlpsupn` as defined in (10.26) are not shown because of lack of space, but they satisfy the stopping criterion of the acceleration process, i.e., they fulfill (10.21), (10.22). As expected, solutions to both models coincide. Regarding the remaining columns in the table, outit is the number of outer iterations, innit is the total number of inner iterations, fcnt, gcnt, and hcnt are, respectively, the number of calls to subroutines `fsub`, `gsub`, and `hsub`. Finally, ccnt, jcnt, and hccnt are the average number of calls to subroutines `csub`, `jacsub`, and `hcsub` per constraint, and Time is the CPU time in seconds.

The figures in Table 13.2 call attention to all instances of model (13.13) and to the last instance of model (13.14) in which the ratio between the number of functional evaluations (fcnt) and the number of inner iterations (innit) is relatively high. This is a symptom of poor descent directions when trying to solve the Augmented Lagrangian subproblems. Along those directions, many functional evaluations are spent in painful backtracking processes with a very slow decrease of the objective function. Many times the backtracking processes and, in consequence, the solver of the Augmented Lagrangian subproblems stop because of lack of progress in the objective function value or a very small step in the line search. This kind of behavior may be related to ill-conditioned (sub)problems. (However, if you observe a similar behavior of Algencan in your problem, the first step is to check the derivatives of the objective function and the constraints!) A detailed output to check the behavior of the subproblems' solver may be obtained with the keyword ITERATIONS-OUTPUT-DETAIL followed by, for example, the printing code number 19.

Table 13.2. *Computational effort measures of the application of Algencan to small and medium-scale instances of the control problems (13.13) and (13.14).*

Optimal control model (13.13)										
N	outit	innit	fcnt	gcnt	hcnt	ccnt	jcnt	hccnt	Time	$f(x^*)$
10	15	99	603	176	104	642	176	102	0.03	4.6139D+00
100	14	110	559	188	116	600	188	114	0.19	5.4477D+00
1,000	45	979	10,889	1,142	982	10,956	1142	979	18.19	5.5349D+00
2,000	24	142	1,120	326	239	1,147	326	234	18.29	5.5397D+00
3,000	20	105	861	213	145	883	213	140	32.24	5.5413D+00

Optimal control model (13.14)										
N	outit	innit	fcnt	gcnt	hcnt	ccnt	jcnt	hccnt	Time	$f(x^*)$
10	9	60	98	100	65	112	100	63	0.01	4.6139D+00
100	9	80	136	125	86	154	125	84	0.15	5.4477D+00
1,000	7	63	99	94	68	108	94	66	4.90	5.5349D+00
2,000	8	56	98	90	61	108	90	59	18.86	5.5397D+00
3,000	9	342	2,236	381	347	2,249	381	326	476.89	5.5413D+00

The most significant digit equal to 1 means that a single line will be printed for each Augmented Lagrangian (outer) iteration. The less significant digit equal to 9 means that you will get all possible details of the iterations of the subproblems' solver (inner iterations) (see Section 12.2). In order to check the coded derivatives against finite difference approximations, set the logical parameter `checkder` equal to true.

In the particular case of the instances mentioned in the paragraph above, the solver of the Augmented Lagrangian subproblems stops because of lack of progress in most of the subproblems. The meaning of lack of progress (for the subproblems' solver) embedded in Algencan is related to the progress in the objective function value, the norm of its projected gradient, and the size of the step (difference between consecutive iterates). If those quantities appear to be stuck for a certain number of consecutive iterations, the lack of progress is characterized and the subproblems' solver stops. However, the constants associated with determining that some of those values are "stuck" and the amount of consecutive iterations with "no progress" that must occur in order to characterize the lack of progress is arbitrary. Modifying those constants to determine the lack of progress in advance greatly improves the performance of Algencan in those instances. Unfortunately, in some other situations, a premature stop by lack of progress in the subproblems may impair the overall performance of Algencan.

13.3.4 ▪ Early acceleration

The figures in Table 13.2 do not point to a clear advantage of one of the models over the other. (Note that comparing the two models is completely outside the scope of these experiments.) However, the performance of Algencan for solving the control problem is clearly model dependent. When solving instances of model (13.14), solutions are always found in the first acceleration process. (The same does not happen with instances of model (13.13).) Moreover, in those cases, most of the time is spent in the Augmented Lagrangian iterations that precede the acceleration process. The question arises of whether to launch the acceleration process at the very beginning. Note that (refer to Section 10.2.4) setting the threshold parameters $\texttt{efacc} = \sqrt{\texttt{epsfeas}}$ and $\texttt{eoacc} = \sqrt{\texttt{epsopt}}$ (as we did in these numerical experiments) has no practical effect, since it reduces to the already default choice of Algencan, which is to start launching the acceleration process after hav-

ing achieved half the required feasibility and optimality tolerances. In the following experiment, we show the behavior of Algencan under a different choice: launch the acceleration process from the beginning, i.e., starting at the given initial point and even previously to the first Augmented Lagrangian iteration. As described in Section 10.2.4, this run of Algencan can be obtained by setting parameters `efacc` and `eoacc` with large values like `efacc` = `eoacc` = 10^{20}. Table 13.3 shows the results.

Table 13.3. *Computational effort measures of the application of Algencan, with early launching of the acceleration process, to large-scale instances of the control problem model (13.14).*

N	nwtkktit	fcnt	gcnt	hcnt	ccnt	jcnt	hccnt	Time	$f(x^*)$
10	8	10	11	8	10	11	7	0.00	4.6139D+00
100	10	12	13	10	12	13	9	0.02	5.4477D+00
1,000	10	12	13	10	12	13	9	0.12	5.5349D+00
2,000	10	12	13	10	12	13	9	0.23	5.5397D+00
3,000	9	11	12	9	11	12	8	0.31	5.5413D+00
10,000	9	11	12	9	11	12	8	1.03	5.5436D+00
100,000	8	10	11	8	10	11	7	9.00	5.5445D+00
1,000,000	7	9	10	7	9	10	6	88.26	5.5447D+00

Neither outer nor inner iterations of Algencan are done to solve the set of problems considered in Table 13.3. Therefore, in the table we show nwtkktit, the number of iterations of the Newton method applied to the KKT system of the problem. The largest problem in Table 13.2 is the one with $N = 3{,}000$. (N is the number of (sub)intervals in the discretization of the optimal control problem, the number of variables is $n = 4N$, and the number of constraints is $m = 3N$.) Figures in Tables 13.2 and 13.3 show that this problem is solved between three and four orders of magnitude faster with this setting of Algencan parameters than with the previous setting. Moreover, Table 13.3 shows additional instances with $N \in \{10^4, 10^5, 10^6\}$ from which it is very clear that the required CPU time grows linearly with respect to N. Once again, Algencan satisfied the required precision in all the instances and the obtained values of the objective function coincide with those reported in Table 13.2. Figure 13.5 illustrates the solution to the optimal control problem (found when solving the instance with $N = 1{,}000$ of model (13.14)).

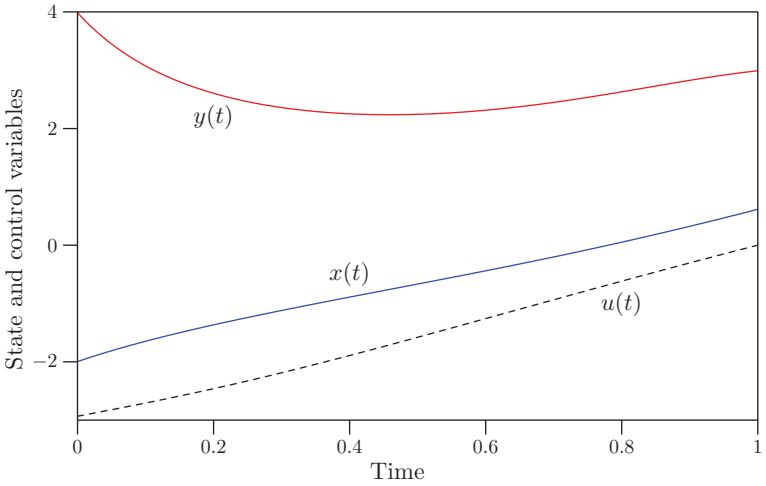

Figure 13.5. *Solution to the optimal control problem.*

13.4 ▪ Grid generation

Grid generation (also called mesh generation) is the art of generating a polygonal or polyhedral mesh that approximates a geometric domain. Typical uses are the representation of a 3D surface on a computer screen or the generation of meshes for finite element computations in computational fluid dynamics.

A curvilinear grid or structured grid is a grid with the same combinatorial structure as a regular grid, in which the cells are quadrilaterals or cuboids rather than rectangles or rectangular parallelepipeds.

Optimal, or well-behaved, meshes are crucial for the solution of partial differential equations in two-dimensional complex regions. A close correlation between the mesh quality and the errors obtained when solving elliptic equations using different meshes has been observed in [216].

Variational methods to generate good conforming meshes on curved regions have shown some difficulties for complicated regions, producing folded, crimped, or rough meshes [216]. Castillo [76] showed that optimizing a combination of several criteria produces better-behaved meshes than optimizing any single criterion alone. In [216], two criteria were employed for the generation of conforming logically rectangular meshes in two-dimensional (2D) curved regions: Minimization of the sum of the squares of all the cell sides and minimization of the sum of squares of all the cell areas.

For a 2D domain, for which a suitable bijection with a rectangle exists, a discrete set of points in its boundary, in the form $P(i,j)$ with $j \in \{1, n_{\text{ord}}\}$ and $i = 1,\ldots,n_{\text{abs}}$ and $i \in \{1, n_{\text{abs}}\}$ and $j = 1,\ldots,n_{\text{ord}}$, is given. The 2D domain is, in fact, defined by this finite set of points in its boundary. The interior points $(P(i,j), i = 2,\ldots,n_{\text{abs}}-1, j = 2, n_{\text{ord}}-1)$ are the unknowns of the problem. See Figure 13.6. The region in Figure 13.6 corresponds to the example in [76, p. 466]. For completeness and reproducibility of the numerical experiments that will be presented below, it is necessary to know that the boundary of the region is given by a unitary-radius half-circle and a half-ellipse with semiaxes $a = 6$ and $b = 3$.

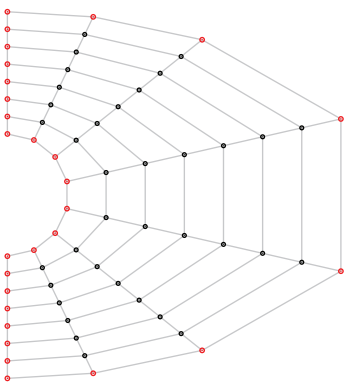

Figure 13.6. *Points in the boundary are given and define the 2D domain. The interior points are the unknowns. The depicted configuration of the interior points is the natural choice of the initial guess for an optimization process.*

The deformed rectangles (cells) have vertices $P(i,j)$, $P(i+1,j)$, $P(i+1,j+1)$, and $P(i,j+1)$ for $i = 1,\ldots,n_{\text{abs}}-1$ and $j = 1,\ldots,n_{\text{ord}}-1$. As is well known, if the Cartesian coordinates of $P(i,j)$ are $p_{ij} = (p_{ij}^x, p_{ij}^y)^T \in \mathbb{R}^2$, the area a_{ij} enclosed by a cell whose vertices in counterclockwise order are $P(i,j)$, $P(i+1,j)$, $P(i+1,j+1)$, and $P(i,j+1)$

is given by

$$\begin{aligned} a_{ij} &= (p^x_{ij}\, p^y_{i+1,j} - p^x_{i+1,j}\, p^y_{ij}) &+ (p^x_{i+1,j}\, p^y_{i+1,j+1} - p^x_{i+1,j+1}\, p^y_{i+1,j}) \\ &\quad + (p^x_{i+1,j+1}\, p^y_{i,j+1} - p^x_{i,j+1}\, p^y_{i+1,j+1}) &+ (p^x_{i,j+1}\, p^y_{i,j} - p^x_{i,j}\, p^y_{i,j+1}). \end{aligned}$$

Let $x = (p^T_{11}, \ldots, p^T_{1,n_{ord}}, p^T_{21}, \ldots, p^T_{2,n_{\text{ord}}}, \ldots p^T_{n_{\text{abs}},1}, \ldots, p^T_{n_{\text{abs}},n_{\text{ord}}})^T \in \mathbb{R}^n$ with $n = 2n_{\text{abs}}n_{\text{ord}}$ and define

$$f_S(x) = \frac{1}{c_S}\left[\frac{1}{2}\sum_{i=1}^{n_{\text{abs}}-1}\sum_{j=1}^{n_{\text{ord}}}\|p_{ij} - p_{i+1,j}\|^2 + \frac{1}{2}\sum_{j=1}^{n_{\text{ord}}-1}\sum_{i=1}^{n_{\text{abs}}}\|p_{ij} - p_{i,j+1}\|^2\right] \tag{13.15}$$

and

$$f_A(x) = \frac{1}{c_A}\left[\frac{1}{2}\sum_{i=1}^{n_{\text{abs}}-1}\sum_{j=1}^{n_{\text{ord}}-1} a_{ij}^2\right], \tag{13.16}$$

where $c_S = (n_{\text{abs}}-1)n_{\text{ord}} + (n_{\text{ord}}-1)n_{\text{abs}}$ and $c_A = (n_{\text{abs}}-1)(n_{\text{ord}}-1)$.

