DOCTOR OF SCIENCES

A trust-region method for constrained derivative-free optimization and worst-case evaluation complexity of non-monotone gradient-related algorithms for unconstrained optimization.

Rodrigues Sampaio, Phillipe

Award date:
2015

Awarding institution:
University of Namur

UNIVERSITÉ DE NAMUR

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUE

A trust-region method for constrained derivative-free optimization and worst-case evaluation complexity of non-monotone gradient-related algorithms for unconstrained optimization

Thèse présentée par
Phillipe Rodrigues Sampaio
pour l'obtention du grade
de Docteur en Sciences

Composition du Jury:

Andrew R. Conn
Serge Gratton
Anne Lemaître (Présidente du Jury)
Annick Sartenaer
Philippe L. Toint (Promoteur)

Août 2015

Imprimé en Belgique

ISBN : 978-2-87037-908-0
Dépôt légal: D / 2015/ 1881/ 39

Université de Namur
Faculté des Sciences
rue de Bruxelles, 61, B-5000 Namur (Belgique)

Une méthode de région de confiance pour l'optimisation sans dérivées avec des contraintes et complexité d'évaluation dans le pire cas d'algorithmes non-monotones du type gradient pour l'optimisation sans contraintes

par Phillipe Rodrigues Sampaio

Résumé: Ce travail est divisé en deux parties liées à deux sujets de recherche qui ont reçus une attention croissante de la communauté d'optimisation au cours des dernières années. La première partie est consacrée à la conception et la mise en œuvre d'une nouvelle méthode de région de confiance pour l'optimisation sans dérivées avec des contraintes. Cette méthode est basée sur des modèles d'interpolation et utilise une procédure d'auto-correction de la géométrie afin de garantir que la géométrie de l'ensemble d'interpolation ne diffère pas trop de l'idéal. Des résultats numériques de la méthode proposée sont également présentés. La deuxième partie analyse le pire cas de la complexité d'évaluation de la classe d'algorithmes non monotones du type gradient pour des problèmes non convexes, lisses et sans contraintes. Nous montrons que cette classe de méthodes nécessite au plus $O(\epsilon^{-2})$ évaluations de fonction pour trouver un point avec la norme du gradient dessous d'un seuil $\epsilon > 0$.

A trust-region method for constrained derivative-free optimization and worst-case evaluation complexity of non-monotone gradient-related algorithms for unconstrained optimization

by Phillipe Rodrigues Sampaio

Abstract: This work is divided into two parts related to two research topics that have received increasing attention of the optimization community over the past years. The first part is concerned with the design and implementation of a new trust-region method for constrained derivative-free optimization. This method is based on interpolation models and employs a self-correcting geometry procedure in order to ensure that the geometry of the interpolation set does not differ too much from the ideal one. Numerical results of the proposed method are also presented. The second part analyzes the worst-case evaluation complexity of the class of non-monotone gradient-related algorithms for smooth nonconvex and unconstrained problems. We show that this class of methods requires at most $O(\epsilon^{-2})$ function evaluations to find a point with the gradient norm below a threshold $\epsilon > 0$.

Thèse de doctorat en Sciences Mathématiques (Ph.D. thesis in Mathematics)
Date: 25/08/2015
Département de Mathématique
Promoteur (Advisor): Prof. Ph. L. Toint

Acknowledgements

First and foremost, I am grateful to God for his close guidance throughout my life, for giving me the strength and faith that it takes to keep moving forward and for surprising me with so many undeserved blessings.

I am extremely grateful to my parents, Paulo César and Kátia, and my brother, Jonathas, for their great love and unwavering support along every step of this journey. Words are not enough to express my gratitude.

I want to thank my advisor, Philippe Toint, for his unrelenting support, inspiration, friendship, patience and for having believed in me. I will be forever indebted. I also thank him and his wife, Claire, for all the pleasant dinners and enjoyable conversations that we had during these years.

I also want to express my great appreciation for the indispensable help of the members of the jury. To Anne Lemaître for her wonderful assistance and attention when I needed the most. To Annick Sartenaer for her help on uncountable matters and advices towards this work. To Serge Gratton for the great time that I spent in Toulouse and discussions on this thesis. To Andrew Conn for his great suggestions and insightful comments made to enrich this work.

I thank Anke Tröltzsch for her invaluable help on the development of our code, which would certainly have consumed a lot more time without her aid, and for all the invitations to give talks at conferences.

I am very grateful to Margherita Porcelli for her friendship, support and great moments that we shared together. I was really blessed for having her as my office mate for one year.

I thank my friend Justin Buhendwa for his help with my French, our endless philosophical discussions and the great time that we had in California.

I want to thank Neto Pires, Fernando Antonio, Yerali Gandica and Anne-Sophie Libert for the nice conversations, advices and good humor. Many thanks also to Eric Cornellis and Pascale Hermans for their great assistance during these years as well as to everyone in the Department of Mathematics of the University of Namur. I am also very grateful to the University of Namur for the doctoral scholarship that allowed me to come to Belgium and to accomplish this work.

Many thanks to Laurent Sulpizio, Ingrid Calier, Logan Sulpizio and Pierre Sulpizio for welcoming me so nicely in Namur. I am very grateful for the

amazing moments we shared over these years, including all the football matches.

I want to thank Lydia Renier and André Chapaux for their friendship, good taste in music, assistance and great moments that we had every weekend for the past four years. I also want to thank Muriel Deconinck, Bernard Laverdure, Patricia Eloot, Nathanaël Laverdure, Lydie Laverdure, Timothée Laverdure, Popa Doru, Quentin Heinen, Alice Wanet, Ruth Fondrieschi, Dorcas Fondrieschi, Marcel Feron, Denis Doulière, Cedric Amoes and Marco Bronckaert for their friendship and amazing time spent together.

Special thanks to Cecile Motte, David Dubreucq, Théo Dubreucq and Chloé Dubreucq for being my family in Belgium and for everything that they have done to make my stay in this country so wonderful.

Last but not least, I wish to thank my friends in Brazil: Carlos Alberto Maia and Socorro Maia, Miquéias Maia and Natanny Medeiros, Gabriela Maia and Jonathan Alves, Junior Maia and Josy Maia, Jonatas Maia, Jamilly Maia, Vitória Maia, Thais Bandeira and Aurilane Carvalho, Milton Zanini, Edvar Caldas and his family, Augusto Maia and his family, Wagner Lima and his family, Paulo Maia and his family, Sílvia Dantas, Isac Maia and Alessandra Freitas, Ciro Maia and Bruna Maia, David Alencar, Wátila Pereira, Ana Raquel Maia and Alexandre Lima, and many others. Thanks for the never-ending support and prayers which have made this journey much easier.

Phillipe

Contents

1 Introduction **1**
- 1.1 Scope and main contributions . . . 1
  - 1.1.1 Constrained derivative-free optimization . . . 1
  - 1.1.2 Worst-case evaluation complexity . . . 2
- 1.2 Organization of the thesis . . . 2
- 1.3 Notation . . . 3

2 Fundamentals and algorithms of nonlinear optimization **5**
- 2.1 Formulation of nonlinear optimization problems . . . 5
- 2.2 Optimality conditions . . . 6
- 2.3 Algorithms for unconstrained optimization . . . 8
  - 2.3.1 Linesearch methods . . . 8
  - 2.3.2 Trust-region methods . . . 12
  - 2.3.3 Conjugate gradient methods . . . 14
  - 2.3.4 Preconditioned CG methods . . . 18
- 2.4 Algorithms for constrained optimization . . . 19
  - 2.4.1 Projected preconditioned CG methods . . . 19
  - 2.4.2 Truncated projected CG methods . . . 21
  - 2.4.3 Projected gradient methods . . . 22
  - 2.4.4 Sequential quadratic programming methods . . . 24
  - 2.4.5 Filter methods . . . 26
  - 2.4.6 Trust-funnel methods . . . 27

3 Introduction to derivative-free optimization **29**
- 3.1 Existing methods . . . 29
- 3.2 Multivariate polynomial interpolation . . . 32
  - 3.2.1 Polynomial bases . . . 32
  - 3.2.2 Poisedness . . . 34
  - 3.2.3 Well poisedness . . . 35
  - 3.2.4 Underdetermined interpolation models . . . 43
  - 3.2.5 Regression nonlinear models . . . 45
- 3.3 Methods based on interpolation models . . . 46
  - 3.3.1 A simple DFO algorithm . . . 46

3.3.2 Self-correcting geometry scheme 47

4 A derivative-free trust-funnel method 51
4.1 Problems with equality constraints 53
4.1.1 The normal step 53
4.1.2 The tangent step 54
4.1.3 Which steps to compute and retain 57
4.1.4 Iterations types 58
4.1.5 The algorithm 60
4.1.6 Implementation and experiments 64
4.2 Problems with general constraints 72
4.2.1 Subspace minimization 78
4.2.2 The normal step 79
4.2.3 The tangent step 80
4.2.4 Which steps to compute and retain 83
4.2.5 Iterations types 84
4.2.6 Application to derivative-free optimization 86
4.2.7 The algorithm 87
4.2.8 Implementation and experiments 92

5 Worst-case evaluation complexity 103
5.1 The problem and the algorithm 104
5.2 Worst-case analysis 106

6 Conclusions and further research perspectives 117
6.1 Constrained derivative-free optimization 117
6.2 Worst-case evaluation complexity 119

List of Figures 120

Index 123

Appendix 129

A Detailed numerical results 129

Bibliography 137

Chapter 1

Introduction

The scope of this thesis is composed of two different topics that have gained the attention of the optimization community of late. The first part is centred on derivative-free optimization, an important developing area with current rapid growth that has applications in many different fields of science and engineering. While a number of algorithms have been designed for unconstrained derivative-free optimization problems, the range of possibilities for the constrained case remains quite open. The second part of this work is concerned with the evaluation complexity analysis of algorithms for optimization. Although global and local convergence results for several optimization methods have long been known, the evaluation complexity of some of them has only recently been investigated. Here, we particularly address the worst-case evaluation complexity of the class of non-monotone gradient-related algorithms for unconstrained optimization.

In the next sections, we briefly present our main contributions in the two areas aforementioned and describe the organization of the thesis.

1.1 Scope and main contributions

1.1.1 Constrained derivative-free optimization

The fast-growing need for optimization methods that do not consider derivatives in fields such as medicine, chemistry, engineering and many others, aroused the optimization community's interest to come up with new algorithms, unleashing the blossom of a new research field on optimization and hatching new ideas for optimizing functions.

Most available algorithms in the domain of derivative-free optimization essentially address the unconstrained case (as those proposed by Powell, 2002, 2006, 2008; Conn, Scheinberg and Toint, 1998; Scheinberg and Toint, 2010) or the bound-constrained one (see Powell, 2009; Lewis and Torczon, 1999; Gratton, Toint and Tröltzsch, 2011; Tröltzsch, 2011). Nevertheless, many al-

gorithmic possibilities have not been considered as yet, either theoretically or practically, for the constrained case. Among the methods developed for problems with more general linear or nonlinear constraints, we may cite the trust-region methods proposed by Powell (1994, 2014) and Colson (2004), and the direct-search methods of Lewis and Torczon (2000, 2002, 2009), Audet and Dennis (2004), Lucidi, Sciandrone and Tseng (2002) and Yu and Li (1981).

In this work, we consider a derivative-free adaptation of the trust-funnel method presented by Gould and Toint (2010) for the solution of equality-constrained nonlinear optimization problems. We describe the complete algorithm and show its numerical results on a set of 29 small-scale equality-constrained problems from the CUTEst collection by Gould, Orban and Toint (2014). We then extend the original trust-funnel method to problems with both equality and inequality constraints and where simple bounds are also considered. Finally, we exploit and incorporate techniques developed for derivative-free optimization to obtain a final method that can also be used to solve problems with general nonlinear constraints without derivatives and we compare its performance to other well-known algorithms on a larger set of problems.

1.1.2 Worst-case evaluation complexity

One way of measuring the complexity and performance of optimization algorithms is investigating the number of function evaluations needed in the worst case to reduce the norm of the gradient of the objective function below a certain threshold ϵ.

The worst-case evaluation complexity of finding an approximate first-order critical point using gradient-related non-monotone methods for smooth nonconvex and unconstrained problems is investigated. The analysis covers a practical linesearch implementation of these popular methods, allowing for an unknown number of evaluations of the objective function (and its gradient) per iteration. It is shown that this class of methods shares the known complexity properties of a simple steepest-descent scheme and that an approximate first-order critical point can be computed in at most $O(\epsilon^{-2})$ function and gradient evaluations, where $\epsilon > 0$ is the user-defined accuracy threshold on the gradient norm.

1.2 Organization of the thesis

Chapter 2 gives the basic concepts in nonlinear optimization such as the formulation of optimization problems and the characterization of solutions, including optimality conditions. It also gives a brief overview of algorithms for unconstrained and constrained optimization. In Chapter 3, we introduce the reader to the field of derivative-free optimization and develop a framework that will serve as the foundation of our algorithm. In Chapter 4, we describe our trust-region method for constrained derivative-free optimization in detail. We also address implementation issues of the proposed method and show some

numerical results from our experiments. In Chapter 5, we present our contribution on worst-case evaluation complexity of first-order linesearch algorithms for unconstrained optimization. Finally, Chapter 6 summarizes the highlights of our work and discusses potential extensions.

1.3 Notation

Matrix: we represent matrices by upper case letters (A, B, ...).

Identity matrix: we represent the identity matrix of dimension n by I_n.

Diagonal matrix: given a $n \times n$ matrix A, we denote by $diag(A)$ the diagonal matrix of dimension $n \times n$ whose non-zero entries are equal to the diagonal entries of A.

Vector: we represent vectors by lower case letters (a, b, x, y, ...). Given any vector x, we denote its i-th component by $[x]_i$.

Vector norm: unless otherwise specified, our norm $\|\cdot\|$ is the standard Euclidean norm, where $\|x\| \stackrel{\text{def}}{=} \sqrt{x^T x}$.

Vector products: given two vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$, we denote their Euclidean inner product by $\langle x, y\rangle \stackrel{\text{def}}{=} y^T x$.

Euclidian Ball: we let $\mathcal{B}(x; \Delta)$ denote the closed Euclidian ball centred at x, with radius $\Delta > 0$, i.e. $\mathcal{B}(x; \Delta) = \{y \,|\, \|y - x\| \leq \Delta\}$.

Cardinality: given any set $\mathcal{A}$, $|\mathcal{A}|$ denotes the cardinality of $\mathcal{A}$.

Polynomials space: we denote by $\mathcal{P}_n^d$ the space of all polynomials of degree at most d in $\mathbb{R}^n$.

Subspace dimension: given any subspace $\mathcal{S}$, we denote its dimension by $dim(\mathcal{S})$.

Positive definiteness: we use the notation $M \succ 0$ to indicate that a matrix M is positive definite, i.e. that $p^T M p > 0$ for all $p \neq 0$, and $M \succeq 0$ to indicate that a matrix M is positive semidefinite, i.e. that $p^T M p \geq 0$ for all $p \neq 0$.

Gradient and Hessian: given a function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient and the Hessian of f at $x \in \mathbb{R}^n$, when they exist, are denoted by $\nabla f(x)$ and $\nabla^2 f(x)$, respectively.

Gradient and Hessian with respect to a variable: given a function $f : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}$ and $(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$, then $\nabla_x f(x)$ and $\nabla_{xx} f(x)$ are the gradient and the Hessian of f with respect to the first variable x.

Jacobian: given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the $m \times n$ Jacobian matrix of f is denoted by J.

Sequences: we let $\{x_k\}$ denote a sequence of vectors $x_k \in \mathbb{R}^n$.

Chapter 2

Fundamentals and algorithms of nonlinear optimization

The purpose of this chapter is to familiarize the reader with the basic concepts of nonlinear optimization and some of the existing methods for solving unconstrained and constrained problems.

2.1 Formulation of nonlinear optimization problems

Given a real function f, an optimization problem consists of choosing values for the input variables from a defined domain that maximize or minimize f. In the parlance of optimization, the function f is commonly called an *objective function*, although in other fields it is also known as a *loss function*, a *cost function*, an *utility function* or an *energy functional*. In this thesis, we consider optimization problems as minimization problems, following the convention by most of the modern literature in the field. However, any maximization problem might be converted into a minimization one due to the equivalence $\max(f) = -\min(-f)$.

An optimization problem may fall into different categories depending on the type of the functions and variables involved and whether there are constraints or not on the variables of the problem. When there are no constraints, it is called an *unconstrained problem*; otherwise, it is called a *constrained problem*.

Consider a function $f : \mathcal{X} \to \mathbb{R}$ and a subset $\mathcal{F} \subset \mathcal{X}$. An optimization

problem may be expressed under the following general form:

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{s.t.:} & x \in \mathcal{F}, \end{cases} \tag{2.1.1}$$

where f is the objective function, $\mathcal{F}$ denotes the *feasible set* of the problem and the points in $\mathcal{F}$ are called *feasible points*. When $\mathcal{F} = \mathbb{R}^n$, (2.1.1) is called an unconstrained problem; otherwise, it is called a constrained problem. In the latter case, the feasible set $\mathcal{F}$ may be written as

$$\mathcal{F} = \{x \in \mathcal{X} \mid c_i(x) = 0,\ i \in \mathcal{E},\ c_i(x) \leq 0,\ i \in \mathcal{I}\}, \tag{2.1.2}$$

where $\mathcal{E}$ and $\mathcal{I}$ are two finite sets of indices, c_i, $i \in \mathcal{E}$, are the *equality constraints* and c_i, $i \in \mathcal{I}$, are the *inequality constraints*.

2.2 Optimality conditions

A *global minimizer* or *global solution* for a optimization problem is a point x^* such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{F}$. Finding such a point might be quite difficult in practice and sometimes unnecessary; such a task is the goal of a sub-area called *global optimization* that has produced many deterministic and stochastic methods to accomplish it. Due to practical reasons, optimizers are often satisfied with local rather than global solutions. A *local minimizer* or *local solution* for a optimization problem is a point x^* such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{N} \cap \mathcal{F}$, where $\mathcal{N}$ is a neighborhood of x^*. In order to distinguish it from the next definition, we call such a point a *weak local minimizer*. Finally, we define a *strict local minimizer* as a point x^* such that $f(x^*) < f(x)$ for all $x \in \mathcal{N} \cap \mathcal{F}$ with $x \neq x^*$. Notice that, in the case where $\mathcal{F}$ is convex and f is convex over $\mathcal{F}$, every local minimizer is also a global minimizer.

Instead of examining all the points neighboring a given point x^* to find out whether it is a local solution, we may use some theoretical results derived from Taylor's theorem to identify local minimizers when the functions are smooth. First, we consider the unconstrained case $\mathcal{F} = \mathbb{R}^n$. The proofs of the following theorems are readily found in most of the nonlinear programming textbooks, such as those of Bertsekas (1999) and Nocedal and Wright (1999).

Theorem 2.2.1. (First-Order Necessary Conditions). If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then

$$\nabla f(x^*) = 0. \tag{2.2.1}$$

When a point x^* satisfies (2.2.1), it is called a *first-order stationary point*. In numerical optimization, most of the algorithms are designed to find stationary points, which might or might not be a local minimizer.

The following theorem states necessary conditions for local optimality by using information from second-order derivatives.

Theorem 2.2.2. (Second-Order Necessary Conditions). If x^* is a local minimizer and $\nabla^2 f$ exists and is continuous in an open neighborhood of x^*, then

$$\nabla f(x^*) = 0 \quad \text{and} \quad \nabla^2 f(x^*) \succeq 0. \tag{2.2.2}$$

We now give a sufficient condition for optimality that makes use of second-order derivatives of the objective function.

Theorem 2.2.3. (Second-Order Suffient Conditions). Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \succ 0$. Then x^* is a strict local minimizer of f.

Note that the conditions in Theorem 2.2.3 are not necessary, that is, a point x^* may be a local minimizer without satisfying them.

As for constrained problems, we consider first the case where the feasible set $\mathcal{F}$ is convex. At a local minimizer x^*, we expect that the first order variation $\nabla f(x^*)^T d$ along a feasible direction d is non-negative. Due to the convexity of $\mathcal{F}$, feasible directions have the form $d = x - x^*$, where $x \in \mathcal{F}$. The following theorem is derived from that observation and also gives a sufficient condition for attaining global minimization.

Theorem 2.2.4. (First-Order Necessary Conditions for Constrained Problems). If x^* is a local minimizer of (2.1.1) and $\mathcal{F}$ is convex, then

$$\nabla f(x^*)^T (x - x^*) \geq 0, \quad \text{for all } x \in \mathcal{F}. \tag{2.2.3}$$

In addition, if f is convex over $\mathcal{F}$, then (2.2.3) is sufficient for x^* to minimize f over $\mathcal{F}$.

In pursuance of establishing necessary optimality conditions for problems with general constraints, we define the Lagrangian function for the problem (2.1.1), where $\mathcal{F}$ is given by (2.1.2), as

$$\mathcal{L}(x, \lambda, \mu) \stackrel{\text{def}}{=} f(x) + \sum_{i \in \mathcal{E} \cup \mathcal{I}} [\mu]_i c_i(x).$$

Moreover, we say that a point x is *regular* if the set of active constraint gradients $\{\nabla c_i(x),\ i \in \mathcal{A}(x)\}$, where $\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \,|\, c_i(x) = 0\}$, is linearly independent.

We now are ready to state the first-order necessary conditions for optimality in the general constrained case. Such conditions are widely known as the *Karush-Kuhn-Tucker conditions*, or simply *KKT conditions*, and are often used by many algorithms as their main stopping criteria.

Theorem 2.2.5. (Karush-Kuhn-Tucker Conditions). Suppose that x^* is a local minimizer of (2.1.1), that the functions f and c_i are continuously differentiable, and that x^* is regular. Then there is a Lagrange multipliers vector μ^* such that the following conditions are satisfied at (x^*, μ^*)

$$\begin{array}{rcll} \nabla_x \mathcal{L}(x^*, \mu^*) & = & 0, & \\ c_i(x^*) & = & 0, & \text{for all } i \in \mathcal{E}, \\ c_i(x^*) & \leq & 0, & \text{for all } i \in \mathcal{I}, \\ [\mu]_i^* \, c_i(x^*) & = & 0, & \text{for all } i \in \mathcal{I}, \\ [\mu]_i^* & \geq & 0, & \text{for all } i \in \mathcal{I}. \end{array} \tag{2.2.4}$$

When a point x^* satisfies the KKT conditions (2.2.4), it is called a *first-order stationary point* or a *KKT point* for the constrained problem (2.1.1). In constrained optimization, the variables $[x]_i$ are called the *primal* variables, while the variables $[\mu]_i$ are called the *dual* variables.

The results of Theorem 2.2.5 still hold in cases other than x being regular. There is a number of weaker conditions on the objective and constraint functions called *constraint qualifications* that guarantee the existence of Lagrange multipliers $[\mu]_i^*$ once they hold at x^*. An introduction to the Lagrange multiplier theory, where other constraint qualifications are also presented, can be found in Chapter 3 of the textbook by Bertsekas (1999).

2.3 Algorithms for unconstrained optimization

2.3.1 Linesearch methods

Linesearch methods are characterized by iterates of the type

$$x_{k+1} = x_k + \alpha_k d_k.$$

First, a *search direction* d_k for the current point x_k is computed; then a steplength α_k is chosen in the hope of reducing the objective function significantly.

The directions d_k are often specified in the form

$$d_k = -B_k^{-1} \nabla f(x_k).$$

A direction d is said to be a *descent direction* for a point x if $\langle \nabla f(x), d\rangle < 0$. Requiring d_k to be a descent direction guarantees that f can be reduced along d_k for a sufficiently small choice of α_k, since we have

$$f(x_k + \alpha d_k) = f(x_k) + \alpha_k \langle \nabla f(x_k), d_k\rangle + O(\alpha^2).$$

When $B_k = I_n$, the direction d_k becomes simply the negative of the gradient of f at x_k. Such a direction, also known as the *Cauchy direction*, is the one computed by the *steepest descent method*, originally proposed by Cauchy (1847) for the solution of systems of equations. Although it does not require many operations and is rather simple, its rate of convergence is only linear and it can be quite slow in many occasions, such as in poorly scaled problems.

When the Hessian $\nabla^2 f(x_k)$ exists and $\nabla^2 f(x_k) \succ 0$, we can define $B_k = \nabla^2 f(x_k)$. In this case, $d_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ is a descent direction, for we have

$$\langle d_k, \nabla f(x_k)\rangle = -\langle d_k, \nabla^2 f(x_k) d_k\rangle \leq -\lambda_k \|d_k\|^2,$$

where $\lambda_k > 0$ is the smallest eigenvalue of $\nabla^2 f(x_k)$. Such a direction is the one computed by Newton's method. Due to the use of second-order derivatives, Newton's method can converge quadratically, which usually makes it much faster than the steepest descent method. However, differently from the steepest descent method (see Curry, 1944), Newton's method is only *locally convergent*, which means that the starting point x_0 must be close enough to x^* in order to achieve convergence. This happens because the steps computed away from the solution x^* with $\nabla^2 f(x^*) \succ 0$ might not even be descent directions, since the Hessian matrix $\nabla^2 f(x_k)$ may not always be positive definite. To obtain *global convergence*, strategies such as modifying the Hessians $\nabla^2 f(x_k)$ for the computation of the directions can be used. In Figure 2.1, we show an example where the directions computed by Newton's method without any globalization strategy are not always descent directions, preventing the method to converge to the solution x^*, whereas the steepest descent method converges to the solution successfully.

Other approaches choose B_k as an approximation of $\nabla^2 f(x_k)$ rather than the exact Hessian, which avoids its explicit computation at every iteration. Such methods are called *quasi-Newton methods*, or *secant methods*, and were first developed by the physicist Davidon in the mid 1950s at the Argonne National Laboratory. Although his paper on this method was not accepted for publication at that time, being published only in 1991 in the *SIAM Journal on Optimization* (see Davidon, 1991), it became a turning point in numerical optimization and gave birth to a wide range of new algorithms.

In quasi-Newton methods, the matrices B_k are updated at every iteration by using a predefined formula that takes into account the knowledge acquired from past iterations. This class of methods is an attractive alternative for large-scale optimization, for instance, where the problems have large number of variables and expensive computations are avoided as much as possible. Quasi-Newton methods usually have superlinear rate of convergence and thus can converge rapidly to a solution x^*, albeit no use of second-order derivatives is made.

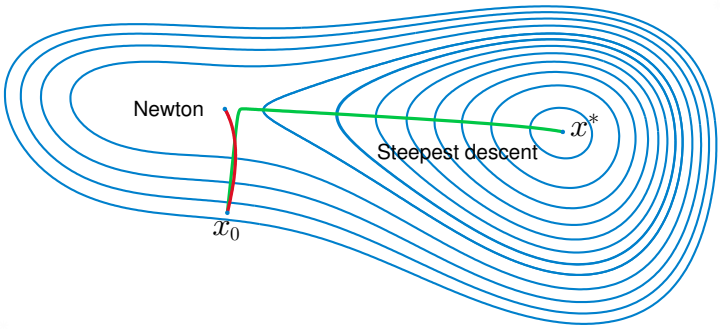


Figure 2.1: Trajectories of the iterates from Newton's method (red line) without any globalization strategy and the steepest descent method (green line) in a 2-dimensional problem. Since Newton's directions are not always descent directions, the method fails to converge to the solution x^*, while the steepest descent method achieves convergence without problems.

In Figure 2.2, an example is shown with the paths followed by Newton's method and the steepest descent method to reach the solution x^* from an initial guess x_0 using small step sizes. Notice that Newton's method goes through a shorter trajectory by exploiting curvature information of the function.

Once the search direction d_k has been computed, one has to choose the length of the step along d_k. As it was mentioned above, the steplength α_k is chosen so that the reduction in f is substantial and drives the method towards convergence. Since asking that $f(x_k+\alpha_k d_k) < f(x_k)$ is not enough to guarantee convergence to a minimizer, a *sufficient decrease condition* is necessary for this purpose. Among the conditions that ensure that α_k will provoke considerable reduction in f along d_k, we can cite two that are commonly employed: the *Wolfe's conditions* and the *Goldstein's conditions*.

Wolfe's conditions are expressed by the following inequalities

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \langle \nabla f(x_k), d_k \rangle, \tag{2.3.1a}$$

$$\langle \nabla f(x_k + \alpha_k d_k), d_k \rangle \geq c_2 \langle \nabla f(x_k), d_k \rangle, \tag{2.3.1b}$$

where $0 < c_1 < c_2 < 1$. The inequality (2.3.1a) is a popular sufficient decrease condition known as the *Armijo's condition*. However, sufficient decrease conditions alone are not enough to ensure a reasonable progress because very short steps can be taken. To avoid this scenario, the curvature condition (2.3.1b) is also demanded.

Goldstein's conditions are quite similar to (2.3.1), differing only at the second inequality. More formally, they are expressed by

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c\, \alpha_k \langle \nabla f(x_k), d_k \rangle, \tag{2.3.2a}$$

$$f(x_k + \alpha_k d_k) \geq f(x_k) + (1 - c) \alpha_k \langle \nabla f(x_k), d_k \rangle, \tag{2.3.2b}$$

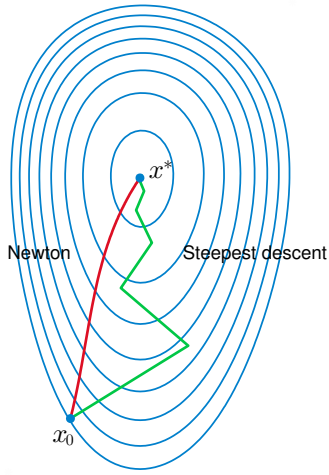


Figure 2.2: Trajectories of the iterates from Newton's method (red line) and the steepest descent method (green line) in a 2-dimensional optimization problem. As opposed to the latter, Newton's method uses curvature information to achieve convergence more rapidly.

where $0 < c < 1/2$.

Another alternative is to combine a sufficient decrease condition, such as the Armijo's condition, with a backtracking linesearch to ensure that the function is sufficiently reduced along the direction d_k. In this case, we obtain the algorithm described below.

Algorithm 2.3.1: Backtracking linesearch.

Step 0: Initialization. Choose $\bar{\alpha} > 0$, $\delta \in (0, 1)$ and $c \in (0, 1)$.

Set $\alpha = \bar{\alpha}$.

Step 1: Repeat until $f(x_k + \alpha d_k) \leq f(x_k) + c\,\alpha\langle \nabla f(x_k), d_k\rangle$

$\alpha \leftarrow \delta\alpha$.

Step 2: Return $\alpha_k = \alpha$.

"Non-monotone" generalizations of these algorithms, where the monotonicity property is abandoned, are also possible (see Grippo, Lampariello and Lucidi, 1986, 1989, and Toint, 1996). In those methods, we impose that the function value of each new iterate satisfies the Armijo's condition with respect to the maximum value of a prefixed number of previous iterates. The resulting

non-monotone backtracking linesearch is stated as the Algorithm 2.3.2 in what follows.

Algorithm 2.3.2: Non-monotone backtracking linesearch.

Step 0: Initialization. Choose $M \geq 0$, $\bar{\alpha} > 0$, $\delta \in (0,1)$ and $c \in (0,1)$.

Set $\alpha = \bar{\alpha}$.

Step 1: Repeat until $f(x_k + \alpha d_k) \leq \max\limits_{0 \leq j \leq M} [f(x_{k-j})] + c\,\alpha \langle \nabla f(x_k), d_k \rangle$

$\alpha \leftarrow \delta \alpha$.

Step 2: Return $\alpha_k = \alpha$.

2.3.2 Trust-region methods

Trust-region methods are a class of iterative methods relatively new when compared to linesearch methods; they date back to 1944, when Kenneth Levenberg proposed a method for nonlinear equations $F(x) = 0$ (see Levenberg, 1944), where $F : \mathbb{R}^n \to \mathbb{R}^m$. In Levenberg's method, the step is calculated as

$$d_k = -(J(x_k)^T J(x_k) + \lambda_k I_n)^{-1} J(x_k)^T F(x_k), \tag{2.3.3}$$

where $J(x_k)$ is the Jacobian matrix of F at x_k and $\lambda_k \geq 0$ is a damping parameter introduced to overcome the ill condition of $J(x_k)$, being adjusted at each iteration. This parameter can be viewed as a means of choosing which type of method is more appropriate for the iteration depending on the progress on reducing $\|F(x)\|$: if λ is large, the term $J(x_k)^T J(x_k)$ becomes negligible and the direction tends to the steepest descent; if λ is small, it tends to Newton's direction where the second-order term is neglected, as in Gauss-Newton methods.

Curiously, the same method was developed independently by Morrison (1960) in the Space Technology Laboratories (STL), California, in a work where the trajectory of a missile is estimated by nonlinear least-squares. With application to STL programs for lunar and interplanetary flights as well as Earth satellite tracking, his subroutine was proposed in a paper published in a seminar on tracking programs and orbit determination sponsored by the Jet Propulsion Laboratory of the California Institute of Technology.

The method was later reprised by the statistician Donald Marquardt in 1963 (see Marquardt, 1963), when he was working at DuPont, an American chemical company. Marquardt realized that for large λ_k, when the algorithm approaches the steepest descent method, we can still benefit from the Hessian approximation matrix $J(x_k)^T J(x_k)$ by using it to scale the gradient, thereby

implying on larger steps when the gradient is small. Marquardt's suggestion for the direction d_k is given by

$$d_k = -(J(x_k)^T J(x_k) + \lambda_k \, diag(J(x_k)^T J(x_k))^{-1} J(x_k)^T F(x_k). \tag{2.3.4}$$

The Levenberg-Morrison-Marquardt method has the property that, for some scalar Δ_k related to the damping parameter λ_k, the vector d_k solves the following problem

$$\begin{cases} \min\limits_{d} & \|F(x_k) + J(x_k)d\| \\ \text{s.t.:} & \|d\| \leq \Delta_k. \end{cases} \tag{2.3.5}$$

Because of the constraint in (2.3.5), it can be viewed as a trust-region method.