The *unconstrained* optimization problem defined in [216] is given by

$$\text{Minimize } \gamma f_S(x) + (1-\gamma) f_A(x), \tag{13.17}$$

where $\gamma \in [0,1]$ is a given constant. It corresponds to a convex combination of the average of the squared sides of the cells and the average of the squared areas of the cells. Note that the points in the boundary are fixed and (although included in x) are not variables of the optimization problem. Hence, cell sides that correspond to a pair of points in the boundary are fixed too and they were included in the objective function for simplicity only. Figures 13.7(a)–13.7(c) show the solutions to problem (13.17) with $\gamma = 0$, $\gamma = 1$, and $\gamma = 0.1$, respectively, for the elliptical region depicted in Figure 13.6 with a 25×25 grid (i.e., $n_{\text{abs}} = n_{\text{ord}} = 25$). This problem has $23 \times 23 \times 2 = 1{,}058$ variables corresponding to the abscissae and ordinates of the grid inner points.

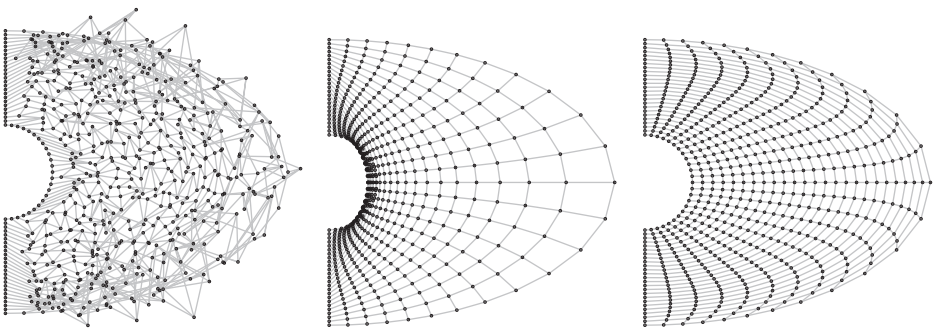

Figure 13.7. *(a) Solution to minimizing the average of the squared areas f_A. (b) Solution to minimizing the average of the squared sides f_S. (c) Solution to minimizing a convex combination with weight $\gamma = 0.1$ for the average of the squared sizes.*

The generated mesh depicted in Figure 13.7(c) illustrates, as observed in [76], that optimizing a combination of the two considered criteria (with the arbitrarily chosen pa-

rameter $\gamma = 0.1$) produces a better behaved mesh than those produced by the optimization of any of the two criteria individually. Observe that meshes in Figures 13.7(a) and 13.7(b) have mesh points outside the 2D region, while the mesh depicted in Figure 13.7(c) does not.

Problem (13.17) is unconstrained and a few words related to its resolution with Algencan are in order. When tackled by Algencan, any unconstrained or bound-constrained problem is solved by Algencan's subproblems' solver Gencan, outside the Augmented Lagrangians framework. Among the available methods for dealing with the subproblems (see Section 12.8), by reasons that will be clear soon, having coded first- and second-order derivatives, we opted for the Newton line-search strategy (keyword NEWTON-LINE-SEARCH-INNER-SOLVER). Other than the choice of the method used to solve the (sub)problem, the two relevant parameters of Algencan in the case of unconstrained or bound-constrained problems are the optimality tolerance ε_{opt} and the maximum number of (inner) iterations. The optimality tolerance corresponds, in the case of unconstrained and bound-constrained problems, to the required tolerance for the sup-norm of the gradient or the projected gradient of the objective function, respectively. In this experiment, it was arbitrarily set to 10^{-8}, i.e., `epsopt = 1.0d-08`. The maximum allowed number of inner iterations is an implicit or additional parameter whose value can be modified with the keyword INNER-ITERATIONS-LIMIT. Its default value, which was not modified in this experiment, is a huge number when the problem is unconstrained or bound constrained.

Regarding the output on the screen, by default Algencan shows a single line of information per Augmented Lagrangian (outer) iteration (see Section 12.2). This means that in the case of unconstrained and bound-constrained problems, in which there are no outer iterations, nothing is shown on the screen during the optimization process. In order to show a single line of information at each iteration of the subproblem's solver (i.e., at each inner iteration), the iterations' output detail should be set to the value 11 using the keyword ITERATIONS-OUTPUT-DETAIL followed by the integer value `11`. The relevant information here is, in fact, the less significant digit. Therefore, any value "ending" with `1` would have the same effect. Moreover, as already explained in Section 12.2, the larger the value of the digit, the larger the amount of information in the output. (The less significant digit corresponds to the inner iterations and the tens digit corresponds to the outer iterations.)

With the settings described in the paragraph above, Algencan found the solution to minimizing $f_A(\cdot)$ depicted in Figure 13.7(a) in 46 inner iterations and using 198 objective function evaluations, 68 gradient evaluations, 46 Hessian evaluations, and 1.01 seconds of CPU time. Naming the solution found as x_A^*, we have that $f_S(x_A^*) \approx 1.87 \times 10^{-1}$ and $f_A(x_A^*) \approx 4.25 \times 10^{-3}$. The solution to minimizing $f_S(\cdot)$ depicted in Figure 13.7(b) was found by Algencan using 2 inner iterations, 3 objective function evaluations, 4 gradient evaluations, 2 Hessian evaluations, and 0.02 seconds of CPU time. Naming the solution found as x_S^*, we have that $f_S(x_S^*) \approx 2.97 \times 10^{-2}$ and $f_A(x_S^*) \approx 1.90 \times 10^{-2}$. The solution to the minimization of the convex combination of both objectives depicted in Figure 13.7(c) was found using 4 iterations of Gencan, 5 objective function evaluations, 6 gradient evaluations, 4 Hessian evaluations, and 0.04 seconds of CPU time. Naming the solution as x_γ^*, we have that $f_S(x_\gamma^*) \approx 4.68 \times 10^{-2}$ and $f_A(x_\gamma^*) \approx 4.91 \times 10^{-3}$. The most relevant information of the numerical results mentioned in this paragraph is that minimizing $f_A(\cdot)$ alone appears to be much more time-consuming than minimizing $f_S(\cdot)$ or a combination of both. It seems as if $f_S(\cdot)$ would play a regularization role in the optimization process.

13.4.1 ▪ Constrained formulations

Consider the problem

$$\begin{array}{ll} \text{Minimize} & \xi_1 f_A(x) + \xi_2 f_S(x) \\ \text{subject to} & \xi_3 f_A(x) + \xi_4 f_S(x) \le \xi_5, \\ & \ell \le x \le u, \end{array} \tag{13.18}$$

where $\xi_1, \ldots, \xi_5 \ge 0$ are given constants. Bound constraints serve only the purpose of fixing the boundary points. This means that for all t, if x_t is a component (abscissa or ordinate) of an interior point, we have $\ell_t = -\infty$ and $u_t = +\infty$, and if x_t is a component of a boundary point, then we have $\ell_t = u_t$. These fixed variables are not variables at all and they are included only to simplify the definition of the problem. As mentioned in Section 12.6, by default Algencan eliminates those variables from the problem to be solved, in a preprocessing stage.

If $\xi_1 + \xi_2 = 1$, for adequate values of ξ_3, ξ_4, and ξ_5 (such as $\xi_3 = \xi_4 = \xi_5 = 0$), problem (13.18) coincides with problem (13.17), which deals with a weighted sum of the objective functions of the biobjective problem of minimizing $f_A(x)$ and $f_S(x)$. If we consider the ϵ-constraint method for multiobjective optimization [184, 141], the problems to be handled are of the form

$$\text{Minimize } f_A(x) \text{ subject to } f_S(x) \le \delta_S \tag{13.19}$$

and

$$\text{Minimize } f_S(x) \text{ subject to } f_A(x) \le \delta_A, \tag{13.20}$$

where δ_S and δ_A are given constants. Both problems are also particular cases of problem (13.18) for suitable (trivial) choices of constants $\xi_1, \ldots, \xi_5$.

The ϵ-constraint method consists of (i) finding a solution x_S^* to minimizing $f_S(x)$ (which corresponds to solving problem (13.18) with $\xi_2 = 1$ and $\xi_1 = \xi_3 = \xi_4 = \xi_5 = 0$), and (ii) solving problem (13.19) with $\delta_S = (1 + \Delta_S) f_S(x_S^*)$ varying $\Delta_S \ge 0$ (which corresponds to solving problem (13.18) with $\xi_1 = \xi_4 = 1$, $\xi_2 = \xi_3 = 0$, and $\xi_5 = \delta_S$). The procedure considering problem (13.20) is analogous.

13.4.2 ▪ Coding the problem

Generating a grid by any of the methods discussed in the previous sections consists of solving one or more instances of problems (13.17), (13.19), and (13.20). All of these are particular cases of problem (13.18). Therefore, it appears that problem (13.18) might be a valuable tool for the generation of meshes. In the rest of this section, we show how to code and solve problem (13.18) using Algencan.

The objective function and the constraint of problem (13.18) differ only in the value of the constants and share *all* nontrivial expressions. Therefore, it is very natural to code them together, opting by coding subroutines `fcsub`, `gjacsub`, and `hlsub` to compute (a) the objective function and the constraint, (b) the gradient of the objective function and the gradient (Jacobian) of the constraint, and (c) the Hessian of the Lagrangian, respectively. See Section 10.2.10.

Problem (13.18) has $n = n_{\text{abs}} n_{\text{ord}}$ variables (although $n_{\text{abs}} n_{\text{ord}} - (n_{\text{abs}} - 1)(n_{\text{ord}} - 1)$ are fixed and will be eliminated by Algencan) and a single inequality constraint. The objective function $f(x)$ is given by

$$f(x) = \xi_1 f_A(x) + \xi_2 f_S(x)$$

and the inequality constraint $c_1(x) \leq 0$ is given by

$$c_1(x) = \xi_3 f_A(x) + \xi_4 f_S(x) - \xi_5.$$

The objective function and the (inequality) constraint are easy to code and do not deserve any special comment. The only relevant decision is that, to avoid thinking in special cases, all $n_{\text{abs}} n_{\text{ord}}$ points in the grid are considered as variables. Then, points in the boundary are fixed (and thus eliminated from the optimization process by Algencan) by appropriately setting their bounds. Variables corresponding to inner points have no bound constraints. The gradients of the objective function and of the constraint are dense n-dimensional arrays. The density of the single-row Jacobian of the constraint will be addressed when dealing with the value of Algencan's parameter `hnnzmax`.

Subroutine `hlsub` must compute the lower triangle of the sparse matrix

$$s_f \nabla^2 f(x) + s_{c_1} \lambda_1 \nabla^2 c_1(x), \tag{13.21}$$

where $\nabla^2 f(x)$ is the Hessian matrix of the objective function $f(x)$ and $\nabla^2 c_1(x)$ is the Hessian matrix of the constraint $c_1(x)$. Parameters s_f, s_{c_1}, and λ_1 are input parameters of subroutine `hlsub` and they correspond to the scaling factor of the objective function, the scaling factor of the constraint, and the Lagrange multiplier of the constraint, respectively (see Section 10.2.10). Since

$$\nabla^2 f(x) = \xi_1 \nabla^2 f_A(x) + \xi_2 \nabla^2 f_S(x)$$

and

$$\nabla^2 c_1(x) = \xi_3 \nabla^2 f_A(x) + \xi_4 \nabla^2 f_S(x),$$

we have that (13.21) coincides with

$$\left(s_f \xi_1 + s_{c_1} \lambda_1 \xi_3\right) \nabla^2 f_A(x) + \left(s_f \xi_2 + s_{c_1} \lambda_1 \xi_4\right) \nabla^2 f_S(x). \tag{13.22}$$

Thus, the simpler way to compute the desired Hessian matrix of the scaled Lagrangian is first to compute $\nabla^2 f_A(x)$ and $\nabla^2 f_S(x)$ and then to multiply them by the constants in (13.22). The possibility of returning more than a single triplet for each matrix element (see Section 10.2.10) releases the user from the cumbersome task of efficiently computing the sum of both matrices.

Needless to say, although first- and second-order derivatives of the problem at hand are simple, coding derivatives (in particular coding sparse Hessians) is prone to error. Therefore, we admit that we made extensive use of the checking derivatives feature of Algencan, setting `checkder = .true.` until obtaining, after several rounds, correct codes for the derivatives.

13.4.3 ▪ Setting Algencan's parameters

By default, without a second thought, we set `epsfeas` = `epsopt` = 10^{-8}, `efstain` = $\sqrt{\texttt{epsfeas}}$, `eostain` = $\texttt{epsfeas}^{1.5}$, `efacc` = $\sqrt{\texttt{epsfeas}}$, `eoacc` = $\sqrt{\texttt{epsopt}}$, `outputfnm = ''`, and `specfnm = ''`. We also set, for a reason that will be elucidated below and is related to the value of parameter `hnnzmax`,

```
nvparam = 1
vparam(1) = 'NEWTON-LINE-SEARCH-INNER-SOLVER'
```

The value of Algencan's input parameter `jcnnzmax`, which must be an upper bound on the number of triplets used to represent the Jacobian of the constraints coded by the user, is set to n, since the gradient of the constraint is dense, i.e., we set `jcnnzmax = n`.

As mentioned in Section 10.2.11, the value of parameter `hnnzmax` depends on which subroutines are coded to represent the problem and on which method is used to solve the (Augmented Lagrangian) subproblems. On the one hand, `hnnzmax` must be an upper bound on the number of triplets needed to store the lower triangle of the Hessian of the scaled Lagrangian computed within subroutine `hlsub`, since this is the way we chose to code the second derivatives of the problem. On the other hand, in addition, extra space may be required to store some information related to the Jacobian $c'(x)$, which, together with the Hessian of the Lagrangian, is needed to obtain the Hessian of the *Augmented* Lagrangian (see (12.3)). If the Euclidean trust-region approach is used to solve the subproblems, then there must be space to store the lower triangle of the (symmetric) matrix $c'(x)^T c'(x)$, which in the present case is a dense $n \times n$ matrix. If the Newton line-search or any other strategy is used to solve the subproblems, no extra space needs to be considered when setting the parameter `hnnzmax`.

The choice of the approach used by Algencan to solve the subproblems is made, by default, by a simple and arbitrary rule based on the number of variables n (see Section 12.8). The Euclidean trust-region approach is used for small problems (the paragraph above explains this choice, at least partially) and the Newton (or truncated Newton) line-search approach is used for large problems. This is why, since we are thinking in large-scale instances of the grid generation problem, for which an $O(n^2)$ memory requirement would not be affordable, we used the Newton line-search approach. Inhibiting the Algencan default choice and fixing the subproblems' solver as the Newton line-search strategy, we are able to set parameter `hnnzmax` as an upper bound on the number of triplets needed to store the lower triangle of the Hessian of the Lagrangian only. Therefore, we set

```
hnnzmaxS = 6 * ( 2 * nabs * nord - nabs - nord )
hnnzmaxA = 36 * ( nabs - 1 ) * ( nord - 1 )
hnnzmax  = hnnzmaxS + hnnzmaxA
```

where `nabs` and `nord` correspond to n_{abs} and n_{ord}, respectively, and `hnnzmaxS` and `hnnzmaxA` correspond to the number of triplets used to store (the lower triangle of) the Hessian matrices of f_S and f_A, respectively. The given values for `hnnzmaxS` and `hnnzmaxA` may not be trivial to see unless you code the Hessians by yourself or at least check the particular implementation of the Hessian of the Lagrangian considered in subroutine `hlsub`. Coded subroutines are part of the Algencan distribution and can be found as file `chap13-grid-ellip.f90` (within folder `sources/examples/f90/`).

13.4.4 ▪ Solving a sequence of ϵ-constrained problems

As an example of the many possibilities of considering problem (13.18) to generate grids, based on the results reported in [76], we considered the problem of minimizing $f_S(x)$ subject to optimality or "near optimality" of $f_A(x)$, i.e., problem (13.20) (or problem (13.18) with $\xi_1 = \xi_4 = 0$, $\xi_2 = \xi_3 = 1$, and $\xi_5 = \delta_A$) with $n_{\text{abs}} = n_{\text{ord}} = 25$, $\delta_A = (1 + \Delta_A) f_S(x_A^*)$, and $\Delta_A \in \{0.0, 0.1, 0.2, \ldots, 0.9\}$, where x_A^* is the solution to minimize $f_A(x)$ (with no constraints) found at the beginning of the present section and depicted in Figure 13.7(a). In the numerical experiments, we considered $f_A(x_A^*) = 4.25251962159732197 \times 10^{-3}$.

Figure 13.8 shows the solution to minimize $f_S(x)$ subject to $f_A(x) \leq f_A(x_A^*)$, i.e., with $\Delta_A = 0$. Of course, the constraint is active at the solution. The value of f_S at the solution is approximately 6.32×10^{-2} and, although it has some irregularities, the generated mesh appears to be similar to the one in Figure 13.7(c). Figure 13.9 shows the solutions to minimize $f_S(x)$ subject to $f_A(x) \leq (1+\Delta_A) f_A(x_A^*)$ for increasing values of Δ_A. As expected, as Δ_A increases, the solutions increasingly resemble the solution to minimize $f_S(x)$ with no constraints, depicted in Figure 13.7(b). Note that, relaxing the optimality of f_A a little bit, the irregularities depicted on Figure 13.8 are avoided.

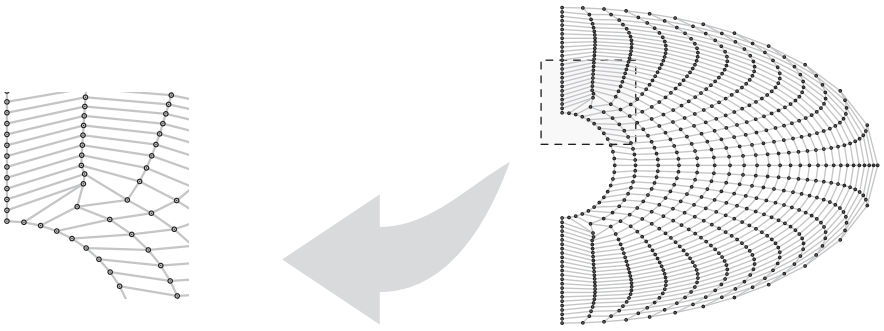

Figure 13.8. *Solution to the constrained problem of minimizing $f_S(x)$ subject to $f_A(x) \leq f_A(x_A^*)$, where x_A^* is the optimal solution of minimizing $f_A(x)$ without constraints. The zoom shows an "irregular" region of the mesh in detail.*

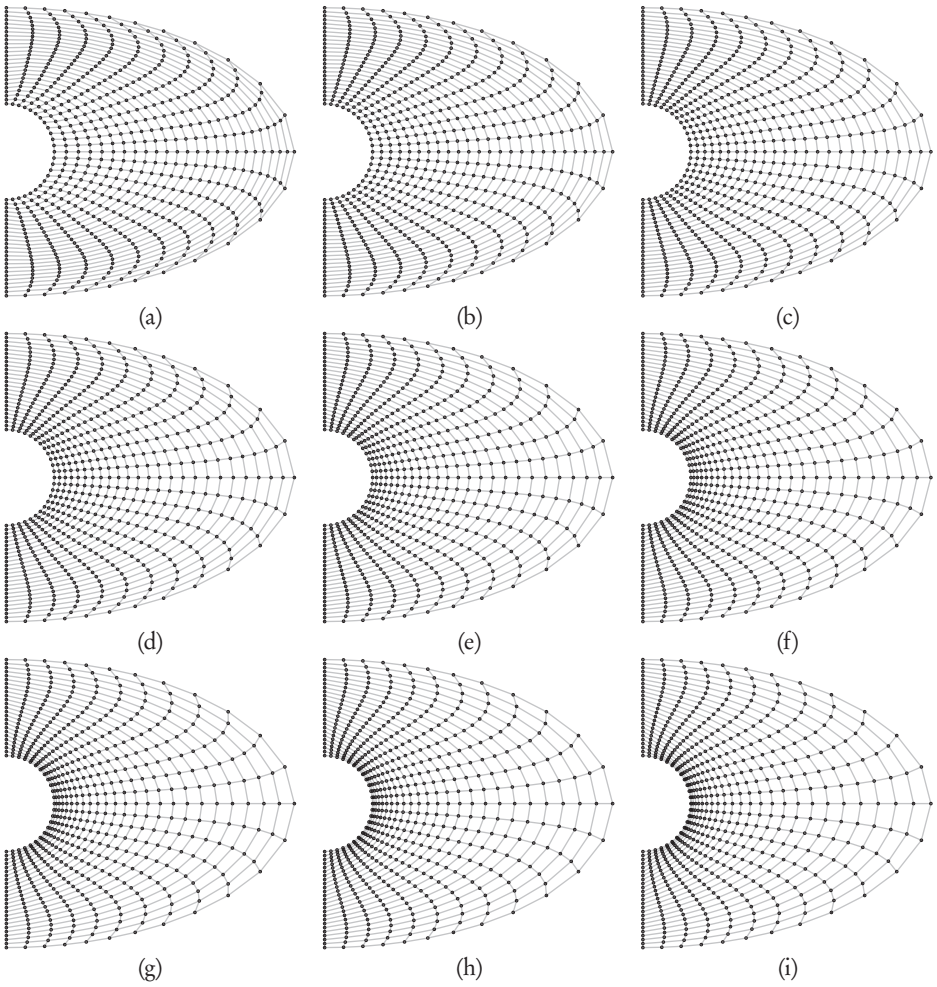

Figure 13.9. *Solutions to the constrained problem of minimizing $f_S(x)$ subject to $f_A(x) \leq (1+\Delta_A)f_A(x_A^*)$, where x_A^* is the optimal solution of minimizing $f_A(x)$ without constraints. Solutions depicted in (a)–(i) correspond to $\Delta_A = 0.1, 0.2, \ldots, 0.9$, respectively.*

Similar results can also be obtained for an 11×11 mesh in the hard horn-shaped 2D region depicted in Figure 13.10, which is known to be more difficult than the previous one, since it has no equal area solution [76]. For the sake of reproducibility, points in the border of the 2D domain are given by (a) a segment from $(1,1)^T$ to $(1,2)^T$ (left); (b) the arc from $(4,0)^T$ to $(4,4)^T$ of a circle centered at $(2.5,2)^T$ with radius 2.5 (right); (c) a parabola of the form $y = ax^2 + bx + c$ with $a = -5/21$, $b = -3.6a$, and $c = 1 + 2.6a$ (bottom); and (d) a parabola with $a = 10/21$, $b = -3.6a$, and $c = 1 + 2.6a$ (top). Figure 13.11 shows the solutions to minimize $f_S(x)$ subject to $f_A(x) \leq (1 + \Delta_A) f_A(x_A^*)$ for increasing values of Δ_A, where x_A^* is the optimal solution to minimizing $f_A(x)$ with no constraints. In the numerical experiments, we considered $f_A(x_A^*) = 1.00260900832775720 \times 10^{-2}$. Analyzing the quality of the generated meshes for this particular problem and determining the most adequate value of Δ_A are outside the scope of this work. Numerical experiments appear to show that problem (13.18) is a useful tool for developing meshes. Source codes related to the horn-shaped 2D region problem depicted in Figure 13.10 are part of the Algencan distribution and can be found as file `chap13-grid-horn.f90` (within folder `sources/examples/f90/`).

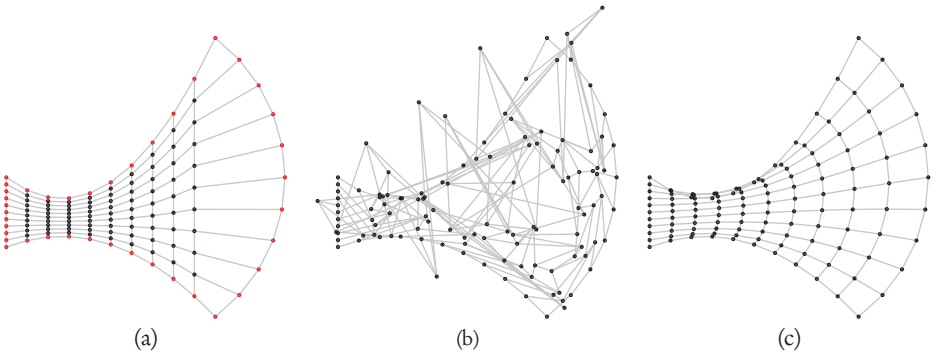

Figure 13.10. *(a) Given boundary points and "natural" initial guess. (b) Solution to minimizing the sum of the squared areas f_A. (c) Solution to minimizing the sum of the squared sides f_S.*

13.4.5 ▪ Solving a larger instance

Both problems considered in the previous section were taken from [76]. Merely increasing the number of points in the mesh (i.e., increasing n_{abs} and/or n_{ord}) is not enough to generate large instances of the optimization problem. This is because, in either of the two problems, if the area of the 2D domain remains fixed while the number of points in the mesh increases, the considered initial guess satisfies the stopping criteria with the prescribed tolerance $\varepsilon_{\text{opt}} = 10^{-8}$. This serves as an alert to the fact that the value 10^{-8} for the optimality tolerance ε_{opt} is an arbitrary choice for a problem-dependent parameter.

In this section, we consider a 100×100 mesh in a scaled version of the horn-shaped region in which the boundary is given by (a) a segment from $(10,10)^T$ to $(10,20)^T$ (left); (b) the arc from $(40,0)^T$ to $(40,40)^T$ of a circle centered at $(25,20)^T$ with radius 25 (right); (c) a parabola of the form $y = ax^2 + bx + c$ with $a = -5/210$, $b = -36a$, and $c = -160a$ (bottom); and (d) a parabola with $a = 1/21$, $b = -36a$, and $c = 40-160a$ (top). The source files are available in the Algencan distribution as file `chap13-grid-horn-large.f90` within folder `sources/examples/f90/`.

Based on the experiments of the previous sections, we aim at first to solve the unconstrained problem of minimizing $f_A(x)$ with no constraints to obtain x_A^* and then to minimize $f_S(x)$ subject to $f_A(x) \leq (1 + 0.1) f_A(x_A^*)$.