As opposed to linesearch methods, where a direction is computed followed by the choice of the steplength, trust-region methods first choose the maximum step size Δ_k and then compute a new step d_k. The direction d_k is calculated through the use of an approximation model $m_k(x_k + d)$ in $\mathcal{B}_k \stackrel{\text{def}}{=} \mathcal{B}(x_k; \Delta_k)$; usually, a quadratic model is employed. At each iteration, the algorithm solves the trust-region problem

$$\begin{cases} \min\limits_{d} & m_k(x_k + d) \\ \text{s.t.:} & \|d\| \leq \Delta_k, \end{cases} \tag{2.3.6}$$

where

$$m_k(x_k + d) \stackrel{\text{def}}{=} f(x_k) + \langle \nabla f(x_k), d_k \rangle + \langle d, B_k d \rangle,$$

B_k is a $n \times n$ symmetric matrix that approximates the Hessian $\nabla^2 f(x_k)$ and $\Delta_k > 0$ is the *trust region radius* that defines the region where the model m_k can be "trusted".

As in the linesearch methods, different choices of B_k produce different directions d_k. When $B_k = I_n$, the direction d_k is the negative of the gradient limited by the trust region radius Δ_k, i.e.

$$d_k = -\Delta_k \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}.$$

When $B_k \succ 0$ and $\|B_k^{-1} \nabla f(x_k)\| \leq \Delta_k$, the solution to (2.3.6) is given by $d_k = -B_k^{-1} \nabla f(x_k)$. In this case, the Newton and quasi-Newton directions are the same as those discussed in the previous section. In other cases, however, the solution to the subproblem (2.3.6) is not direct and algorithms such as the one proposed by Moré and Sorensen (1983) and the Truncated Conjugate Gradient method can be used.

After a trial step has been calculated, the algorithm proceeds by verifying whether it produces a sufficient decrease of f. This is made by analyzing the ratio of the actual reduction over the predicted reduction given by

$$\rho = \frac{f(x_k) - f(x_k + d_k)}{m_k(x_k) - m_k(x_k + d_k)}. \tag{2.3.7}$$

If ρ approaches 1, it means that the model m_k is a good approximation to f in $\mathcal{B}_k$; in this case, the iteration is called a *successful iteration* and the trust region radius may be increased. On the other hand, if ρ is very small or even negative, we conclude that m_k does not approximate f properly in $\mathcal{B}_k$; those iterations are called *unsuccessful iterations* and have the trust region radius decreased by some factor. If ρ is much bigger than 1, the decrease in the objective function was larger than expected, which means that the model is a poor representation of the design space. In this case, however, the increase in the trust region radius is justified as one obtained more reduction than predicted.

To put the discussion above in a more formal description, we present a basic trust-region method in what follows.

Algorithm 2.3.3: Basic trust-region method (BTR).

Step 0: Initialization. An initial point x_0 and an initial trust region radius Δ_0 are given as well as the constants

$$0 < \gamma_1 \leq \gamma_2 < 1 \quad \text{and} \quad 0 < \eta_1 \leq \eta_2 < 1.$$

Compute $f(x_0)$ and set $k = 0$.

Step 1: Model definition. Define a model m_k in $\mathcal{B}_k$.

Step 2: Step calculation. Compute a step d_k that "sufficiently reduces the model" m_k and such that $x_k + d_k \in \mathcal{B}_k$.

Step 3: Acceptance of the trial point. Compute $f(x_k + d_k)$ and define ρ_k as in (2.3.7). If $\rho \geq \eta_1$, then define $x_{k+1} = x_k + d_k$; otherwise, define $x_{k+1} = x_k$.

Step 4: Trust-region radius update. Set

$$\Delta_{k+1} = \begin{cases} [\Delta_k, \infty) & \text{if } \rho_k \geq \eta_2, \\ [\gamma_2 \Delta_k, \Delta_k] & \text{if } \rho_k \in [\eta_1, \eta_2), \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if } \rho_k < \eta_1. \end{cases} \tag{2.3.8}$$

Increment k by 1 and go to Step 1.

For a thorough analysis of trust-region methods, the reader is refereed to the textbook of Conn, Gould and Toint (2000).

2.3.3 Conjugate gradient methods

The conjugate gradient (CG) method was originally developed by Hestenes and Stiefel (1952) as an algorithm for solving systems of linear equations

$$Ax = b, \tag{2.3.9}$$

where A is a symmetric positive definite matrix. It can be also applied in optimization for solving unconstrained problems where the objective function is quadratic with its Hessian being symmetric positive definite. In other words, it may be used to solve the problem

$$\begin{cases} \min\limits_{x} & f(x) \stackrel{\text{def}}{=} \langle b, x\rangle + \langle x, Ax\rangle \\ \text{s.t.:} & x \in \mathbb{R}^n. \end{cases} \tag{2.3.10}$$

Since $\nabla f(x) = Ax - b$, requiring $\nabla f(x) = 0$ is equivalent to solve a linear system as in (2.3.9).

The CG method is an iterative algorithm that computes directions holding a *conjugacy* property with respect the Hessian matrix. Given a set of nonzero vectors $\{d_0, d_1, \ldots, d_p\}$, we say that these vectors are conjugate with respect to the symmetric positive definite matrix A if

$$\langle d_i, Ad_j\rangle = 0, \quad \text{for all } \; i \neq j. \tag{2.3.11}$$

It can be proven that, with only n of such vectors, it is possible to find the solution x^* to the problem (2.3.10) in at most n steps (see Theorem 5.1 in Nocedal and Wright, 1999, for instance). However, computing an entire set of conjugate directions might be very expensive and thus, for large-scale problems, it turns out to be a bad idea. The CG method provides a way of generating these directions without keeping the entire conjugate directions set by only using the gradient and the direction computed at the last iteration.

Let x_0 be a initial guess to (2.3.10) and define the residual as

$$r_k \stackrel{\text{def}}{=} Ax_k - b.$$

It is reasonable to take the negative of the gradient of f at x_0 as the first direction, as the steepest descent method would do. Thus, we set

$$d_0 = -r_0 = b - Ax_0.$$

If the user does not have any information about the solution, the zero vector $x_0 = 0$ can be used.

Consider now the iteration k and suppose that the directions $\{d_0, \ldots, d_{k-1}\}$ are conjugate with respect to A. We shall try to find a direction d_k to compute x_{k+1}. The negative residual $-r_k$ may be used again, but we need d_k to be conjugate to all the previous directions. By defining the direction d_k as a linear combination of the steepest descent direction $-r_k$ and the previous direction d_{k-1} and imposing the conjugacy property between d_k and d_{k-1}, we can obtain a set of conjugate directions $\{d_0, d_1, \ldots, d_{k-1}, d_k\}$ (see Theorem 5.3 in Nocedal and Wright, 1999). In other words, the direction that we seek is expressed by

$$d_k = -r_k + \beta_k\, d_{k-1}, \tag{2.3.12}$$

where β_k is a coefficient appropriately chosen to make the directions d_k and d_{k-1} conjugate. By pre-multiplying (2.3.12) by $d_{k-1}^T A$, we obtain

$$\langle d_{k-1}, Ad_k \rangle = -\langle d_{k-1}, Ar_k \rangle + \beta_k \langle d_{k-1}, Ad_{k-1} \rangle. \tag{2.3.13}$$

By imposing the condition $\langle d_{k-1}, Ad_k \rangle = 0$, we have

$$\beta_k = \frac{\langle d_{k-1}, Ar_k \rangle}{\langle d_{k-1}, Ad_{k-1} \rangle}. \tag{2.3.14}$$

As it can be seen, the CG method only needs the current gradient $-r_k$ and the previous direction d_{k-1} at each iteration, which can lead to enormous savings of calculations and memory for large-scale problems.

Once the direction d_k has been computed, the steplength α_k is calculated as the exact minimizer of f along $x_k + \alpha d_k$. Since f is quadratic, the expression for α_k is easily obtained by

$$\alpha_k = -\frac{\langle r_k, d_k \rangle}{\langle d_k, Ad_k \rangle}. \tag{2.3.15}$$

By using (2.3.12) and the orthogonality of the gradient with respect to all the previous directions (see Theorem 5.2 in Nocedal and Wright, 1999), i.e,

$$\langle r_k, d_i \rangle = 0, \quad \text{for all} \quad i = 0, \ldots, k-1,$$

we may replace (2.3.15) by

$$\alpha_k = \frac{\langle r_k, r_k \rangle}{\langle d_k, Ad_k \rangle}. \tag{2.3.16}$$

If we also use the fact that $\alpha_k A d_k = d_{k+1} - d_k$, then β_{k+1} may be written as

$$\beta_{k+1} = \frac{\langle r_{k+1}, r_{k+1} \rangle}{\langle r_k, r_k \rangle}. \tag{2.3.17}$$

The complete CG method is stated here as the Algorithm 2.3.4.

Algorithm 2.3.4: Conjugate gradient method.

Initialization. Choose an initial point x_0. Set $r_0 = Ax_0 - b$, $d_0 = -r_0$ and $k = 0$.

Repeat until $r_k \neq 0$

$$\alpha_k \leftarrow \frac{\langle r_k, r_k \rangle}{\langle d_k, Ad_k \rangle}; \tag{2.3.18a}$$

$$x_{k+1} \leftarrow x_k + \alpha_k d_k; \tag{2.3.18b}$$

$$r_{k+1} \leftarrow r_k + \alpha_k \, Ad_k; \tag{2.3.18c}$$

$$\beta_{k+1} \leftarrow \frac{\langle r_{k+1}, r_{k+1} \rangle}{\langle r_k, r_k \rangle}; \tag{2.3.18d}$$

$$d_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} d_k; \tag{2.3.18e}$$

$$k \leftarrow k + 1. \tag{2.3.18f}$$

In Figure 2.3, a 2-dimensional example is shown with the paths followed by the CG method and the steepest descent method to reach the solution x^* from an initial guess x_0. Notice that the CG method takes only 2 iterations to find the solution, as at most n iterations are necessary for this method to converge for convex quadratics.

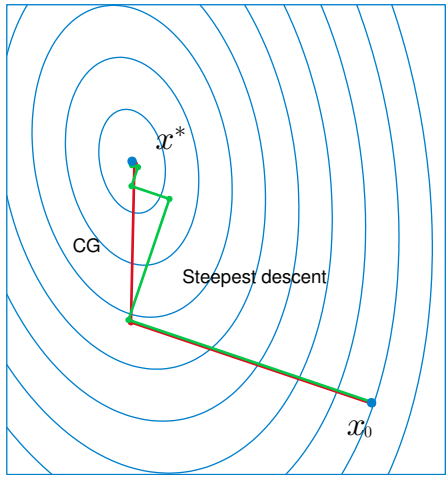


Figure 2.3: Trajectories of the iterates from the CG method (red line) and the steepest descent method (green line) in a 2-dimensional optimization problem. As expected, the CG method takes no more than 2 iterations to converge.

The rate of convergence of the method can be stated in terms of the eigenvalues of the Hessian matrix A, as it is shown in the next theorem (proof in Theorem 5.1.6, Conn et al., 2000).

Theorem 2.3.1. If A has ℓ distinct eigenvalues, the conjugate gradient Algorithm 2.3.4 will terminate with $x_j = x^*$ for some $j \leq \ell$.

In addition to the eigenvalues of the matrix A, its condition number $\kappa(A) = \|A\| \, \|A^{-1}\|$ also plays an important role in the convergence of the method, as the theorem below shows (proof in Theorem 5.1.7, Conn et al., 2000).

Theorem 2.3.2. The sequence $\{x_k\}$ of iterates generated by the Algorithm 2.3.4 satisfies the inequality

$$\|x_{k+1} - x^*\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|x_0 - x^*\|_A. \tag{2.3.19}$$

We will see in the next subsection that the Theorems 2.3.1 and 2.3.2 are very useful when one aims to accelerate the convergence of CG methods.

2.3.4 Preconditioned CG methods

The rate of convergence of CG methods can be improved by means of *preconditioning* of the Hessian matrix A. This is done by choosing an invertible matrix R and making a change of variables as

$$\bar{x} = Rx.$$

The resulting preconditioned problem is given by

$$\min_{\bar{x}} \quad \bar{f}(\bar{x}) \stackrel{\text{def}}{=} \langle \bar{b}, \bar{x} \rangle + \langle \bar{x}, \bar{A}\bar{x} \rangle \tag{2.3.20}$$

where

$$\bar{H} = R^{-T} H R^{-1} \text{ and } \bar{b} = R^{-T} b.$$

The intention is to choose R such that the eigenvalues of the matrix $R^{-T} H R^{-1}$ are clustered and/or its condition number is much smaller than that of H and ideally close to one. The motivation behind this is in the Theorems 2.3.1 and 2.3.2.

Fortunately, it is not necessary to carry out computations involving $\bar{H}$ and $\bar{c}$. It is possible to derive a method where only H and c participate in the operations and we still (implicitly) use the preconditioning matrix R. For such purpose, we define the preconditioner $M = R^T R$. Note that the eigenvalues of $R^{-T} H R^{-1}$ are the same of those of $M^{-1} H$, which implies that we can affect the eigenvalues of $\bar{H}$ by directly working with the preconditioner M instead

of R. After a change of variables, the preconditioned CG method can then be described as follows.

Algorithm 2.3.5: Preconditioned CG method.

Initialization. An initial point x_0 and preconditioner M. Set $r_0 = Ax_0 - b$ and let $y_0 = M^{-1}r_0$. Set $d_0 = -y_0$ and $k = 0$.

Repeat until $r_k \neq 0$

$$\alpha_k \leftarrow \frac{\langle r_k, y_k\rangle}{\langle d_k, Ad_k\rangle}; \tag{2.3.21a}$$

$$x_{k+1} \leftarrow x_k + \alpha_k d_k; \tag{2.3.21b}$$

$$r_{k+1} \leftarrow r_k + \alpha_k\, Ad_k; \tag{2.3.21c}$$

$$y_{k+1} \leftarrow M^{-1}r_{k+1}; \tag{2.3.21d}$$

$$\beta_{k+1} \leftarrow \frac{\langle r_{k+1}, y_{k+1}\rangle}{\langle r_k, y_k\rangle}; \tag{2.3.21e}$$

$$d_{k+1} \leftarrow -y_{k+1} + \beta_{k+1} d_k; \tag{2.3.21f}$$

$$k \leftarrow k + 1. \tag{2.3.21g}$$

2.4 Algorithms for constrained optimization

We describe in this section a few existing methods for solving constrained optimization problems that are related in some extent to the work developed in this thesis. We note that there are many other popular methods for constrained optimization, such as interior point and augmented Lagrangian methods, that are not discussed here but are readily found in several nonlinear programming textbooks.

2.4.1 Projected preconditioned CG methods

The conjugate gradient (CG) method can also be used to solve linearly constrained problems by using preconditioning and projection. With a view to simplify the description of the generalized method, we shall assume that A is a $m \times n$ full-rank matrix, where $m \leq n$. The linearly constrained, quadratic optimization problem of interest is given as

$$\begin{cases} \min\limits_{x} & f(x) \stackrel{\text{def}}{=} \langle c, x\rangle + \langle x, Hx\rangle \\ \text{s.t.:} & Ax = b. \end{cases} \tag{2.4.1}$$

If the problem contains inequalities, one can use an active-set approach to predict the indices of those constraints which will be active at the solution. In this case, the algorithms tries to solve a equality-constrained problem of the type (2.4.1) whose constraints are defined by the working set, a subset of the indices of constraints which are active at the current point.

Let Z be a $n \times (n-m)$ matrix whose columns span the null space of A. Then the columns of the matrix $(A^T\ Z)$ form a basis for $\mathbb{R}^n$, and any vector x such that $Ax = b$ can be written as

$$x = A^T x_A + Z x_Z, \tag{2.4.2}$$

for some vectors $x_A \in \mathbb{R}^m$ and $x_Z \in \mathbb{R}^{n-m}$. Using (2.4.2) and the fact that $AZ = 0$, we have

$$b = Ax = AA^T x_A + AZx_Z = AA^T x_A, \tag{2.4.3}$$

from which the vector x_A can be determined by

$$x_A = (AA^T)^{-1} b. \tag{2.4.4}$$

Substituting (2.4.2) into the problem (2.4.1), the latter becomes equivalent to the reduced problem in x_Z given by

$$\begin{cases} \min\limits_{x_Z} & \langle c_Z, x_Z \rangle + \langle x_Z, H_{ZZ} x_Z \rangle \\ \text{s.t.:} & x_Z \in \mathbb{R}^{n-m}, \end{cases} \tag{2.4.5}$$

where

$$H_{ZZ} = Z^T H Z, \quad c_Z = Z^T (HA^T x_A + c),$$

and the constant terms involving x_A have been omitted. We assume here that the matrix H_{ZZ} is positive definite. In the next section, we analyze the case where it may be indefinite. Since (2.4.5) is an unconstrained convex quadratic problem, we can apply the Algorithm 2.3.4 on page 16 to obtain a solution x_Z^*. Once we have x_Z^*, we can retrieve the solution x^* to the constrained problem (2.4.1) by (2.4.2).

With the purpose of improving the rate of convergence of the method, a preconditioning (symmetric, positive definite) matrix W_{ZZ} may be used to cluster the eigenvalues of $W_{ZZ}^{-1} H_{ZZ}$ around ℓ values, for some $\ell \ll (n-m)$, and to obtain a condition number $\kappa(W_{ZZ}^{-1} H_{ZZ})$ much smaller than that of H_{ZZ}. Since W_{ZZ} should approximate H_{ZZ}, the ideal choice would be one that makes $W_{ZZ}^{-1} H_{ZZ} = I_{(n-m)}$, which implies that

$$W_{ZZ} = Z^T H Z. \tag{2.4.6}$$

For this reason, the preconditioner W_{ZZ} is usually written in the form

$$W_{ZZ} = Z^T G Z, \tag{2.4.7}$$

where G is any symmetric matrix such that Z^TGZ is positive definite.

By a change of variables, we can avoid working in the $(n-m)$-dimensional space and making operations with the null-space matrix Z. Let P be a $n \times n$ projection matrix defined by $P = Z(Z^TGZ)^{-1}Z^T$, where G is the preconditioning matrix from (2.4.7). Suppose we are given an initial guess x satisfying $Ax = b$. The projected preconditioned CG algorithm for (2.4.1) is given below.

Algorithm 2.4.1: Projected preconditioned CG method.

Initialization. Given x such that $Ax = b$, set $r = Hx+c$, $g = Pr$ and $d = -g$.

Repeat until stopping criteria is satisfied

$$\alpha \leftarrow \frac{\langle r, g\rangle}{\langle d, Hd\rangle}; \tag{2.4.8a}$$

$$x \leftarrow x + \alpha d; \tag{2.4.8b}$$

$$r^+ \leftarrow r + \alpha\, Hd; \tag{2.4.8c}$$

$$g^+ \leftarrow Pr^+; \tag{2.4.8d}$$

$$\beta \leftarrow \frac{\langle r^+, g^+\rangle}{\langle r, g\rangle}; \tag{2.4.8e}$$

$$d \leftarrow -g^+ + \beta d; \tag{2.4.8f}$$

$$g \leftarrow g^+; \quad r \leftarrow r^+. \tag{2.4.8g}$$

In Gould, Hribar and Nocedal (2001), the authors show that the use of projections can cause significant rounding errors and propose different approaches using iterative refinement and a residual update strategy in order to reduce those errors.

2.4.2 Truncated projected CG methods

It is possible to extend CG methods to constrained problems where the objective function is not strictly convex. Consider the following trust-region quadratic problem

$$\begin{cases} \min\limits_{x} & f(x) \stackrel{\text{def}}{=} \langle c, x\rangle + \langle x, Hx\rangle \\ \text{s.t.:} & Ax = b, \\ & \|x\| \leq \Delta, \end{cases} \tag{2.4.9}$$

where $\Delta > 0$ is the trust region radius. Suppose that we apply the Projected Preconditioned CG method to (2.4.9) regardless of whether H is positive definite or not. If it happens that the steplength α_k is such that $\|x_k + \alpha_k d_k\| \geq \Delta$

at iteration k, we simply stop at the boundary of the trust-region when walking along d_k, i.e. we choose another steplength σ_k as the positive root of

$$\|x_k + \sigma d_k\| = \Delta, \tag{2.4.10}$$

and take the point $x_k + \sigma_k d_k$ as the new iterate. In cases where we encounter $\langle d_k, Hd_k\rangle \leq 0$, the method can benefit from the negative curvature by reducing f along $x_k + \alpha d_k$ as much as possible while staying within the trust region, which implies that the new point x_{k+1} will be at the trust-region boundary along this line, just as in (2.4.10).

In a more formal description, the truncated projected CG method, also referred to as the Steihaug-Toint Conjugate Gradient method (see Toint, 1981, and Steihaug, 1983), is stated as follows.

Algorithm 2.4.2: Truncated projected CG method.

Initialization. Given x such that $Ax = b$, set $r = Hx+c$, $g = Pr$ and $d = -g$.

Repeat until stopping criteria is satisfied

Set $\kappa_{cu} \leftarrow \langle d, Hd\rangle$ and $\alpha \leftarrow \langle r, g\rangle/\kappa_{cu}$.

If $\kappa_{cu} \leq 0$ or $\|x+\alpha d\| \geq \Delta$, compute σ as the positive root of $\|x+\sigma d\| = \Delta$ and set $x \leftarrow x + \sigma d$. Otherwise, set $x \leftarrow x + \alpha d$.

$$\begin{aligned} r^+ &\leftarrow r + \alpha\, Hd; \\ g^+ &\leftarrow Pr^+; \\ \beta &\leftarrow \frac{\langle r^+, g^+\rangle}{\langle r, g\rangle}; \\ d &\leftarrow -g^+ + \beta d; \\ g &\leftarrow g^+; \quad r \leftarrow r^+. \end{aligned}$$

2.4.3 Projected gradient methods

Consider the constrained problem

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{s.t.:} & x \in \mathcal{F}, \end{cases} \tag{2.4.11}$$

where $\mathcal{F}$ is a convex set, and denote by $P_{\mathcal{F}}[\cdot]$ the projection on the set $\mathcal{F}$. Projected gradient methods or gradient projection methods are characterized by iterates of the type

$$x_{k+1} = x_k + \alpha_k(\bar{x}_k - x_k),$$

where

$$\bar{x}_k = P_{\mathcal{F}}[x_k - \beta_k \nabla f(x_k)],$$

and $\alpha_k > 0$ and $\beta_k > 0$ may be viewed as steplengths. Basically, the algorithm consists on projecting a point obtained through the steepest descent method onto the feasible set in order to obtain a feasible point $\bar{x}_k$ that will be used to define the new direction $d_k = \bar{x}_k - x_k$. Thus, in its most simple form, this class of methods may be interpreted as the counterpart of steepest descent methods in constrained optimization.

Projection onto a convex set $\mathcal{F}$ might be an easy task depending on the nature of the constraint set. For example, it is straightforward when $\mathcal{F}$ is expressed by simple bounds on x as in

$$\mathcal{F} = \{x \,|\, l \leq x \leq u\}.$$

In this case, the projection is given componentwise by

$$[P_{\mathcal{F}}[x]]_i = \begin{cases} [l]_i & \text{if } [x]_i < [l]_i\,, \\ [x]_i & \text{if } [l]_i \leq [x]_i \leq [u]_i\,, \\ [u]_i & \text{if } [x]_i > [u]_i\,. \end{cases} \tag{2.4.12}$$

In other cases, however, it may not be so direct and may even require the solution of an optimization problem.

Similarly to the unconstrained case, the steplengths can be computed by means of a backtracking linesearch algorithm coupled with the Armijo's condition. The main difference is that we have now two possibilities: we can apply it along the feasible direction d_k or on the projection arc. In the former case, we set $\beta_k = \beta$, where β is a constant, and choose α_k from a backtracking linesearch along $x_k + \alpha d_k$. In the latter case, we fix $\alpha_k = 1$ and apply the backtracking linesearch on the projection arc

$$\mathcal{PA} = \{x_k(\beta) \,|\, \beta > 0\},$$

where, for all $\beta > 0$, $x_k(\beta)$ is defined by

$$x_k(\beta) = P_{\mathcal{F}}[x_k - \beta \nabla f(x_k)].$$

Figure 2.4 illustrates a step of the projected gradient method and the projection arc in an example where the feasible set is a polygon. Finally, the algorithm stops when it encounters a point x^* such that

$$x^* = P_{\mathcal{F}}[x^* - \beta \nabla f(x^*)]$$

for all $\beta > 0$, since this implies that x^* is a stationary point.

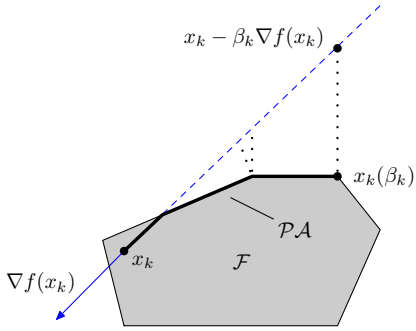


Figure 2.4: Graphical representation of the projected gradient method.

An in-depth analysis of projected gradient methods is encountered in the Chapter 12 of Conn et al. (2000) and in the Chapter 2 of Bertsekas (1999).

2.4.4 Sequential quadratic programming methods

Sequential quadratic programming (SQP) methods are one of the most popular methods for the solution of smooth nonconvex, nonlinear optimization problems; they were first proposed by Wilson (1963) in his Ph.D. thesis. At each iteration, a quadratic optimization problem is solved, where a quadratic approximation for the objective function and for the active constraints at the current point x_k are used to compute a new direction d_k. Their applicability to a broad class of nonlinear optimization problems and strong convergence results has driven many researchers from an exhaustive theoretical study to the development of several SQP codes, such as SNOPT (Gill, Murray and Saunders, 2005), NPSOL (Gill, Murray, Saunders and Wright, 2001), NLPQL (Schittkowski, 1986), KNITRO (Byrd, Nocedal and Waltz, 2006), LOQO (Vanderbei, 1999) and IPOPT (Wächter and Biegler, 2006).

SQP methods are strongly related to Newton's method. In fact, they can be viewed as an application of Newton's method to constrained optimization problems. For the sake of simplification, we consider the following constrained problem with only equality constraints to develop the connection between both methods:

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{s.t.:} & c(x) = 0. \end{cases} \tag{2.4.13}$$

The Lagrangian function for the problem (2.4.13) is defined by

$$\mathcal{L}(x, \mu) \stackrel{\text{def}}{=} f(x) + \langle \mu, c(x) \rangle.$$

We know from Theorem (2.2.5) that the first-order necessary conditions for the vector (x, μ) to be a primal-dual solution for (2.4.13) are

$$\nabla_x \mathcal{L}(x, \mu) = 0 \quad \text{and} \quad c(x) = 0. \tag{2.4.14}$$

Consider one iteration of Newton's method, starting at estimates x_k and μ_k of the primal and dual variables, and let d_k and d_k^μ be their corrections calculated by solving the following system of equations

$$\begin{pmatrix} \nabla^2_{xx}\mathcal{L}(x_k,\mu_k) & J^T(x_k) \\ J(x_k) & 0 \end{pmatrix} \begin{pmatrix} d_k \\ -d_k^\mu \end{pmatrix} = \begin{pmatrix} \nabla_x \mathcal{L}(x_k,\mu_k) \\ c(x_k) \end{pmatrix}, \tag{2.4.15}$$

where $J(x_k)$ is the Jacobian of c at x_k. Again using Theorem 2.2.5, we notice that the equations (2.4.15) represent the KKT conditions for the primal and dual solution (x_k, μ_k) of the quadratic optimization problem

$$\begin{cases} \min\limits_d & \langle \nabla_x \mathcal{L}(x_k,\mu_k), d\rangle + \frac{1}{2}\langle d, \nabla^2_{xx}\mathcal{L}(x_k,\mu_k)d\rangle \\ \text{s.t.:} & c(x_k) + J(x_k)d = 0. \end{cases} \tag{2.4.16}$$

By writing the dual variables as $\mu_{k+1} = \mu_k + d_k^\mu$, the following equations are analogous to (2.4.15)

$$\begin{pmatrix} \nabla^2_{xx}\mathcal{L}(x_k,\mu_k) & J^T(x_k) \\ J(x_k) & 0 \end{pmatrix} \begin{pmatrix} d_k \\ -\mu_{k+1} \end{pmatrix} = \begin{pmatrix} \nabla f(x_k) \\ c(x_k) \end{pmatrix}. \tag{2.4.17}$$

Finally, the equations (2.4.17) are the first-order optimality conditions for the primal and dual solution (x_k, μ_k) of the problem

$$\begin{cases} \min\limits_d & \langle \nabla f(x_k), d\rangle + \frac{1}{2}\langle d, \nabla^2_{xx}\mathcal{L}(x_k,\mu_k)d\rangle \\ \text{s.t.:} & c(x_k) + J(x_k)d = 0, \end{cases} \tag{2.4.18}$$

which is the quadratic optimization subproblem solved at each iteration in an SQP algorithm.

Consider now a more general problem where inequalities are also present:

$$\begin{cases} \min\limits_x & f(x) \\ \text{s.t.:} & c_{\mathcal{E}}(x) = 0, \\ & c_{\mathcal{I}}(x) \leq 0. \end{cases} \tag{2.4.19}$$

One way of handling (2.4.19) consists in solving first the following quadratic optimization subproblem

$$\begin{cases} \min\limits_d & \langle \nabla f(x_k), d\rangle + \frac{1}{2}\langle d, \nabla^2_{xx}\mathcal{L}(x_k,\mu_k)d\rangle \\ \text{s.t.:} & c_{\mathcal{E}}(x_k) + J_{\mathcal{E}}(x_k)d = 0, \\ & c_{\mathcal{I}}(x_k) + J_{\mathcal{I}}(x_k)d \leq 0, \end{cases} \tag{2.4.20}$$

and then use the active set for this subproblem as a prediction of that for (2.4.19). Another possibility is to define the active set $\mathcal{A}$ *a priori* (based on the inequalities that are close to be active and whose Lagrange multipliers estimates are positive) and solve the equality-constrained subproblem

$$\begin{cases} \min\limits_d & \langle \nabla f(x_k), d\rangle + \frac{1}{2}\langle d, \nabla^2_{xx}\mathcal{L}(x_k,\mu_k)d\rangle \\ \text{s.t.:} & c_{\mathcal{A}}(x_k) + J_{\mathcal{A}}(x_k)d = 0. \end{cases} \tag{2.4.21}$$

Although SQP methods have good local convergence, they may not converge if we simply take steps from the solution to (2.4.18). This may happen, for instance, when the starting point is far away from the solution. SQP methods may be divided into two classes of algorithms based on how they overcome this difficulty: linesearch SQP methods and trust-region SQP methods. In the first class, once the step d_k has been computed, a linesearch is performed by means of a *merit function*, a function used to ensure a balance between reducing the objective function and reducing infeasibility, and a steplength that reduces the value of the merit function is calculated similarly to the unconstrained case. In trust-region SQP methods, a trust-region constraint is added to (2.4.18) for the computation of the step. In this case, the trust-region radius is updated at each iteration based on the ratio of the actual reduction over the predicted reduction of a merit function.

A review of SQP methods can be found in the Chapter 15 of Conn et al. (2000) and in the survey papers of Gould and Toint (2000) and Gill and Wong (2012).

2.4.5 Filter methods

Filter methods were first proposed by Fletcher and Leyffer (2002) as an alternative to penalty methods, such as the augmented Lagrangian method. In penalty methods, the original constrained problem is replaced by a sequence of unconstrained subproblems in which a measure of constraint violation is added to the objective function. This measure is multiplied by a penalty parameter ρ that can be chosen adaptively. One of the difficulties encountered in these methods is the choice of an initial value for ρ, for a bad choice can make the unconstrained problem to be unbounded below even if the original constrained problem has a solution. Besides, depending on the choice of the measure of infeasibility, the unconstrained problem may be non-smooth regardless of smoothness of the constraints. In contrast to penalty methods, filter methods make no use of merit functions and thus are free from issues related to penalties; instead, convergence is controlled by a technique based on the concept of *dominance* from multiobjective optimization.

Consider a measure of infeasibility for any point x given by $v(x) \stackrel{\text{def}}{=} \|c(x)\|$. The two goals in constrained optimization are to achieve optimality and feasibility at some point x, which means that $v(x) = 0$ for the latter. This can be interpreted as a multiobjective optimization problem where the objective functions f and v are to be minimized. Notice that, if we have two vectors x and y such that $f(x) \leq f(y)$ and $v(x) \leq v(y)$, then we may dispose of y, since x is at least as "good" as y. This gives rise to the notion of dominance, where a pair $(f(x), v(x))$ is said to *dominate* another pair $(f(y), v(y))$ if $f(x) \leq f(y)$ and $v(x) \leq v(y)$.

A filter is a list of pairs $(f(x_k), v(x_k))$ stored by the algorithm such that no pair dominates any other. After a trial point $x^+ = x_k + d_k$ has been computed, the algorithm accepts it only if no pair in the filter dominates $(f(x^+), v(x^+))$.

In this case, $(f(x^+), v(x^+))$ is added to the filter and a new iteration begins. Figure 2.5 illustrates a filter with four pairs.

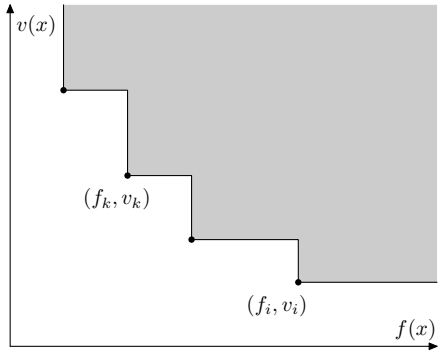


Figure 2.5: Graphical representation of the filter method. The shaded area contain all the points dominated by the four pairs (f_j, v_j) in the filter, which are indicated by a dot.

Because of the good numerical results obtained by Fletcher and Leyffer with a Filter-SQP method, many other papers were addressed later on to the implementation and convergence of algorithms that rely on the filter idea, such as Fletcher, Leyffer and Toint (2002*b*), Fletcher, Gould, Leyffer, Toint and Wächter (2002*a*), Fletcher, Leyffer, Ralph and Scholtes (2006), Biegler and Wächter (2005) and Ulbrich, Ulbrich and Vicente (2004). In particular, Colson (2004) and Audet and Dennis (2004) have developed filter methods where no derivatives are used.