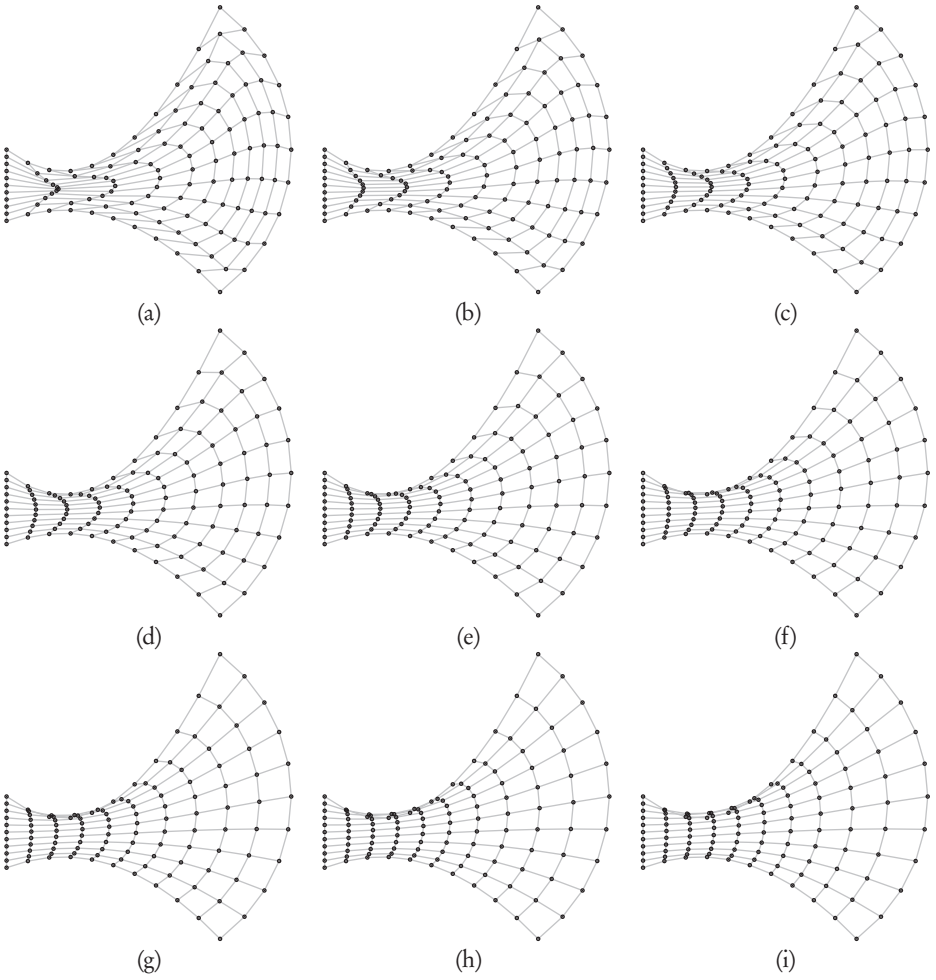

Figure 13.11. *Solutions to the constrained problem of minimizing $f_S(x)$ subject to $f_A(x) \leq (1+\Delta_A)f_A(x_A^*)$, where x_A^* is the optimal solution of minimizing $f_A(x)$ without constraints. Solutions depicted in (a)–(i) correspond to $\Delta_A = 0.1, 0.2, \ldots, 0.9$, respectively.*

The problem of minimizing $f_A(x)$ is an unconstrained problem with $n = 19{,}208$ variables. When tackled by Algencan, it is solved by the bound constraints solver Gencan. Setting $\varepsilon_{\text{opt}} = 10^{-8}$, Gencan took 565 iterations, 3,490 functional evaluations, 841 gradient evaluations, 565 Hessian evaluations, and 1,500.21 seconds of CPU time. Note that, in the case of unconstrained and bound-constrained problems, the number of (inner) iterations and the number of Hessian evaluations coincide if the inner-to-the-face strategy is the Newton's method with line search. At the solution x_A^* found, we have $f_A(x_A^*) \approx 1.04403487484171029 \times 10^{-2}$. Note that it is not very clear whether such precision would be needed in order to be used in the second step of the process of generating the mesh. A rough approximation $\bar{x}_A$, with $f_A(\bar{x}_A) \approx 1.08 \times 10^{-2}$, would have been found in less than half the time (177 inner iterations) setting $\varepsilon_{\text{opt}} = 10^{-4}$.

In the second stage of the mesh generation problem, we solved the problem of minimizing $f_S(x)$ subject to $f_A(x) \leq (1+0.1)f_A(x_A^*)$ with $f_A(x_A^*) = 1.04403487484171029 \times 10^{-2}$.

Algencan solved the problem using 14 outer iterations, 75 inner iterations, 293 calls to subroutine `fcsub`, 167 calls to subroutine `gjacsub`, 105 calls to subroutine `hlpsub`, and 93.19 seconds of CPU time. In the case of this constrained problem, although the inner solver Gencan is using the Newton's method with line searches, the number of Hessians evaluations is larger than the number of inner iterations because some Hessians evaluations are used in unsuccessful (therefore, discarded) trials of the acceleration process. At the solution x_γ^* found, we have $f_S(x_\gamma^*) \approx 8.40936730531235827 \times 10^{-2}$ and $f_A(x_\gamma^*) \approx 1.14843878681980868 \times 10^{-2}$. Figures 13.12 and 13.13 illustrate the solution found. Note that, while the first-stage unconstrained problem appears to be a bit time-consuming (a drawback that can be circumvented by requiring a looser stopping-criterion tolerance), the constrained problem of the second stage is solved by Algencan relatively fast.

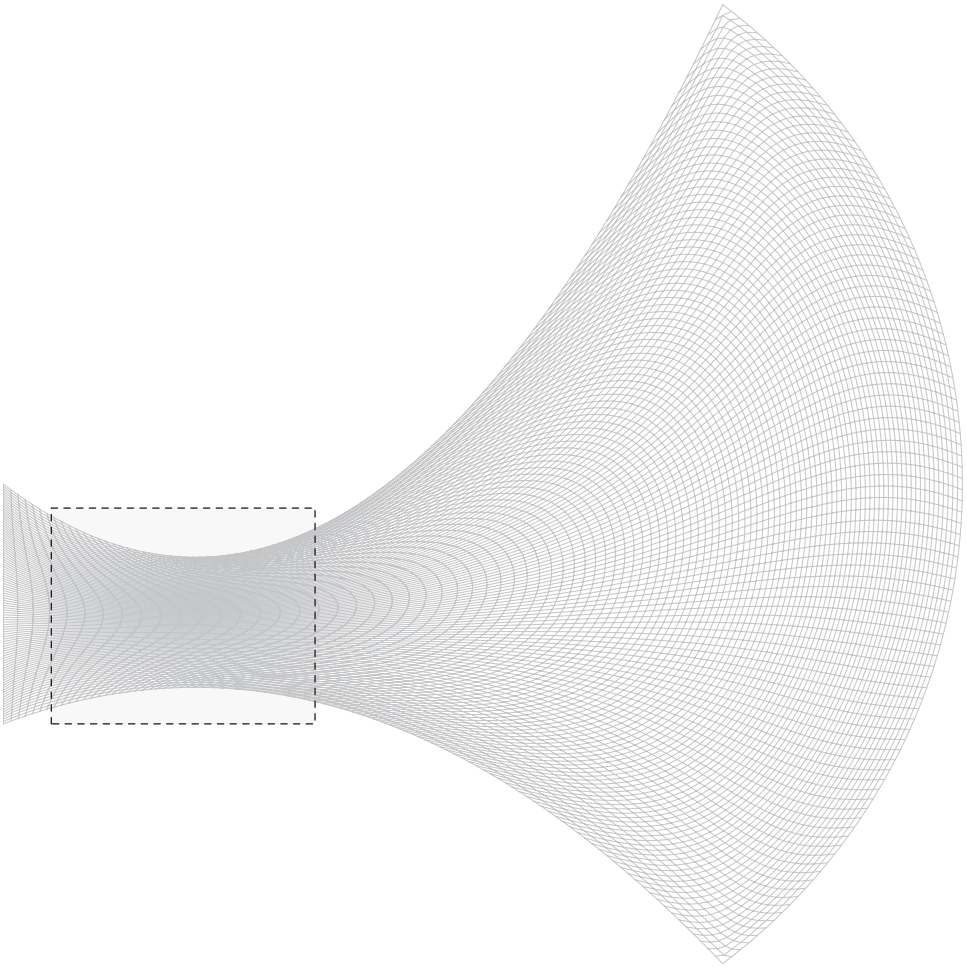

Figure 13.12. *Solution to the large horn-shaped 2D region mesh generation problem.*

13.5 ▪ Review and summary

In this chapter, we tackled practical problems using the Augmented Lagrangian software Algencan. Models were fully described, were the main code and the user-provided sub-

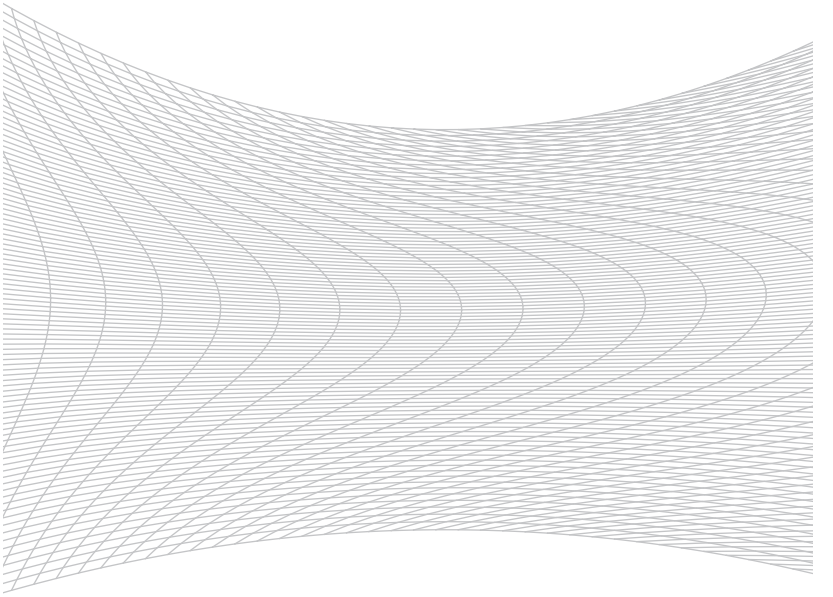

Figure 13.13. *Zoom of the shaded part of Figure 13.12.*

routines needed to solve each problem. Users were oriented toward clever choice of algorithmic parameters. Problems were used as examples, while the main focus was in the use of Algencan.

13.6 ▪ Further reading

The practical application of computing initial points for molecular dynamics simulations is fully described in [187, 193]. The Packmol package implements these ideas and is available at `http://www.ime.unicamp.br/~martinez/packmol/`.

The "sparse" implementation of the nonoverlapping objective function is described in detail in [65]. The problem of map projections appears to be a very controversial subject. In this chapter, we tackled the problem of drawing proportional maps from the optimization point of view, ignoring every possible subject that concerns cartographers, with the single purpose of illustrating the use of Algencan. A complete source of information on many approaches to this subject appears to be the book [241]. The main references to the control and mesh generation problems were given in the text.

13.7 ▪ Problems

13.1 The possibilities for problems in this chapter are enormous. Check the available codes related to the examples described in the present chapter. Reproduce the presented results. Modify the Algencan parameters, rerun, and compare the results.

13.2 Create your own examples. For each problem, tune the Algencan parameters to obtain something that you may consider a good approximation to a solution with a "satisfactory" performance of Algencan.

13.3 The unconstrained collection of Moré, Garbow, and Hillstrom (MGH) [205] describes 35 test functions $f : \mathbb{R}^n \to \mathbb{R}^m$. Use Algencan to solve the 35 feasibility

problems of the form $f(x) = 0$. (Note that $m > n$ is not an issue for the application of Algencan.)

13.4 Consider the 35 test functions $f : \mathbb{R}^n \to \mathbb{R}^m$ in MGH. This time solve the least-squares formulations given by minimizing $\sum_{i=1}^m f_i(x)^2$. Compare the effort needed to code the user-provided subroutines in this case and in the case of Problem 13.3 (see Section 12.9). Compare the results.

13.5 Consider the feasibility problems of Problem 13.3 and minimize the objective function $\sum_{i=1}^n x_i^2$. Add bounds and maximize $\sum_{i=1}^n x_i^2$.

13.6 Combining the test functions in MGH, create your own nonlinear programming problems and solve them with Algencan. Take care to build some feasible problems considering constraints of the form $f(x) \leq f(e)$, where $e = (1, \ldots, 1)^T \in \mathbb{R}^n$.

Chapter 14
Final Remarks

Numerical optimization deals with algorithms to find minimizers of functions on given domains. Strictly speaking, "minimizers" means "global minimizers" because, in most real-life models that use optimization, scientists and decision makers want to approximate the lowest possible value of an objective function subject to different types of constraints. In this context, the concept of "stationary point" or even "local minimizer" makes little sense for most practitioners.

Unfortunately, unless we can exploit specific knowledge about the function and the constraints (convexity, algebraic structure, separability, small-dimension), it is impossible to guarantee, in affordable computational time, that an approximate global minimizer has been found. For this reason, most "affordable algorithms" are designed in order to guarantee convergence to points that satisfy necessary optimality conditions.

In this book, we emphasized the role of AKKT and other sequential optimality conditions, which are satisfied by every local minimizer (independently of regularity properties of the constraints) and seem to be the computable conditions that any practical optimization software may test to declare success of a minimization process. Weak constraint qualifications guarantee that AKKT points satisfy the popular KKT condition and for this reason also play an important role in convergence analysis.

For most applications, obtaining a feasible point is a major goal. This is a difficult objective to accomplish because achieving feasibility is also a global optimization problem. The structural difference between this problem and general global optimization is that we can recognize (up to some precision) whether we have been successful. Mathematically, all the constraints have the same status, but practitioners distinguish between soft and hard constraints. Hard constraints may be "definitions," which need to be fulfilled with high precision in order to preserve the essence of the problem. Evolution constraints, which express the way in which a state variable in the present depends on its value in the past, are constraints of this type. Very small errors in their fulfillment may accumulate and cause disastrous decisions.