2.4.6 Trust-funnel methods

The trust-funnel method was firstly introduced by Gould and Toint (2010) as an SQP algorithm for equality-constrained optimization problems whose convergence is driven by an adaptive bound $v_k^{\max}$ imposed on the allowed infeasibility $v(x) = \frac{1}{2}\|c(x)\|^2$ at each iteration. This bound is monotonically decreased as the algorithm progresses, assuring its global convergence whilst seeking optimality and hence originating the name "trust funnel". It belongs to the class of trust-region methods and makes use of a composite-step approach to calculate a new direction at each iteration: a normal step is firstly computed in the hope of reducing the infeasibility measure ensuing from the constraint functions' values, and a tangent step is subsequently calculated with the aim of improving optimality of the iterates with regard to the objective function. These computations are carried out with the use of two different trust regions, one for each step component. The main idea is to consider the objective function and the constraints as independently as possible. The method is noticeable among others for constrained problems as a parameter-free alternative, for neither filter nor penalties are needed, freeing the user from common difficulties

encountered when choosing the initial penalty parameter, for instance. An extension to problems with both equalities and inequalities was developed of late by Curtis, Gould, Robinson and Toint (2014), who presented an interior-point trust-funnel algorithm for solving large-scale problems that may be characterized as a barrier-SQP method.

As mentioned previously, this method employs a composite-step approach of the type suggested by Omojokun (1989) in his doctoral thesis under the supervision of R. H. Byrd as well as in the thesis by Andrew R. Conn in 1971 in a linesearch context (see Conn, 1976; Conn and Pietrzykowski, 1977; Conn and Coleman, 1982). In this technique, each full SQP step is decomposed as

$$d_k = n_k + t_k,$$

where the *normal step* component n_k aims to improve feasibility and the *tangent step* component t_k reduces the model while not jeopardizing the infeasibility reduction we have just obtained. The trial point $x_k^+ = x_k + d_k$ is accepted only if the constraint violation satisfies the funnel condition

$$v(x_k^+) \leq v_k^{\max}. \tag{2.4.22}$$

As feasibility is improved during the optimization process, the value of $v_k^{\max}$ is reduced according to the infeasibility of the iterates. Figure 2.6 illustrates the idea of funnel, exemplifying a path followed by the iterates while satisfying (2.4.22). The complete description of the algorithm, including the computation of each step component and the updating strategy of the funnel, is given together with our derivative-free adaptation presented in Chapter 4.

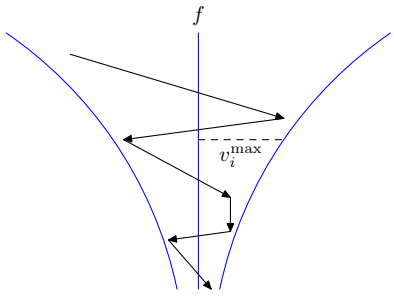


Figure 2.6: Graphical representation of the trust-funnel method.

Chapter 3

Introduction to derivative-free optimization

In this chapter, we introduce the reader to the field of derivative-free optimization (DFO). Firstly, we give a short summary of the main classes of existing algorithms for this area. Then, we elaborate on one of these classes in the interest of building a framework that will be the underlying support of our own algorithm presented in Chapter 4.

3.1 Existing methods

Within the range of DFO methods devised thus far, three main classes may be distinguished, namely: direct-search methods, derivative estimation by finite differences and model-based algorithms. The first class, also called *zero-order methods*, is rooted on the exploration of the variable space by generating a set of trial points at each iteration and having their function values compared to the best solution previously obtained. Hence these methods neither require nor attempt to approximate derivatives for the problem to be solved. The generation of the sample set usually follows a predefined geometric pattern, although one can also make use of directions randomly generated to explore the neighborhood of the iterate (see Gratton, Royer, Vicente and Zhang, 2014). In a general form, a direct-search method has two major steps, called *search* and *poll*. In the search step, the user can use some insight or information about the function to compute a trial point; for instance, interpolating models may be used to obtain new directions. If the search step is unsuccessful, the poll step evaluates the objective function at some neighbor points defined by a pattern (or random process) and a step size parameter. If no point with lower function value is found, the step size is reduced and a new iteration begins. Figure 3.1 gives a graphical representation of a direct-search method where the coordinate directions define the pattern used in the poll step. Note that the search step is

not encompassed by all direct-search methods and thus consists of an optional step where one attempts to widen the search process.

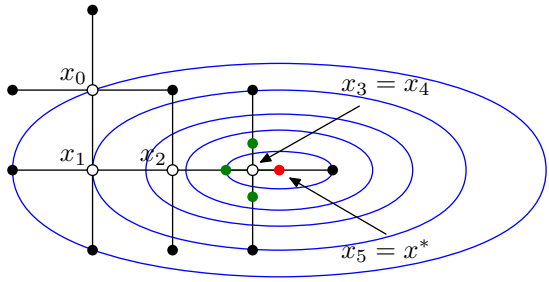


Figure 3.1: Illustration of a direct-search method in which coordinate directions are used. The level sets of the objective function are portrayed by the ellipses. Note that, at x_3, the algorithm fails at finding a descent direction in the poll step, causing reduction of the step size. The points obtained with the new step size are depicted by green dots and a red dot. The iterates are represented by hollow circles, with the exception of the minimizer $x_5 = x^*$, which is red.

The popularity of this class of methods is ascribed to its simplicity and to the fact that it often works reasonably well in practice, besides that no assumption of smoothness of the objective function is demanded, which makes it applicable to a wide range of problems. Nevertheless, a relatively large number of function evaluations is often performed. Besides that, the number of function evaluations needed to perform the method increases rapidly as the number of variables grows.

Although the general term *direct-search method* was conceived in a 1961 paper by Hooke and Jeeves, pioneer research works in this field were led before by Box and Wilson (1951), with the culminating *response surface methodology*, and by Box (1957), with a less sophisticated procedure based upon the latter called *evolutionary operation*. Afterwards, simplex-based direct-search methods were introduced by Spendley, Hext and Himsworth (1962), in which a pattern of $n + 1$ points in $\mathbb{R}^n$ in the form of a simplex is constructed by the algorithm. Among the methods of this class, we may cite the popular *Nelder-Mead* algorithm or *simplex search* algorithm (see Nelder and Mead, 1965) for unconstrained optimization without derivatives. Since it is easy to understand and to code, the Nelder-Mead algorithm has been broadly used in many fields of science and technology, such as chemistry and medicine. At the time of its development, it gained popularity very quickly due to its lack of sophistication and low storage requirements, being ideally suited for use on minicomputers, especially in laboratories. Despite being commonly used, the Nelder-Mead algorithm may not converge in some cases (see Lagarias, Reeds, Wright and Wright, 1998, and Singer and Singer, 2001). Other variants of the original methods by Nelder and Mead (1965) and Spendley et al. (1962) furnished with convergence theory have been proposed later by Yu (1979*a*, 1979*b*), who first introduced

a sufficient decrease condition in direct search and was the first to use the theory of positive bases in the convergence proof of a pattern-search method, and by Price, Coope and Byatt (2002) and Bürmen, Puhan and Tuma (2006). Later, Torczon (1997) introduced generalized pattern-search (GPS) methods for unconstrained optimization where positive-basis techniques are used in the convergence theory. Other examples of this class include the mesh adaptive direct search methods (MADS) proposed by Audet and Dennis (2006) that modify the poll step of GPS algorithms to have a dense set of directions in the space of optimization variables. One of the appealing features of these pattern-search methods in contrast to other derivative-free algorithms (such as the Nelder-Mead algorithm) is the existence of a supporting convergence theory. Other extensions for mixed variable programming (see Audet and Dennis, 2000) and problems involving constraints (Yu and Li, 1981; Lewis and Torczon, 1999, 2000, 2002, 2009; Lucidi et al., 2002; Audet and Dennis, 2004) were also proposed.

The second class of DFO methods concerns the use of finite differences along with quasi-Newton methods (see Chapter 2 of this thesis for a brief overview on the latter). An inherent drawback of this approach is that a single gradient estimation requires at least the number of variables plus one function evaluations, which is expensive in cases where the function evaluations are costly. Another one is when the functions are noisy, in which case gradient estimation by finite differences is often useless.

The third class of methods was introduced by Winfield (1969, 1973) with the minimization of quadratic interpolation models in a neighbourhood of the current iteration where the models are assumed to be valid. Later, Powell (1994, 1998) proposed COBYLA, a method for constrained optimization, which supports arbitrary nonlinear inequality and equality constraints by using linear multivariate interpolation-based models for both of them in a trust-region framework. Afterwards, Powell devised algorithms for both the unconstrained and the bound-constrained cases using Lagrange polynomials, whose implementations are the softwares UOBYQA (2002), NEWUOA (2006, 2008) and BOBYQA (2009).

The introduction of the criticality step and the first interpolation-based derivative-free trust-region method with global convergence to first-order stationary points is due to Conn, Scheinberg and Toint (1997), while the analysis of convergence for second-order stationary points were addressed by Conn, Scheinberg and Vicente (2009*a*). Conn, Scheinberg and Vicente (2008*a*, 2008*b*) analyzed the relation between the geometry of sample sets and the validity of the model for determined interpolation, polynomial regression and underdetermined interpolation cases. They also showed how the bounds on the error between an interpolating polynomial and the true function can be used in the convergence theory of derivative-free sampling methods.

Since the cost to maintain the quality of the geometry of the interpolation set all the time is expensive, Fasano, Nocedal and Morales (2009) suggested to ignore any geometry control and obtained good performance in practice. Later,

Scheinberg and Toint (2010) proved that their method may lose the property of provable global convergence to first-order stationary points and showed that we cannot afford to have it without maintaining the quality of the geometry of the interpolation set. They proposed then a suitable choice of interpolation points yielding a self-correcting geometry scheme, which is the cornerstone of our derivative-free trust-funnel method for constrained nonlinear optimization problems.

Several other trust-region methods were also proposed, e.g. the WEDGE algorithm by Marazzi and Nocedal (2002), the least Frobenius norm updating algorithm by Powell (2004), the DFO algorithm of Conn et al. (1997, 1998), a derivative-free SQP-filter algorithm by Colson (2004) for constrained problems, and the algorithms BC-DFO of Gratton et al. (2011) and BCDFO+ of Tröltzsch (2011) for bound-constrained problems. For a thorough survey on direct search, interpolation models and other derivative-free methods, we refer the reader to the paper of Lewis, Torczon and Trosset (2000) and the textbooks of Conn, Scheinberg and Vicente (2009*b*) and Conn et al. (2000).

Another category of methods that were not discussed above but we would like to mention is that of stocasthic derivative-free methods such as evolutionary algorithms (Bäck and Schwefel, 1993; Beyer and Schwefel, 2002; Holland, 1992), particle swarm optimization (Kennedy and Eberhart, 1995) and simulated annealing (Kirkpatrick, Gelatt and Vecchi, 1983). These methods have shown good results in the search for solutions of difficult optimization problems (e.g. problems where the objective function is non-smooth and multi-modal) that local deterministic DFO methods might experience complications to solve.

The algorithm developed in this work belongs to the third class of methods, i.e. it is based on models built from multivariate polynomial interpolation of the objective and constraint functions. Thus, the remainder of this chapter aims at the development of a DFO trust-region framework where interpolating models are employed that will serve as a basis for our algorithm introduced in Chapter 4.

3.2 Multivariate polynomial interpolation

Before going further into the details of the algorithms, we first introduce some concepts and results from multivariate polynomial interpolation theory that we make use throughout and that can be found to a more extent in Conn et al. (2009*b*). For the sake of simplicity and following the notation in that reference, we will denote the i-th component of a vector x by x_i in this section.

3.2.1 Polynomial bases

Consider $\mathcal{P}_n^d$, the space of polynomials of degree less than or equal to d in $\mathbb{R}^n$ with its dimension denoted by $q_1 = q+1$, and $\phi(x) = \{\phi_0(x), \phi_1(x), \ldots, \phi_q(x)\}$ a basis for $\mathcal{P}_n^d$. Let $m(x)$ be any polynomial of degree less than or equal do d.

Since $\phi \stackrel{\text{def}}{=} \phi(x)$ is a basis of $\mathcal{P}_n^d$ and $m(x) \in \mathcal{P}_n^d$, we can express $m(x)$ as

$$m(x) = \sum_{j=0}^{q} \alpha_j \phi_j(x), \tag{3.2.1}$$

for some coefficients $\alpha_j \in \mathbb{R}$, $j = 0, \ldots, q$.

A simple example of a basis for $\mathcal{P}_n^d$ is the set of all monomials, called the *monomial basis* or the *natural basis*, which can be expressed as

$$\bar{\phi}(x) = \{1, x_1, x_2, \ldots, x_n, x_1^2/2, x_1x_2, \ldots, x_{n-1}^{d-1}x_n/(d-1)!, x_n^d/d!\}.$$

For instance, when $n = 3$ and $d = 2$, $\bar{\phi}$ is given by

$$\bar{\phi}(x) = \{1, x_1, x_2, x_3, x_1^2/2, x_1x_2, x_2^2/2, x_1x_3, x_2x_3, x_3^2/2\}.$$

Since the monomials appear just like the polynomials in the Taylor expansion, it turns out to be easy to work with, thus being commonly used as the starting step for the construction of other bases.

Let $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$ be a sample set and $\phi(x) = \{\phi_0(x), \phi_1(x), \ldots, \phi_q(x)\}$ a basis for $\mathcal{P}_n^d$. Let $p_1 = p + 1$ denote the number of sample points and assume that $p_1 = q_1$ for now. The first aim of an interpolation model-based algorithm is to find a surrogate model m for the function f with the following interpolation conditions being satisfied

$$m(y^i) = \sum_{j=0}^{p} \alpha_j \phi_j(y^i) = f(y^i), \quad \text{for all } \; y^i \in \mathcal{Y}. \tag{3.2.2}$$

The coefficients $\{\alpha_j\}_{j=0}^p$ are determined by solving the interpolation linear system

$$M(\phi, \mathcal{Y})\alpha_\phi = f(\mathcal{Y}), \tag{3.2.3}$$

where

$$M(\phi, \mathcal{Y}) = \begin{pmatrix} \phi_0(y^0) & \phi_1(y^0) & \cdots & \phi_p(y^0) \\ \phi_0(y^1) & \phi_1(y^1) & \cdots & \phi_p(y^1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(y^p) & \phi_1(y^p) & \cdots & \phi_p(y^p) \end{pmatrix}, \quad f(\mathcal{Y}) = \begin{pmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^p) \end{pmatrix}.$$

If we consider the natural basis $\bar{\phi}$ with $d = 2$, thereby having a quadratic model of the form

$$m(x) = c + \langle g, x \rangle + \tfrac{1}{2}\langle x, Hx \rangle,$$

then the cardinality of $\mathcal{Y}$ must be at least

$$p_1 = 1 + n + \tfrac{1}{2}n(n+1) = \tfrac{1}{2}(n+1)(n+2)$$

to ensure that the quadratic interpolation model is entirely determined by the equations (3.2.3). Even so, the last condition is not sufficient to guarantee the existence or uniqueness of an interpolant. For instance, six points on a circle in $\mathbb{R}^2$ do not determine a two-dimensional quadratic, because any quadratic that is a multiple of the equation of the circle can be added to the interpolant without affecting the interpolation conditions. Therefore, some additional *geometric condition* on $\mathcal{Y}$ is required for ensuring the existence and uniqueness of the model. As we will see at the next subsection, it is related to the set $\mathcal{Y}$ and the approximation space.

3.2.2 Poisedness

In this subsection, we give the condition that guarantees the existence and uniqueness of an interpolation model. We start by introducing the concept of poisedness where the sought condition relies.

Definition 3.2.1. *The set $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$ is poised for polynomial interpolation in $\mathbb{R}^n$ if the corresponding matrix $M(\phi, \mathcal{Y})$ is nonsingular for some basis ϕ in $\mathcal{P}_n^d$.*

If we consider the linear interpolation case ($d = 1$) with the natural basis $\bar{\phi}(x) = \{1, x_1, x_2, \ldots, x_n\}$, we obtain

$$M(\phi, \mathcal{Y}) = \begin{pmatrix} 1 & y_1^0 & \cdots & y_n^0 \\ 1 & y_1^1 & \cdots & y_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & y_1^n & \cdots & y_n^n \end{pmatrix}.$$

By applying one step of Gaussian elimination, we arrive at the matrix

$$\begin{pmatrix} 1 & y_1^0 & \cdots & y_n^0 \\ 0 & y_1^1 - y_1^0 & \cdots & y_n^1 - y_n^0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & y_1^n - y_1^0 & \cdots & y_n^n - y_n^0 \end{pmatrix},$$

which can be expressed by blocks as

$$\begin{pmatrix} 1 & (y^0)^T \\ 0 & L \end{pmatrix},$$

where

$$L = \begin{pmatrix} y^1 - y^0 & \cdots & y^n - y^0 \end{pmatrix}^T.$$

Hence the matrix $M(\phi, \mathcal{Y})$ is nonsingular if and only if L is nonsingular, which is equivalent to say that the set is $\{y^0, y^1, \ldots, y^n\}$ is affinely independent.

For the general interpolation case, the matrix $M(\phi, \mathcal{Y})$ is singular if and only if there exists $\gamma \in \mathbb{R}^{p_1}$ such that $\gamma \neq 0$ and a polynomial, of degree at most d, expressed as

$$m(x) = \sum_{j=0}^{p} \gamma_j \phi_j(x),$$

such that $m(y) = 0$ for all $y \in \mathcal{Y}$. In other words, the matrix $M(\phi, \mathcal{Y})$ is singular if and only if the sample points lie on a "polynomial manifold" of degree d or less. Finally, it can be shown that if $M(\phi, \mathcal{Y})$ is nonsingular for some basis ϕ, then it is nonsingular for any basis of $\mathcal{P}_n^d$.

Using the concept of poisedness of sample sets, we have the following result.

Lemma 3.2.2. Given a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and a poised set $\mathcal{Y} \in \mathbb{R}^n$, the interpolating polynomial $m(x)$ exists and is unique.

Now that we have established the condition to ensure existence and uniqueness of interpolation models, we are also interested to know how well poised is a sample set. A first thought could be taking the condition number of the matrix $M(\phi, \mathcal{Y})$ as a measure of poisedness. However, such value depends on the choice of the basis ϕ and can equal any number between 1 and $+\infty$ for a suitable choice. Moreover, for a fixed choice of ϕ, the condition number depends on the scaling of $\mathcal{Y}$ as well. The establishment of such measure must then be reached by other means. In what follows, we show how it can be done properly.

3.2.3 Well poisedness

The notion of well poisedness of a sample set can be described by the use of Lagrange polynomials, as it is shown next. For that reason, we first present the Lagrange form of the interpolating polynomial.

Definition 3.2.3. *Given a set of interpolation points $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$, a basis of $p_1 = p + 1$ polynomials $\ell_j(x)$, $j = 0, \ldots, p$, in $\mathcal{P}_n^d$ is called a basis of Lagrange polynomials if*

$$\ell_j(y^i) = \delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j, \\ 0 & \text{if} \quad i \neq j. \end{cases} \tag{3.2.4}$$

Given a poised set $\mathcal{Y}$, the interpolation polynomial in the Lagrange form $m(x)$ can then be expressed by a linear combination of Lagrange basis polynomials as follows

$$m(x) = \sum_{i=0}^{p} f(y^i) \ell_i(x). \tag{3.2.5}$$

Figure 3.1 shows an example where a function is interpolated by a quadratic model built from Lagrange polynomials using six sample points.

Using the fact that each Lagrange polynomial $\ell_j(x)$ is an interpolating polynomial of a function that vanishes at all points in $\mathcal{Y}$ except at y^j, where it is equal to one, and the Lemma 3.2.2, the following result is immediate.

Lemma 3.2.4. If $\mathcal{Y}$ is poised, then the basis of Lagrange polynomials exists and is uniquely defined.

It is possible to use these polynomials to derive a bound on the distance between f and m at a point x. In Ciarlet and Raviart (1972), it is shown that, for any x in the convex hull of $\mathcal{Y}$, one has

$$\|\mathcal{D}^r f(x) - \mathcal{D}^r m(x)\| \leq \frac{1}{(d+1)!}\nu_d \sum_{i=0}^{p} \|y^i - x\|^{d+1}\|\mathcal{D}^r \ell_i(x)\|,$$

where $\mathcal{D}^r$ denotes the r-th derivative of a function and ν_d is an upper bound on $\mathcal{D}^{d+1} f(x)$, which means that this error bound requires $f(x)$ to have a bounded $(d+1)$st derivative. For $r = 0$, we have the reduced case

$$|f(x) - m(x)| \leq \frac{1}{(d+1)!} p_1 \nu_d \Lambda_\ell \Delta^{d+1}, \tag{3.2.6}$$

where

$$\Lambda_\ell = \max_{0 \leq i \leq p} \max_{x \in B(\mathcal{Y})} |\ell_i(x)|,$$

and Δ is the diameter of the smallest ball $B(\mathcal{Y})$ containing $\mathcal{Y}$. In Sauer and Yuan (1995), an equivalent result to (3.2.6) is given using the Newton fundamental polynomials as basis instead of the Lagrange polynomials.

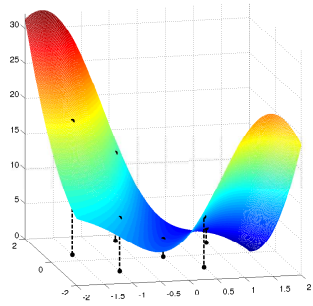
Since the values that the Lagrange polynomials can assume within the region $B(\mathcal{Y})$ have major impact on the error bound, a classical measure of poisedness of $\mathcal{Y}$ in $\mathcal{B}$ is the upper bound on the absolute values of the Lagrange polynomials in $B(\mathcal{Y})$, as we will describe formally in the next definition.

Different yet equivalent approaches for measuring poisedness with Lagrange polynomials exist. Assuming that the matrix $M(\phi, \mathcal{Y})$ is nonsingular, we can also express the vector $\phi(x)$ uniquely in terms of the vectors $\phi(y^i)$, $i = 0, \ldots, p$, as

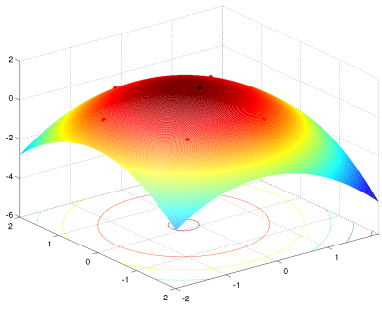
$$\sum_{i=0}^{p} \lambda_i(x)\phi(y^i) = \phi(x) \tag{3.2.7}$$

or, equivalently,

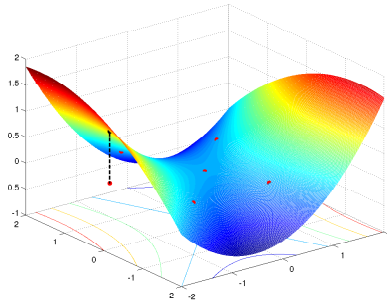
$$M(\phi, y)^T \lambda(x) = \phi(x), \quad \text{where } \lambda(x) = (\lambda_0(x), \ldots, \lambda_p(x))^T. \tag{3.2.8}$$



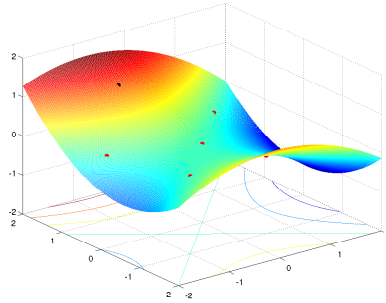
(a) The original function and the interpolation set



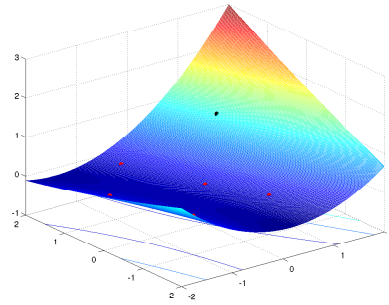
(b) The first Lagrange polynomial



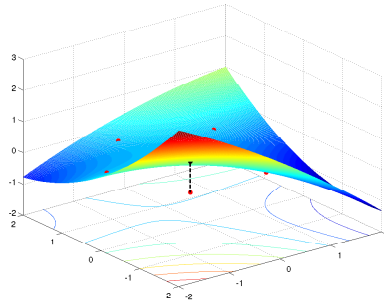
(c) The second Lagrange polynomial



(d) The third Lagrange polynomial

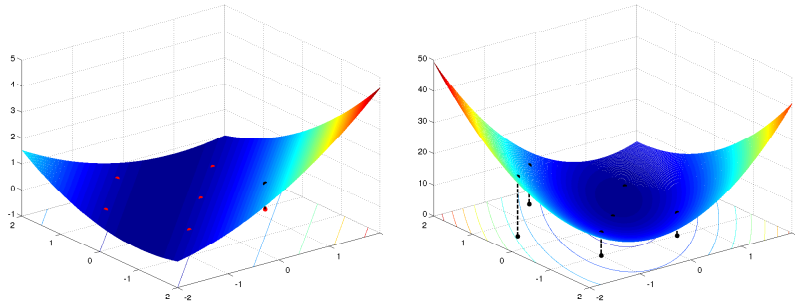


(e) The fourth Lagrange polynomial



(f) The fifth Lagrange polynomial

From (3.2.7) and (3.2.8), it is clear that $\lambda(x)$ is not only a vector of polynomials in $\mathcal{P}_n^d$, but it is also the vector of Lagrange polynomials for $\mathcal{Y}$. In our measure of poisedness, we would also like to know how well the vectors $\phi(y^i)$, $i = 0, \ldots, p$, span all the set $\phi(x)$ in a region of interest $\mathcal{B}$. Therefore, such measure should



(g) The sixth Lagrange polynomial (h) The final interpolating quadratic

Figure 3.1: Construction of a quadratic interpolation model using Lagrange polynomials.

also take into account the region of interest besides the polynomial space. It means that, if the smallest ball that contains the set $\mathcal{Y}$ is substantially smaller than the region $\mathcal{B}$, then $\mathcal{Y}$ is not supposed to be well poised in $\mathcal{B}$. Besides, the coefficients of the linear combination of the vectors $\phi(y^i)$ in (3.2.7), i.e. the values of the Lagrange polynomials at $\mathcal{B}$, should not be very large when $\mathcal{Y}$ is well poised for $\mathcal{B}$.

For instance, consider the linear interpolation case for the sample set $\mathcal{Y} = \{(0,0),(0,1),(1,0)\}$ in the region defined by $\mathcal{B}(0;1)$, the ball centred at the origin with radius one, and the natural basis $\bar{\phi}$. We know that the vector of coefficients of each Lagrange polynomial λ_i must satisfy the linear system

$$M(\bar{\phi}, \mathcal{Y})\lambda_i = e_i, \tag{3.2.9}$$

where e_i denotes the i-th unit vector ($i = 0, 1, 2$), whose i-th component is one and all others are zero. For $\lambda_0 = (\lambda_0^0, \lambda_0^1, \lambda_0^2)^T$, the system (3.2.9) then becomes

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_0^2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix},$$

whose solution is the vector $\lambda_0 = (1, -1, -1)^T$. By using the natural basis $\bar{\phi}(x) = \{1, x_1, x_2\}$ to express the Lagrange polynomial $\lambda_0(x)$, we obtain

$$\lambda_0(x) = \sum_{j=0}^{2} \lambda_0^j \, \bar{\phi}_j(x) = 1 - x_1 - x_2. \tag{3.2.10}$$

Since $\mathcal{B}(0;1)$ denotes the closed Euclidian ball centred at $(0,0)$ with radius one, we have from (3.2.10) that $|\lambda_0(x)| < 3$ for any point $x \in \mathcal{B}(0;1)$. As for the

vector λ_1, its components are given by $\lambda_1 = (0, 0, 1)^T$, thereby having

$$\lambda_1(x) = \sum_{j=0}^{2} \lambda_1^j \, \bar{\phi}_j(x) = x_2. \tag{3.2.11}$$

In this case, for any point $x \in \mathcal{B}(0; 1)$, we have that $|\lambda_1(x)| \leq 1$. Finally, the solution λ_2 for (3.2.9) is given by the vector $\lambda_2 = (0, 1, 0)^T$, from which we obtain

$$\lambda_2(x) = \sum_{j=0}^{2} \lambda_2^j \, \bar{\phi}_j(x) = x_1. \tag{3.2.12}$$

This then implies that $|\lambda_2(x)| \leq 1$ for any point $x \in \mathcal{B}(0; 1)$. Thus, from (3.2.10)-(3.2.12), we can derive the bound

$$\|\lambda(x)\|_\infty < 3, \;\; \text{for all} \;\; x \in \mathcal{B}(0; 1). \tag{3.2.13}$$

On the other hand, by choosing another region $\mathcal{B}(0; 10^3)$ and the point $\bar{x} = (0, -10^3)^T \in \mathcal{B}(0; 10^3)$, but keeping the same set $\mathcal{Y}$, we have

$$\|\lambda(\bar{x})\|_\infty = |\lambda_0(\bar{x})| = 1001. \tag{3.2.14}$$

The bounds (3.2.13) and (3.2.14) exemplify the discussion above and indicate that $\phi(\mathcal{Y})$ "spans well" the set $\phi(x)$ in $\mathcal{B}(0; 1)$, with moderate coefficient values for (3.2.7), but not in the region defined by $\mathcal{B}(0; 10^3)$. In other words, the sample set $\mathcal{Y} = \{(0, 0), (0, 1), (1, 0)\}$ is well poised in $\mathcal{B}(0; 1)$, but not in $\mathcal{B}(0; 10^3)$.

Finally, we present a geometric approach for the measure of poisedness of a sample set. Consider the set $\mathcal{Y}_i(x) = \mathcal{Y} \setminus \{y^i\} \cup \{x\}$, $i = 0, \ldots, p$, for a given point x. By applying Cramer's rule to the system (3.2.8), we obtain

$$\lambda_i(x) = \frac{\det(M(\phi, \mathcal{Y}_i(x)))}{\det(M(\phi, \mathcal{Y}))}. \tag{3.2.15}$$

From this expression, it is clear that $\lambda(x)$ is a polynomial in $\mathcal{P}_n^d$. In addition, we have that

$$\lambda_i(y^i) = \frac{\det(M(\phi, \mathcal{Y}_i(y^i)))}{\det(M(\phi, \mathcal{Y}))} = \frac{\det(M(\phi, \mathcal{Y}))}{\det(M(\phi, \mathcal{Y}))} = 1 \tag{3.2.16}$$

and

$$\lambda_i(y^j) = \frac{\det(M(\phi, \mathcal{Y}_i(y^j)))}{\det(M(\phi, \mathcal{Y}))} = \frac{0}{\det(M(\phi, \mathcal{Y}))} = 0, \tag{3.2.17}$$

for all $j \neq i$. From (3.2.16) and (3.2.16), we conclude that $\lambda(x)$ is exactly the set of Lagrange polynomials. It also follows that $\lambda(x)$ does not depend on the choice of ϕ as long as the polynomial space $\mathcal{P}_n^d$ is fixed.

The expression (3.2.15) can be interpreted as follows. Consider a set $\phi(\mathcal{Y}) = \{\phi(y^0), \ldots, \phi(y^p)\}$ in $\mathbb{R}^{p_1}$. Let $\text{vol}(\phi(\mathcal{Y}))$ be the volume of the simplex of vertices in $\phi(\mathcal{Y})$, given by

$$\text{vol}(\phi(\mathcal{Y})) = \frac{|\det(M(\phi, \mathcal{Y}))|}{p_1!}.$$

Then

$$|\lambda_i(x)| = \frac{\text{vol}(\phi(\mathcal{Y}_i(x)))}{\text{vol}(\phi(\mathcal{Y}))}. \tag{3.2.18}$$

This means that the absolute value of the i-th Lagrange polynomial at a given point x is the change in the volume of (the p_1-dimensional convex hull of) $\phi(\mathcal{Y})$ when y^i is replaced by x.

We now are able to formally define well poisedness in terms of the Lagrange polynomials.

Definition 3.2.5. *Let $\Lambda > 0$ and a set $\mathcal{B} \in \mathbb{R}^n$ be given. Let ϕ be a basis in $\mathcal{P}_n^d$ with $\phi = \{\phi_0(x), \phi_1(x), \ldots, \phi_p(x)\}$. A poised set $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$ is said to be Λ-poised in $\mathcal{B}$ (in the interpolation sense) if and only if*

1. *for the basis of Lagrange polynomials associated with $\mathcal{Y}$*

$$\Lambda \geq \max_{0 \leq i \leq p} \max_{x \in \mathcal{B}} |\ell_i(x)|, \tag{3.2.19}$$

   or, equivalently,

2. *for any $x \in \mathcal{B}$ there exists $\lambda(x) \in \mathbb{R}^{p_1}$ such that*

$$\sum_{i=0}^{p} \lambda_i(x)\phi(y^i) = \phi(x) \quad \textit{with} \quad \|\lambda(x)\|_\infty \leq \Lambda, \tag{3.2.20}$$

   or, equivalently,

3. *replacing any point in $\mathcal{Y}$ by any $x \in \mathcal{B}$ can increase the volume of the set $\{\phi(y^0), \ldots, \phi(y^p)\}$ at most by a factor Λ.*

Now that we have established a measure of poisedness using Lagrange polynomials, we must consider the algorithmic aspects that its use entails. Unless some procedure to estimate lower and upper bounds on the Lagrange polynomials in the region of interest is used, the calculations involved imply the global optimization of the Lagrange polynomials at each step and thus may dominate the overall computational effort. Therefore, using the Λ-poisedness measure at each iteration for controlling the quality of the interpolation models might turn out to be unaffordable. The question that naturally arises is how to control

poisedness iteratively without costly additional steps and in a way related to Λ-poisedness.

As we already know, the condition number of the interpolation matrix $M(\phi, \mathcal{Y})$ is not ideal for measuring poisedness of a given sample set $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$. $M(\phi, \mathcal{Y})$ depends on the basis ϕ and on the scaling of $\mathcal{Y}$, but not on the region of interest, whereas Λ-poisedness depends on the latter, but is independent of scaling. Although they seem quite different, it is possible to link them by fixing a basis $\phi = \bar{\phi}$ and shifting and scaling $\mathcal{Y}$. For this purpose, we first shift $\mathcal{Y}$ by $-y^0$ to center the new set at the origin:

$$\{0, y^1 - y^0, \ldots, y^p - y^0\}.$$

Then we scale this new set by

$$\Delta = \Delta(\mathcal{Y}) = \max_{1 \leq 1 \leq p} \|y^i - y^0\|,$$

from which we obtain

$$\hat{\mathcal{Y}} = \{0, \hat{y}^1, \ldots, \hat{y}^p\} = \{0, (y^1 - y^0)/\Delta, \ldots, (y^p - y^0)/\Delta\} \subset \mathcal{B}(0; 1).$$

Now let $\hat{M} \stackrel{\text{def}}{=} M(\bar{\phi}, \hat{\mathcal{Y}})$, where $\bar{\phi}$ is the natural basis, and denote the condition number of $\hat{M}$ by $\kappa(\hat{M}) = \|\hat{M}\| \, \|\hat{M}^{-1}\|$. To bound $\kappa(\hat{M})$ in terms of Λ, it is sufficient to bound $\|\hat{M}^{-1}\|$. Likewise, to bound Λ in terms of $\kappa(\hat{M})$, it is sufficient to bound it in terms of $\|\hat{M}^{-1}\|$. We then have the following unifying result whose proof can be found in the Section 3.4 of Conn et al. (2009*b*).

Theorem 3.2.6. If $\hat{M}$ is nonsingular and $\|\hat{M}^{-1}\| \leq \Lambda$, then the set $\hat{\mathcal{Y}}$ is $\sqrt{p_1}\Lambda$-poised in the unit ball $\mathcal{B}(0; 1)$. Conversely, if the set $\hat{\mathcal{Y}}$ is Λ-poised in the unit ball $\mathcal{B}(0; 1)$, then

$$\|\hat{M}^{-1}\| \leq \theta p_1{}^{\frac{1}{2}} \Lambda, \tag{3.2.21}$$

where $\theta > 0$ is dependent on n and d, but independent of $\hat{\mathcal{Y}}$ and Λ.

A reasonably cheap algorithmic strategy may be using the condition number of $\hat{M}$ for monitoring the error between the real function f and the current model m_k, while considering Λ-poisedness for operations on the sample set, such as improving well poisedness of the sample set.

For improving well poisedness via Lagrange polynomials, we assume that the given sample set $\mathcal{Y}$ is poised. We then want to verify whether $\mathcal{Y}$ is Λ-poised or not, for some $\Lambda > 1$. The first step is to compute the maximum absolute value of the Lagrange polynomials on a region $\mathcal{B}$. If such value is below Λ, then $\mathcal{Y}$ is Λ-poised. If there is an index $j \in \{0, \ldots, p\}$ such that

$$\max_{0 \leq i \leq p} \max_{x \in \mathcal{B}} |\ell_i(x)| = |\ell_j(\bar{x})| > \Lambda,$$

where

$$\bar{x} = \arg\max_{x \in \mathcal{B}} |\ell_j(x)|,$$

then the corresponding interpolation point y^j is replaced by $\bar{x}$. The complete algorithm is given in detail below.

Algorithm 3.2.1: Improving well poisedness via Lagrange polynomials.

Step 0: Initialization. An initial constant $\Lambda > 1$ and poised set $\mathcal{Y} = \mathcal{Y}_0$, with $|\mathcal{Y}| = p_1$. Compute the associated Lagrange polynomials $\ell_i(x)$, $i = 0, \ldots, p$, and set $k = 1$.

Step 1: Poisedness measurement. Estimate

$$\Lambda_{k-1} = \max_{0 \le i \le p} \max_{x \in \mathcal{B}} |\ell_i(x)|.$$

Step 2: Point exchange. If $\Lambda_{k-1} > \Lambda$, then let $i_k \in \{0, \ldots, p\}$ be an index for which

$$\max_{x \in \mathcal{B}} |\ell_{i_k}(x)| > \Lambda,$$

and let $y_*^{i_k} \in \mathcal{B}$ be a point that (approximately) maximizes $|\ell_{i_k}(x)|$ in $\mathcal{B}$. Update $\mathcal{Y}_k$ by

$$\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{i_k}\} \cup \{y_*^{i_k}\}.$$

Otherwise ($\Lambda_{k-1} \le \Lambda$), return the Λ-poised set $\mathcal{Y}_k$.

Step 3: Update the Lagrange polynomials. Update all Lagrange polynomial coefficients. Set $k = k + 1$ and go to Step 1.

The next theorem from Conn et al. (2009*b*) states the correctness of the Algorithm 3.2.1.

Theorem 3.2.7. For any given $\Lambda > 1$, a closed ball $\mathcal{B}$, and a fixed polynomial basis ϕ, Algorithm 3.2.1 terminates with a Λ-poised set $\mathcal{Y}$ after at most $N = N(\Lambda, \phi)$ iterations, where N is a constant which depends on Λ and ϕ.

3.2.4 Underdetermined interpolation models

The algorithm developed in this work employs the commonly used idea of starting with incomplete interpolation models with linear accuracy and then enhancing them with curvature information, thereby having an actual accuracy at least as good as that for linear models and, hopefully, better. We thus consider underdetermined quadratic interpolation, i.e.

$$n+1 \leq |\mathcal{Y}| \leq (n+1)(n+2)/2,$$

with initial sample sets that are poised for linear interpolation.

We now have the following linear system to solve in order to build the interpolation model:

$$M(\phi, \mathcal{Y})\alpha_\phi = f(\mathcal{Y}), \tag{3.2.22}$$

where

$$M(\phi, \mathcal{Y}) = \begin{pmatrix} \phi_0(y^0) & \phi_1(y^0) & \cdots & \phi_q(y^0) \\ \phi_0(y^1) & \phi_1(y^1) & \cdots & \phi_q(y^1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(y^p) & \phi_1(y^p) & \cdots & \phi_q(y^p) \end{pmatrix}, \quad f(\mathcal{Y}) = \begin{pmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^p) \end{pmatrix}, \quad p \leq q.$$

Since the linear system (3.2.22) is potentially underdetermined, the resulting interpolating polynomials will be no longer unique, which means that there are different ways to construct the model $m(x)$.

One simple approach, called *subbasis selection*, consists in considering only $p+1$ columns of the matrix $M(\phi, \mathcal{Y})$ in the linear system (3.2.22), having then a square matrix. Basically, this means that one is choosing a *subbasis* $\tilde{\phi}$ of ϕ with only $p+1$ elements. By removing the other $q-p$ columns, it will cause $q-p$ elements of α_ϕ to be zero. An inherent shortcoming of this approach is that the selected $p+1$ columns might be linearly dependent, which could be taken as a lack of poisedness of the sample set and thus cause the exchange of some of the points in the set. This may happen even if a different set of columns could have provided well poisedness without exchanging any point. A good advantage of this approach is that information about f, e.g. sparsity structure of the derivatives, can be taken into account in the choice of a set of columns over another.

Another way to build the model is to take the minimum ℓ_2-norm solution of (3.2.22). In Conn et al. (2009*b*), the authors show that this approach is more robust with respect to small perturbations of the data than choosing a subbasis.

A third possibility stems from the fact that we want to build models for which the norm of the Hessian is moderate, as it plays a relevant role on the error bounds for quadratic interpolation models (see Theorem 5.4 in Conn et al., 2009*b*). For that purpose, the interpolation model can be written as

$$m(x) = \alpha_L^T \phi_L(x) + \alpha_Q^T \phi_Q(x),$$

where α_L and ϕ_L are related to the linear components of the natural basis ϕ, while α_Q and ϕ_Q, to the quadratic ones. The minimum Frobenius norm solution α^{mfn} is then the solution of the following optimization problem in α_L and α_Q:

$$\begin{array}{ll} \min & \frac{1}{2}\|\alpha_Q\|^2 \\ \text{s.t.:} & M(\phi_L, \mathcal{Y})\alpha_L + M(\phi_Q, \mathcal{Y})\alpha_Q = f(\mathcal{Y}). \end{array}$$

Due to the choice of the basis ϕ and the separation $\alpha = (\alpha_L, \alpha_Q)$, minimizing the norm of α_Q is equivalent to minimizing the Frobenius norm of the Hessian of $m(x)$, thereby originating the name of the model.

Another two ways of working with interpolation models are noteworthy. The first one is the least Frobenius norm updating approach proposed by Powell (2004), which aims at providing a model Hessian closest, in the Frobenius norm sense, to that one obtained through the last calculation of the model. The idea is similar to that used in quasi-Newton methods and showed good results with $p = 2n$. The second manner is by minimizing the ℓ_1-norm of the entries of the Hessian model instead of the ℓ_2-norm in the problem above. This might be interesting when there is a sparse structure in the Hessian matrix and that structure is not known in advance, for fully quadratic models can be recovered with high probabilty using much less than $(n+2)(n+1)/2$ random points in such cases, as it has been proved in Bandeira, Scheinberg and Vicente (2012). Since considering all the possibilities of model building is beyond our scope in this work, the last two approaches are not included in our experiments.

We also require the following assumption, which is readily achieved, for instance, by applying the procedures described in Conn et al. (2009*b*) once the interpolant model is built with the Lagrange polynomials.

Assumption 3.2.8. *Assume we are given any set $\mathcal{Y} \subset \mathcal{B}(z;\Delta)$ with $n+1 \leq |\mathcal{Y}| \leq (n+1)(n+2)/2$ and $z \in \mathbb{R}^n$. Then we can apply a finite number of substitutions of the points in $\mathcal{Y}$, in fact, at most $|\mathcal{Y}| - 1$, such that the new resultant set is Λ-poised in $B(z;\Delta)$ for a polynomial space $\mathcal{P}$, with dimension $|\mathcal{Y}|$ and $\mathcal{P}_n^1 \subseteq \mathcal{P} \subseteq \mathcal{P}_n^2$.*

The next lemma states the error bounds for at most fully quadratic models. As one might expect, the accuracy inherent in undetermined quadratic interpolation models is similar to the linear interpolation ones, where the error bounds are linear in Δ for the first derivatives and quadratic for the function values. The proof of the lemma can be found in Conn, Scheinberg and Zang (2010).

Lemma 3.2.9. Given any $\Delta > 0$ and $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\} \subset \mathcal{B}(y^0; \Delta)$ Λ-poised in $\mathcal{B}(y^0; \Delta)$ with $n + 1 \leq |\mathcal{Y}| \leq (n + 1)(n + 2)/2$, let $m(x) \in \mathcal{P}_n^2$ be an interpolating polynomial of f on $\mathcal{Y}$, i.e.

$$m(y^i) = f(y^i), \quad i = 0, \ldots, p.$$

If $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable in an open domain Ω containing $\mathcal{B}(y^0; \Delta)$ and ∇f is Lipschitz continuous in Ω with constant L, then for any point $y \in \mathcal{B}(y^0; \Delta)$ we have

$$\begin{aligned} \|\nabla f(y) - \nabla m(y)\| &\leq \kappa_{eg}(n, \Lambda, L)(\|\nabla^2 m\| + 1)\Delta, \\ |f(y) - m(y)| &\leq \kappa_{ef}(n, \Lambda, L)(\|\nabla^2 m\| + 1)\Delta^2, \end{aligned}$$

where κ_{eg} and κ_{ef} are positive constants depending only on n, Λ and L.

3.2.5 Regression nonlinear models

In cases where the functions are noisy but relatively cheap, one can sample more local points than it would be necessary for complete interpolation. In such cases, we have $p_1 > q_1$, and the model can be built using regression. If the least-squares approach is used, the interpolation conditions become then

$$M(\phi, \mathcal{Y})\alpha \overset{\text{l.s.}}{=} f(\mathcal{Y}) \tag{3.2.23}$$

or, equivalently,

$$\min_{\alpha} \|M(\phi, \mathcal{Y})\alpha - f(\mathcal{Y})\|. \tag{3.2.24}$$

The above solution has unique solution if the matrix $M(\phi, \mathcal{Y})$ has full column rank, in which case the set $\mathcal{Y}$ is said to be poised for polynomial least-squares regression in $\mathbb{R}^n$.

Another approach, known as Least Absolute Deviations (LAD) regression or *robust regression*, makes use of the ℓ_1 norm instead of the ℓ_2 norm in (3.2.24). Differently from the ordinary least-squares regression where the error can be largely increased by the squaring operation if it is bigger than 1, LAD is more robust in that it is more resistant to outliers in the data as the errors only have their absolute values taken rather than augmented. Although it is more robust, LAD is known to be less stable than the least-squares regression as LAD is more sensitive to small perturbations of the data points.

The unique model $m(x)$ built from least-squares regression to approximate a function $f(x)$ on a poised set $\mathcal{Y}$ can be expressed under the same form as (3.2.5). Error bounds for quadratic regression models ($d = 2$) are stated in the next theorem by Conn et al. (2008*b*).

Lemma 3.2.10. Given any $\Delta > 0$ and $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\} \subset \mathcal{B}(y^0; \Delta)$ Λ-poised (in the regression sense) in $\mathcal{B}(y^0; \Delta)$ with $p_1 > (n+1)(n+2)/2$, let $m(x) \in \mathcal{P}_n^2$ be a least-squares regression polynomial built from (3.2.24). If $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable in an open domain Ω containing $\mathcal{B}(y^0; \Delta)$ and ∇f is Lipschitz continuous in Ω with constant L, then for any point $y \in \mathcal{B}(y^0; \Delta)$ we have

$$\begin{aligned} \|\nabla f(y) - \nabla m(y)\| &\leq \kappa_{eg}(p_1, q_1, \Lambda, L)\Delta^2, \\ |f(y) - m(y)| &\leq \kappa_{ef}(p_1, q_1, \Lambda, L)\Delta^3, \end{aligned}$$

where κ_{eg} and κ_{ef} are positive constants depending only on p_1, q_1, Λ and L.

3.3 Methods based on interpolation models

3.3.1 A simple DFO algorithm

Two relevant algorithmic aspects, having both practical and theoretical implications, must be taken into consideration when building a model-based trust-region algorithm, namely: a geometry phase whose goal is to ensure the adequacy of the current sample set $\mathcal{Y}_k$ and the trust region management. As it has been proven in Scheinberg and Toint (2010), the former is deemed of fundamental importance to ensure global convergence. The variations suggested to the trust region management are usually centred on unsuccessful iterations, which may be due to a large trust region or lack of poisedness.

In the algorithm developed by Conn et al. (1997), model improvement steps are considered at unsuccessful iterations whenever the sample set is not poised. Such steps aim to improve the poisedness of the interpolation set by calculating new points that might enhance the quality of geometry. As we discussed in the last section, these steps often involve the global optimization of the Lagrange polynomials and thus may be quite expensive. In their algorithm, the trust region is shrunk at unsuccessful iterations only if the model is "adequate" therein, or, in other words, the current sample set is sufficiently well poised in that region. In order to diminish the cost of possibly unnecessary geometry improvement steps, Fasano et al. (2009) decided to avoid any geometry control mechanism and obtained good results in practice. Since their algorithm is a "simple" DFO trust-region method, we present it here as the basis of our DFO trust-region framework to be further explored.

Algorithm 3.3.1: Simple DFO trust-region algorithm

Step 0: Initialization. An initial trust-region radius Δ_0 and an initial poised interpolation set $\mathcal{Y}_0$ containing the starting point x_0 are given. This interpolation set defines an (at most quadratic) interpolation model m_0 around x_0. Constants $\eta \in (0,1)$ and $0 < \gamma_1 \leq \gamma_2 < 1$ are also given. Set $k = 0$.

Step 1: Compute a trial point. Compute x_k^+ that "sufficiently reduces" $m_k(x)$ in $\mathcal{B}(x_k; \Delta_k)$.

Step 2: Evaluate the objective function at the trial point. Compute $f(x_k^+)$ and ρ_k from (2.3.7).

Step 3: Define the next iterate. Let $y_{k,\max} = \arg\max_{y \in \mathcal{Y}_k} \|y - x_k\|$.

Step 3.1: Successful iteration. If $\rho_k \geq \eta$, define $x_{k+1} = x_k^+$ and choose $\Delta_{k+1} \geq \Delta_k$. Set $\mathcal{Y}_{k+1} = \mathcal{Y} \setminus \{y_{k,\max}\} \cup \{x_k^+\}$.

Step 3.2: Unsuccessful iteration. If $\rho_k < \eta$, define $x_{k+1} = x_k$ and choose $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$. Set

$$\mathcal{Y}_{k+1} = \begin{cases} \mathcal{Y}_k & \text{if } \|y_{k,\max} - x_k\| \leq \|x_k^+ - x_k\|, \\ \mathcal{Y}_k \setminus \{y_{k,\max}\} \cup \{x_k^+\} & \text{otherwise.} \end{cases}$$

Step 4: Update the model and Lagrange polynomials. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation model m_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$. Increment k by one and go to Step 1.

By computing a point x_k^+ that sufficiently reduces $m_k(x)$ in $\mathcal{B}(x_k, \Delta_k)$, we mean that it must satisfy the well-known Cauchy condition for that problem

$$m_k(x_k) - m_k(x_k^+) \geq \kappa_{\mathrm{C}} \|g_k\| \min\left[\frac{\|g_k\|}{1 + \|H_k\|}, \Delta_k\right], \tag{3.3.1}$$

where we define $g_k \overset{\text{def}}{=} \nabla m_k(x_k)$ and $H_k \overset{\text{def}}{=} \nabla^2 m_k(x_k)$, and where κ_{C} is some constant in $(0,1)$.

3.3.2 Self-correcting geometry scheme

In Scheinberg and Toint (2010), the authors probed into the question of whether it is possible to ignore geometry considerations and still have a globally convergent algorithm. Rather than suggesting a new model improvement scheme, their main objective was to analyze the role played by the geometry in model-based DFO methods. They gave two counter-examples where the algorithm of Fasano et al. (2009) could converge to a non-stationary point and

showed that it is impossible to have global convergence without resorting to the upkeep of the geometry of the sample set. On the other hand, they also proposed a solution to the unresolved dilemma between avoiding costly geometry maintenance steps and having global convergence, which is based on a self-correction mechanism for the geometry. The solution relies mainly on the idea that unsuccessful iterations might not be as disheartening as they seem and that one can still use the trial point x_k^+ to maintain the quality of geometry by taking special cares for the final criticality step and monitoring the geometry with the use of the Lagrange polynomials.

A slight modification on the scheme was suggested later by Tröltzsch (2011) in regard to the trust region management. In the original approach, the trust region radius is only reduced at unsuccessful iterations where it is impossible to improve poisedness by replacing a suitable choice of interpolation points. Tröltzsch then proposed to shrink the trust region at unsuccessful iterations whenever the trust region radius is greater than a predefined constant value, regardless of the update of the sample set. The numerical experiments conducted with the modified self-correction mechanism for the unconstrained and bound-constrained cases presented a better performance than those with the original scheme. Therefore, we decided to incorporate such modification in our final algorithm, hoping that similar improvements for the constrained case could be attained.

We introduce now the modified scheme in detail.

Algorithm 3.3.2: A DFO trust-region algorithm with self-correcting geometry scheme

---

Step 0: Initialization. An initial trust-region radius Δ_0, an initial accuracy threshold ϵ_0 and an initial poised interpolation set $\mathcal{Y}_0$ containing the starting point x_0 are given, as well as the maximum number of interpolation points $p_{\max} \geq |\mathcal{Y}_0|$ in $\mathcal{Y}_k$ at the end. Let p_k denote the cardinality of $\mathcal{Y}_k$. This interpolation set defines an interpolation model m_0 around x_0 and associated Lagrange polynomials $\{\ell_{0,j}\}_{j=0}^{p}$. Constants $\eta \in (0,1)$, $0 < \gamma_1 \leq \gamma_2 < 1$, $\alpha \in (0,1)$, $\beta > 0$, $\zeta \geq 1$ and $\Lambda > 1$ are also given. Choose $v_0 \neq x_0$ and set $k = 0$ and $i = 0$.

Step 1: Criticality step.

Step 1.1: Define $\hat{m}_i = m_k$.

Step 1.2: If $\|\nabla \hat{m}_i(x_k)\| < \epsilon_i$, set $\epsilon_{i+1} = \alpha \|\nabla \hat{m}_i(x_k)\|$ and modify $\mathcal{Y}_k$ as needed to make sure it is Λ-poised in $\mathcal{B}(x_k; \epsilon_{i+1})$, increment i by one and start Step 1.2 again.

Step 1.3: Set $m_k = \hat{m}_i$, $\Delta_k = \beta \|\nabla m_k(x_k)\|$ and define $v_i = x_k$ if a new model has been computed.

Step 2: Compute a trial point. Compute $x_k^+ \in \mathcal{B}(x_k; \Delta_k)$ such that (3.3.1) holds.

Step 3: Evaluate the objective function at the trial point. Compute $f(x_k^+)$ and ρ_k from (2.3.7).

Step 4: Define next iterate.

Step 4.1: Augmenting interpolation set. If $p_k < p_{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$. If $\rho_k \geq \eta$, set $x_{k+1} = x_k^+$ and choose $\Delta_{k+1} \geq \Delta_k$. If $\rho_k < \eta$, set $x_{k+1} = x_k$ and $\Delta_{k+1} = \Delta_k$.

Step 4.1: Successful iteration. If $p_k = p_{\max}$ and $\rho_k \geq \eta$, set $x_{k+1} = x_k^+$, choose $\Delta_{k+1} \geq \Delta_k$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.3.2}$$

Step 4.2: Replace a far interpolation point. If $p_k = p_{\max}$, $\rho_k < \eta$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \,|\, \|y^{k,j} - x_k\| > \zeta\Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\}$$

is non-empty, then set $x_{k+1} = x_k$, $\Delta_{k+1} = \Delta_k$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ where r is an index for any point in $\mathcal{F}_k$, for instance, such that

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.3.3}$$

Step 4.3: Replace a close interpolation point. If $p_k = p_{\max}$, $\rho_k < \eta$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \,|\, \|y^{k,j} - x_k\| \leq \zeta\Delta \text{ and } |\ell_{k,j}(x_k^+)| > \lambda\}$$

is non-empty, then set $x_{k+1} = x_k$, $\Delta_{k+1} = \Delta_k$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ where r is an index for any point in $\mathcal{C}_k$, for instance, such that

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.3.4}$$

Step 4.4: Reduce the trust-region radius. If $p_k = p_{\max}$, $\rho_k < \eta$ and either $x_k = v_i$ and $\Delta_k > \epsilon_i$ or $\mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then set $x_{k+1} = x_k$, $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

Step 5: Update the model and Lagrange polynomials. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation model m_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$ and the associated Lagrange polynomials $\{l_{k+1,j}\}_{j=0}^p$. Increment k by one and go to Step 1.

In order to present the fundamental lemma that states the self-correction property of the original algorithm, we make use of the following assumptions.

Assumption 3.3.1. *The objective function f is continuoulsy differentiable in an open set $\mathcal{V}$ containing all iterates generated by the algorithm, and its gradient ∇f is Lipschitz continuous in V with constant $\frac{1}{2}L$.*

Assumption 3.3.2. *There exists a constant $\kappa_{\mathrm{H}} \geq L$ such that $1 + \|H_k\| \leq \kappa_{\mathrm{H}}$ for every $k \geq 0$.*

We now give the aforementioned result whose proof can be found in the original paper by Scheinberg and Toint (2010).

Lemma 3.3.3. Suppose that the Assumptions 3.3.1 and 3.3.2 hold, that $p_k = p = p_{\max}$, for all k, and that m_k is of degree one or higher. Then, for any constant $\Lambda > 1$, if iteration k is unsuccessful,

$$\mathcal{F}_k = \emptyset$$

and

$$\Delta_k \leq \min\left[\frac{1}{\kappa_{\mathrm{H}}}, \frac{(1-\eta)\kappa_{\mathrm{C}}}{2\kappa_{\mathrm{ef}}(\zeta+1)^2(p\Lambda+1)}\right]\|g_k\| \stackrel{\mathrm{def}}{=} \kappa_\Lambda \|g_k\|,$$

then

$$\mathcal{C}_k \neq \emptyset.$$

Based on the above result, the geometry is said to be self-correcting due to the fact that, provided the trust-region radius is sufficiently small when compared to the model's gradient and all the significant interpolation points are contained in the trust region, every unsuccessful iteration must result in an improvement of the interpolation set geometry.

Chapter 4

A derivative-free trust-funnel method

The algorithm developed in this work, named *DEFT-FUNNEL* (DErivative-Free Trust FUNNEL), is of the trust-funnel variety and employs surrogate models built from multivariate polynomial interpolation of the objective and constraint functions. In addition, it uses a self-correcting geometry procedure in the same spirit of that one presented in Section 3.3.2 in order to ensure that the geometry of the interpolation set does not differ too much from the ideal one.

The reliance on the algorithm proposed by Gould and Toint (2010) allows its main features to be preserved:

- independence between the objective and constraint functions by using different models and trust regions for each one;
- a sequential quadratic programming approach to compute the step;
- the flexibility resulting from the fact that the algorithm doesn't necessarily compute both normal and tangent steps at every iteration (the computation is done for whichever is/are likely to improve feasibility and optimality significantly);
- the specific nature of the algorithm which uses neither merit functions (penalty or otherwise) nor filters;
- the trust-funnel driven convergence — the gist of the algorithm and what makes it different from other composite-step approaches —, a progressively decreasing limit on the permitted infeasibility of the successive iterates.

After having presented different DFO techniques in the previous chapter from which our algorithm gleans, we now explain how DEFT-FUNNEL assembles them all into the trust funnel framework.

Given a poised set of sample points $\mathcal{Y}_0 = \{y^0, y^1, \ldots, y^p\}$ with an initial point $x_0 \in \mathcal{Y}_0$, the first step of our algorithm is to replace the objective function $f(x)$ and the constraint functions $c(x) = (c_1(x), c_2(x), \ldots, c_m(x))$ by surrogate models $m_0^f(x)$ and $m_0^c(x) = (m_{01}^c(x), m_{02}^c(x), \ldots, m_{0m}^c(x))$ built from polynomial interpolation of f and c on $\mathcal{Y}_0$, having therefore the following required interpolation conditions being satisfied

$$\begin{aligned} m_0^f(y^i) &= f(y^i), \\ m_0^c(y^i) &= c(y^i), \end{aligned} \tag{4.0.1}$$

for all $y^i \in \mathcal{Y}_0$.

At each iteration k, we have $x_k \in \mathcal{Y}_k$. Depending on the optimality and feasibility of the point x_k, a new step d_k is computed. As discussed in Section 2.4.6, each full step of the trust-funnel algorithm is decomposed as

$$d_k = n_k + t_k,$$

where the normal step component n_k aims to improve feasibility and the tangent step component t_k reduces the objective function's model without worsening the constraint violation up to first order and abandoning the gains obtained through the former without good reasons. This is done by requiring the tangent step to lie in the null space of the Jacobian of the constraints and by requiring the predicted improvement in the objective function obtained in the tangent step to not be negligible compared to the predicted change in f resulting from the normal step. After having computed a trial point $x_k + d_k$, the algorithm proceeds by checking whether the iteration was successful in a sense yet to be defined. The iterate is then updated correspondingly, while the sample set and the trust regions are updated according to the self-correcting geometry scheme. If $\mathcal{Y}_k$ has been modified, the models $m_k^f(x)$ and $m_k^c(x)$ are updated to satisfy the interpolation conditions (4.0.1) for the new set $\mathcal{Y}_{k+1}$, implying that new function evaluations of f and c are carried out for the additional point obtained at iteration k.

We first give a description of the method for nonlinear optimization problems with equality constraints only. Implementation issues of the proposed method are discussed and numerical results from our experiments on a set of 29 small-scale problems are presented. Then, we extend the original derivative-based trust-funnel method to problems with both equality and inequality constraints as well as simple bounds. At last, we modify DEFT-FUNNEL in order to obtain a final method that can be used to solve problems with general nonlinear constraints where the derivatives are unavailable and we compare its performance to well-known algorithms on a larger set of problems.

4.1 Problems with equality constraints

We consider the equality-constrained nonlinear optimization problem

$$\begin{cases} \min & f(x) \\ \text{s.t.:} & c(x) = 0, \end{cases} \tag{4.1.1}$$

where we assume that $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable, and that f is bounded below on the feasible domain.

We now show how each component is computed in our algorithm.

4.1.1 The normal step

We measure the constraint violation at any point x by

$$v(x) \stackrel{\text{def}}{=} \tfrac{1}{2}\|c(x)\|^2. \tag{4.1.2}$$

Analogous to the trust-funnel method for equality-constrained problems proposed by Gould and Toint, we also have a funnel bound $v_k^{\max}$ for v such that, for each iteration k,

$$v_k \leq v_k^{\max},$$

where $v_k \stackrel{\text{def}}{=} v(x_k)$. As this bound is monotonically decreased, the algorithm is driven towards feasibility, guaranteeing the convergence of the algorithm.

If, at iteration k, the constraint violation is significant with respect to the measure of optimality, yet to be defined, a normal step n_k is computed by reducing the Gauss-Newton model of $\frac{1}{2}\|m_k^c(x)\|^2$ within a trust region, i.e., by solving the following trust-region linear least-squares problem

$$\begin{cases} \min & \tfrac{1}{2}\|c_k + J_k n\|^2 \\ \text{s.t.:} & n_k \in \mathcal{N}_k, \end{cases} \tag{4.1.3}$$

where $c_k \stackrel{\text{def}}{=} c(x_k) = m_k^c(x_k)$, $J_k \stackrel{\text{def}}{=} J(x_k)$ is the Jacobian of m_k^c at x_k and

$$\mathcal{N}_k \stackrel{\text{def}}{=} \{z \in \mathbb{R}^n \mid \|z\| \leq \Delta_k^c\}, \tag{4.1.4}$$

for some radius $\Delta_k^c > 0$. An exact Gauss-Newton step is not required; rather, the computed step n_k must reduce sufficiently the model within $\mathcal{N}_k$ in the sense that it satisfies the Cauchy condition for the problem (4.1.3)

$$\begin{aligned} \delta_k^{c,n} &\stackrel{\text{def}}{=} \tfrac{1}{2}\|c_k\|^2 - \tfrac{1}{2}\|c_k + J_k n_k\|^2 \\ &\geq \kappa_{nC}\|J_k^T c_k\| \min\left[\frac{\|J_k^T c_k\|}{1 + \|W_k\|}, \Delta_k^c\right] \geq 0, \end{aligned} \tag{4.1.5}$$

where $W_k = J_k^T J_k$ is the symmetric Gauss-Newton approximation of the Hessian of $\frac{1}{2}\|m_k^c(x)\|^2$ at x_k and $\kappa_{nC} \in (0, \frac{1}{2}]$. In practice, DEFT-FUNNEL solves

the problem (4.1.3) exactly by using the Moré-Sorensen algorithm (Moré and Sorensen, 1983).

Since there is nothing assuring that the normal step is indeed "normal", we add the following condition requiring that it mostly lies in the space spanned by the columns of the matrix J_k^T

$$\|n_k\| \leq \kappa_n \|c_k\|, \tag{4.1.6}$$

for some $\kappa_n \geq 1$. Such imposed conditions are very reasonable in practice, being satisfied, for instance, if one applies a truncated conjugate-gradient method. If x_k is feasible, we choose a null normal step ($n_k = 0$). When $c(x_k) \neq 0$ and $J(x_k)^T c_k = 0$, we call x_k an *infeasible stationary point*. If such a point is found, the algorithm stops. This might happen when the problem is infeasible, for instance. Note that it is also possible that the algorithm stops at an infeasible stationary point even if the problem has a global minimum as the algorithm only aims at finding stationary points by using local models.

4.1.2 The tangent step

Once the normal step n_k has been calculated, the computation of the tangent step to reduce the model m_k^f is carried out while the algorithm tries not to deteriorate the improvement on feasibility obtained through the former.

We define the quadratic model function

$$\psi_k(x_k + s) \stackrel{\text{def}}{=} f_k + \langle g_k, s\rangle + \tfrac{1}{2}\langle s, G_k s\rangle, \tag{4.1.7}$$

where $f_k \stackrel{\text{def}}{=} f(x_k) = m_k^f(x_k)$, $g_k \stackrel{\text{def}}{=} \nabla m_k^f(x_k)$ and G_k is a symmetric approximation of the Hessian of the Lagrangian $\mathcal{L}(x, y) = m_k^f(x) + \langle \mu, m_k^c(x)\rangle$ given by

$$G_k \stackrel{\text{def}}{=} H_k + \sum_{i=1}^{m} [\hat{\mu}_k]_i C_{ik}. \tag{4.1.8}$$

In the last definition, H_k is a bounded symmetric approximation of $\nabla^2 m_k^f(x_k)$, the matrices C_{ki} are bounded symmetric approximations of the constraints' models Hessians $\nabla^2 m_{ki}^c(x_k)$ and the vector $\hat{\mu}_k$ may be viewed as a bounded local approximation of the Lagrange multipliers, in the sense that we require that

$$\|\hat{\mu}_k\| \leq \kappa_\mu, \tag{4.1.9}$$

for some $\kappa_\mu > 0$.