Affordable algorithms cannot guarantee finding feasible points without specific knowledge of the structure of the constraints. One can employ an optimization algorithm to solve a problem that has no feasible points and, in this case, all we can ensure is that a stationary point for the infeasibility is found. Moreover, even if a problem is feasible, when there are infeasible stationary points of the infeasibility, these points are natural attractors and, in theory, nonconvergence to them cannot be guaranteed. This is why convergence theories for affordable algorithms generally begin proving convergence to

(feasible or not) stationary points of infeasibility and finish proving (KKT-like) stationarity of the feasible limit points. Substantial variations of this schedule necessarily assume rather strong hypotheses on the problem or on the algorithm. (For example, it could be assumed that infeasible stationary points of infeasibility do not exist, or that a particular subalgorithm of the main algorithm is always successful, or that a sequence of algorithmic internal quantities is naturally bounded without safeguards.)

The success (or failure) of an algorithm for solving a practical problem must be evaluated by the user. It is not uncommon for a practitioner to be quite unhappy with the solution obtained by an optimization algorithm that satisfied convergence criteria with high precision. On the other hand, the situation in which the practitioner finds acceptable a solution painfully obtained and poorly qualified by optimization software is not rare, either. However, in the process of developing algorithms, numerical optimization people need to develop their own criteria for declaring success or failure. Moreover, many times one has to decide which of two or more algorithms has been the most successful for solving a particular (test) problem. Two alternatives are given below:

1. A method is successful when it finds a feasible point (up to some given precision) and satisfies an approximate KKT condition (up to some given precision).

2. A method is successful when it finds a feasible point (up to some given precision) and obtains "the lowest value" of the objective function (among the considered methods).

The argument favoring the first position is that affordable optimization algorithms aim to find not global minimizers but KKT points. Therefore, it should be unfair to consider that an algorithm fails for doing what it is supposed to do. The second position is based on the assumption that finding low values of the objective function among feasible points is the real objective of optimization and that approximate KKT points are merely signals that the real goals are probably accomplished.

Consider the problem

$$\text{Minimize } -x_1 \text{ subject to } x_1 x_2 = 0,\ x_1 + x_2 \geq 0, \text{ and } x_2 \geq (x_1 - 1)^3.$$

All points of the form $(0, x_2)^T$ with $x_2 > 0$ are KKT points and local minimizers of this problem. However, these points are also global maximizers and the global minimizer is the point $(1,0)^T$ that does not satisfy the KKT condition. Of course, being a minimizer, $(1,0)^T$ is an AKKT point. If Algorithm A finds the global minimizer $(1,0)^T$ and Algorithm B finds $(0,1)^T$, the decision of which has been successful (A, B, or both) depends on the alternative chosen above. It can be expected that algorithms that find $(0,1)^T$ should be fast, since all constraint qualifications are satisfied at this point, whereas algorithms that find $(1,0)^T$ may be slow because this is a highly degenerate point. The comparison could be even more controversial if one imagines the existence of a third Algorithm C that converges to $(0.8,0)^T$, a feasible point at which even AKKT does not hold, but where the objective function value is smaller than its value at all KKT points. Although it is clear that Algorithm B performed better than Algorithm C in this problem, the comparison between Algorithms A and C is not obvious.

This example shows how important it is to make evaluation criteria explicit in numerical comparisons. Fortunately, this topic does not seem to be relevant for practical users. In the presence of a real-life problem, practitioners probably have their own criteria for deciding whether the output of optimization software is acceptable or not. On the other hand, their choice of software depends on such variables as clarity, usability, and

maintenance that are not subject to numerical controversy. As a project in permanent development, Algencan depends decisively on the contribution of users. Algencan developers aim for improvements to this software that extend far beyond their mathematical careers through the participation of their students, collaborators, and users of the software. The software that began as an orthodox Augmented Lagrangian algorithm may evolve to incorporate other constrained optimization techniques, as has begun to occur under the Newton acceleration improvements.

Bibliography

[1] J. Abadie and J. Carpentier. Generalization of the Wolfe reduced-gradient method to the case of nonlinear constraints. In R. Fletcher, editor, *Optimization*, pages 37–47. Academic Press, New York, 1968. (Cited on p. 57)

[2] S. Adhikari and R. Baer. Augmented Lagrangian method for order-N electronic structure. *Journal of Chemical Physics*, 115(1):11–14, 2001. (Cited on p. 6)

[3] C. S. Adjiman, I. P. Androulakis, C. D. Maranas, and C. A. Floudas. A global optimization method αBB for process design. *Computers and Chemical Engineering*, 20:S419–S424, 1996. (Cited on p. 45)

[4] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaler. A global optimization method αBB for general twice-differentiable constrained NLPs-I. Theoretical advances, *Computers and Chemical Engineering*, 22:1137–1158, 1998. (Cited on p. 45)

[5] R. R. Allran and S. E. J. Johnsen. An algorithm for solving nonlinear programming problems subject to nonlinear inequality constraints. *The Computer Journal*, 13(2):171–177, 1970. (Cited on p. 37)

[6] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*, volume 9 of *Software, Environments, and Tools*. SIAM, Philadelphia, 1996. (Cited on p. 137)

[7] R. Andrade, E. G. Birgin, I. Chambouleyron, J. M. Martínez, and S. D. Ventura. Estimation of the thickness and the optical parameters of several superimposed thin films using optimization. *Applied Optics*, 47(28):5208–5220, 2008. (Cited on p. 110)

[8] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt. On augmented Lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18(4):1286–1309, 2007. (Cited on pp. 9, 34, 106)

[9] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt. Second-order negative-curvature methods for box-constrained and general constrained optimization. *Computational Optimization and Applications*, 45(2):209–236, 2010. (Cited on pp. 28, 111, 156)

[10] R. Andreani, C. E. Echagüe, and M. L. Schuverdt. Constant-rank condition and second-order constraint qualification. *Journal of Optimization Theory and Applications*, 146(2):255–266, 2010. (Cited on p. 28)

[11] R. Andreani, J. A. R. Flor, J. M. Martínez, and P. J. S. Silva. Constraint qualifications and approximate-KKT sequences. Contributed talk, X Brazilian Workshop on Continuous Optimization, Florianopolis, SC, Brazil, 2014. (Cited on p. 25)

[12] R. Andreani, G. Haeser, and J. M. Martínez. On sequential optimality conditions for smooth constrained optimization. *Optimization*, 60(5):627–641, 2011. (Cited on p. 28)

[13] R. Andreani, G. Haeser, M. L. Schuverdt, and P. J. S. Silva. Two new weak constraint qualifications and applications. *SIAM Journal on Optimization*, 22(3):1109–1135, 2012. (Cited on pp. 25, 28, 29)

[14] R. Andreani, G. Haeser, M. L. Schuverdt, and P. J. S. Silva. A relaxed constant positive linear dependence constraint qualification and applications. *Mathematical Programming*, 135(1-2):255–273, 2012. (Cited on pp. 21, 22, 25, 28, 29)

[15] R. Andreani, J. M. Martínez, L. T. Santos, and B. F. Svaiter. On the behavior of constrained optimization methods when Lagrange multipliers do not exist. *Optimization Methods and Software*, 29(3):646–657, 2014. (Cited on p. 56)

[16] R. Andreani, J. M. Martínez, and M. L. Schuverdt. On the relation between constant positive linear dependence condition and quasinormality constraint qualification. *Journal of Optimization Theory and Applications*, 125(2):473–483, 2005. (Cited on pp. 20, 28)

[17] R. Andreani, J. M. Martínez, and M. L. Schuverdt. On second-order optimality conditions for nonlinear programming. *Optimization*, 56(5–6):529–542, 2007. (Cited on p. 28)

[18] R. Andreani, J. M. Martínez, and B. F. Svaiter. A new sequential optimality condition for constrained optimization and algorithmic consequences. *SIAM Journal on Optimization*, 20(6):3533–3554, 2010. (Cited on pp. 28, 56)

[19] M. Andretta, E. G. Birgin, and J. M. Martínez. Practical active-set Euclidian trust-region method with spectral projected gradients for bound-constrained minimization. *Optimization*, 54(3):305–325, 2005. (Cited on pp. 106, 156)

[20] T. M. Apostol. *Mathematical Analysis: A Modern Approach to Advanced Calculus*. Addison-Wesley, Reading, MA, 1957. (Cited on pp. 15, 62)

[21] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, volume 13 of *Classics in Applied Mathematics*. SIAM, Philadelphia, 1995. (Cited on p. 178)

[22] J. Ashburner, C. Hutton, R. Frackowiak, I. Johnsrude, C. Price, and K. Friston. Identifying global anatomical differences: Deformation-based morphometry. *Human Brain Mapping*, 6(5–6):348–357, 1998. (Cited on p. 7)

[23] C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM Journal on Matrix Analysis and Applications*, 20(2):513–561, 1998. (Cited on p. 91)

[24] C. Audet and J. E. Dennis, Jr. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472, 2009. (Cited on p. 34)

[25] A. Auslender, M. Teboulle, and S. Ben-Tiba. Interior proximal and multiplier methods based on second order homogeneous kernels. *Mathematics of Operations Research*, 24(3):645–668, 1999. (Cited on p. 37)

[26] D. E. Azofeifa, N. Clark, and W. E. Vargas. Optical and electrical properties of terbium films as a function of hydrogen concentration. *Physica Status Solidi (B)*, 242(10):2005–2009, 2005. (Cited on p. 110)

[27] N. Banihashemi and C. Y. Kaya. Inexact restoration for Euler discretization of box-constrained optimal control problems. *Journal of Optimization Theory and Applications*, 156(3):726–760, 2013. (Cited on p. 179)

[28] J. F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*, volume 30 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 1998. (Cited on p. 45)

[29] M. Bartholomew-Biggs. *Nonlinear Optimization with Financial Applications*. Kluwer Academic Publishers, Norwell, MA, 2005. (Cited on p. 13)

[30] J. Barzilai and J. M. Borwein. Two point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988. (Cited on pp. 92, 98)

[31] L. Bello and M. Raydan. Convex constrained optimization for the seismic reflection tomography problem. *Journal of Applied Geophysics*, 62(2):158–166, 2007. (Cited on p. 110)

[32] A. Ben-Tal, I. Yuzefovich, and M. Zibulevsky. Penalty/barrier multiplier methods for minimax and constrained smooth convex programs. Technical report, Optimization Laboratory, Faculty of Industrial Engineering Management, Technion, Haifa, Israel, 1992. (Cited on p. 37)

[33] A. Ben-Tal and M. Zibulevsky. Penalty/barrier multiplier methods for convex programming problems. *SIAM Journal on Optimization*, 7(2):347–366, 1997. (Cited on p. 37)

[34] J. D. Benson, A. J. Kearsley, and A. Z. Higgins. Mathematical optimization of procedures for cryoprotectant equilibration using a toxicity cost function. *Cryobiology*, 64(3):144–151, 2012. (Cited on pp. 8, 9)

[35] F. Benvenuto, R. Zanella, L. Zanni, and M. Bertero. Nonnegative least-squares image deblurring: Improved gradient projection approaches. *Inverse Problems*, 26(2):025004, 2010. (Cited on p. 110)

[36] E. van den Berg and M. P. Friedlander. Probing the Pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008. (Cited on pp. 110, 145)

[37] E. van den Berg and M. P. Friedlander. Sparse optimization with least-squares constraints. *SIAM Journal on Optimization*, 21(4):1201–1229, 2011. (Cited on p. 110)

[38] D. P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21(2):174–184, 1976. (Cited on p. 98)

[39] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, Belmont, MA, 1996. (Cited on pp. 34, 38)

[40] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 2nd edition, 1999. (Cited on pp. 13, 18, 20, 28, 110)

[41] E. G. Birgin, R. Biloti, M. Tygel, and L. T. Santos. Restricted optimization: A clue to a fast and accurate implementation of the common reflection surface stack method. *Journal of Applied Geophysics*, 42(3–4):143–155, 1999. (Cited on p. 110)

[42] E. G. Birgin, L. F. Bueno, N. Krejić, and J. M. Martínez. Low order-value approach for solving VaR-constrained optimization problems. *Journal of Global Optimization*, 51(4):715–742, 2011. (Cited on pp. 3, 9, 13, 38)

[43] E. G. Birgin, E. V. Castelani, A. L. Martínez, and J. M. Martínez. Outer trust-region method for constrained optimization. *Journal of Optimization Theory and Applications*, 150(1):142–155, 2011. (Cited on p. 38)

[44] E. G. Birgin, R. A. Castillo, and J. M. Martínez. Numerical comparison of Augmented Lagrangian algorithms for nonconvex problems. *Computational Optimization and Applications*, 31(1):31–55, 2005. (Cited on p. 37)

[45] E. G. Birgin, I. Chambouleyron, and J. M. Martínez. Estimation of the optical constants and the thickness of thin films using unconstrained optimization. *Journal of Computational Physics*, 151(2):862–880, 1999. (Cited on p. 110)

[46] E. G. Birgin, I. Chambouleyron, J. M. Martínez, and S. D. Ventura. Estimation of optical parameters of very thin films. *Applied Numerical Mathematics*, 47(2):109–119, 2003. (Cited on p. 110)

[47] E. G. Birgin and Y. G. Evtushenko. Automatic differentiation and spectral projected gradient methods for optimal control problems. *Optimization Methods and Software*, 10(2):125–146, 1998. (Cited on p. 110)