By using the decomposition $d_k = n_k + t_k$, we have

$$\psi_k(x_k + n_k) = f_k + \langle g_k, n_k\rangle + \tfrac{1}{2}\langle n_k, G_k n_k\rangle \tag{4.1.10}$$

and

$$\begin{array}{rcl} \psi_k(x_k+n_k+t) & = & f_k+\langle g_k, n_k+t\rangle+\frac{1}{2}\langle n_k+t, G_k(n_k+t)\rangle \\ & = & \psi_k(x_k+n_k)+\langle g_k^n, t\rangle+\frac{1}{2}\langle t, G_k t\rangle \end{array} \tag{4.1.11}$$

where

$$g_k^n \stackrel{\text{def}}{=} g_k + G_k n_k. \tag{4.1.12}$$

We thus have that (4.1.11) is a quadratic model of the function $m_k^f(x_k+n_k+t)$. In the interest of assuring that it is a proper local approximation, the complete step $d = n_k + t$ must belong to

$$\mathcal{T}_k \stackrel{\text{def}}{=} \{d \in \mathbb{R}^n \mid \|d\| \leq \Delta_k^f\}, \tag{4.1.13}$$

for some radius Δ_k^f. The minimization of (4.1.11) should then be restricted to the intersection of $\mathcal{N}_k$ and $\mathcal{T}_k$, which imposes that the *tangent step* t_k results in a complete step $d_k = n_k + t_k$ that satisfies the inclusion

$$d_k \in \mathcal{R}_k \stackrel{\text{def}}{=} \mathcal{N}_k \cap \mathcal{T}_k \stackrel{\text{def}}{=} \{d \in \mathbb{R}^n \mid \|d\| \leq \Delta_k\}, \tag{4.1.14}$$

where the radius Δ_k of $\mathcal{R}_k$ is thus given by

$$\Delta_k = \min[\Delta_k^c, \Delta_k^f]. \tag{4.1.15}$$

Due to (4.1.14), we first check if n_k belongs to $\mathcal{R}_k$ before computing t_k by asking that

$$\|n_k\| \leq \kappa_{\mathcal{R}} \Delta_k, \tag{4.1.16}$$

for some $\kappa_{\mathcal{R}} \in (0,1)$. If (4.1.16) holds, which means that there is "enough space left" to make another step without crossing the trust region border, the tangent step is finally computed by (approximately) solving the following problem

$$\left\{ \begin{array}{ll} \min & \langle g_k^n, t\rangle + \frac{1}{2}\langle t, G_k t\rangle \\ \text{s.t.:} & J_k t = 0, \\ & \|n_k + t\| \leq \Delta_k. \end{array} \right. \tag{4.1.17}$$

In practice, we do not solve the problem (4.1.17) exactly, rather we only require a "sufficient" reduction of (4.1.11) within the hyperplane tangent to the constraints intersected to the trust region. Note that the original proposal of Gould and Toint also allows for "tangent" steps which do not lie exactly in the null space of the Jacobian J_k, but we neglect this possibility here because the computation of exactly tangent steps (i.e. satisfying $J_k t = 0$) is acceptable in the context of small-scale problems.

In order to compute an approximate gradient at $x_k + n_k$, we first compute a new local estimate of the Lagrange multipliers μ_k by solving the least-squares problem

$$\min_{\mu} \frac{1}{2}\|g_k^n + J_k^T \mu\|^2. \tag{4.1.18}$$

Since the number of variables of the problems considered in this work is small, we allow (4.1.18) to be solved exactly in our experiments, although again only an approximation could be required. The orthogonal projection of g_k^n onto the null space of J_k is then denoted by

$$r_k \stackrel{\text{def}}{=} g_k^n + J_k^T \mu_k, \tag{4.1.19}$$

which motivates that we require the tangent step to produce a reduction in the model ψ_k which is at least a fraction of that achieved by solving the modified Cauchy point subproblem

$$\min_{\substack{\tau>0 \\ x_k+n_k-\tau r_k \in \mathcal{R}_k}} \psi_k(x_k + n_k - \tau r_k), \tag{4.1.20}$$

where we have assumed that $\|r_k\| > 0$. This procedure ensures (see Section 8.1.5 of Conn et al., 2000), for some $\kappa_{tC1} \in (0, 1]$, the modified Cauchy condition

$$\begin{aligned} \delta_k^{f,t} &\stackrel{\text{def}}{=} \psi_k(x_k + n_k) - \psi_k(x_k + n_k + t_k) \\ &\geq \kappa_{tC1}\pi_k \min\left[\frac{\pi_k}{1 + \|G_k\|}, \tau_k\|r_k\|\right] > 0 \end{aligned} \tag{4.1.21}$$

on the decrease of the objective function model within $\mathcal{R}_k$, where we have set

$$\pi_k \stackrel{\text{def}}{=} \frac{\langle g_k^n, r_k\rangle}{\|r_k\|} \geq 0, \tag{4.1.22}$$

where τ_k is the maximal step length along $-r_k$ from $x_k + n_k$ which remains in the trust-region $\mathcal{R}_k$. If $r_k = 0$, we simply set $\pi_k = 0$. We also have that

$$\tau_k\|r_k\| \geq (1 - \kappa_{\mathcal{R}})\Delta_k$$

by construction and thus the modified Cauchy condition (4.1.21) may now be rewritten as

$$\begin{aligned} \delta_k^{f,t} &\stackrel{\text{def}}{=} \psi_k(x_k + n_k) - \psi_k(x_k + n_k + t_k) \\ &\geq \kappa_{tC}\pi_k \min\left[\frac{\pi_k}{1 + \|G_k\|}, \Delta_k\right], \end{aligned} \tag{4.1.23}$$

with $\kappa_{tC} \stackrel{\text{def}}{=} \kappa_{tC1}(1 - \kappa_{\mathcal{R}}) \in (0, 1)$. As it can be seen in (4.1.23), π_k may be considered as an optimality measure in the sense that it measures how much decrease could be obtained locally along the negative of the approximate projected gradient r_k. Figure 4.1 on the facing page depicts the components of the final step $d_k = n_k + t_k$ computed by the algorithm and illustrates one of its iterations.

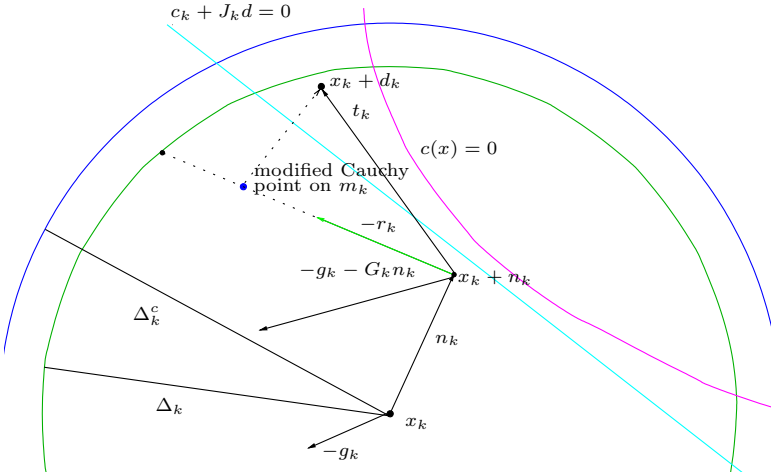


Figure 4.1: The components of the step d_k with $\Delta_k = \Delta_k^f$.

4.1.3 Which steps to compute and retain

We now explain when the normal step n_k and the tangent step t_k should be computed at iteration k given the constraint violation and the measure of optimality described in the previous subsection.

The normal step is computed when $k = 0$ or the current violation is "significant", which is now formally defined by the conditions

$$\|c_k\| > \omega_n(\pi_{k-1}) \text{ or } v_k > \kappa_{vv} v_k^{\max}, \tag{4.1.24}$$

where ω_n is some bounding function of π_{k-1} and $\kappa_{vv} \in (0, 1)$ is a constant. If (4.1.24) fails, the computation of the normal step is not required and so we set $n_k = 0$.

When $r_k = 0$, the computation of tangent step is not needed, and we simply define $\pi_k = 0$ and $t_k = 0$. If π_k is small compared to the current infeasibility, i.e., for a given a monotonic bounding function ω_t of $\|c_k\|$, the condition

$$\pi_k > \omega_t(\|c_k\|) \tag{4.1.25}$$

fails, then we should worry more about feasibility than optimality. Thus, we set $t_k = 0$ in this case. Notice also that both conditions (4.1.25) and (4.1.24) may fail, in which case we have that $d_k = n_k + t_k = 0$, and the new vector μ_k of approximate Lagrange multipliers is the only one to have been computed in the iteration.

We also require that

$$\omega_n(t) = 0 \iff t = 0 \quad \text{and} \quad \omega_t(\omega_n(t)) \leq \kappa_\omega t, \tag{4.1.26}$$

for all $t \geq 0$ and for some $\kappa_\omega \in (0, 1)$. Examples for the functions $\omega_n(t)$ and $\omega_t(t)$ might be

$$\omega_n(t) \stackrel{\text{def}}{=} 0.01 \min[1,\ t] \quad \text{and} \quad \omega_t(t) \stackrel{\text{def}}{=} 0.01 \min[1,\ t^2]. \tag{4.1.27}$$

The conditions (4.1.26) are important to prove the following lemma in Gould and Toint (2010).

Lemma 4.1.1. For all k such that $d_k = 0$ and x_k is not an infeasible stationary point,

$$\pi_k \leq \kappa_\omega \pi_{k-1}.$$

It is argued in their paper that this lemma, the initial assumption that f is bounded below on the feasible domain and the fact that the algorithm terminates when an infeasible stationary point is found imply that such behavior ($d_k = 0$) cannot persist unless x_k is optimal.

The ideal tangent step t_k should not be small compared to the normal step; besides, it should cause a good reduction in the model (4.1.11). Conversely, if it is large and produces no considerable improvement on optimality but undoes the gains in feasibility obtained through the normal step, it is not worth using it. Such considerations are taken into account by verifying whether the conditions

$$\|t_k\| > \kappa_{\mathcal{CS}} \|n_k\| \tag{4.1.28}$$

and

$$\delta_k^f \stackrel{\text{def}}{=} \delta_k^{f,t} + \delta_k^{f,n} \geq \kappa_\delta \delta_k^{f,t}, \tag{4.1.29}$$

where

$$\delta_k^{f,n} \stackrel{\text{def}}{=} \psi_k(x_k) - \psi_k(x_k + n_k), \tag{4.1.30}$$

are satisfied for some $\kappa_{\mathcal{CS}} > 1$ and for $\kappa_\delta \in (0, 1)$. The inequality (4.1.29) indicates that the *predicted* improvement in the objective function obtained in the tangent step is not negligible compared to the *predicted* change in f resulting from the normal step. If (4.1.28) holds but (4.1.29) fails, the tangent step is not useful as we have argued above, and we simply reset $t_k = 0$.

4.1.4 Iterations types

After the computation of the trial point step

$$x_k^+ \stackrel{\text{def}}{=} x_k + d_k,$$

we need to decide whether the algorithm should accept it or reject it. If $n_k = t_k = 0$, iteration k is said to be a *μ-iteration* because the only computation

performed is that of a new vector of Lagrange multiplier estimates. We call iteration k an *f-iteration* if $t_k \neq 0$, (4.1.29) holds, and

$$v_k^+ \leq v_k^{\max}, \tag{4.1.31}$$

where $v_k^+ \stackrel{\text{def}}{=} v(x_k^+, s_k^+)$. Condition (4.1.31) ensures that the step keeps feasibility within reasonable bounds. Thus the iteration's expected major achievement is, in this case, a decrease in the value of the objective function f, hence its name. If iteration k is neither a μ-iteration nor an f-iteration, then it is said to be a *c-iteration*. If (4.1.29) fails, then the expected major achievement (or failure) of iteration k is to improve feasibility, which is also the case when the step only contains its normal component.

We now describe the conditions for the acceptation of the trial point, which are based on the major expected achievement of the iteration.

- If iteration k is a μ-iteration, the only choice left is to restart with $x_{k+1} = x_k$ using the new multipliers. We then define

$$\Delta_{k+1}^f = \Delta_k^f \text{ and } \Delta_{k+1}^c = \Delta_k^c \tag{4.1.32}$$

  and keep the current value of the maximal infeasibility $v_{k+1}^{\max} = v_k^{\max}$.

- If iteration k is an f-iteration, we accept the trial point (i.e., $x_{k+1} = x_k^+$) if

$$\rho_k^f \stackrel{\text{def}}{=} \frac{f(x_k) - f(x_k^+)}{\delta_k^f} \geq \eta_1, \tag{4.1.33}$$

  and reject it (i.e., $x_{k+1} = x_k$), otherwise. The value of the maximal infeasibility measure is left unchanged, that is, $v_{k+1}^{\max} = v_k^{\max}$. Note that $\delta_k^f > 0$ (because of (4.1.23) and (4.1.29)) unless x_k is first-order critical, and hence that condition (4.1.33) is well-defined.

- If iteration k is a c-iteration, we accept the trial point if the improvement in feasibility is comparable to its predicted value

$$\delta_k^c \stackrel{\text{def}}{=} \tfrac{1}{2}\|c_k\|^2 - \tfrac{1}{2}\|c_k + J_k d_k\|^2,$$

  and the latter is itself comparable to its predicted decrease along the normal step, that is,

$$n_k \neq 0, \quad \delta_k^c \geq \kappa_{cn}\delta_k^{c,n} \text{ and } \rho_k^c \stackrel{\text{def}}{=} \frac{v_k - v_k^+}{\delta_k^c} \geq \eta_1, \tag{4.1.34}$$

  for some $\kappa_{cn} \in (0, 1-\kappa_{tg}]$. If (4.1.34) fails, the trial point is rejected. We update the value of the maximal infeasibility by

$$v_{k+1}^{\max} = \begin{cases} \max\left[\kappa_{tx1} v_k^{\max}, v_k^+ + \kappa_{tx2}(v_k - v_k^+)\right] & \text{if (4.1.34) hold,} \\ v_k^{\max} & \text{otherwise,} \end{cases} \tag{4.1.35}$$

  for some $\kappa_{tx1} \in (0,1)$ and $\kappa_{tx2} \in (0,1)$.

We do not describe here the updating strategy of the trust regions radii in iterations of type f and c as it is embedded in the self-correcting geometry scheme; we leave its details to the next subsection, where the complete algorithm is given.

As we justify now, the last condition in (4.1.34) is well-defined. Firstly, we only check the third condition *after* the first two conditions have been verified. Assuming that $n_k \neq 0$, the Cauchy condition (4.1.5) and $c(x_k) \neq 0$ ensure that $\delta_k^{c,n} > 0$ provided $J_k^T c_k \neq 0$. Thus the third condition is well defined, unless x_k is an infeasible stationary point, in which case the algorithm is terminated.

4.1.5 The algorithm

We are now ready to state our complete algorithm, which puts the above discussion in a more formal context.

Algorithm 4.1.1: DEFT-FUNNEL

Step 0: Initialization. An initial accuracy threshold ϵ_0, an initial vector of multipliers μ_{-1} and positive initial trust-region radii Δ_0^f and Δ_0^c are given, as well as the constants

$$\alpha \in (0,1), \quad 0 < \gamma_1 < 1 < \gamma_2, \quad \zeta \geq 1, \quad 0 < \eta_1 < \eta_2 < 1 \quad \text{and} \quad \beta, \eta_3 > 0.$$

An initial set of interpolation points is also given, $\mathcal{Y}_0$, with $x_0 \in \mathcal{Y}_0 \subset \mathcal{B}(x_0; \Delta_0)$ and $|\mathcal{Y}_0| \geq n+1$, as well as the maximum number of interpolation points $p_{\max} \geq |\mathcal{Y}_0|$ in $\mathcal{Y}_k$ at the end. Let p_k denote the cardinality of $\mathcal{Y}_k$. This interpolation set defines interpolation models m_0^f and m_0^c around x_0 and associated Lagrange polynomials $\{\ell_{0,j}\}_{j=0}^p$. Define $\Delta_0 = \min[\Delta_0^f, \Delta_0^c] \leq \Delta^{\max}$, and $v_0^{\max} = \max[\kappa_{ca}, \kappa_{cr} v(x_0)]$ for some constants $\kappa_{ca} > 0$ and $\kappa_{cr} > 1$. Define $\nu_f^{\max} > 0$ and $\nu_c^{\max} > 0$, the maximum number of times that the tangential and normal trust regions sizes can be reduced when an interpolation point is replaced at unsuccessful iterations. Initialize the corresponding counters $\nu_f = \nu_c = 0$. Define $k = 0$ and $i = 0$.

Step 1: Criticality step. Define $\hat{m}_i^f = m_k^f$, $\hat{m}_i^c = m_k^c$, $\nabla \hat{g}_i = \nabla g_k$, $\hat{J}_i = J_k$ and $\nabla \hat{\mathcal{L}}_i = \nabla \hat{g}_i + \hat{J}_i^T \mu_k$.

Step 1.1: If $\|c_k\| \leq \epsilon_i$ and $\|\nabla \hat{\mathcal{L}}_i\| \leq \epsilon_i$, set $\epsilon_{i+1} = \max[\alpha \|c_k\|, \alpha \|\nabla \hat{\mathcal{L}}_i\|, \epsilon]$ and modify $\mathcal{Y}_k$ as needed to ensure it is Λ-poised in $\mathcal{B}(x_k, \epsilon_{i+1})$. If $\mathcal{Y}_k$ was modified, compute new models $\hat{m}_i^f$ and $\hat{m}_i^c$, calculate $\nabla \hat{g}_i$, $\hat{J}_i$ and $\nabla \hat{\mathcal{L}}_i$ associated to these models and increment i by one. If $\|c_k\| \leq \epsilon$ and $\|\nabla \hat{\mathcal{L}}_i\| \leq \epsilon$, return x_k; otherwise, start Step 1.1 again;

Step 1.2: Set $m_k^f = \hat{m}_i^f$, $m_k^c = \hat{m}_i^c$, $\nabla g_k = \nabla \hat{g}_i$, $J_k = \hat{J}_i$, $\nabla \mathcal{L}_k = \nabla \hat{\mathcal{L}}_i$ and $\Delta_k = \beta \max[\|c_k\|, \|\nabla \mathcal{L}_k\|]$, and define $v_i = x_k$ if a new model has been computed.

Step 2: Normal step. Compute a normal step n_k that sufficiently reduces the linearized infeasibility (in the sense that (4.1.5) holds), under the constraint that (4.1.4) and (4.1.6) also hold. This computation must be performed if $k = 0$ or if (4.1.24) holds when $k > 0$.

If n_k has not been computed, set $n_k = 0$.

Step 3: Tangent step. If (4.1.16) holds, then

Step 3.1: select a vector $\hat{\mu}_k$ satisfying (4.1.9) and define G_k by (4.1.8);

Step 3.2: compute μ_k by solving (4.1.18) and r_k by (4.1.19);

Step 3.3: if (4.1.25) holds, compute a tangent step t_k that sufficiently reduces the model (4.1.11) (in the sense that (4.1.23) holds) and such that the complete step $d_k = n_k + t_k$ satisfies (4.1.14).

If (4.1.16) fails, set $\mu_k = \mu_{k-1}$. In this case, or if (4.1.25) fails, or if (4.1.28) holds but (4.1.29) fails, set $t_k = 0$ and $d_k = n_k$. In all cases, define $x_k^+ = x_k + d_k$.

Step 4: Conclude a μ-iteration. If $n_k = t_k = 0$, then

Step 4.1: set $x_{k+1} = x_k$;

Step 4.2: define $\Delta_{k+1}^f = \Delta_k^f$ and $\Delta_{k+1}^c = \Delta_k^c$;

Step 4.3: set $v_{k+1}^{\max} = v_k^{\max}$ and $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^c]$.

Step 5: Conclude an f-iteration. If $t_k \neq 0$ and (4.1.29) and (4.1.31) hold,

Step 5.1: Augment the interpolation set. If $p_k < p_{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$.

- If $\rho_k^f \geq \eta_1$, set $x_{k+1} = x_k^+$ and $\nu_f = 0$.
  If $\rho_k^f \geq \eta_2$, set $\Delta_{k+1}^f = \min[\max[\gamma_2 \|d_k\|, \Delta_k^f], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$.
  If $v(x_k^+) < \eta_3\, v_k^{\max}$, set $\Delta_{k+1}^c = \min[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.
- If $\rho_k^f < \eta_1$, set $x_{k+1} = x_k$ and $\Delta_{k+1}^c = \Delta_k^c$.
  If $\nu_f \leq \nu_f^{\max}$, set $\Delta_{k+1}^f = \gamma_1 \Delta_k^f$ and $\nu_f = \nu_f + 1$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$.

Step 5.2: Successful iteration. If $p_k = p_{\max}$, $\rho_k^f \geq \eta_1$, set $x_{k+1} = x_k^+$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.1.36}$$

Set $\nu_f = 0$. If $\rho_k^f \geq \eta_2$, set $\Delta_{k+1}^f = \min[\max[\gamma_2\|d_k\|, \Delta_k^f], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$.
If $v(x_k^+) < \eta_3\, v_k^{\max}$, then set $\Delta_{k+1}^c = \min[\max[\gamma_2\|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.

Step 5.3: Replace a far interpolation point. If $p_k = p_{\max}$, $\rho_k^f < \eta_1$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| > \zeta\Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\}$$

is non-empty, then define $x_{k+1} = x_k$, and set $\Delta_{k+1}^f = \gamma_1\|d_k\|$ if $\nu_f \leq \nu_f^{\max}$ or $\Delta_{k+1}^f = \Delta_k^f$ otherwise.
Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg\max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.1.37}$$

If $\nu_f \leq \nu_f^{\max}$, update $\nu_f = \nu_f + 1$.

Step 5.4: Replace a close interpolation point. If $p_k = p_{\max}$, $\rho_k^f < \eta_1$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| \leq \zeta\Delta \text{ and } |\ell_{k,j}(x_k^+)| > \lambda\}$$

is non-empty, then define $x_{k+1} = x_k$ and set $\Delta_{k+1}^f = \gamma_1\|d_k\|$ if $\nu_f \leq \nu_f^{\max}$ or $\Delta_{k+1}^f = \Delta_k^f$ otherwise.
Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg\max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.1.38}$$

If $\nu_f \leq \nu_f^{\max}$, update $\nu_f = \nu_f + 1$.

Step 5.5: Reduce the trust-region radius. If $p_k = p_{\max}$, $\rho_k^f < \eta_1$ and either $x_k = v_i$ and $\Delta_k^f > \epsilon_i$ or $\mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then define $x_{k+1} = x_k$, $\Delta_{k+1}^c = \Delta_k^c$, choose $\Delta_{k+1}^f = \gamma_1\|d_k\|$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

Step 5.6: Update the combined radius. Set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^c]$ and $v_{k+1}^{\max} = v_k^{\max}$.

Step 6: Conclude a c-iteration. If either $n_k \neq 0$ and $t_k = 0$, or either one of (4.1.29) or (4.1.31) fails,

Step 6.1: Augment the interpolation set. If $p_k < p_{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$.

- If $\rho_k^c \geq \eta_1$, set $x_{k+1} = x_k^+$, $\Delta_{k+1}^f = \Delta_k^f$ and $\nu_c = 0$.
  If $\rho_k^c \geq \eta_2$, set $\Delta_{k+1}^c = \min[\max[\gamma_2\|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.

- If $\rho_k^c < \eta_1$, set $x_{k+1} = x_k$ and $\Delta_{k+1}^f = \Delta_k^f$.
  If $\nu_c \leq \nu_c^{\max}$, set $\Delta_{k+1}^c = \gamma_1 \|n_k\|$ if $\|n_k\| \neq 0$, and $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$ otherwise ($\|n_k\| = 0$). Update $\nu_c = \nu_c + 1$.
  If $\nu_c > \nu_c^{\max}$, set $\Delta_{k+1}^c = \Delta_k^c$.

Step 6.2: Successful iteration. If $p_k = p_{\max}$ and (4.1.34) holds, set $x_{k+1} = x_k^+$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.1.39}$$

Set $\Delta_{k+1}^f = \Delta_k^f$ and $\nu_c = 0$. Set $\Delta_{k+1}^c = \min[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}]$ if $\rho_k^c \geq \eta_2$ or $\Delta_{k+1}^c = \Delta_k^c$ otherwise.

Step 6.3: Replace a far interpolation point. If $p_k = p_{\max}$, (4.1.34) fails, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k \mid > \zeta\Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\}$$

is non-empty, then define $x_{k+1} = x_k$ and set $\Delta_{k+1}^f = \Delta_k^f$. If $\nu_c \leq \nu_c^{\max}$, then set $\Delta_{k+1}^c = \gamma_1 \|n_k\|$ if $\|n_k\| \neq 0$, or $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$ otherwise ($\|n_k\| = 0$). If $\nu_c > \nu_c^{\max}$, set $\Delta_{k+1}^c = \Delta_k^c$.
Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.1.40}$$

If $\nu_c \leq \nu_c^{\max}$, update $\nu_c = \nu_c + 1$.

Step 6.4: Replace a close interpolation point. If $p_k = p_{\max}$, (4.1.34) fails, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| \leq \zeta\Delta \text{ and } |\ell_{k,j}(x_k^+)| > \lambda\}$$

is non-empty, then set $x_{k+1} = x_k$ and $\Delta_{k+1}^f = \Delta_k^f$. If $\nu_c \leq \nu_c^{\max}$, then set $\Delta_{k+1}^c = \gamma_1 \|n_k\|$ if $\|n_k\| \neq 0$, or $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$ otherwise ($\|n_k\| = 0$). If $\nu_c > \nu_c^{\max}$, set $\Delta_{k+1}^c = \Delta_k^c$.
Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.1.41}$$

If $\nu_c \leq \nu_c^{\max}$, update $\nu_c = \nu_c + 1$.

Step 6.5: Reduce the trust-region radius. If $p_k = p_{\max}$, (4.1.34) fails and either $x_k = v_i$ and $\Delta_k^c > \epsilon_i$ or $\mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then set $x_{k+1} = x_k$ and $\Delta_{k+1}^f = \Delta_k^f$. If $\|n_k\| \neq 0$, set $\Delta_{k+1}^c = \gamma_1 \|n_k\|$, otherwise set $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$. Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

Step 6.6: Update the combined radius and the funnel bound. Set $\Delta_{k+1} = \min[\Delta^f_{k+1}, \Delta^c_{k+1}]$ and update $v_k^{\max}$ using (4.1.35).

Step 7: Update the model and Lagrange polynomials. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation models m^f_{k+1} and m^c_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$ and the associated Lagrange polynomials $\{l_{k+1,j}\}_{j=0}^p$. Increment k by one and go to Step 1.

4.1.6 Implementation and experiments

4.1.6.1 Building the models

The construction of the models and the management of the interpolation set in our algorithm benefit to a great extent from the framework developed in Tröltzsch (2011) and Gratton et al. (2011). As the details of such steps can be readily found in those references, we give here only a brief description of the main elements involved.

- The interpolation models $m^f(x)$ and $m^c_j(x)$ are built from the solution of the interpolation system

$$M(\bar{\phi}, \mathcal{Y})\alpha_{\bar{\phi}} = h(\mathcal{Y}), \tag{4.1.42}$$

where

$$M(\bar{\phi}, \mathcal{Y}) = \begin{pmatrix} \bar{\phi}_0(y^0) & \bar{\phi}_1(y^0) & \cdots & \bar{\phi}_q(y^0) \\ \bar{\phi}_0(y^1) & \bar{\phi}_1(y^1) & \cdots & \bar{\phi}_q(y^1) \\ \vdots & \vdots & \ddots & \vdots \\ \bar{\phi}_0(y^p) & \bar{\phi}_1(y^p) & \cdots & \bar{\phi}_q(y^p) \end{pmatrix}, \quad h(\mathcal{Y}) = \begin{pmatrix} h(y^0) \\ h(y^1) \\ \vdots \\ h(y^p) \end{pmatrix}, \quad p \leq q,$$

where $\bar{\phi}$ is the basis of monomials and $h(x)$ is replaced by the objective function $f(x)$ or some constraint function $c_j(x)$. Following the discussion in Section 3.2.3 on how to measure poisedness cheaply and efficiently, we make use of the matrix $\hat{M} = M(\bar{\phi}, \hat{\mathcal{Y}})$ when solving (4.1.42), where $\hat{\mathcal{Y}}$ is the shifted and scaled version of $\mathcal{Y}$.

If $p_0 = |\mathcal{Y}_0| = n + 1$, a linear model rather than an underdetermined quadratic model is built for each function. The reason is that, despite both having error bounds that are linear in Δ for the first derivatives, the error bound for the latter includes also the norm of the model's Hessian, as it has been seen in Lemma 3.2.9, which makes it worse than the former. Whenever

$$n + 1 < p_k \leq (n + 1)(n + 2)/2 = p^{\max},$$

the algorithm builds underdetermined quadratic models based on the choice of the user between the approaches described in Section 3.2.4. If

regression models are considered instead, we set $p^{\max} = (n+1)(n+2)$, which means that we allow the sample set to have twice the number of sample points required for fully quadratic interpolation models. Notice that having a number of sample points much larger than the required for quadratic interpolation can also worsen the quality of the interpolation models as the sample set could contain points that are too far from the iterate, which is not ideal for models built for local approximation.

- Although the self-correcting geometry scheme guarantees (under certain assumptions) the improvement of the quality of the geometry of the sample set at unsuccessful iterations, it does not prevent the deterioration of the geometry along successful iterations, which may cause the interpolation matrix $\hat{M}$ to become ill-conditioned. This can also happens when

$$p_k < (n+1)(n+2)/2 = p^{\max},$$

in which case every trial point is added into the interpolation set without caring for poisedness. In this case, checking the Lagrange polynomials' values is not applicable, since there is no point to be replaced. To overcome these difficulties, Tröltzsch proposed to monitor the condition number $\kappa(\hat{M})$ at each iteration and use its singular value decomposition to replace all singular values smaller than a threshold δ whenever one has $\kappa(\hat{M}) > \kappa_{\text{illcond}}$, for some large value κ_{illcond}. She then proved that the error bound on the gradient of the perturbed model for the linear interpolation case remains quite similar. In our algorithm, we also apply this strategy for the underdetermined quadratic interpolation case, although no equivalent theoretical result has been proven.

- The coefficients of each Lagrange polynomial $\lambda_i(x)$ are calculated by solving the linear system

$$\hat{M}\lambda_i = e_i, \tag{4.1.43}$$

where e_i denotes the i-th unit vector ($i = 0, 1, 2$), whose i-th component is one and all others are zero. The maximization of the absolute value of $\lambda_i(x)$ in a region $\mathcal{B}$, which is part of the Algorithm 3.2.1 on page 42 for improving well-poisedness of the sample set, is made by minimizing first $\lambda_i(x)$ within $\mathcal{B}$ and then $-\lambda_i(x)$. Each minimization uses a modified version of the Moré-Sorensen algorithm (Moré and Sorensen, 1983) developed by Tröltzsch for problems where bound constraints are also considered.

4.1.6.2 Solving the subproblems

The normal steps are computed with the Moré-Sorensen algorithm, while the tangent steps are calculated with a truncated projected CG method (see Section 2.4.2 for a brief overview). In the latter, we consider the projection

operator $P = Z(Z^T G Z)^{-1} Z^T$ with $G = I_n$. This implies that P is the orthogonal projection operator onto the null space of A and thus can be rewritten as $P = I_n - A^T(AA^T)^{-1}A$. Using this formula, we can express the system of the preconditioned residual

$$g^+ = Pr^+$$

by the augmented system

$$\begin{pmatrix} I_n & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} g^+ \\ v^+ \end{pmatrix} = \begin{pmatrix} r^+ \\ 0 \end{pmatrix}. \tag{4.1.44}$$

As pointed out by Gould et al. (2001), this augmented system approach suffers from significant round-off errors as g^+ becomes gradually small while r^+ does not. Since the magnitude of the errors generated in the solution of (4.1.44) is dominated by v^+ and v^+ remains large, the error in the computation of g^+ tends to be large as well. To reduce these errors, the authors proposed a projected preconditioned CG method that makes use of iterative refinement and a residual update strategy that redefines r^+ so that its norm is closer to that of g^+. Because similar errors appeared in the solution of the tangent step subproblem in our numerical experiments, we decided to incorporate these strategies into our algorithm. The reformulated truncated projected CG method used in DEFT-FUNNEL is given below.

Algorithm 4.1.2: Reformulated truncated projected CG method.

Initialization. Given x such that $Ax = b$, compute $r = Hx + c$, $r = Pr$, $g = Pr$, and set $d = -g$. Choose a tolerance $\theta_{\max}$.

Repeat until stopping criteria is satisfied

Set $\kappa_{cu} \leftarrow \langle d, Hd \rangle$ and $\alpha \leftarrow \langle r, g \rangle / \kappa_{cu}$.

If $\kappa_{cu} \leq 0$ or $\|x+\alpha d\| \geq \Delta$, compute σ as the positive root of $\|x+\sigma d\| = \Delta$ and set $x \leftarrow x + \sigma d$. Otherwise, set $x \leftarrow x + \alpha d$.

$$r^+ \leftarrow r + \alpha\, Hd; \tag{4.1.45a}$$

$$g^+ \leftarrow Pr^+; \tag{4.1.45b}$$

Apply iterative refinement to Pr^+, if necessary,

$$\text{until } \max_i \left\{ \frac{A_i^T g^+}{\|A_i\| \|g^+\|} \right\} < \theta_{\max}; \tag{4.1.45c}$$

$$\beta \leftarrow \frac{\langle r^+, g^+ \rangle}{\langle r, g \rangle}; \tag{4.1.45d}$$

$$d \leftarrow -g^+ + \beta d; \tag{4.1.45e}$$

$$g \leftarrow g^+; \quad r \leftarrow r^+. \tag{4.1.45f}$$

In Algorithm 4.1.2, A_i denotes the i-th row of A, which means that (4.1.45c) measures the angle between g and the rows of A. In exact arithmetic, this value should be zero as it indicates that the CG iterates remain in the constraint manifold $Ax = b$. In our experiments, we set the tolerance $\theta_{\max} = 10^{-12}$. The algorithm stops when $\sqrt{\langle r, g\rangle} \leq 10^{-12}$ or the number of iterations exceeds $2(n-m)$, where $n-m$ is the dimension of the reduced problem (2.4.5). The iterative refinement, in turn, is done by applying the following algorithm.

Algorithm 4.1.3: Iterative refinement.

Repeat until stopping criteria is satisfied

$$\text{Compute} \quad \rho_g = r^+ - g^+ - A^T v^+ \ \text{ and } \ \rho_v = -Ag^+,$$

$$\text{solve} \quad \begin{pmatrix} I_n & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta g^+ \\ \Delta v^+ \end{pmatrix} = \begin{pmatrix} \rho_g \\ \rho_v \end{pmatrix},$$

$$\text{and update} \quad g^+ \leftarrow g^+ + \Delta g^+ \ \text{ and } \ v^+ \leftarrow v^+ + \Delta v^+.$$

Finally, we estimate the Lagrange multipliers by solving the subproblem (4.1.18) with the standard CG algorithm described in Section 2.3.3.

4.1.6.3 Improving well-poisedness in μ-iterations

In our experiments, we noticed that null steps $d_k = 0$ might be caused by the poor quality of the interpolation models. For this reason, we verify the Λ-poisedness in μ-iterations and attempt to improve it whenever we have

$$\Lambda\, \Delta(\mathcal{Y}_k) > \epsilon, \tag{4.1.46}$$

where

$$\Delta(\mathcal{Y}_k) \stackrel{\text{def}}{=} \max_j \|y^{k,j} - x_k\|$$

and $\epsilon > 0$ is a threshold chosen by the user. The inequality (4.1.46) gives an estimate of the error bound for the models based on the Lemma 3.2.9 on page 45. If (4.1.46) holds, we try to reduce the value at the left side by modifying the sample set $\mathcal{Y}_k$. Firstly, we choose a constant $\xi \in (0,1)$ and replace all points $y^{k,j} \in \mathcal{Y}_k$ such that

$$\|y^{k,j} - x_k\| > \xi \Delta(\mathcal{Y}_k)$$

by new points $y_*^{k,j}$ that (approximately) maximizes $|\ell_{j_k}(x)|$ in $\mathcal{B}(x_k; \xi\Delta(\mathcal{Y}_k))$. Then we use the Algorithm 3.2.1 on page 42 with the smaller region $\mathcal{B}$ to

improve Λ-poisedness of the new sample set. The use of this strategy in this type of iteration presented good results in our experiments. The fact that a very small number of μ-iterations occurred during the tests may have compensated for the additional function evaluations performed because of the change in $\mathcal{Y}_k$.

4.1.6.4 Code specifications

In this subsection, we do not intend to describe the code in detail as it is composed by several functions that amount to thousand of lines; rather, we provide some knowledge to the reader of the main components of the trust-funnel framework.

We have implemented DEFT-FUNNEL as well as the integrated methods for solving the subproblems using Matlab (R2013a) in a Linux environment. The call to DEFT-FUNNEL is made through the function `deft_funnel`, which mainly sets the initial values for all the parameters, defines an initial interpolation set and builds the associated models to replace the objective function and the constraints. Only four arguments are required from the user: a function handle to the objective function, a function handle to the constraint functions, an initial starting point and a vector of estimates for the Lagrange multipliers. A simple example of usage is given below.

```
[ x, fx, mu, norm_gradlag, norm_cx, Delta, nit, nfeval] = deft_funnel(
    @objfun, @cons, x0, mu0)
```

The default output of the algorithm is given by:

- `x`: point returned by the algorithm;
- `fx`: value of the objective function at `x`;
- `mu`: vector of estimates of the Lagrange multipliers;
- `norm_gradlag`: norm of the gradient of the Lagrangian function at (`x`, `mu`);
- `norm_cx`: norm of c at `x`;
- `Delta`: final trust region radius;
- `nit`: number of iterations required;
- `nfeval`: number of calls made to the subroutine that evaluates the objective function and the constraints.

The user may also set other constants' values through optional input parameters while calling `deft_funnel`. For instance, it is possible to choose the initial degree of the models (fully linear, quadratic with a diagonal Hessian or fully quadratic), the approach to build the underdetermined quadratic models (subbasis selection, minimum Frobenius norm, minimum ℓ_2-norm or regression),

the parameters' values related to the trust-funnel algorithm, the threshold ϵ for declaring convergence and the verbosity level. Any optional input parameters can be set by the user by specifying first the parameter (for example, the threshold for declaring convergence which is defined in our code by the constant `epsilon`) followed by the value to be attributed to it (for instance, `1.0e-04`) in the call to `deft_funnel`. An example of call to the algorithm with some optional input parameters' values specified in the argument list is shown below.

```
[ x, fx, mu, norm_gradlag, norm_cx, Delta, nit, nfeval ] = deft_funnel(
    @objfun, @cons, x0, mu0, 'initialDegree', 'linear' , 'whichmodel', '
    subbasis', 'eta1', 0.01, 'eta2', 0.9, 'gamma1', 0.5, 'gamma2', 2.5,
    'epsilon', 1.0e-04)
```

After the interpolation set has been set up and the functions have been evaluated to build the models, `deft_funnel` calls the function `deft_funnel_main`, which implements the main body of the algorithm. The computation of the normal step is done through a call from `deft_funnel_main` to the function `lstr`, a Matlab implementation of the Moré-Sorensen algorithm from the GALAHAD package (Gould, Orban and Toint, 2003*b*). We show below the context where this function is called, which happens when the constraint violation is not sufficiently small. In our experiments with equality-constrained problems, we noticed that the algorithm yielded better results when it computed a normal step whenever the constraint violation was larger than the convergence threshold ϵ. This slight modification is included in the following piece of code.

```
% check if a normal step is needed by verifying the
% constraint violation at the current iterate
if (norm_cx <= epsilon)
       nstep = zeros(n, 1);
else
       % define the trust region radius of the subproblem
       n_max = min(Delta_c, kappa_n * norm_cx);
       % compute a normal step
       [ lstr_status, nstep ] = lstr(cx, J, n_max, verbosity_lstr);
end
norm_nstep = norm(nstep);
```

Lagrange multipliers are estimated by calling the function `lscg`, which implements a standard CG method for solving least-squares problems, while the tangent step is computed through the function `trprojcg` that implements the truncated projected CG method. Both `lscg` and `trprojcg` functions are called from within `deft_funnel_main`. In what follows, we show how the Lagrange multipliers and the tangent step are calculated after the computation of the normal step using these functions.

```
% Before attempting to compute a tangent step, check
% if there is enough space left in the trust region
```

```
% after the computation of the normal step
if (norm_nstep <= kappa_b * Delta)
        g_n = gfx + H * nstep;

        % estimate the vector 'mu' of Lagrange multipliers
        [ lscg_status, mu ] = lscg(g_n, J, verbosity_lscg);

        % define the projected gradient r_k
        r = g_n + J.' * mu;
        norm_gn = norm(g_n);

        % define the measure of optimality
        if (norm_r > 0.0)
                pi_measure = abs(g_n.' * r)/norm(r);
        else
                pi_measure = 0.0;
        end

        % compare the actual value of the measure of optimality
        % with that of infeasibility given by 'v'
        if (pi_measure > forcing(3, v))

                % consider the space left in the trust region after
                % walking along the normal step when defining the
                % trust region radius for the tangent step subproblem
                Delta_within = Delta - norm_nstep;

                % compute the tangent step
                [ trprojcg_status, tstep ] = trprojcg(g_n, H, J, ...
                                        Delta_within, verbosity_trprojcg);

                norm_tstep = norm(tstep);
                d = nstep + tstep;
        else
                norm_tstep = 0.0;
                d = nstep;
        end
else
        norm_tstep = 0.0;
        d = nstep;
end
```

4.1.6.5 Numerical results

We tested DEFT-FUNNEL on a set of problems from the CUTEst collection (Gould, Orban and Toint, 2003*a*). The first selected test set contains 29 small-scale equality-constrained optimization problems used by Colson (2004) in his numerical tests with the software CDFO, a derivative-free adaptation of

the Filter-SQP method. We compare here our results to those obtained with CDFO and those obtained with the software COBYLA by Powell (1994), a trust-region method for constrained problems that models the objective and constraint functions by linear interpolation. The criterion for comparison between the methods is solely based on the number of calls to the single subroutine that evaluates the objective function and the constraints at the same time at the required points.

Since the stopping criteria present in the three methods differ from each other, we decided to make two types of comparison between them. In the first one, we used the built-in convergence conditions of each algorithm and attempted to balance them by varying their parameters' values. In CDFO, convergence is declared when both conditions $\|\nabla\mathcal{L}(x_k)\| \leq 10^{-3}$ and $\|c(x_k)\| \leq 10^{-5}$ are satisfied, which are also the default values in the original code. Since we use the same threshold ϵ for verifying feasibility and optimality in DEFT-FUNNEL, we decided to pick up a value in between and thus we set $\epsilon = 10^{-4}$. However, we noticed that we obtained $\|c(x_k)\| \leq 10^{-5}$ or even better for most of the problems. In practice, DEFT-FUNNEL also terminates when the sample set is well poised — i.e., the error between the models and the real functions is sufficiently small — and either $\Delta_k \leq 10^{-7}$ or $\|s_k\| \leq 10^{-7}$ occurs. The stopping criterion used in COBYLA is based on the trust region radius ρ from the interval $[\rho_{\text{end}}, \rho_{\text{beg}}]$, where ρ_{beg} and ρ_{end} are constants predefined by the user. The parameter ρ is decreased by a constant factor during the execution of the algorithm and is never increased. The algorithm stops when $\rho = \rho_{\text{end}}$. Therefore, ρ_{end} should have the magnitude of the required accuracy in the final values of the variables. In our experiments, we set $\rho_{\text{end}} = 10^{-4}$.

As for the second type of comparison, much for benchmark purposes, we assume that the optimal objective function value f^* of each problem is known *a priori* and, thus, we declare convergence for a method at iteration k if and only if one has

$$|f(x_k) - f^*| \leq 10^{-4}\,|f(x_0) - f^*| \quad \text{and} \quad \|c(x_k)\| \leq 10^{-4}\,\|c(x_0)\|,$$

thereby providing a common criterion for optimality as well.

In DEFT-FUNNEL, we fixed the trust-region parameters to $\Delta_0 = 1$, $\eta_1 = 0.0001$, $\eta_2 = 0.9$, $\eta_3 = 0.5$, $\gamma_1 = 0.5$, $\gamma_2 = 2.5$ and $\Delta^{\max} = 10^{10}$. The parameter ζ used in the definition of the sets $\mathcal{F}_k$ and $\mathcal{C}_k$ of far points and close points, respectively, is set to $\zeta = 1$. For the limit number of times to reduce the trust regions sizes when a far or close interpolation point is replaced at unsuccessful iterations, we choose $\nu_f^{\max} = \nu_c^{\max} = 10$. We set $p_{\max} = (n+1)(n+2)/2$ for the subbasis, minimum Frobenius norm and minimum ℓ_2-norm approaches, and $p_{\max} = (n+1)(n+2)$ for the regression case. Finally, we set $\alpha = 0.1$, $\beta = 1$ and $\epsilon_0 = 0.01$ as the initial value for the loop in the criticality step. The limit of function evaluations imposed in our experiments was $300 \times n$, where n is the number of variables in the problem. Tables A.1 and A.2 in the appendix report the number of function evaluations required by the methods to solve each one of the problems.

We also present *performance profiles* of all the methods. Such profiles were introduced by Dolan and Moré (2002) as a manner to compare the performance of a set of solvers $\mathcal{S}$ on a test $\mathcal{P}$. Let n_s denote the number of solvers, and n_p, the number of problems. We are interested in using the number of function evaluations as a performance measure. For each problem p and solver s, we define

$t_{p,s} =$ number of function evaluations required to solve problem p by solver s.

We compare the performance on problem p by solver s with the best performance by any solver on this problem; that is, we use the *performance ratio*

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}.$$

If we define

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

then $\rho_s(\tau)$ is an approximation to the probability for solver $s \in \mathcal{S}$ that a performance ratio $r_{p,s}$ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio. The function ρ_s is the (cumulative) distribution function for the performance ratio.

The performance profiles for the first type of comparison are shown in Figures 4.2, 4.3 and 4.4. In the first one, the four different approaches to build the models in DEFT-FUNNEL are compared to each other, while in the latter two each of these approaches are individually compared to CDFO and COBYLA. The performance profiles for the second type of comparison, in turn, are shown in Figures 4.5 and 4.6, where each variant of DEFT-FUNNEL is individually compared to CDFO and COBYLA as well. We remind the reader here that COBYLA uses linear models only, which may be a disadvantage and might bias the comparison somewhat against this approach. Besides, it is implemented in single precision, while DEFT-FUNNEL is implemented in double precision.

As it can be seen in the performance profiles, the four variants of our method surpassed CDFO and COBYLA in the set of problems. Among these variants, the minimum ℓ_2-norm model variant outperformed the subbasis selection and also, somewhat surprisingly, the minimum Frobenius norm approach. Globally, the results obtained by DEFT-FUNNEL for equality-constrained problems are encouraging and motivated us to extend the method to problems with general nonlinear constraints, which is the subject of the next section.

4.2 Problems with general constraints

We now consider a trust-funnel method for the solution of the nonlinear optimization problem

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{s.t.:} & l^s \leq c(x) \leq u^s, \\ & l^x \leq x \leq u^x, \end{cases} \tag{4.2.1}$$

Figure 4.2: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models for the first type of comparison on a set of problems from the CUTEst collection.

where we assume that $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable, and that f is bounded below on the feasible domain. The vectors l^s and u^s are lower and upper bounds, respectively, on the constraints' values $c(x)$, while l^x and u^x are bounds on the x variables, with $l^s \in (\mathbb{R} \cup -\infty)^m$, $u^s \in (\mathbb{R} \cup \infty)^m$, $l^x \in (\mathbb{R} \cup -\infty)^n$ and $u^x \in (\mathbb{R} \cup \infty)^n$.

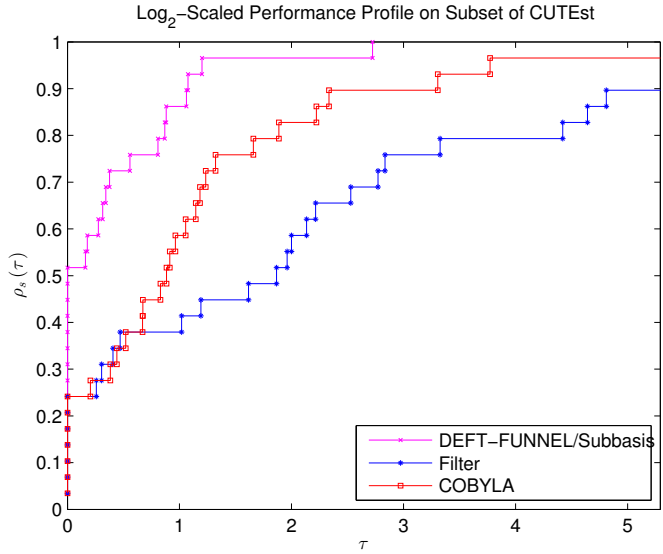
By defining $f(x, s) \stackrel{\text{def}}{=} f(x)$ and $c(x, s) \stackrel{\text{def}}{=} c(x) - s$, the problem above may be rewritten as the following equality-constrained optimization problem with simple bounds

$$\begin{cases} \min\limits_{(x,s)} & f(x, s) \\ \text{s.t.:} & c(x, s) = 0, \\ & l^s \leq s \leq u^s, \\ & l^x \leq x \leq u^x, \end{cases} \tag{4.2.2}$$

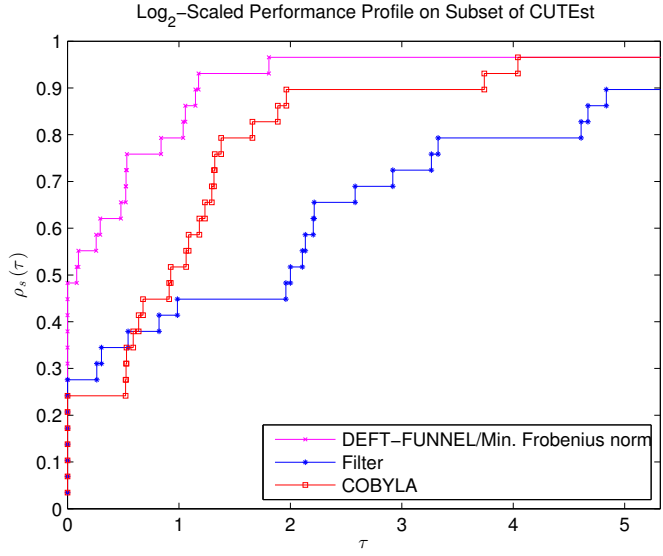
which is the one we will address.

We first present the method for the case where the derivatives can be used; then we apply it to derivative-free optimization problems using the techniques developed in the last section.

The final method described here features four main steps to solve problems with general nonlinear constraints, namely: a subspace minimization approach to handle the bounds on the x variables, which makes it an active-set method,

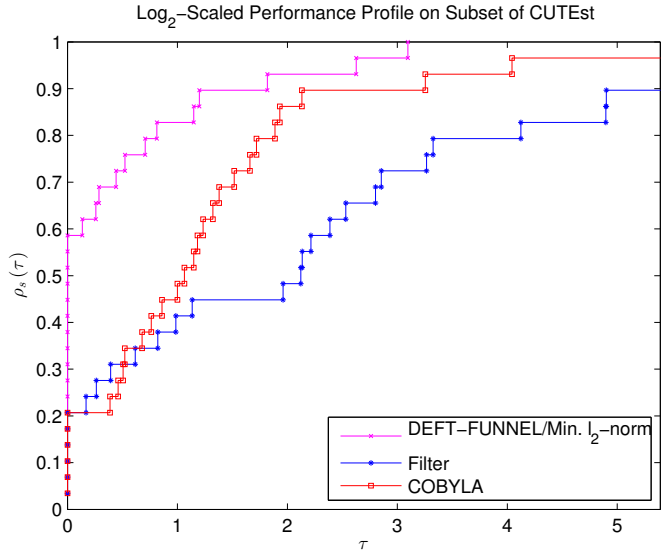


(a) Subbasis

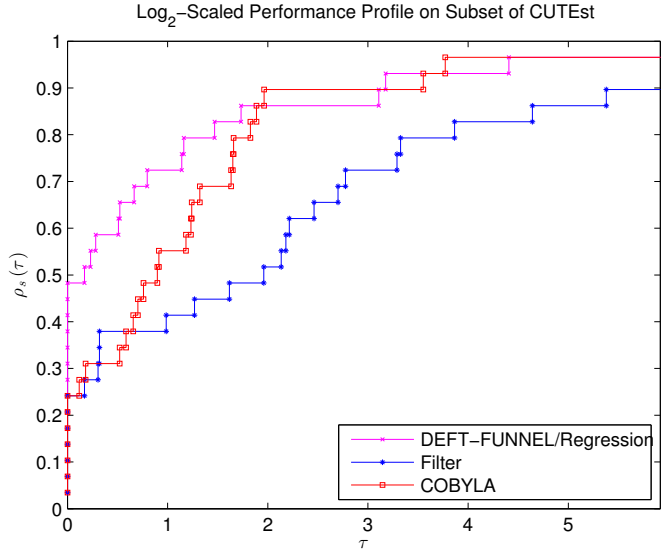


(b) Min. Frobenius norm

Figure 4.3: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the first type of comparison on a set of problems from the CUTEst collection.

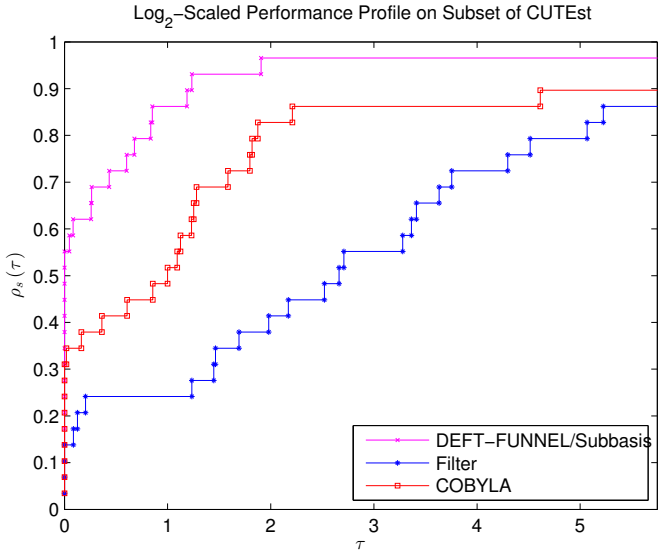


(a) Min. ℓ_2-norm

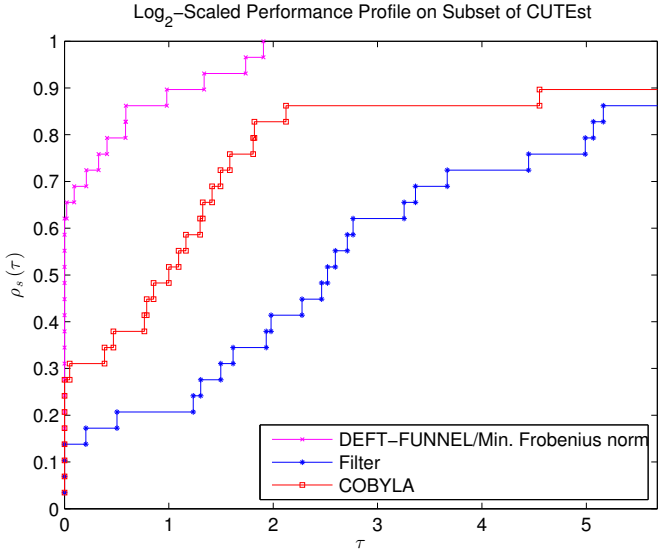


(b) Regression

Figure 4.4: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the first type of comparison on a set of problems from the CUTEst collection.

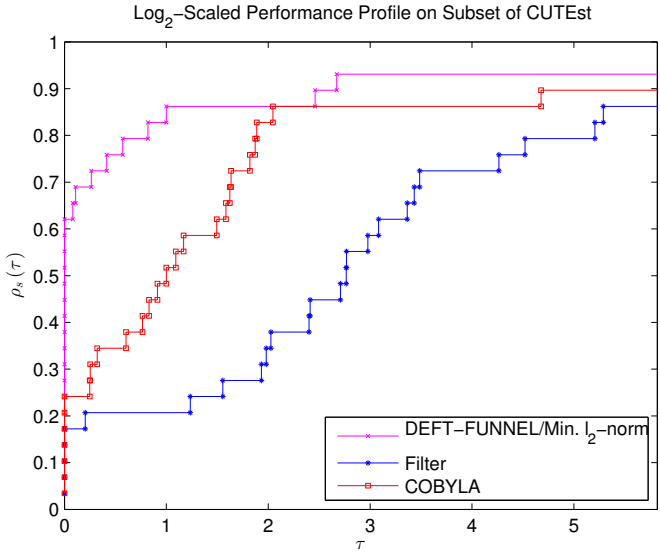


(a) Subbasis

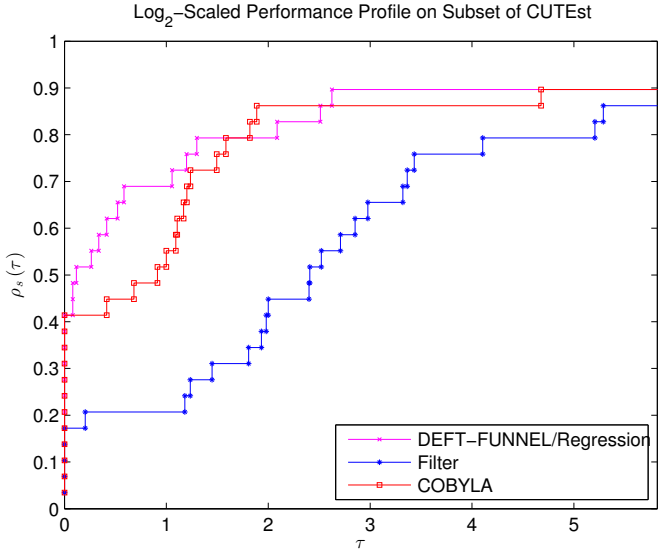


(b) Min. Frobenius norm

Figure 4.5: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the second type of comparison on a set of problems from the CUTEst collection.



(a) Min. ℓ_2-norm



(b) Regression

Figure 4.6: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the second type of comparison on a set of problems from the CUTEst collection.

a bounded linear least-squares solver to calculate the normal step, a projected gradient method to calculate the tangent step and the control of the permitted infeasibility of the iterates through the funnel bound. The reason behind the choice of exploring subspaces defined by the active bounds is dual. Besides the fact that we aim to avoid treating the bounds on the x variables as general inequality constraints, the reduction of the dimension of the problem after having identified the active bounds helps to thwart a possible degeneration of the interpolation set when the sample points become close to each other and thus affinely dependent, which happens often as the optimal solution is approached. The fact that the s variables play no role on the choice of the interpolation set vindicates the construction of the subspaces based upon the x variables only.

Our method generates a sequence of points $\{(x_k, s_k)\}$ such that, at each iteration k, the bound constraints below are satisfied

$$l^s \leq s_k \leq u^s, \tag{4.2.3}$$
$$l^x \leq x_k \leq u^x. \tag{4.2.4}$$

By using a composite-step approach, each trial step $d_k \stackrel{\text{def}}{=} (d_k^x, d_k^s)^T$ is decomposed as

$$d_k = \begin{pmatrix} d_k^x \\ d_k^s \end{pmatrix} = \begin{pmatrix} n_k^x \\ n_k^s \end{pmatrix} + \begin{pmatrix} t_k^x \\ t_k^s \end{pmatrix} = n_k + t_k,$$

where n_k is the normal step and t_k is the tangent step.

In the following subsections, we describe how each component is computed. Before that, we briefly explain how the subspace minimization is employed in our algorithm.

4.2.1 Subspace minimization

As in the method proposed by Gratton et al. (2011) for bound-constrained optimization problems, our algorithm makes use of an active-set approach where the minimization is restricted to subspaces defined by the active x variables.

At each iteration k, we define the subspace $\mathcal{S}_k$ as follows

$$\mathcal{S}_k \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n \mid [x]_i = [l^x]_i \;\text{ for }\; i \in \mathcal{L}_k \;\text{ and }\; [x]_i = [u^x]_i \;\text{ for }\; i \in \mathcal{U}_k\},$$

where $\mathcal{L}_k \stackrel{\text{def}}{=} \{i \mid [x_k]_i - [l^x]_i \leq \epsilon_b\}$ and $\mathcal{U}_k \stackrel{\text{def}}{=} \{i \mid [u^x]_i - [x_k]_i \leq \epsilon_b\}$ define the index sets of (nearly) active variables at their bounds, for some small constant $\epsilon_b > 0$ defined *a priori*. After that $\mathcal{S}_k$ has been defined, the minimization at iteration k is then restricted to the new subspace $\mathcal{S}_k$. Once a direction d_k for (x_k, s_k) has been computed, we set $(x_{k+1}, s_{k+1}) = (x_k, s_k) + d_k$ if k is a successful iteration; otherwise, we set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$.

If a solution $(\tilde{x}_k, \tilde{s}_k)$ for the subproblem defined by $\mathcal{S}_k$ satisfies the optimality conditions for the subproblem, we check whether it is also optimal for the

original problem (4.1.2). If it is, the solution encountered is returned to the user and the algorithm halts; otherwise, it proceeds in the full space by computing a new direction for $(\tilde{x}_k, \tilde{s}_k)$ and repeats the above process at iteration $k+1$ by defining $\mathcal{S}_{k+1}$.

4.2.2 The normal step

For any point (x, s), we measure the constraint violation by

$$v(x,s) \stackrel{\text{def}}{=} \tfrac{1}{2}\|c(x,s)\|^2. \tag{4.2.5}$$

At each iteration k, the constraint violation is bounded above by the funnel $v_k^{\max}$, i.e.,

$$v_k \leq v_k^{\max},$$

where $v_k \stackrel{\text{def}}{=} v(x_k, s_k)$.

If the constraint violation is significant, a normal step n_k can be computed by reducing the Gauss-Newton model of v at (x_k, s_k) within a trust region while care is taken to ensure that the conditions (4.2.3) and (4.2.4) are fulfilled.

As before, we add the following condition to ensure that the normal step is truly "normal":

$$\|n_k\|_\infty \leq \kappa_n \|c(x_k, s_k)\|, \tag{4.2.6}$$

for some $\kappa_n \geq 1$. We then perform the calculation of n_k by solving the bound-constrained linear least-squares problem

$$\left\{ \begin{array}{rc} \min\limits_{n=(n^x,n^s)} & \tfrac{1}{2}\|c(x_k,s_k) + J(x_k,s_k)n\|^2 \\ \text{s.t.:} & l^s \leq s_k + n^s \leq u^s, \\ & l^x \leq x_k + n^x \leq u^x, \\ & x_k + n^x \in \mathcal{S}_k, \\ & n \in \mathcal{N}_k, \end{array} \right. \tag{4.2.7}$$

where $J(x,s) \stackrel{\text{def}}{=} (J(x) \;\; -I_m)$ represents the Jacobian of $c(x,s)$ with respect to (x,s) and

$$\mathcal{N}_k \stackrel{\text{def}}{=} \{ z \in \mathbb{R}^{n+m} \mid \|z\|_\infty \leq \min[\, \Delta_k^c,\, \kappa_n \|c(x_k,s_k)\| \,] \}, \tag{4.2.8}$$

for some trust-region radius $\Delta_k^c > 0$.

Rather than solving (4.2.7) exactly, it suffices to compute a step n_k that produces a reduction in the linear part of the Gauss-Newton model of v at (x_k, s_k) which is at least a fraction of that achieved by the projected Cauchy direction. The following modified Cauchy condition then results from the projection procedure:

$$\begin{aligned} \delta_k^{c,n} &\stackrel{\text{def}}{=} \tfrac{1}{2}\|c(x_k,s_k)\|^2 - \tfrac{1}{2}\|c(x_k,s_k) + J(x_k,s_k)n_k\|^2 \\ &\geq \kappa_{nC}\pi_k^v \min\left[\frac{\pi_k^v}{1+\|W_k\|}, \Delta_k^c, 1 \right] \geq 0, \end{aligned} \tag{4.2.9}$$

where $W_k = J(x_k, s_k)^T J(x_k, s_k)$ is the symmetric Gauss-Newton approximation of the Hessian of v at (x_k, s_k), $\kappa_{nC} \in (0, \frac{1}{2}]$ and

$$\pi_k^v \stackrel{\text{def}}{=} -\langle J(x_k, s_k)^T c(x_k, s_k), b_k \rangle$$

is a v-criticality measure that indicates how much decrease could be obtained locally along the projection of the negative gradient of the Gauss-Newton model of v at (x_k, s_k) onto the region delimited by the bounds. The projected Cauchy direction b_k is, in turn, given by the solution of

$$\begin{cases} \min\limits_{b=(b^x,b^s)} & \langle J(x_k, s_k)^T c(x_k, s_k), b \rangle \\ \text{s.t.:} & l^s \leq s_k + b^s \leq u^s, \\ & l^x \leq x_k + b^x \leq u^x, \\ & x_k + b^x \in \mathcal{S}_k, \\ & \|b\|_\infty \leq 1. \end{cases} \tag{4.2.10}$$

In practice, DEFT-FUNNEL solves the problem (4.2.7) exactly by using a projected gradient method combined with an active-set approach, as it is further discussed in Section 4.2.8.

Finally, we call (x_k, s_k) an infeasible stationary point if $c(x_k, s_k) \neq 0$ and $\pi_k^v = 0$, in which case the algorithm terminates.

4.2.3 The tangent step

The SQP model for the function f is defined as

$$\psi_k((x_k, s_k) + d) \stackrel{\text{def}}{=} f_k + \langle g_k, d \rangle + \tfrac{1}{2}\langle d, B_k d \rangle, \tag{4.2.11}$$

where $f_k \stackrel{\text{def}}{=} f(x_k, s_k)$, $g_k \stackrel{\text{def}}{=} \nabla_{(x,s)} f(x_k, s_k)$, and B_k is the approximate Hessian of the Lagrangian function

$$\begin{aligned} \mathcal{L}(x, s, \mu, z^s, w^s, z^x, w^x) = f(x) + \langle \mu, c(x, s)) \rangle + \langle w^s, s - u^s \rangle + \langle z^s, l^s - s \rangle \\ + \langle w^x, x - u^x \rangle + \langle z^x, l^x - x \rangle \end{aligned} \tag{4.2.12}$$

with respect to (x, s), given by

$$B_k = \begin{pmatrix} G_k & 0 \\ 0 & 0 \end{pmatrix},$$

where G_k is a symmetric approximation of the Hessian of the Lagrangian with respect to x defined as

$$G_k \stackrel{\text{def}}{=} H_k + \sum_{i=1}^{m} [\hat{\mu}_k]_i C_{ik}, \tag{4.2.13}$$

z^s and w^s are the Lagrange multipliers associated to the lower and upper bounds, respectively, on the slack variables s, and z^x and w^x are the Lagrange multipliers associated to the lower and upper bounds on the x variables. In (4.2.13), H_k is a bounded symmetric approximation of $\nabla^2_{xx} f(x_k, s_k) = \nabla^2 f(x_k)$, the matrices C_{ik} are bounded symmetric approximations of the constraints' Hessians $\nabla^2_{xx} c_{ik}(x_k, s_k) = \nabla^2 c_{ik}(x_k)$ and the vector $\hat{\mu}_k$ may be viewed as a bounded local approximation of the Lagrange multipliers with respect to the equality constraints $c(x, s)$, in the sense that we require that

$$\|\hat{\mu}_k\| \leq \kappa_\mu, \tag{4.2.14}$$

for some $\kappa_\mu > 0$.