[48] E. G. Birgin, D. Fernández, and J. M. Martínez. The boundedness of penalty parameters in an Augmented Lagrangian method with constrained subproblems. *Optimization Methods and Software*, 27(6):1001–1024, 2012. (Cited on pp. 34, 59, 70)

[49] E. G. Birgin, C. A. Floudas, and J. M. Martínez. Global minimization using an Augmented Lagrangian method with variable lower-level constraints. *Mathematical Programming*, 125(1):139–162, 2010. (Cited on pp. 2, 45)

[50] E. G. Birgin and J. M. Gentil. Evaluating bound-constrained minimization software. *Computational Optimization and Applications*, 53(2):347–373, 2012. (Cited on p. 156)

[51] E. G. Birgin and J. M. Martínez. A box-constrained optimization algorithm with negative curvature directions and spectral projected gradients. *Computing [Suppl]*, 15:49–60, 2001. (Cited on pp. 106, 156)

[52] E. G. Birgin and J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Computational Optimization and Applications*, 23(1):101–125, 2002. (Cited on pp. 106, 156)

[53] E. G. Birgin and J. M. Martínez. Local convergence of an inexact-restoration method and numerical experiments. *Journal of Optimization Theory and Applications*, 127(2):229–247, 2005. (Cited on p. 57)

[54] E. G. Birgin and J. M. Martínez. Improving ultimate convergence of an Augmented Lagrangian method. *Optimization Methods and Software*, 23(2):177–195, 2008. (Cited on p. 118)

[55] E. G. Birgin and J. M. Martínez. Structured minimal-memory inexact quasi-Newton method and secant preconditioners for Augmented Lagrangian optimization. *Computational Optimization and Applications*, 39(1):1–16, 2008. (Cited on p. 156)

[56] E. G. Birgin and J. M. Martínez. Augmented Lagrangian method with nonmonotone penalty parameters for constrained optimization. *Computational Optimization and Applications*, 51(3):941–965, 2012. (Cited on p. 37)

[57] E. G. Birgin, J. M. Martínez, L. Martínez, and G. B. Rocha. Sparse projected-gradient method as a linear-scaling low-memory alternative to diagonalization in self-consistent field electronic structure calculations. *Journal of Chemical Theory and Computation*, 9(2):1043–1051, 2013. (Cited on p. 6)

[58] E. G. Birgin, J. M. Martínez, and L. F. Prudente. Augmented Lagrangians with possible infeasibility and finite termination for global nonlinear programming. *Journal of Global Optimization*, 58:207–242, 2014. (Cited on pp. 2, 46)

[59] E. G. Birgin, J. M. Martínez, and L. F. Prudente. Optimality properties of an Augmented Lagrangian method on infeasible problems. Technical report MCDO110214, University of Campinas, Campinas, SP, Brazil, 2014. (Cited on pp. 44, 146)

[60] E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000. (Cited on pp. 92, 97, 98, 110)

[61] E. G. Birgin, J. M. Martínez, and M. Raydan. Algorithm 813: SPG—Software for convex-constrained optimization. *ACM Transactions on Mathematical Software*, 27(3):340–349, 2001. (Cited on pp. 97, 98, 110)

[62] E. G. Birgin, J. M. Martínez, and M. Raydan. Inexact spectral projected gradient methods on convex sets. *IMA Journal of Numerical Analysis*, 23(4):539–559, 2003. (Cited on pp. 97, 98, 103, 110)

[63] E. G. Birgin, J. M. Martínez, and M. Raydan. Spectral projected gradient methods. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3652–3659. Springer, New York, 2nd edition, 2009. (Cited on p. 97)

[64] E. G. Birgin, J. M. Martínez, and M. Raydan. Spectral projected gradient methods: Review and perspectives. Technical report, State University of Campinas, Campinas, SP, Brazil, 2012. (Cited on p. 110)

[65] E. G. Birgin and F. N. C. Sobral. Minimizing the object dimensions in circle and sphere packing problems. *Computers & Operations Research*, 35(7):2357–2375, 2008. (Cited on pp. 167, 168, 192)

[66] L. S. Blackford, J. Demmel, J. Dongarra, I. S. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002. (Cited on p. 137)

[67] S. F. Bockman. Generalizing the formula for areas of polygons to moments. *American Mathematical Monthly*, 96(2):131–132, 1989. (Cited on p. 172)

[68] S. Bonettini, R. Zanella, and L. Zanni. A scaled gradient projection method for constrained image deblurring. *Inverse Problems*, 25(1):015002, 2009. (Cited on p. 110)

[69] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal. *Numerical Optimization: Theoretical and Practical Aspects*. Universitext. Springer, New York, 2nd edition, 2006. (Cited on p. 79)

[70] C. Brezinski and M. R. Zaglia. *Extrapolation Methods Theory and Practice*. North-Holland, Amsterdam, 1991. (Cited on p. 58)

[71] J. Bunch and B. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, 1971. (Cited on p. 91)

[72] R. S. Burachik and C. Y. Kaya. An augmented penalty function method with penalty parameter updates for nonconvex optimization. *Nonlinear Analysis: Theory, Methods & Applications*, 75(3):1158–1167, 2012. (Cited on p. 46)

[73] J. D. Buys. *Dual Algorithms for Constrained Optimization Problems*. PhD thesis, University of Leiden, Leiden, the Netherlands, 1972. (Cited on p. 2)

[74] E. Cancès, C. L. Bris, and Y. Maday. *Méthodes mathématiques en Chimie Quantique. Une Introduction*. Mathématiques et Applications. Springer, New York, 2006. (Cited on p. 5)

[75] E. V. Castelani, A. L. Martinez, J. M. Martínez, and B. F. Svaiter. Addressing the greediness phenomenon in nonlinear programming by means of proximal Augmented Lagrangians. *Computational Optimization and Applications*, 46(2):229–245, 2010. (Cited on p. 38)

[76] J. Castillo. A discrete variational grid generation method. *SIAM Journal on Scientific and Statistical Computing*, 12(2):454–468, 1991. (Cited on pp. 182, 183, 187, 189)

[77] R. A. Castillo. *Métodos de Lagrangiano Aumentado usando penalidades generalizadas para programação não linear*. PhD thesis, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 1998. (Cited on p. 37)

[78] A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte-Rendu de l'Académie des Sciences*, 27:536–538, 1847. (Cited on p. 110)

[79] R. M. Chamberlain, M. J. D. Powell, C. Lemarechal, and H. C. Pedersen. The watchdog technique for forcing convergence in algorithms for constrained optimization. In A. G. Buckley and J. L. Goffin, editors, *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, volume 16 of *Mathematical Programming Studies*, pages 1–17. Springer, New York, 1982. (Cited on pp. 76, 103)

[80] I. Chambouleyron, S. D. Ventura, E. G. Birgin, and J. M. Martínez. Optical constants and thickness determination of very thin amorphous semiconductor films. *Journal of Applied Physics*, 92(6):3093–3102, 2009. (Cited on p. 110)

[81] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998. (Cited on p. 146)

[82] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Lancelot: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics*. Springer-Verlag, New York, 1992. (Cited on pp. 2, 151)

[83] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust Region Methods*, volume 1 of *MOS-SIAM Series on Optimization*. SIAM, Philadelphia, 2000. (Cited on pp. 2, 18, 28, 76, 91, 94, 95)

[84] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*, volume 8 of *MOS-SIAM Series on Optimization*. SIAM, Philadelphia, 2009. (Cited on p. 79)

[85] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, New York, 3rd edition, 1999. (Cited on pp. 167, 168)

[86] D. Cores, R. Escalante, M. Gonzalez-Lima, and O. Jimenez. On the use of the spectral projected gradient method for support vector machines. *Computational and Applied Mathematics*, 28(3):327–364, 2009. (Cited on p. 110)

[87] D. Cores and M. C. Loreto. A generalized two point ellipsoidal anisotropic ray tracing for converted waves. *Optimization and Engineering*, 8(4):373–396, 2007. (Cited on p. 110)

[88] R. Courant. Variational methods for the solution of problems with equilibrium and vibration. *Bulletin of the American Mathematical Society*, 49:1–23, 1943. (Cited on pp. 28, 59)

[89] F. Curiel, W. E. Vargas, and R. G. Barrera. Visible spectral dependence of the scattering and absorption coefficients of pigmented coatings from inversion of diffuse reflectance spectra. *Applied Optics*, 41(28):5969–5978, 2002. (Cited on p. 110)

[90] Y.-H. Dai. Convergence properties of the BFGS algorithm. *SIAM Journal on Optimization*, 13(3):693–701, 2002. (Cited on pp. 77, 94)

[91] Y.-H. Dai. A perfect example for the BFGS method. *Mathematical Programming*, 138(1–2):501–530, 2013. (Cited on pp. 77, 94)

[92] Y. H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Mathematical Programming*, 106(3):403–421, 2006. (Cited on p. 110)

[93] Y. H. Dai and Y. Yuan. A nonlinear conjugate gradient method with a strong global convergence property. *SIAM Journal on Optimization*, 10(1):177–182, 1999. (Cited on p. 94)

[94] T. A. Davis. *Direct Methods for Sparse Linear Systems*, volume 2 of *Fundamentals of Algorithms*. SIAM, Philadelphia, 2006. (Cited on p. 123)

[95] G. P. Deidda, E. Bonomi, and C. Manzi. Inversion of electrical conductivity data with Tikhonov regularization approach: Some considerations. *Annals of Geophysics*, 46(3):549–558, 2003. (Cited on p. 110)

[96] S. Dempe. *Foundations of Bilevel Programming*, volume 61 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, the Netherlands, 2002. (Cited on p. 27)

[97] J. E. Dennis, Jr., and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, 1977. (Cited on p. 83)

[98] J. E. Dennis, Jr., and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, volume 16 of *Classics in Applied Mathematics*. SIAM, Philadelphia, 1996. (Cited on pp. 86, 93, 94)

[99] M. A. Diniz-Ehrhardt, J. M. Martínez, and L. G. Pedroso. Derivative-free methods for nonlinear programming with general lower-level constraints. *Computational and Applied Mathematics*, 30(1):19–52, 2011. (Cited on p. 2)

[100] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002. (Cited on p. 164)

[101] Z. Dostál. Box constrained quadratic programming with proportioning and projections. *SIAM Journal on Optimization*, 7(3):871–887, 1997. (Cited on p. 111)

[102] Z. Dostál. *Optimal Quadratic Programming Algorithms*, volume 23 of *Optimization and Its Applications*. Springer Science+Business Media, New York, 2009. (Cited on pp. 1, 37)

[103] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983. (Cited on p. 91)

[104] J. Eckstein and P. J. S. Silva. A practical relative error criterion for Augmented Lagrangians. *Mathematical Programming*, 141(1–2):319–348, 2013. (Cited on p. 37)

[105] T. F. Edgar, D. M. Himmelblau, and L. S. Lasdon. *Optimization of Chemical Processes*. McGraw-Hill, New York, 2001. (Cited on p. 13)

[106] F. Facchinei and J.-S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. *Springer Series in Operations Research and Financial Engineering*. Springer, New York, 2004. (Cited on p. 27)

[107] D. Fernández, E. A. Pilotta, and G. A. Torres. An inexact restoration strategy for the globalization of the sSQP method. *Computational Optimization and Applications*, 54(3):595–617, 2013. (Cited on p. 3)

[108] D. Fernández and M. V. Solodov. Local convergence of exact and inexact augmented Lagrangian methods under the second-order sufficient optimality condition. *SIAM Journal on Optimization*, 22(2):384–407, 2012. (Cited on pp. 59, 70)

[109] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, volume 4 of *Classics in Applied Mathematics*. SIAM, Philadelphia, 1990. (Cited on pp. 28, 59)

[110] M. A. Figueiredo, R. D. Nowak, and S. J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Journal of Selected Topics in Signal Processing*, 1:586–597, 2007. (Cited on pp. 110, 145)

[111] A. Fischer and A. Friedlander. A new line search inexact restoration approach for nonlinear programming. *Computational Optimization and Applications*, 46(2):333–346, 2010. (Cited on p. 57)

[112] P. Van Fleet. *Discrete Wavelet Transformations: An Elementary Approach with Applications*. John Wiley and Sons, Inc., New York, 2008. (Cited on p. 146)

[113] R. Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, Ltd., London, 2nd edition, 1987. (Cited on pp. 13, 18, 88)

[114] R. Fletcher. Low storage methods for unconstrained optimization. *Lectures in Applied Mathematics*, 26:165–179, 1990. (Cited on p. 110)

[115] R. Fletcher. A limited memory steepest descent method. *Mathematical Programming*, 135(1–2):413–436, 2012. (Cited on p. 110)

[116] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964. (Cited on p. 94)

[117] C. A. Floudas. *Deterministic Global Optimization: Theory, Methods and Application*, volume 37 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 2000. (Cited on p. 45)

[118] C. A. Floudas, I. G. Akrotirianakis, S. Caratzoulas, C. A. Meyer, and J. Kallrath. Global optimization in the 21st century: Advances and challenges. *Computers & Chemical Engineering*, 29(6):1185–1202, 2005. (Cited on p. 45)

[119] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A modeling language for mathematical programming*. Brooks/Cole Publishing Company, Pacific Grove, CA, 2002. (Cited on p. 135)

[120] J. B. Francisco, J. M Martínez, L. Martínez, and F. I. Pisnitchenko. Inexact restoration method for minimization problems arising in electronic structure calculations. *Computational Optimization and Applications*, 50(3):555–590, 2011. (Cited on p. 29)