By using the decomposition $d_k = n_k + t_k$, we then have that

$$\psi_k((x_k, s_k) + n_k) = f_k + \langle g_k, n_k \rangle + \tfrac{1}{2}\langle n_k, B_k n_k \rangle \tag{4.2.15}$$

and

$$\begin{array}{rcl} \psi_k((x_k, s_k) + n_k + t) & = & f_k + \langle g_k, n_k + t \rangle + \frac{1}{2}\langle n_k + t, B_k\,(n_k + t) \rangle \\ & = & \psi_k((x_k, s_k) + n_k) + \langle g_k^N, t \rangle + \frac{1}{2}\langle t, B_k t \rangle, \end{array} \tag{4.2.16}$$

where

$$g_k^N \stackrel{\rm def}{=} g_k + B_k\, n_k. \tag{4.2.17}$$

To make sure that (4.2.16) approximates the function $f((x_k, s_k)+n_k+t)$ locally well, we ask that the complete step $d = n_k + t$ must belong to

$$\mathcal{T}_k \stackrel{\rm def}{=} \{d \in \mathbb{R}^{n+m} \mid \|d\|_\infty \leq \Delta_k^f\}, \tag{4.2.18}$$

for some radius Δ_k^f. The minimization of (4.2.16) should then be restricted to the intersection of $\mathcal{N}_k$ and $\mathcal{T}_k$, which imposes that the *tangent step* t_k results in a complete step $d_k = n_k + t_k$ that satisfies the inclusion

$$d_k \in \mathcal{R}_k \stackrel{\rm def}{=} \mathcal{N}_k \cap \mathcal{T}_k \stackrel{\rm def}{=} \{d \in \mathbb{R}^{n+m} \mid \|d\|_\infty \leq \Delta_k\}, \tag{4.2.19}$$

where the radius Δ_k of $\mathcal{R}_k$ is thus given by

$$\Delta_k = \min[\Delta_k^c, \Delta_k^f]. \tag{4.2.20}$$

Similarly to the computation of the normal step, we intend to remain in the subspace $\mathcal{S}_k$ after walking along the tangent direction. We accomplish that by imposing the following condition

$$x_k + n_k^x + t^x \in \mathcal{S}_k.$$

Additionally, the conditions (4.2.3) and (4.2.4) must be satisfied at the final point $(x_k, s_k) + d_k$, i.e., we must have

$$\begin{array}{c} l^s \leq s_k + n_k^s + t^s \leq u^s, \\ l^x \leq x_k + n_k^x + t^x \leq u^x. \end{array}$$

Before calculating the tangent step, we verify if there is still enough space left to move within the region $\mathcal{R}_k$ after the computation of the normal step n_k by checking the condition

$$\|n_k\|_\infty \leq \kappa_{\mathcal{R}} \Delta_k, \tag{4.2.21}$$

for some $\kappa_{\mathcal{R}} \in (0,1)$. If (4.2.21) holds, we compute the tangent step by (approximately) solving the problem

$$\begin{cases} \min\limits_{t=(t^x,t^s)} & \langle g_k^N, t\rangle + \frac{1}{2}\langle t, B_k t\rangle \\ \text{s.t.:} & J(x_k,s_k)t = 0, \\ & l^s \leq s_k + n_k^s + t^s \leq u^s, \\ & l^x \leq x_k + n_k^x + t^x \leq u^x, \\ & x_k + n_k^x + t^x \in \mathcal{S}_k. \\ & n_k + t \in \mathcal{R}_k. \end{cases} \tag{4.2.22}$$

Although we use a projected gradient method in our code to solve the problem (4.2.22) exactly, which is justified by the small size of the problems tested, it suffices to compute a step t_k that produces a reduction in the model ψ_k which is at least a fraction of that achieved by solving the modified Cauchy point subproblem

$$\min_{\substack{\tau>0 \\ (x_k,s_k)+n_k+\tau r_k \in \mathcal{R}_k}} \psi_k((x_k,s_k)+n_k+\tau r_k), \tag{4.2.23}$$

where r_k is the projected Cauchy direction obtained by solving the linear optimization problem

$$\begin{cases} \min\limits_{r=(r^x,r^s)} & \langle g_k^N, r\rangle \\ \text{s.t.:} & J(x_k,s_k)r = 0, \\ & l^s \leq s_k + n_k^s + r^s \leq u^s, \\ & l^x \leq x_k + n_k^x + r^x \leq u^x, \\ & x_k + n_k^x + r^x \in \mathcal{S}_k. \\ & \|r\|_\infty \leq 1. \end{cases} \tag{4.2.24}$$

We then define our f-criticality measure as

$$\pi_k^f \stackrel{\text{def}}{=} -\langle g_k^N, r_k\rangle. \tag{4.2.25}$$

By definition, π_k^f measures how much decrease could be obtained locally along the projection of the negative of the approximate gradient g_k^N onto the null space of $J(x_k,s_k)$ intersected to the region delimited by the bounds. This procedure ensures, for some $\kappa_{tC} \in (0,1]$, the modified Cauchy condition

$$\begin{aligned} \delta_k^{f,t} &\stackrel{\text{def}}{=} \psi_k((x_k,s_k)+n_k) - \psi_k((x_k,s_k)+n_k+t_k) \\ &\geq \kappa_{tC}\pi_k^f \min\left[\frac{\pi_k^f}{1+\|B_k\|}, \Delta_k, 1\right] > 0. \end{aligned} \tag{4.2.26}$$

A new local estimate of the Lagrange multipliers $(\mu_k, z_k^s, w_k^s, z_k^x, w_k^x)$ are computed by solving the following bound-constrained linear least-squares problem

$$\left\{ \begin{array}{rc} \min\limits_{(\mu,\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x)} & \frac{1}{2}\|\mathcal{M}_k(\mu,\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x)\|^2. \\ \text{s.t.:} & \hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x \geq 0, \end{array} \right. \tag{4.2.27}$$

where

$$\begin{aligned} \mathcal{M}_k(\mu,\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x) \stackrel{\text{def}}{=} & \begin{pmatrix} g_k^N \\ 0 \end{pmatrix} + \begin{pmatrix} J(x_k)^T \\ -I_m \end{pmatrix} \mu + \begin{pmatrix} 0 \\ I_w^s \end{pmatrix} \hat{w}^s \\ & + \begin{pmatrix} 0 \\ -I_z^s \end{pmatrix} \hat{z}^s + \begin{pmatrix} I_w^x \\ 0 \end{pmatrix} \hat{w}^x + \begin{pmatrix} -I_z^x \\ 0 \end{pmatrix} \hat{z}^x, \end{aligned}$$

the matrices I_z^s and I_w^s are obtained from I_m by removing the columns whose indices are not associated to any active (lower and upper, respectively) bound at $s_k+n_k^s$, the matrices I_z^x and I_w^x are obtained from I_n by removing the columns whose indices are not associated to any active (lower and upper, respectively) bound at $x_k + n_k^x$, and the Lagrange multipliers $(\hat{z}^s, \hat{w}^s, \hat{z}^x, \hat{w}^x)$ are those in (z^s, w^s, z^x, w^x) associated to active bounds at $s_k + n_k^s$ and $x_k + n_k^x$. All the other Lagrange multipliers are set to zero.

4.2.4 Which steps to compute and retain

The normal step is computed when $k = 0$ or the current violation is relatively substantial, which is verified by the following conditions

$$\|c(x_k, s_k)\| > \omega_n(\pi_{k-1}^f) \text{ or } v_k > \kappa_{vv} v_k^{\max}. \tag{4.2.28}$$

If (4.2.28) fails, the computation of a normal step is unnecessary, which allows us to set $n_k = 0$.

If the solution of (4.2.24) is $r_k = 0$, then by (4.2.25) we have $\pi_k^f = 0$. In this case, the computation of the tangent step is skipped, and we simply set $t_k = 0$. If π_k^f is unsubstantial compared to the current infeasibility, i.e., the condition

$$\pi_k^f > \omega_t(\|c(x_k, s_k)\|) \tag{4.2.29}$$

fails, we set $t_k = 0$.

As in the case for equality-constrained problems, we verify whether the conditions

$$\|t_k\| > \kappa_{\mathcal{CS}} \|n_k\| \tag{4.2.30}$$

and

$$\delta_k^f \stackrel{\text{def}}{=} \delta_k^{f,t} + \delta_k^{f,n} \geq \kappa_\delta \delta_k^{f,t}, \tag{4.2.31}$$

where

$$\delta_k^{f,n} \stackrel{\text{def}}{=} \psi_k(x_k, s_k) - \psi_k((x_k, s_k) + n_k), \tag{4.2.32}$$

are satisfied for some $\kappa_{\mathcal{CS}} > 1$ and for $\kappa_\delta \in (0, 1)$. If (4.2.30) holds but (4.2.31) fails, the tangent step plays no major role as the predicted decrease on the objective function's model is not significant compared to its possible increase due to the normal step. In this case, we merely reset $t_k = 0$.

4.2.5 Iterations types

Once we have computed the step d_k, we define the trial point as

$$(x_k^+, s_k^+) \stackrel{\text{def}}{=} (x_k, s_k) + d_k. \tag{4.2.33}$$

If $n_k = t_k = 0$, then iteration k is said to be a μ-iteration. If $t_k \neq 0$, (4.2.31) holds, and

$$v_k^+ \leq v_k^{\max}, \tag{4.2.34}$$

where $v_k^+ \stackrel{\text{def}}{=} v(x_k^+, s_k^+)$, we call iteration k an f-iteration. If iteration k is neither a μ-iteration nor an f-iteration, then it is called a c-iteration.

The acceptation of the trial point and the trust regions management follow the same ideas of the previous version of the method and thus are based on the major expected achievement of the iteration.

- If iteration k is a μ-iteration, we restart with $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ using the new multipliers. We then define

  $$\Delta_{k+1}^f = \Delta_k^f \text{ and } \Delta_{k+1}^c = \Delta_k^c \tag{4.2.35}$$

  and keep the current value of the maximal infeasibility $v_{k+1}^{\max} = v_k^{\max}$.

- If iteration k is an f-iteration, we accept the trial point (i.e., $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$) if

  $$\rho_k^f \stackrel{\text{def}}{=} \frac{f(x_k, s_k) - f(x_k^+, s_k^+)}{\delta_k^f} \geq \eta_1, \tag{4.2.36}$$

  and reject it (i.e., $(x_{k+1}, s_{k+1}) = (x_k, s_k)$), otherwise.

  The radius of $\mathcal{T}_k$ is then updated by

  $$\Delta_{k+1}^f = \begin{cases} \min\left[\max[\gamma_2\|d_k\|, \Delta_k^f], \Delta^{\max}\right] & \text{if } \rho_k^f \geq \eta_2, \\ \Delta_k^f & \text{if } \rho_k^f \in [\eta_1, \eta_2), \\ \gamma_1\|d_k\| & \text{if } \rho_k^f < \eta_1, \end{cases} \tag{4.2.37}$$

where the constants $\Delta^{\max}, \eta_1, \eta_2, \gamma_1$, and γ_2 are given and satisfy the conditions

$$\Delta^{\max} > 1, \quad 0 < \eta_1 \leq \eta_2 < 1 \quad \text{and} \quad 0 < \gamma_1 < 1 < \gamma_2.$$

The radius of $\mathcal{N}_k$, in turn, is updated by

$$\Delta_{k+1}^c = \begin{cases} \min\left[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}\right] & \text{if } \rho_k^f \geq \eta_1 \text{ and} \\ & \quad v_k^+ < \eta_3\, v_k^{\max}, \\ \Delta_k^c & \text{if } \rho_k^f < \eta_1, \end{cases} \tag{4.2.38}$$

for some constant $\eta_3 > 0$.

The value of the maximal infeasibility measure is left unchanged, that is, $v_{k+1}^{\max} = v_k^{\max}$. Note that $\delta_k^f > 0$ (because of (4.2.26) and (4.2.31)) unless (x_k, s_k) is first-order critical, and hence that condition (4.2.36) is well-defined.

- If iteration k is a c-iteration, we accept the trial point if the improvement in feasibility is comparable to its predicted value

$$\delta_k^c \stackrel{\text{def}}{=} \tfrac{1}{2}\|c(x_k, s_k)\|^2 - \tfrac{1}{2}\|c(x_k, s_k) + J(x_k, s_k)d_k\|^2,$$

and the latter is itself comparable to its predicted decrease along the normal step, that is,

$$n_k \neq 0, \quad \delta_k^c \geq \kappa_{cn}\delta_k^{c,n} \text{ and } \rho_k^c \stackrel{\text{def}}{=} \frac{v_k - v_k^+}{\delta_k^c} \geq \eta_1, \tag{4.2.39}$$

for some $\kappa_{cn} \in (0, 1 - \kappa_{tg}]$. If (4.2.39) fails, the trial point is rejected.

The radius of $\mathcal{N}_k$ is then updated by setting Δ_{k+1}^c to

$$\begin{cases} \min\left[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}\right] & \text{if } \rho_k^c \geq \eta_2 \text{ and } \delta_k^c \geq \kappa_{\text{cn}}\delta_k^{c,n}, \\ \Delta_k^c & \text{if } \rho_k^c \in [\eta_1, \eta_2) \text{ and } \delta_k^c \geq \kappa_{\text{cn}}\delta_k^{c,n}, \\ \gamma_1 \|n_k\| & \text{if } \rho_k^c < \eta_1 \text{ or } \delta_k^c < \kappa_{\text{cn}}\delta_k^{c,n}, \\ & \quad \text{and } \|n_k\| \neq 0, \\ \gamma_1 \Delta_k^c & \text{if } \rho_k^c < \eta_1 \text{ or } \delta_k^c < \kappa_{\text{cn}}\delta_k^{c,n}, \\ & \quad \text{and } \|n_k\| = 0, \end{cases} \tag{4.2.40}$$

while that of $\mathcal{T}_k$ is left unchanged ($\Delta_{k+1}^f = \Delta_k^f$). Notice that the last case in (4.2.40) is possible in a c-iteration due to the fact that one might have $\|d_k\| \neq 0$, with $\|n_k\| = 0$ and $\|t_k\| \neq 0$, and the condition (4.2.34) does not hold, which makes the iteration k of type c rather than of f.

We update the value of the maximal infeasibility by

$$v_{k+1}^{\max} = \begin{cases} \max\left[\kappa_{tx1} v_k^{\max}, v_k^+ + \kappa_{tx2}(v_k - v_k^+)\right] & \text{if (4.2.39) hold,} \\ v_k^{\max} & \text{otherwise,} \end{cases} \tag{4.2.41}$$

for some $\kappa_{tx1} \in (0, 1)$ and $\kappa_{tx2} \in (0, 1)$.

4.2.6 Application to derivative-free optimization

In a similar way to what we have done with the original trust-funnel method for equality-constrained problems, we now adapt the general method to the case where the derivatives are not available. Essentially, we modify the DEFT-FUNNEL method as needed to make it able to handle simple bounds as well. Although it might seem a simple additional task, the treatment of the bounds unfurls new implications for the management of the geometry of the sample set. As we mentioned before, there might be a possible degeneration of the interpolation set when the sample points become close to each other and thus affinely dependent, which happens often as the optimal solution is approached. A possible scenario is illustrated in Figure 4.7, where the bound constraint $[x]_2 \geq 0$ is active at the solution. Eventually, the sets of active bounds of the iterates $\{x_k\}$ and the solution x^* become identical, causing affine dependence between the points sufficiently close to the solution. Since the iterates are added into the interpolation set as the algorithm progresses, the quality of the geometry of the sample set decays as a result of the affine dependence, impoverishing the interpolation models.

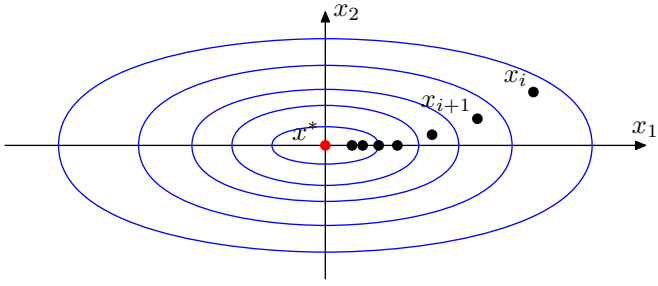


Figure 4.7: Illustration of a scenario where the interpolation set becomes degenerated as the optimal solution is approached. In this example, we consider a 2-dimensional problem with the bound constraint $[x]_2 \geq 0$, which is active at the solution x^* and at the iterates close to it.

In an attempt to diminish the chances of degeneration, we can reduce the dimension of the problem by working on the subspace defined by the active bounds. As we explain next, this is done in DEFT-FUNNEL by means of recursion.

4.2.6.1 Recursive call in subspaces

In our algorithm, we apply the recursive approach found in the derivative-free method proposed by Gratton et al. (2011) for bound-constrained optimization problems. Once the subspace $\mathcal{S}_k$ has been defined at iteration k, the algorithm calls itself recursively and the dimension of the problem is then reduced to $\hat{n} = n - |\mathcal{L}_k \cup \mathcal{U}_k|$, where n denotes here the dimension of $\mathbb{R}^n$. The recursive

call considers only the $\hat{n}$ indices of non-active bounds of each point involved in the procedure, which means that now we have $x_k \in \mathbb{R}^{\hat{n}}$. A new well-poised interpolation set $\mathcal{Z}_k \subset \mathbb{R}^{\hat{n}}$ is then constructed from a suitable choice of points in $\mathcal{X}_k \cap \mathcal{S}_k$, where $\mathcal{X}_k$ is the set of all points obtained up to iteration k, as we explain now. In order to save function evaluations in the building process of the new interpolation set, all the points in $\mathcal{X}_k$ that are nearly but not in $\mathcal{S}_k$ are projected onto $\mathcal{S}_k$ and used to build $\mathcal{Z}_k$ with their model values instead of their real function values. In a more formal description, we define the set of the points that are close to (but not on) the active bounds at x_k as

$$\mathcal{A}_k \stackrel{\text{def}}{=} \left\{ y \in \mathcal{X}_k \;\middle|\; \begin{array}{c} 0 \leq |\, [y]_i - [l^x]_i \,| \leq \epsilon_b \;\; \text{for} \;\; i \in \mathcal{L}_k \;\; \text{and} \\ 0 \leq |\, [u^x]_i - [y]_i \,| \leq \epsilon_b \;\; \text{for} \;\; i \in \mathcal{U}_k \end{array} \right\},$$

where, for at least one i, the strict inequality

$$0 < |\, [y]_i - [l^x]_i \,|, \; i \in \mathcal{L}_k,$$

or

$$0 < |\, [u^x]_i - [y]_i \,|, \; i \in \mathcal{U}_k,$$

must hold. We then project all the points $y \in \mathcal{A}_k$ onto $\mathcal{S}_k$, obtaining new "dummy" points y_s that are added to $\mathcal{X}_k$ with associated values $m_k^f(y_s)$ and $m_k^c(y_s)$ rather than the values of the original functions. These dummy points are progressively replaced by other points with true function values with high priority during the minimization in $\mathcal{S}_k$.

Convergence in a subspace is only declared if the interpolation set contains no dummy points. If a solution has been found for a subspace and there are still dummy points in the interpolation set, evaluations of the original functions $f(x)$ and $c(x)$ at such points are carried out and the interpolating models are recomputed from the original function values. Once convergence has been declared in a subspace $\mathcal{S}_k$, the $|\mathcal{L}_k \cup \mathcal{U}_k|$ fixed components $[x]_i$ associated with the active bounds and the component x of the approximate solution found in $\mathcal{S}_k$ of dimension $\hat{n} = n - |\mathcal{L}_k \cup \mathcal{U}_k|$ are assembled to compose a full-dimensional vector $x^*_{\mathcal{S}}$ in $\mathbb{R}^n$. The algorithm then checks whether $(x^*_{\mathcal{S}}, s^*_{\mathcal{S}})$ is optimal for the full-dimensional problem or not. Firstly, a full-space interpolation set of degree $n+1$ is built in an ϵ-neighborhood around the point $x^*_{\mathcal{S}}$. Subsequently, the corresponding interpolating models m_k^f and m_k^c are recomputed and the f-criticality measure π_{k-1}^f is calculated anew using information of the updated models. Finally, the criticality step in the full space is then entered.

4.2.7 The algorithm

We now provide a formal description of our complete algorithm for solving nonlinear optimization problems with general nonlinear constraints without using derivatives.

Algorithm 4.2.1: DEFT-FUNNEL($\mathcal{S}$, $\mathcal{X}$, $\mathcal{Y}$, (x, s), Δ^f, Δ^c, $v^{\max}$)

Step 0: Initialization. An initial accuracy threshold ϵ_0, an initial vector of multipliers μ_{-1} and positive initial trust-region radii Δ_0^f and Δ_0^c are given, as well as the constants

$$\alpha \in (0,1), \quad 0 < \gamma_1 < 1 < \gamma_2, \quad \zeta \geq 1, \quad 0 < \eta_1 < \eta_2 < 1 \text{ and } \beta, \eta_3 > 0.$$

An initial set of interpolation points is also given, $\mathcal{Y}_0$, with $x_0 \in \mathcal{Y}_0 \subset \mathcal{B}(x_0; \Delta_0)$ and $|\mathcal{Y}_0| \geq n+1$, as well as the maximum number of interpolation points $p_{\max} \geq |\mathcal{Y}_0|$ in $\mathcal{Y}_k$ at the end. Let p_k denote the cardinality of $\mathcal{Y}_k$. This interpolation set defines interpolation models m_0^f and m_0^c around x_0 and associated Lagrange polynomials $\{\ell_{0,j}\}_{j=0}^p$. Define $\Delta_0 = \min[\Delta_0^f, \Delta_0^c] \leq \Delta^{\max}$, and $v_0^{\max} = \max[\kappa_{ca}, \kappa_{cr} v(x_0, s_0)]$ for some constants $\kappa_{ca} > 0$ and $\kappa_{cr} > 1$. Compute r_{-1} by solving (4.2.24) with normal step $n_{-1} = 0$ and define π_{-1}^f as in (4.2.25). Define $\nu_f^{\max} > 0$ and $\nu_c^{\max} > 0$, the maximum number of times that the tangential and normal trust regions sizes can be reduced when an interpolation point is replaced at unsuccessful iterations. Initialize the corresponding counters $\nu_f = \nu_c = 0$. Define $k = 0$ and $i = 0$.

Step 1: Subspace minimization. Check for (nearly) active bounds at x_k and define $\mathcal{S}_k$.

Step 1.1: If there is no (nearly) active bound or if $\mathcal{S}_k$ has already been explored, go to Step 1.6. If all bounds are active, go to Step 1.5.

Step 1.2: Project points in $\mathcal{X}_k$ which lie close to the (nearly) active bounds on $\mathcal{S}_k$ and associate with them suitable function values estimates.

Step 1.3: Build a new interpolation set $\mathcal{Z}_k$ in $\mathcal{S}_k$ including the projected points, if any.

Step 1.4: Call recursively DEFT-FUNNEL($\mathcal{S}_k$, $\mathcal{X}_k$, $\mathcal{Z}_k$, (x_k, s_k), Δ_k^f, Δ_k^c, $v_k^{\max}$) and let $(x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$ be the solution of the subspace problem after adding the fixed components.

Step 1.5: If $dim(\mathcal{S}_k) < n$ (where n denotes here the dimension of $\mathbb{R}^n$), return $(x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$. Otherwise, reset $(x_k, s_k) = (x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$, construct a new interpolation set $\mathcal{Y}_k$ around x_k, build the corresponding models m_k^f and m_k^c and recompute π_{k-1}^f using information of the new models.

Step 1.6: If $\mathcal{S}_k$ has already been explored, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, reduce the trust regions radii $\Delta_{k+1}^f = \gamma_1 \Delta_k^f$ and $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$, set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^c]$ and build a new poised set $\mathcal{Y}_{k+1}$ in

$\mathcal{B}(x_{k+1}; \Delta_{k+1})$. Compute models m_{k+1}^f and m_{k+1}^c and increment k by one.

Step 2: Criticality step. Define $\hat{m}_i^f = m_k^f$, $\hat{m}_i^c = m_k^c$ and $\hat{\pi}_i^f = \pi_{k-1}^f$.

Step 2.1: If $\|c(x_k, s_k)\| \leq \epsilon_i$ and $\hat{\pi}_i^f \leq \epsilon_i$, set $\epsilon_{i+1} = \max[\alpha\|c(x_k, s_k)\|, \alpha\hat{\pi}_i^f, \epsilon]$ and modify $\mathcal{Y}_k$ as needed to ensure it is Λ-poised in $\mathcal{B}(x_k, \epsilon_{i+1})$. If $\mathcal{Y}_k$ was modified, compute new models $\hat{m}_i^f$ and $\hat{m}_i^c$, calculate $\hat{r}_i$ and $\hat{\pi}_i^f$ associated to these models and increment i by one. If $\|c(x_k, s_k)\| \leq \epsilon$ and $\hat{\pi}_i^f \leq \epsilon$, return (x_k, s_k); otherwise, start Step 2.1 again.

Step 2.2: Set $m_k^f = \hat{m}_i^f$, $m_k^c = \hat{m}_i^c$ and $\pi_{k-1}^f = \hat{\pi}_i^f$. Update $\Delta_k = \beta \max[\|c(x_k, s_k)\|, \pi_{k-1}^f]$ and define $v_i = x_k$ if a new model has been computed.

Step 3: Normal step. If $c(x_k, s_k) \neq 0$ and $\pi_k^v = 0$, STOP (infeasible stationary point). Otherwise, compute a normal step n_k by solving the problem (4.2.7). This computation must be performed if $k = 0$ or if (4.2.28) holds when $k > 0$. If n_k has not been computed, set $n_k = 0$.

Step 4: Tangent step. If (4.2.21) holds, then

Step 4.1: select a vector $\hat{\mu}_k$ satisfying (4.2.14) and define G_k as in (4.1.8) to obtain B_k;

Step 4.2: compute μ_k by solving (4.2.27);

Step 4.3: compute the modified Cauchy direction r_k by solving (4.2.24) and define π_k^f as (4.2.25);

Step 4.4: if (4.2.29) holds, compute a tangent step t_k by solving (4.2.22).

If (4.2.21) fails, set $\mu_k = \mu_{k-1}$. In this case, or if (4.2.29) fails, or if (4.2.30) holds but (4.2.31) fails, set $t_k = 0$ and $d_k = n_k$. In all cases, define $(x_k^+, s_k^+) = (x_k, s_k) + d_k$.

Step 5: Conclude a μ-iteration. If $n_k = t_k = 0$, then

Step 5.1: set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$;

Step 5.2: define $\Delta_{k+1}^f = \Delta_k^f$ and $\Delta_{k+1}^c = \Delta_k^c$;

Step 5.3: set $v_{k+1}^{\max} = v_k^{\max}$, $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^c]$ and $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

Step 6: Conclude an f-iteration. If $t_k \neq 0$ and (4.2.31) and (4.2.34) hold,

Step 6.1: Augment the interpolation set. If $p_k < p_{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$.

- If $\rho_k^f \geq \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$ and $\nu_f = 0$.
  If $\rho_k^f \geq \eta_2$, set $\Delta_{k+1}^f = \min[\max[\gamma_2 \|d_k\|, \Delta_k^f], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$.
  If $v(x_k^+, s_k^+) < \eta_3\, v_k^{\max}$, set $\Delta_{k+1}^c = \min[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.
- If $\rho_k^f < \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta_{k+1}^c = \Delta_k^c$.
  If $\nu_f \leq \nu_f^{\max}$, set $\Delta_{k+1}^f = \gamma_1 \Delta_k^f$ and $\nu_f = \nu_f + 1$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$.

Step 6.2: Successful iteration. If $p_k = p_{\max}$ and $\rho_k^f \geq \eta_1$, then set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.2.42}$$

Set $\nu_f = 0$. If $\rho_k^f \geq \eta_2$, set $\Delta_{k+1}^f = \min[\max[\gamma_2 \|d_k\|, \Delta_k^f], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$. If $v(x_k^+, s_k^+) < \eta_3\, v_k^{\max}$, set $\Delta_{k+1}^c = \min[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.

Step 6.3: Replace a far interpolation point. If $p_k = p_{\max}$, $\rho_k^f < \eta_1$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| > \zeta\Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\}$$

is non-empty, then define $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, and set $\Delta_{k+1}^f = \gamma_1 \|d_k\|$ if $\nu_f \leq \nu_f^{\max}$ or $\Delta_{k+1}^f = \Delta_k^f$ otherwise.
Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.2.43}$$

If $\nu_f \leq \nu_f^{\max}$, update $\nu_f = \nu_f + 1$.

Step 6.4: Replace a close interpolation point. If $p_k = p_{\max}$, $\rho_k^f < \eta_1$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| \leq \zeta\Delta \text{ and } |\ell_{k,j}(x_k^+)| > \lambda\}$$

is non-empty, then define $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and set $\Delta_{k+1}^f = \gamma_1 \|d_k\|$ if $\nu_f \leq \nu_f^{\max}$ or $\Delta_{k+1}^f = \Delta_k^f$ otherwise.
Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.2.44}$$

If $\nu_f \leq \nu_f^{\max}$, update $\nu_f = \nu_f + 1$.

Step 6.5: Reduce the trust-region radius. If $p_k = p_{\max}$, $\rho_k^f < \eta_1$ and either $x_k = v_i$ and $\Delta_k^f > \epsilon_i$ or $\mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, $\Delta_{k+1}^c = \Delta_k^c$, $\Delta_{k+1}^f = \gamma_1 \|d_k\|$ and $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

Step 6.6: Update the combined radius. Set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^c]$ and $v_{k+1}^{\max} = v_k^{\max}$.

Step 7: Conclude a c-iteration. If either $n_k \neq 0$ and $t_k = 0$, or either one of (4.2.31) or (4.2.34) fails,

Step 7.1: Augment the interpolation set. If $p_k < p_{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$.

- If $\rho_k^c \geq \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$, $\Delta_{k+1}^f = \Delta_k^f$ and $\nu_c = 0$. If $\rho_k^c \geq \eta_2$, set $\Delta_{k+1}^c = \min[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.
- If $\rho_k^c < \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta_{k+1}^f = \Delta_k^f$. If $\nu_c \leq \nu_c^{\max}$, then set $\Delta_{k+1}^c = \gamma_1 \|n_k\|$ if $\|n_k\| \neq 0$ and $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$ otherwise ($\|n_k\| = 0$). Update $\nu_c = \nu_c + 1$. If $\nu_c > \nu_c^{\max}$, set $\Delta_{k+1}^c = \Delta_k^c$.

Step 7.2: Successful iteration. If $p_k = p_{\max}$, (4.2.39) holds, then set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.2.45}$$

Set $\Delta_{k+1}^f = \Delta_k^f$ and $\nu_c = 0$. Set $\Delta_{k+1}^c = \min[\max[\gamma_2 \|n_k\|, \Delta_k^c], \Delta^{\max}]$ if $\rho_k^c \geq \eta_2$ or $\Delta_{k+1}^c = \Delta_k^c$ otherwise.

Step 7.3: Replace a far interpolation point. If $p_k = p_{\max}$, (4.2.39) fails, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k \mid > \zeta \Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\}$$

is non-empty, then define $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and set $\Delta_{k+1}^f = \Delta_k^f$. If $\nu_c \leq \nu_c^{\max}$, then set $\Delta_{k+1}^c = \gamma_1 \|n_k\|$ if $\|n_k\| \neq 0$, or $\Delta_{k+1}^c = \gamma_1 \Delta_k^c$ otherwise ($\|n_k\| = 0$). If $\nu_c > \nu_c^{\max}$, set $\Delta_{k+1}^c = \Delta_k^c$. Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.2.46}$$

If $\nu_c \leq \nu_c^{\max}$, update $\nu_c = \nu_c + 1$.

Step 7.4: Replace a close interpolation point. If $p_k = p_{\max}$, (4.2.39) fails, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| \leq \zeta \Delta \text{ and } |\ell_{k,j}(x_k^+)| > \lambda\}$$

is non-empty, then set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta^f_{k+1} = \Delta^f_k$. If $\nu_c \leq \nu_c^{\max}$, then set $\Delta^c_{k+1} = \gamma_1 \|n_k\|$ if $\|n_k\| \neq 0$, or $\Delta^c_{k+1} = \gamma_1 \Delta^c_k$ otherwise ($\|n_k\| = 0$). If $\nu_c > \nu_c^{\max}$, set $\Delta^c_{k+1} = \Delta^c_k$.

Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{4.2.47}$$

If $\nu_c \leq \nu_c^{\max}$, update $\nu_c = \nu_c + 1$.

Step 7.5: Reduce the trust-region radius. If $p_k = p_{\max}$, (4.2.39) fails and either $x_k = v_i$ and $\Delta^c_k > \epsilon_i$ or $\mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta^f_{k+1} = \Delta^f_k$. If $\|n_k\| \neq 0$, set $\Delta^c_{k+1} = \gamma_1 \|n_k\|$, otherwise set $\Delta^c_{k+1} = \gamma_1 \Delta^c_k$. Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

Step 7.6: Update the combined radius and the funnel bound. Set $\Delta_{k+1} = \min[\Delta^f_{k+1}, \Delta^c_{k+1}]$ and update $v_k^{\max}$ using (4.2.41).

Step 8: Update the models and the Lagrange polynomials. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation models m^f_{k+1} and m^c_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$ and the associated Lagrange polynomials $\{l_{k+1,j}\}_{j=0}^p$. Increment k by one and go to Step 1.

4.2.8 Implementation and experiments

4.2.8.1 Solving the subproblems

For the calculation of the normal step and the approximate Lagrange multipliers, we used a Matlab code named BLLS, developed in collaboration with Philippe Toint and Anke Tröltzsch for solving bound-constrained linear least-squares problems. This method is intended for small-dimensional problems and is an active-set algorithm where the unconstrained problem is solved at each iteration in the subspace defined by the currently active bounds, which are determined by a projected Cauchy step. Two strategies are available for the computation of the Cauchy point: (1) a simple Armijo backtracking linesearch method starting from the stepsize corresponding to the unconstrained minimizer along the steepest descent; (2) a successive piecewise quadratic minimization, where the quadratic model is successively minimized on each segment of the projected steepest-descent path until a (first) local minimizer is found. As for the computation of the tangent step, we used a non-monotone spectral projected gradient method (Birgin, Martínez and Raydan, 2000) to solve the (possibly) indefinite quadratic subproblems (4.2.22).