[121] A. Friedlander and J. M. Martínez. On the maximization of a concave quadratic function with box constraints. *SIAM Journal on Optimization*, 4(1):177–192, 1994. (Cited on p. 111)

[122] A. A. Gaivoronski and G. Pflug. Finding optimal portfolios with constraints on value at risk. Technical report, Department of Industrial Economics and Technology Management, NTNU—Norwegian University of Science and Technology, Norway, 1999. (Cited on p. 13)

[123] D. Y. Gao. *Duality Principles in Nonconvex Systems: Theory, Methods, and Applications*, volume 39 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 2000. (Cited on p. 45)

[124] R. N. Gasimov. Augmented Lagrangian duality and nondifferentiable optimization methods in nonconvex programming. *Journal of Global Optimization*, 24(2):187–203, 2002. (Cited on p. 46)

[125] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, 1986. (Cited on p. 13)

[126] A. A. Goldstein. Convex programming in Hilbert space. *Bulletin of the American Mathematical Society*, 70(5):709–710, 1964. (Cited on pp. 98, 110)

[127] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 4th edition, 2013. (Cited on p. 82)

[128] C. C. Gonzaga and R. A. Castillo. A nonlinear programming algorithm based on non-coercive penalty functions. *Mathematical Programming*, 96(1):87–101, 2003. (Cited on p. 37)

[129] C. C. Gonzaga, E. Karas, and M. Vanti. A globally convergent filter method for nonlinear programming. *SIAM Journal on Optimization*, 14(3):646–669, 2004. (Cited on p. 57)

[130] N. I. M. Gould. On the accurate determination of search directions for simple differentiable penalty functions. *IMA Journal of Numerical Analysis*, 6(3):357–372, 1986. (Cited on p. 2)

[131] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst and SIFDec, A Constrained and Unconstrained Testing Environment with Safe Threads, Technical report RAL-TR-2013-005, Rutherford Appleton, Laboratory, Chilton, Oxfordshire, England, 2013 (Cited on p. 135)

[132] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986. (Cited on pp. 98, 103, 110)

[133] A. A. Groenwold, L. F. P. Etman, S. Kok, D. W. Wood, and S. Tosserams. An Augmented Lagrangian approach to non-convex SAO using diagonal quadratic approximations. *Structural and Multidisciplinary Optimization*, 38(4):415–421, 2009. (Cited on p. 12)

[134] J. Guddat, F. G. Vásquez, and H. Th. Jongen. *Parametric Optimization: Singularities, Path-following and Jumps*. John Wiley and Sons, Inc., New York, 1999. (Cited on pp. 27, 88)

[135] J. Guerrero, M. Raydan, and M. Rojas. A hybrid-optimization method for large-scale nonnegative full regularization in image restoration. *Inverse Problems in Science and Engineering*, 21(5):741–766, 2013. (Cited on p. 110)

[136] P. C. Haarhoff and J. D. Buys. A new method for the optimization of a nonlinear function subject to nonlinear constraints. *The Computer Journal*, 13(2):178–184, 1970. (Cited on p. 2)

[137] W. W. Hager. Rates of convergence for discrete approximations to unconstrained control problems. *SIAM Journal on Numerical Analysis*, 13(4):449–472, 1976. (Cited on p. 178)

[138] W. W. Hager and H. C. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, 16(1):170–192, 2005. (Cited on p. 94)

[139] W. W. Hager and H. C. Zhang. A new active set algorithm for box constrained optimization. *SIAM Journal on Optimization*, 17(2):526–557, 2006. (Cited on pp. 94, 110)

[140] W. W. Hager and H. C. Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2(1):35–58, 2006. (Cited on p. 94)

[141] Y. Y. Haimes, L. S. Lasdon, and D. Wismer. On a bicriterion formulation of the problem of integrated system identification and system optimization. *IEEE Transactions on Systems, Man and Cybernetics*, 1(3):296–297, 1971. (Cited on p. 185)

[142] Z. He, T. Ogawa, and M. Haseyama. The simplest measurement matrix for compressed sensing of natural images. In *17th IEEE International Conference on Image Processing (ICIP)*, pages 4301–4304, 2010. (Cited on p. 146)

[143] T. Helgaker, P. Jorgensen, and J. Olsen. *Molecular Electronic-Structure Theory*. John Wiley and Sons, Inc., New York, 2000. (Cited on p. 5)

[144] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969. (Cited on pp. 2, 31, 37)

[145] M. R. Hestenes. *Optimization Theory: The Finite Dimensional Case*. John Wiley and Sons, Inc., New York, 1975. (Cited on p. 28)

[146] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952. (Cited on p. 92)

[147] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. McGraw-Hill, New York, 1981. (Cited on p. 169)

[148] R. Horst, P. M. Pardalos, and M. V. Thoai. *Introduction to Global Optimization*, volume 3 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 1995. (Cited on p. 45)

[149] HSL(2013). A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk. (Cited on pp. 118, 137)

[150] C. Humes, Jr. and P. J. S. Silva. Strict convex regularizations, proximal points and augmented Lagrangians. *RAIRO Operations Research*, 34(3):283–303, 2000. (Cited on p. 37)

[151] A. N. Iusem. Augmented Lagrangian and proximal point methods for convex optimization. *Investigación Operativa*, 8(1–3):11–49, 1999. (Cited on pp. 34, 37)

[152] A. F. Izmailov. On the limiting properties of dual trajectories in the Lagrange multipliers method. *Computational Mathematics and Mathematical Physics*, 51(1):1–20, 2011. (Cited on p. 71)

[153] A. F. Izmailov, A. S. Kurennoy, and M. V. Solodov. Local convergence of the method of multipliers for variational and optimization problems under the sole noncriticality assumption. http://www.optimization-online.org/DB_HTML/2013/08/3999.html, 2013. (Cited on p. 71)

[154] A. F. Izmailov and M. V. Solodov. *Otimização—Volume 1: Condições de Otimalidade, Elementos de Análise Convexa e de Dualidade*. IMPA—Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, RJ, Brazil, 2009. (Cited on p. 28)

[155] A. F. Izmailov, M. V. Solodov, and E. I. Uskov. Global convergence of Augmented Lagrangian methods applied to optimization problems with degenerate constraints, including problems with complementarity constraints, 2012. (Cited on p. 71)

[156] Z. Jiang. Applications of conditional nonlinear optimal perturbation to the study of the stability and sensitivity of the Jovian atmosphere. *Advances in Atmospheric Sciences*, 23(5):775–783, 2006. (Cited on p. 110)

[157] H. Th. Jongen and G.-W. Weber. On parametric nonlinear programming. *Annals of Operations Research*, 27(1):253–283, 1990. (Cited on pp. 27, 88)

[158] W. Karush. *Minima of Functions of Several Variables with Inequalities as Side Conditions*. Master's thesis, Department of Mathematics, University of Chicago, 1939. (Cited on p. 28)

[159] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. (Cited on p. 137)

[160] C. Y. Kaya. Inexact restoration for Runge Kutta discretization of optimal control problems. *SIAM Journal on Numerical Analysis*, 48(4):1492–1517, 2010. (Cited on pp. 57, 178, 179)

[161] C. Y. Kaya and J. M. Martínez. Euler discretization and inexact restoration for optimal control. *Journal of Optimization Theory and Applications*, 134(2):191–206, 2007. (Cited on p. 178)

[162] H. B. Keller, A. K. Nandakumaran, and M. Ramaswamy. *Lectures on Numerical Methods in Bifurcation Problems*. Springer, New York, 1987. (Cited on p. 27)

[163] M. Kocvara and M. Stingl. PENNON—A generalized Augmented Lagrangian method for semidefinite programming. In G. Di Pillo and A. Murli, editors, *High Performance Algorithms and Software for Nonlinear Optimization*, volume 24 of *Applied Optimization*, pages 297–315. Kluwer Academic Publishers, Dordrecht, the Netherlands, 2003. (Cited on p. 39)

[164] T. G. Kolda, R. M. Lewis, and V. Torczon. A generating set direct search augmented Lagrangian algorithm for optimization with a combination of general and linear constraints. Technical report SAND2006-3515, Sandia National Laboratories, Albuquerque, NM, 2006. (Cited on p. 2)

[165] B. W. Kort and D. P. Bertsekas. Multiplier methods for convex programming. In *volume 12 of IEEE Conference on Decision and Control Including the 12th Symposium on Adaptive Processes*, pages 428–432, 1973. (Cited on p. 37)

[166] B. W. Kort and D. P. Bertsekas. Combined primal dual and penalty methods for convex programming. *SIAM Journal on Control and Optimization*, 14(2):268–294, 1976. (Cited on p. 37)

[167] M. Kočvara and M. Stingl. PENNON—A code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, 18(3):317–333, 2003. (Cited on p. 37)

[168] M. Kočvara and M. Stingl. Solving nonconvex SDP problems of structural optimization with stability control. *Optimization Methods and Software*, 19(5):595–609, 2004. (Cited on p. 37)

[169] M. Kočvara and M. Stingl. On the solution of large-scale SDP problems by the modified barrier method using iterative solvers. *Mathematical Programming*, 109(2–3):413–444, 2007. (Cited on p. 37)

[170] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, University of California Press, Berkeley, 1951. (Cited on p. 28)

[171] W. La Cruz, J. M. Martínez, and M. Raydan. Spectral residual method without gradient information for solving large-scale nonlinear systems of equations. *Mathematics of Computation*, 75(255):1429–1448, 2006. (Cited on p. 110)

[172] W. La Cruz and M. Raydan. Nonmonotone spectral methods for large-scale nonlinear systems. *Optimization Methods and Software*, 18(5):583–599, 2003. (Cited on p. 110)

[173] L. S. Lasdon. Reduced gradient methods. In M. J. D. Powell, editor, *Nonlinear Optimization*, pages 235–242. Academic Press, New York, 1982. (Cited on p. 57)

[174] E. S. Levitin and B. T. Polyak. Constrained minimization methods. *USSR Computational Mathematics and Mathematical Physics*, 6(5):1–50, 1966. (Cited on pp. 98, 110)

[175] R. M. Lewis and V. Torczon. A globally convergent Augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4):1075–1089, 2002. (Cited on p. 2)

[176] R. M. Lewis and V. Torczon. A direct search approach to nonlinear programming problems using an augmented Lagrangian method with explicit treatment of linear constraints. Technical report WM-CS-2010-01, Department of Computer Sciences, College of William and Mary, Williamsburg, VA, 2010. (Cited on p. 2)

[177] Z. Li and M. G. Ierapetritou. Capacity expansion planning through augmented Lagrangian optimization and scenario decomposition. *AIChE Journal*, 58(3):871–883, 2012. (Cited on pp. 10, 11, 46)

[178] M.-L. Liu and N. V. Sahinidis. Long range planning in the process industries: A projection approach. *Computers & Operations Research*, 23(3):237–253, 1996. (Cited on p. 10)

[179] I. Loris, M. Bertero, C. De Mol, R. Zanella, and L. Zanni. Accelerating gradient projection methods for ℓ_1-constrained signal recovery by steplength selection rules. *Applied and Computational Harmonic Analysis*, 27(2):247–254, 2009. (Cited on p. 110)

[180] S. Lu. Implications of the constant rank constraint qualification. *Mathematical Programming*, 126(2):365–392, 2011. (Cited on pp. 25, 29)

[181] S. Lu. Relation between the constant rank and the relaxed constant rank constraint qualifications. *Optimization*, 61(5):555–566, 2012. (Cited on pp. 25, 29)

[182] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*, volume 116 of *International Series in Operations Research & Management Science*. Springer Science+Business Media, New York, 2008. (Cited on pp. 13, 18, 28, 76)

[183] O. L. Mangasarian. *Nonlinear Programming*, volume 10 of *Classics in Applied Mathematics*. SIAM, Philadelphia, 1994. (Cited on p. 28)

[184] S. A. Marglin. *Public Investment Criteria*. MIT Press, Cambridge, MA, 1967. (Cited on p. 185)

[185] J. M. Martínez. BOX-QUACAN and the implementation of augmented Lagrangian algorithms for minimization with inequality constraints. *Computational and Applied Mathematics*, 19:31–56, 2000. (Cited on p. 171)

[186] J. M. Martínez. Inexact-restoration method with Lagrangian tangent decrease and new merit function for nonlinear programming. *Journal of Optimization Theory and Applications*, 111(1):39–58, 2001. (Cited on p. 57)

[187] J. M. Martínez and L. Martínez. Packing optimization for automated generation of complex system's initial configurations for molecular dynamics and docking. *Journal of Computational Chemistry*, 24(7):819–825, 2003. (Cited on pp. 6, 7, 165, 166, 167, 192)

[188] J. M. Martínez and E. A. Pilotta. Inexact-restoration algorithm for constrained optimization. *Journal of Optimization Theory and Applications*, 104(1):135–163, 2000. (Cited on p. 57)

[189] J. M. Martínez and L. F. Prudente. Handling infeasibility in a large-scale nonlinear optimization algorithm. *Numerical Algorithms*, 60(2):263–277, 2012. (Cited on pp. 56, 66)

[190] J. M. Martínez and L. Qi. Inexact Newton methods for solving nonsmooth equations. *Journal of Computational and Applied Mathematics*, 60(1–2):127–145, 1995. (Cited on pp. 86, 88)

[191] J. M. Martínez and L. T. Santos. New theoretical results on recursive quadratic programming algorithms. *Journal of Optimization Theory and Applications*, 97(2):435–454, 1998. (Cited on p. 157)

[192] J. M. Martínez and B. F. Svaiter. A practical optimality condition without constraint qualifications for nonlinear programming. *Journal of Optimization Theory and Applications*, 118(1):117–133, 2003. (Cited on pp. 28, 48)

[193] L. Martínez, R. Andrade, E. G. Birgin, and J. M. Martínez. PACKMOL: A package for building initial configurations for molecular dynamics simulations. *Journal of Computational Chemistry*, 30(13):2157–2164, 2009. (Cited on pp. 6, 7, 165, 166, 167, 170, 192)

[194] W. F. Mascarenhas. The BFGS method with exact line searches fails for non-convex objective functions. *Mathematical Programming*, 99(1):49–61, 2004. (Cited on pp. 77, 94)

[195] W. F. Mascarenhas. On the divergence of line search methods. *Computational and Applied Mathematics*, 26(1):129–169, 2007. (Cited on pp. 77, 94)

[196] L. C. Matioli. *Uma nova metodologia para construção de funções de penalização para algoritmos de Lagrangiano Aumentado*. PhD thesis, Universidade Federal de Santa Catarina, Florianópolis, Brazil, 2001. (Cited on p. 37)

[197] E. J. McShane. The Lagrange multiplier rule. *American Mathematical Monthly*, 80(8):922–925, 1973. (Cited on p. 28)

[198] A. Miele, H. Y. Huang, and J. C. Heideman. Sequential gradient-restoration algorithm for the minimization of constrained functions ordinary and conjugate gradient versions. *Journal of Optimization Theory and Applications*, 4(4):213–243, 1969. (Cited on p. 57)

[199] A. Miele, A. V. Levy, and E. E. Cragg. Modifications and extensions of the conjugate gradient-restoration algorithm for mathematical programming problems. *Journal of Optimization Theory and Applications*, 7(6):450–472, 1971. (Cited on p. 57)

[200] A. Miele, E. M. Sims, and V. K. Basapur. Sequential Gradient-Restoration Algorithm for Mathematical Programming Problems with Inequality Constraints: Part 1, Theory. Technical report 168, Rice University, Houston, TX, 1983. (Cited on p. 57)

[201] M. I. Miller, G. E. Christensen, Y. Amit, and U. Grenander. Mathematical textbook of deformable neuroanatomies. *Proceedings of the National Academy of Sciences*, 90(24):11944–11948, 1993. (Cited on p. 7)

[202] M. Millstone. *Real-Space Localization Methods for minimizing the Kohn-Sham Energy*. PhD thesis, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, 2011. (Cited on p. 6)

[203] L. Minchenko and S. Stakhovski. On relaxed constant rank regularity condition in mathematical programming. *Optimization*, 60(4):429–440, 2011. (Cited on pp. 25, 29)

[204] L. Minchenko and S. Stakhovski. Parametric nonlinear programming problems under the relaxed constant rank condition. *SIAM Journal on Optimization*, 21(1):314–332, 2011. (Cited on pp. 25, 28, 29)

[205] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981. (Cited on p. 192)

[206] P. Moulin, R. Krishnamurthy, and J. W. Woods. Multiscale modeling and estimation of motion fields for video coding. *IEEE Transactions on Image Processing*, 6(12):1606–1620, 1997. (Cited on p. 7)

[207] M. Mu, W. S. Duan, and B. Wang. Conditional nonlinear optimal perturbation and its applications. *Nonlinear Processes in Geophysics*, 10:493–501, 2003. (Cited on p. 110)

[208] M. Mulato, I. Chambouleyron, E. G. Birgin, and J. M. Martínez. Determination of thickness and optical constants of a-Si:H films from transmittance data. *Applied Physics Letters*, 77(14):2133–2135, 2000. (Cited on p. 110)

[209] A. B. Murphy. Optical properties of an optically rough coating from inversion of diffuse reflectance measurements. *Applied Optics*, 46:3133–3143, 2007. (Cited on p. 110)

[210] F. H. Murphy. A class of exponential penalty functions. *SIAM Journal on Control*, 12(4):679–687, 1974. (Cited on p. 37)

[211] H. Nakayama, H. Sayama, and Y. Sawaragi. A generalized Lagrangian function and multiplier method. *Journal of Optimization Theory and Applications*, 17(3–4):211–227, 1975. (Cited on p. 37)

[212] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 13:271–369, 2004. (Cited on p. 45)

[213] J. Nocedal and S. J. Wright. *Numerical Optimization*. *Springer Series in Operations Research and Financial Engineering*. Springer Science+Business Media, New York, 2nd edition, 2006. (Cited on pp. 13, 28)

[214] S. S. Oren. On the selection of parameters in self scaling variable metric algorithms. *Mathematical Programming*, 7(1):351–367, 1974. (Cited on p. 110)

[215] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*, volume 30 of *Classics in Applied Mathematics*. SIAM, Philadelphia, 2000. (Cited on p. 94)

[216] V. Pereyra, M. A. Saunders, and J. Castillo. Equispaced pareto front construction for constrained bi-objective optimization. *Mathematical and Computer Modelling*, 57(9–10):2122–2131, 2013. (Cited on pp. 182, 183)

[217] E. Polak and G. Ribière. Note sur la convergence de methodes de directions conjugées. *RAIRO Recherche Operationelle*, 16(R1):35–43, 1969. (Cited on p. 94)

[218] M. J. D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298. Academic Press, New York, 1969. (Cited on pp. 2, 31, 37)

[219] P. Pulay. Convergence acceleration of iterative sequences: The case of SCF iteration. *Chemical Physics Letters*, 73(2):393–398, 1980. (Cited on p. 58)

[220] L. Qi and J. Sun. A nonsmooth version of Newton's method. *Mathematical Programming*, 58(1–3):353–367, 1993. (Cited on pp. 86, 88)

[221] L. Qi and Z. Wei. On the constant positive linear dependence condition and its application to SQP methods. *SIAM Journal on Optimization*, 10(4):963–981, 2000. (Cited on p. 20)

[222] A. Ramirez-Porras and W. E. Vargas. Transmission of visible light through oxidized copper films: Feasibility of using a spectral projected gradient method. *Applied Optics*, 43:1508–1514, 2004. (Cited on p. 110)

[223] W. Ratcliff, P. A. Kienzle, J. W. Lynn, S. Li, P. Dai, G. F. Chen, and N. L. Wang. Magnetic form factor of $SrFe_2As_2$: Neutron diffraction measurements. *Physical Review B*, 81(14):140502(R), 2010. (Cited on p. 12)

[224] M. Raydan. On the Barzilai and Borwein choice of steplength for the gradient method. *IMA Journal of Numerical Analysis*, 13(3):321–326, 1993. (Cited on pp. 92, 98, 110)

[225] M. Raydan. The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal on Optimization*, 7(1):26–33, 1997. (Cited on pp. 92, 98, 110)

[226] W. C. Rheinboldt. *Numerical Analysis of Parametrized Nonlinear Equations*, volume 7 of *University of Arkansas Lecture Notes in the Mathematical Sciences*. John Wiley and Sons, Inc., New York, 1986. (Cited on p. 27)

[227] A. M. A. C. Rocha, T. F. M. C. Martins, and E. M. G. P. Fernandes. An augmented Lagrangian fish swarm based method for global optimization. *Journal of Computational and Applied Mathematics*, 235(16):4611–4620, 2011. (Cited on p. 45)

[228] R. T. Rockafellar. The multiplier method of Hestenes and Powell applied to convex programming. *Journal of Optimization Theory and Applications*, 12(6):555–562, 1973. (Cited on p. 37)

[229] R. T. Rockafellar. Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12(2):268–285, 1974. (Cited on pp. 2, 31, 34, 37)

[230] M. Rom and M. Avriel. Properties of the sequential gradient-restoration algorithm (SGRA), Part 1: Introduction and comparison with related methods. *Journal of Optimization Theory and Applications*, 62(1):77–98, 1989. (Cited on p. 57)

[231] M. Rom and M. Avriel. Properties of the sequential gradient-restoration algorithm (SGRA), Part 2: Convergence analysis. *Journal of Optimization Theory and Applications*, 62(1):99–125, 1989. (Cited on p. 57)

[232] J. Rosen. The gradient projection method for nonlinear programming. Part I. Linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8(1):181–217, 1960. (Cited on p. 57)

[233] A. Ruszczyński. On convergence of an Augmented Lagrangian decomposition method for sparse convex optimization. *Mathematics of Operations Research*, 20(3):634–656, 1995. (Cited on p. 11)

[234] T. Schormann and M. Kraemer. Voxel-guided morphometry (VGM) and application to stroke. *IEEE Transactions on Medical Imaging*, 22(1):62–74, 2003. (Cited on p. 7)

[235] L. Schrage. A more portable Fortran random number generator. *ACM Transactions on Mathematical Software*, 5(2):132–138, 1979. (Cited on p. 128)

[236] M. Sdika. A fast nonrigid image registration with constraints on the Jacobian using large scale constrained optimization. *IEEE Transactions on Medical Imaging*, 27(2):271–281, 2008. (Cited on pp. 7, 8)

[237] P. Seferlis. *Collocation Models for Distillation Units and Sensitivity Analysis Studies in Process Optimization*. PhD thesis, Chemical Engineering, McMaster University, Hamilton, ON, Canada, 1995. (Cited on p. 27)

[238] T. Serafini, G. Zanghirati, and L. Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20(2–3):353–378, 2005. (Cited on p. 110)

[239] A. Shapiro. Sensitivity analysis of parameterized variational inequalities. *Mathematics of Operations Research*, 30(1):109–126, 2005. (Cited on p. 30)

[240] H. D. Sherali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*, volume 31 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 1999. (Cited on p. 45)

[241] J. P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University of Chicago Press, Chicago, 1993. (Cited on p. 192)

[242] M. Stingl. On the Solution of Nonlinear Semidefinite Programs by Augmented Lagrangian methods. Master's thesis, Universität Erlangen-Nürnberg, Aachen, Germany, 2006. (Cited on p. 39)

[243] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, volume 12 of *Texts in Applied Mathematics*. Springer Science+Business Media, New York, 1993. (Cited on p. 178)

[244] E. Süli and D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, Cambridge, UK, 2003. (Cited on p. 9)

[245] R. Szeliski and H.-Y. Shum. Motion estimation with quadtree splines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(12):1199–1210, 1996. (Cited on p. 7)

[246] R. Tavakoli and H. Zhang. A nonmonotone spectral projected gradient method for large-scale topology optimization problems. *Numerical Algebra, Control and Optimization*, 2(2):395–412, 2012. (Cited on p. 110)

[247] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming*, volume 65 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 2003. (Cited on p. 45)

[248] J.-P. Thirion. Image matching as a diffusion process: an analogy with Maxwell's demons. *Medical Image Analysis*, 2(3):243–260, 1998. (Cited on p. 7)

[249] J.-P. Thirion, S. Prima, G. Subsol, and N. Roberts. Statistical analysis of normal and abnormal dissymmetry in volumetric medical images. *Medical Image Analysis*, 4(2):111–121, 2000. (Cited on p. 7)

[250] P. Tseng and D. P. Bertsekas. On the convergence of the exponential multiplier method for convex programming. *Mathematical Programming*, 60(1–3):1–19, 1993. (Cited on p. 37)

[251] H. Tuy. *Convex Analysis and Global Optimization*, volume 22 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 1998. (Cited on p. 45)

[252] W. E. Vargas. Inversion methods from Kiabelka-Munk analysis. *Journal of Optics A: Pure and Applied Optics*, 4(4):452–456, 2002. (Cited on p. 110)

[253] W. E. Vargas, D. E. Azofeifa, and N. Clark. Retrieved optical properties of thin films on absorbing substrates from transmittance measurements by application of a spectral projected gradient method. *Thin Solid Films*, 425(1):1–8, 2003. (Cited on p. 110)

[254] S. D. Ventura, E. G. Birgin, J. M. Martínez, and I. Chambouleyron. Optimization techniques for the estimation of the thickness and the optical parameters of thin films using reflectance data. *Journal of Applied Physics*, 97(4):043512, 2005. (Cited on p. 110)

[255] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. (Cited on p. 117)

[256] L. T. Watson, S. C. Billups, and A. P. Morgan. Algorithm 652: HOMPACK: A suite of codes for globally convergent homotopy algorithms. *ACM Transactions on Mathematical Software*, 13(3):281–310, 1987. (Cited on p. 27)

[257] Z. Wu, G. N. Phillips, R. A. Tapia, and Y. Zhang. A fast Newton algorithm for entropy maximization in phase determination. *SIAM Review*, 43(4):623–642, 2001. (Cited on p. 12)

[258] A. E. Xavier. *Penalização hiperbolica*. PhD thesis, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 1992. (Cited on p. 37)

[259] Z. B. Zabinsky. *Stochastic Adaptive Search for Global Optimization*, volume 72 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Dordrecht, the Netherlands, 2003. (Cited on p. 45)

[260] N. Zeev, O. Savasta, and D. Cores. Nonmonotone spectral projected gradient method applied to full waveform inversion. *Geophysical Prospecting*, 54(5):525–534, 2006. (Cited on p. 110)

[261] X. Z. Zhu, J. C. Zhou, L. L. Pan, and W. L. Zhao. Generalized quadratic augmented Lagrangian methods with nonmonotone penalty parameters. *Journal of Applied Mathematics*, 2012:181629, 2012. (Cited on p. 37)

Author Index

Subject Index