4.2.8.2 Code specifications

The user may specify lower and upper bounds on the variables x by setting values to the constants `lxbounds` and `uxbounds`, respectively, in the call to `deft_funnel`. Lower and upper bounds on the slacks s can be defined likewise by setting values to the constants `lsbounds` and `usbounds`, respectively. An example of usage of the extended DEFT-FUNNEL to solve the problem HS21 from CUTEst is given below.

```
[ x, fx, mu, pi_f, norm_gradlag, norm_cxs, Delta, nit, nfeval] = ...
       deft_funnel(@hs21obj, @hs21cons, [-1 -1] , 0, 'lsbounds', 0, ...
       'usbounds', Inf, 'lxbounds', [2 -50], 'uxbounds', [50 50], ...
       'whichmodel', 'regression', 'epsilon', 1.0e-04)
```

The default output of the algorithm is given by:

- `x`: point returned by the algorithm;
- `s`: values of the slack variables;
- `fx`: value of the objective function at `x`;
- `mu`: vector of estimates of the Lagrange multipliers;
- `pi_f`: value of the measure of optimality at (`x`, `s`);
- `norm_gradlag`: norm of the gradient of the Lagrangian function at (`x`, `s`, `mu`);
- `norm_cxs`: norm of c at (`x`, `s`);
- `Delta`: final trust region radius;
- `nit`: number of iterations required;
- `nfeval`: number of calls made to the subroutine that evaluates the objective function and the constraints.

At each iteration, `deft_funnel_main` calls the function `ident_actv_bnds`, which identifies the active bounds at the current iterate and implements the recursive approach described in Section 4.2.6. After that a new subspace has been defined and a new interpolation set has been built, `ident_actv_bnds` then calls the function `deft_funnel_main` and the algorithm proceeds by working in the reduced dimension.

The value of the measure of optimality defined by `pi_f` is calculated by calling the function `computeOptimality` from the functions `deft_funnel_main` and `ident_actv_bnds`. In the latter, it is done when some bound becomes active, which implies that the measure of optimality must be recomputed while considering the information of the new models. We describe below the function `computeOptimality` in detail.

```
function pi_f = computeOptimality( x, s, g_n, J_s, ls, us, lx, ux )

% Compute the dual optimiality measure by solving the problem
%
%  min  g_n.' * dr
%
%  s.t.: J_s * dr = 0,
%   lx - x <= dr^x <= ux - x,
%   ls - s <= dr^s <= us - s,
%      ||dr|| < = 1,
%
% and by defining pi_f = -<g_n,dr*>, where 'dr*' denotes the solution.

    n = length(x);
    m = length(s);

    % define the bounds for the direction 'dr'

    lb(1:n+m) = -1.0;
    ub(1:n+m) = 1.0;

    dlx = lx - x;
    dlx = dlx';
    dux = ux - x;
    dux = dux';
    lb(1:n) = max(lb(1:n), dlx);
    ub(1:n) = min(ub(1:n), dux);

    dls = ls - s;
    dls = dls';
    dus = us - s;
    dus = dus';
    lb(n+1:n+m) = max(lb(n+1:n+m),dls);
    ub(n+1:n+m) = min(ub(n+1:n+m),dus);

    initPoint = zeros(n+m, 1);
    b = zeros(m, 1);

    options = optimset('LargeScale','off','Simplex','on');
    [dr, fevallinp, exitflag, output] = linprog(g_n, ...
    [], [], J_s, b, lb', ub', initPoint, options);

    % compute the dual optimality measure

    if (norm(dr) > 1.0e-14)
        pi_f = -g_n.' * dr;
    else
        pi_f = 0.0;
```

```
    end

end
```

As we mentioned before, the normal step and the Lagrange multipliers are computed by the function `blls`, our solver for bound-constrained linear least-squares problems. The tangent step is then calculated by calling the function `spg`, which implements the non-monotone spectral projected gradient method. The projections onto the convex feasible set are made in `spg` by calling the function `quadprog` of Matlab.

4.2.8.3 Numerical results

We tested the extended DEFT-FUNNEL firstly on a set of 80 small-scale constrained problems from the CUTEst collection. The problems contain at least one equality or inequality constraint, many of them containing both types and some containing simple bounds as well. Among the 80 problems, 63 are in the range of the first 113 test examples from the Hock-Schittowski collection (Hock and Schittkowski, 1980), while the remaining are nonlinearly constrained optimization problems found in Boggs and Tolle (1989). The choice of the test problems is mostly based on the fact that we wanted to test DEFT-FUNNEL on a set of benchmark problems with a small number of variables and with the three types of constraints (bounds, equalities and inequalities) being present in the set.

We compare the results of the four variants of our method with regard to the way of building the interpolating models — subbasis selection, mininum Frobenius norm, minimum ℓ_2-norm and regression — to those obtained with COBYLA. As before, the only criterion for comparison is the number of calls to the subroutine that evaluates the objective function and the constraints at the same time at the required points.

The threshold ϵ for declaring convergence in the criticality step in DEFT-FUNNEL was set to $\epsilon = 10^{-4}$, while the stopping criterion in COBYLA was set to $\rho_{\text{end}} = 10^{-4}$. The same values for the DEFT-FUNNEL parameters used in the experiments with equality-constrained problems in Section 4.1.6 were kept. In the appendix, we report the number of function evaluations required by the methods to solve the 80 problems in the Tables A.3, A.4 and A.5.

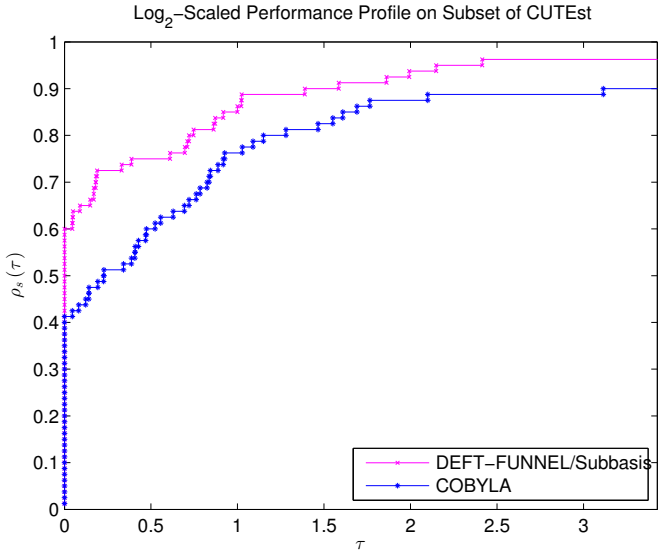
Figure 4.8 shows the performance profiles of the four variants of DEFT-FUNNEL. Each of these variants is then compared to COBYLA individually in Figures 4.9 and 4.10. As it can be seen, DEFT-FUNNEL has shown superior results on the set of test problems when the interpolating models are built from subbasis selection, minimum Frobenius norm and minimum ℓ_2-norm approaches. For the regression variant, Figure 4.10b reveals that COBYLA was faster than DEFT-FUNNEL in most of the problems, although the latter was able to solve more problems than the former for large values of τ.

We also tested DEFT-FUNNEL on a set of 20 small-scale linearly constrained optimization problems from CUTEst and compared its results with

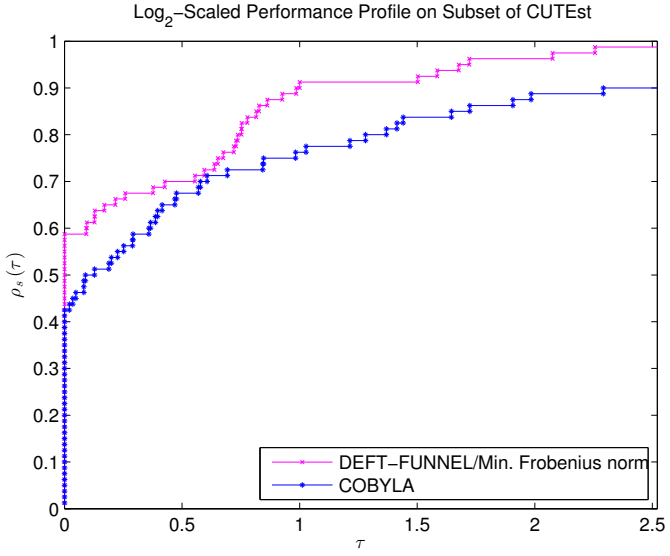
Figure 4.8: Log_2-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models on a set of 80 problems from the CUTEst collection.

those of the software LINCOA, a newly developed trust-region method by Powell (2014) for linearly constrained optimization without derivatives. His software combines active-set methods with truncated conjugate gradients and uses quadratic interpolation models for the objective function. The user is required to provide a feasible starting point, the Jacobian matrix and the vector of constants in the right side of the constraints as inputs. Since many of CUTEst problems provide an infeasible starting point, we chose a feasible one instead for the sake of comparison. The stopping criterion used in LINCOA is the same found in COBYLA, i.e., the algorithm stops when the trust region radius $\rho \in [\rho_{\text{end}}, \rho_{\text{beg}}]$ reaches the smallest value allowed ρ_{end}. In our experiments, we set $\rho_{\text{end}} = 10^{-4}$. Regarding the parameter setting in DEFT-FUNNEL, the same values were kept for comparison with LINCOA. Table A.6 in the appendix show the number of function evaluations required by the methods to solve the 20 linearly constrained problems.

In Figure 4.11, performance profiles of the four variants of DEFT-FUNNEL for the 20 linearly constrained problems are given. We compare each variant of our method to LINCOA in Figures 4.12 and 4.13. The results reported in Figures 4.12a and 4.12b reveal a superior performance of LINCOA over DEFT-FUNNEL/Subbasis and DEFT-FUNNEL/Frobenius. On the other hand, Figure 4.13a shows that DEFT-FUNNEL/Min. ℓ_2-norm was faster than LINCOA. In Figure 4.13b, it can be seen that DEFT-FUNNEL/Regression also

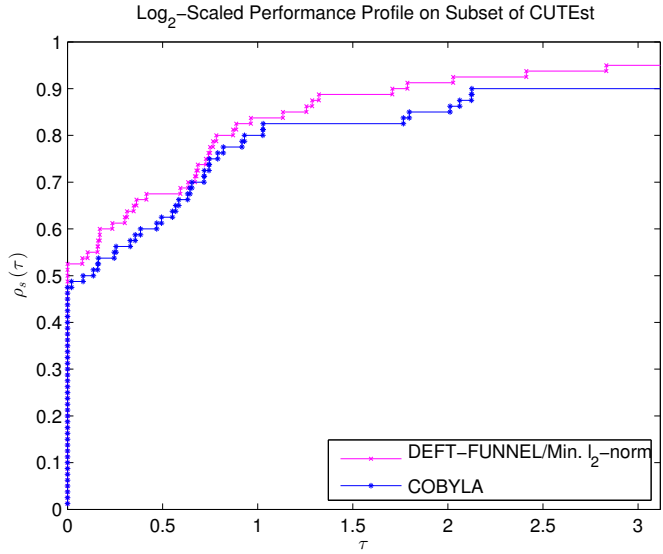


(a) Subbasis

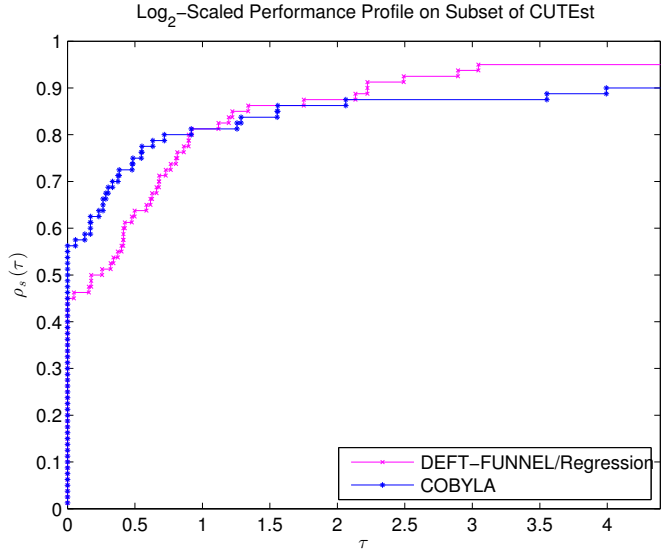


(b) Min. Frobenius norm

Figure 4.9: Log_2-scaled performance profiles of the methods DEFT-FUNNEL and COBYLA on a set of 80 problems from the CUTEst collection.



(a) Min. ℓ_2-norm



(b) Regression

Figure 4.10: *Log*$_2$-scaled performance profiles of the methods DEFT-FUNNEL and COBYLA on a set of 80 problems from the CUTEst collection.

was slightly faster, although LINCOA presented superior performance for large values of τ, which indicates more robustness.

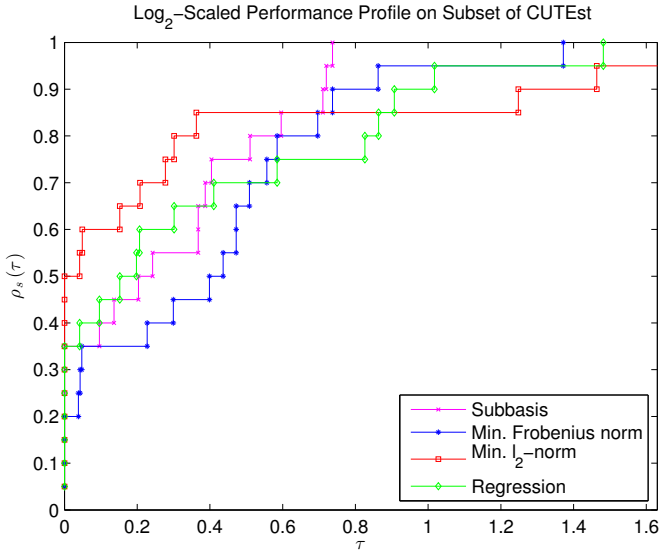
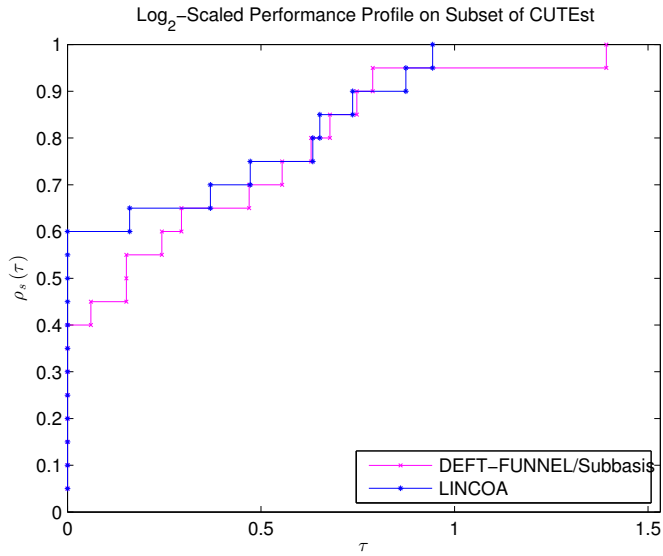
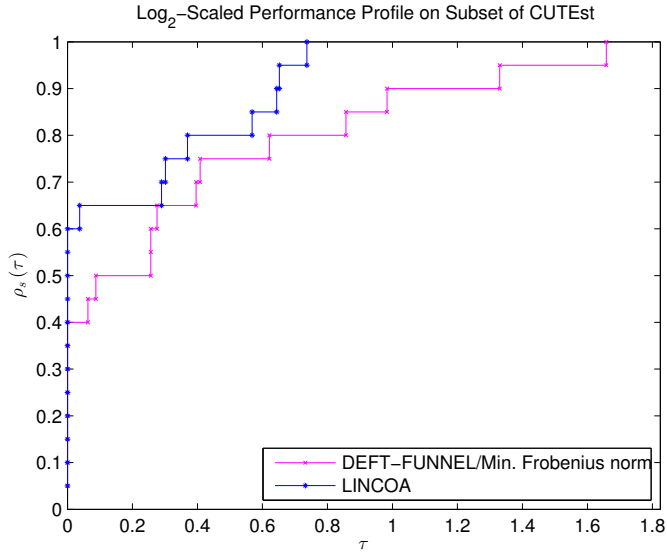


Figure 4.11: Log_2-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models on a set of 20 linearly constrained problems from the CUTEst collection.

We conclude this section by noticing that the performance profiles presented here give clear indication that DEFT-FUNNEL provides encouraging results for small-scale nonlinear optimization problems with general nonlinear constraints. For the case where the user knows that the constraint functions are linear and he is able to provide their gradients, it might be interesting to the handle those constraints separately rather than lumping them and other general nonlinear constraints together in $c(x)$.

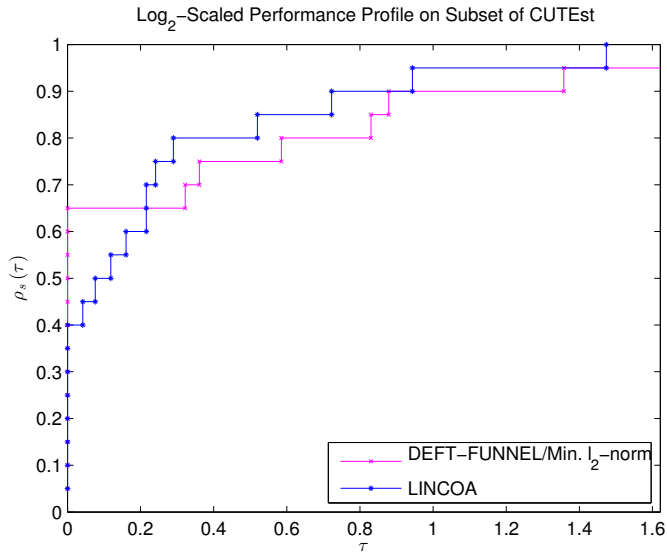


(a) Subbasis

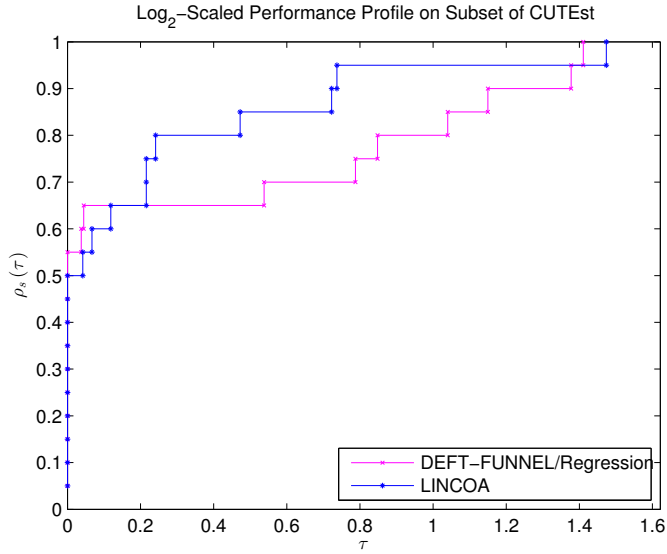


(b) Min. Frobenius norm

Figure 4.12: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL and COBYLA on a set of 20 linearly constrained problems from the CUTEst collection.



(a) Min. ℓ_2-norm



(b) Regression

Figure 4.13: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL and COBYLA on a set of 20 linearly constrained problems from the CUTEst collection.

Chapter 5

Worst-case evaluation complexity

The worst-case evaluation complexity of optimization algorithms applied on nonlinear and potentially nonconvex problems has been studied in a sequence of recent papers, both for the unconstrained case (Nesterov, 2004; Gratton, Sartenaer and Toint, 2008; Nesterov and Polyak, 2006; Cartis, Gould and Toint, 2011*a*) and for the constrained one (Cartis, Gould and Toint, 2012*a*, 2014). Of particular interest here are the results of Nesterov (2004), page 29, in which this author analyzes the worst-case behaviour of the steepest descent method for unconstrained minimization (both for exact and approximate linesearches) and shows that an approximate first-order stationary point, that is, a point at which the norm of the gradient of the objective function is less than $\epsilon > 0$, must be obtained in at most $O(\epsilon^{-2})$ iterations. Nesterov's analysis of the steepest-descent variants therefore effectively assumes that a single objective function value per iteration is computed, or at least that the number of such evaluations in the course of a single iteration is bounded. His bounds thus specify iteration-complexity rather than evaluation complexity. At variance, more typical implementations use a linesearch to compute a suitable steplength, with the possible drawback that an unknown number of additional function evaluations may be required during the course of a single iteration. The question of the worst-case objective-function evaluation complexity of linesearch implementations of this type has not yet been considered specifically. Interestingly, a worst-case complexity analysis is available for other first-order algorithms, such as first-order trust-region methods (Gratton et al., 2008) and first-order regularization algorithms (Cartis, Gould and Toint, 2011*b*).

In parallel, it has long been known that "gradient-related" minimization methods share a number of their convergence properties with the steepest-descent algorithm (see Ortega and Rheinboldt, 2000, for an early reference). In these methods, a linesearch is performed along a direction whose angle with

the negative gradient is bounded away from orthogonality. This class covers a wide range of practical algorithms, including for instance variable-metric techniques or finite-difference schemes when Hessian approximations have bounded conditioning (see Nocedal and Wright, 1999, page 40, for instance). Despite their close connection with steepest descent, their worst-case analysis remains so far an open question.

Standard linesearch methods are usually defined in a way which ensures monotonically decreasing objective-function values as the iterations proceed. However, "non-monotone" generalizations of these algorithms, where this monotonicity property is abandoned, have gained respect in practice because of their often better performance (see Section 2.3.1 for more details on these methods). Again, the worst-case performance of this interesting class of algorithms is so far unexplored.

In the second part of this thesis, we bring together these three questions (standard linesearch, gradient-related directions and non-monotonicity) and provide an analysis which covers them all. We therefore consider non-monotone gradient-related linesearch optimization methods and show that, as for steepest-descent, their objective-function evaluation complexity is $O(\epsilon^{-2})$. Note that standard monotone variants are also covered by this analysis.

5.1 The problem and the algorithm

We consider the nonlinear and possibly nonconvex smooth unconstrained minimization problem

$$\left\{ \begin{array}{ll} \min\limits_{x} & f(x) \\ \text{s.t.:} & x \in \mathbb{R}^n, \end{array} \right. \tag{5.1.1}$$

for which we assume the following:

AF0 $f(x)$ is bounded below on $\mathbb{R}^n$, that is, there exists a constant[(1)] $\kappa_{\rm lbf}$ such that, for all $x \in \mathbb{R}^n$, $f(x) \geq \kappa_{\rm lbf}$.

AF1 $f(x)$ is continuously differentiable on $\mathbb{R}^n$.

As stated in the introduction of this chapter, we consider a class of algorithm in which the search directions are "gradient-related" (see Ortega and Rheinboldt, 2000, page 495, and Bertsekas, 1999, page 35). This terminology means that, at iteration k, an approximate unidimensional minimization of the objective function is performed along a direction d_k whose angle with the steepest descent is controlled by the conditions

$$\langle g_k, d_k \rangle \leq -\kappa_1 \|g_k\|^2 \text{ and } \|d_k\| \leq \kappa_2 \|g_k\|, \tag{5.1.2}$$

(1) "lbf" stands for "lower bound on the objective function".

where $g_k \stackrel{\text{def}}{=} g(x_k) \stackrel{\text{def}}{=} \nabla f(x_k)$ and κ_1 and κ_2 are positive constants independent of k.

Once the direction is fixed, it is then used in a non-monotone linesearch. We choose here a Goldstein-Armijo variant (see Section 2.3.1), in which a stepsize t_k (yielding a new iterate $x_{k+1} = x_k + t_k d_k$) is accepted whenever the conditions

$$f(x_k + t_k d_k) \leq \max_{0 \leq j \leq M} [f(x_{k-j})] + \alpha\, t_k \langle g_k, d_k \rangle, \tag{5.1.3}$$

and

$$f(x_k + t_k d_k) \geq \max_{0 \leq j \leq M} [f(x_{k-j})] + \beta\, t_k \langle g_k, d_k \rangle \tag{5.1.4}$$

hold, where $M \geq 0$, $\alpha \in (0, 1)$ and $\beta \in (\alpha, 1)$ are constants independent of k, and where, by convention, $x_{-M} = \ldots x_{-1} = x_0$. Note that $M = 0$ corresponds to the monotone case.

The class of algorithms of interest may now be stated formally as Algorithm 5.1.1.

Algorithm 5.1.1: Gradient-related non-monotone linesearch.

Step 0: Initialization. An initial point x_0 is given, as well as an accuracy level $\epsilon > 0$. The constants t_{ini}, M, α and β are also given, satisfying $t_{\text{ini}} > 0$, $M \geq 0$ and $0 < \alpha < \beta < 1$. Compute $f(x_0)$, g_0 and set $k = 0$.

Step 1: Test for termination. If $\|g_k\| \leq \epsilon$, terminate.

Step 2: Select a search direction. Choose d_k such that (5.1.2) holds.

Step 3: Linesearch: test initial stepsize.

1. Set $t_k = t_{\text{ini}} > 0$, $t_{\text{low}} = 0$ and compute $f(x_k + t_k d_k)$.
2. If (5.1.3) fails, go to Step 4.
3. If (5.1.4) fails, go to Step 5.
4. Else go to Step 7.

Step 4: Linesearch: backtracking.

1. While (5.1.3) fails, set $t_{\text{up}} \longleftarrow t_k$, $t_k \longleftarrow \frac{1}{2} t_k$ and compute $f(x_k + t_k d_k)$.
2. If (5.1.4) holds, go to Step 7, or set $t_{\text{low}} \longleftarrow t_k$ and go to Step 6 otherwise.

Step 5: Linesearch: look ahead.

1. While (5.1.4) fails, set $t_{\text{low}} \longleftarrow t_k$, $t_k \longleftarrow 2t_k$ and compute $f(x_k + t_k d_k)$.
2. If (5.1.3) holds, go to Step 7, or set $t_{\text{up}} \longleftarrow t_k$ and go to Step 6 otherwise.

Step 6: Linesearch: bisect inside bracket.

1. Set $t_k \longleftarrow \frac{1}{2}(t_{\rm low} + t_{\rm up})$ and compute $f(x_k + t_k d_k)$.
2. If (5.1.3) fails, set $t_{\rm up} \longleftarrow t_k$ and return to Step 6.
3. If (5.1.4) fails, set $t_{\rm low} \longleftarrow t_k$ and return to Step 6.

Step 7: Compute the new iterate and gradient. Set $x_{k+1} = x_k + t_k d_k$ and compute $g_{k+1} = g(x_{k+1})$. Increment k by one and return to Step 1.

Note that the successive phases of the Goldstein-Armijo technique are apparent in the algorithm's description: a bracket containing the desired step is first identified by backtracking (Step 4) or look-ahead (Step 5), and the final step is then computed by bisection (Step 6).

5.2 Worst-case analysis

We now analyze the worst-case behaviour of Algorithm 5.1.1. A first step in this analysis is to specify our assumptions.

AF2 Assume that $g(x)$ is Lipschitz continuous on $\mathbb{R}^n$, that is there exists a constant $L_g > 0$ such that, for all $x, y \in \mathbb{R}^n$,

$$\|g(x) - g(y)\| \leq L_g \|x - y\|.$$

The first simple but crucial property that can be deduced from these assumptions is that the stepsize is bounded below by a constant inversely proportional to the Lipschitz constant L_g.

Lemma 5.2.1. Suppose that AF0–AF2 hold. Then any value of $t > 0$ such that (5.1.4) holds for $t_k = t$ also satisfies the inequality

$$t \geq \frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2}. \tag{5.2.1}$$

Proof. We successively use the mean value theorem, the Cauchy-Schwarz

inequality and AF2 to obtain that

$$\begin{aligned} f(x_k + td_k) &= f(x_k) + t\langle g_k, d_k\rangle + \int_0^1 \langle g(x_k + \tau t d_k) - g_k, td_k\rangle d\tau \\ &\leq f(x_k) + t\langle g_k, d_k\rangle + t\|d_k\| \int_0^1 \|g(x_k + \tau t d_k) - g_k\| d\tau \\ &\leq f(x_k) + t\langle g_k, d_k\rangle + \tfrac{1}{2} t^2 L_g \|d_k\|^2 \\ &\leq \max_{0\leq j\leq M} [f(x_{k-j})] + t\langle g_k, d_k\rangle + \frac{1}{2} t^2 L_g \|d_k\|^2. \end{aligned} \tag{5.2.2}$$

Combining this relation with (5.1.4) and (5.1.2), we have that

$$t \geq \frac{2\langle g_k, d_k\rangle(\beta - 1)}{L_g \|d_k\|^2} \geq \frac{2(1-\beta)\kappa_1 \|g_k\|^2}{L_g \|g_k\|^2 \kappa_2^2} = \frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2}. \tag{5.2.3}$$

□

We now prove that there is a finite and non-empty interval of acceptable stepsizes.

Lemma 5.2.2. Suppose that AF0-AF1 hold and that $g_k \neq 0$. Then there exists an interval $[t_k^\beta, t_k^\alpha]$ such that

$$0 < t_k^\beta < t_k^\alpha < +\infty \tag{5.2.4}$$

and (5.1.3)-(5.1.4) hold for every value of $t_k \in [t_k^\beta, t_k^\alpha]$.

Proof. Observe first that the slope of $f(x_k + td_k)$ is steeper than that of the straight lines $f(x_k) + \alpha t\langle g_k, d_k\rangle$ and $f(x_k) + \beta t\langle g_k, d_k\rangle$, $(t \geq 0)$, since $\alpha < 1$ and $\beta < 1$. Thus, for all $t > 0$ sufficiently small,

$$f(x_k + td_k) < f(x_k) + \alpha t\langle g_k, d_k\rangle \leq \max_{0\leq j\leq M} f(x_{k-j}) + \alpha t\langle g_k, d_k\rangle \tag{5.2.5}$$

and

$$f(x_k + td_k) < f(x_k) + \beta t\langle g_k, d_k\rangle \leq \max_{0\leq j\leq M} f(x_{k-j}) + \beta t\langle g_k, d_k\rangle. \tag{5.2.6}$$

It follows from (5.2.5) that (5.1.3) holds for all t_k sufficiently small. Furthermore, (5.1.3) does not hold in the limit as $t_k = t \to \infty$ since

$$f(x_k) + \alpha t\langle g_k, d_k\rangle \leq f(x_k) - \alpha t \kappa_1 \|g_k\|^2 \to -\infty$$

(because of (5.1.2)), while $f(x_k + td_k) \geq \kappa_{\text{lbf}}$ for all t due to AF0. Thus there exists a value $0 < t_k^\alpha < \infty$ such that

$$f(x_k + t_k^\alpha d_k) = \max_{0\leq j\leq M} f(x_{k-j}) + \alpha t_k^\alpha \langle g_k, d_k\rangle. \tag{5.2.7}$$

For simplicity, let us choose the smallest t_k^α that satisfies (5.2.7) so that (5.2.5) holds for all $t \in (0, t_k^\alpha)$. Since $\alpha < \beta < 1$, we note that

$$\max_{0\leq j\leq M} f(x_{k-j}) + \beta t\langle g_k, d_k\rangle < \max_{0\leq j\leq M} f(x_{k-j}) + \alpha t\langle g_k, d_k\rangle \quad \text{for all} \quad t > 0.$$

Letting $t = t_k^\alpha$ in this inequality and using (5.2.7), we deduce that (5.1.4) must continue to hold for $0 < t_k = t < t_k^\alpha$ sufficiently close to t_k^α. However, (5.2.6) implies that (5.1.4) must fail for sufficiently small $t > 0$, and using again AF1, we conclude that there exists $0 < t_k^\beta < t_k^\alpha$ such that

$$f(x_k + t_k^\beta d_k) = \max_{0\leq j\leq M} f(x_{k-j}) + \beta t_k^\beta \langle g_k, d_k\rangle, \tag{5.2.8}$$

and (5.1.4) holds for all $t_k \in [t_k^\beta, t_k^\alpha]$. (Clearly, t_k^β must be distinct from $t_k^\alpha < \infty$ due to (5.2.7), (5.2.8) and $\alpha < \beta$.) This conclude the proof since (5.1.3) holds for t_k in the same interval due to (5.2.5) and the definition of t_k^α. □

Having proved the existence of an interval of acceptable stepsizes, we now verify that the measure of this interval is bounded below by some positive constant.

Lemma 5.2.3. Suppose that AF0-AF2 hold, and define t_k^α and t_k^β to be any solutions of (5.2.7) and (5.2.8), respectively, such that (5.1.3) and (5.1.4) hold for each $t \in [t_k^\beta, t_k^\alpha]$. Then the interval $[t_k^\beta, t_k^\alpha]$ has a strictly positive measure in the sense that there exists a constant $\kappa_{\text{int}} > 0$ only depending on α, β, κ_1, κ_2 and L_g such that

$$t_k^\alpha - t_k^\beta \geq \kappa_{\text{int}}. \tag{5.2.9}$$

Proof. Assume first that $f(x_k + t_k^\alpha d_k) \leq f(x_k + t_k^\beta d_k)$. Then (5.2.7) and (5.2.8) imply that $\alpha t_k^\alpha > \beta t_k^\beta$, and so, using also Lemma 5.2.1,

$$t_k^\alpha - t_k^\beta \geq \frac{\beta - \alpha}{\alpha} t_k^\beta \geq \frac{2(\beta - \alpha)(1 - \beta)\kappa_1}{\alpha L_g \kappa_2^2}. \tag{5.2.10}$$

Suppose now that $f(x_k + t_k^\alpha d_k) > f(x_k + t_k^\beta d_k)$. Applying the mean value theorem to $f(x + td)$ on $[t_k^\beta, t_k^\alpha]$ yields that

$$\begin{aligned} f(x_k + t_k^\alpha d_k) - f(x_k + t_k^\beta d_k) &= (t_k^\alpha - t_k^\beta)\langle g(x_k + t_\xi d_k), d_k\rangle \\ &\leq (t_k^\alpha - t_k^\beta)\|g(x_k + t_\xi d_k)\|\,\|d_k\| \\ &\leq (t_k^\alpha - t_k^\beta)\kappa_2\|g(x_k + t_\xi d_k)\|\,\|g_k\|, \end{aligned}$$

where $t_\xi \in (t_k^\beta, t_k^\alpha)$ and where the first inequality follows from the Cauchy-Schwarz inequality and the second from (5.1.2). Furthermore, the Lipschitz continuity of g (AF2), (5.1.2) and the bound $t_\xi < t_k^\alpha$ give that

$$\begin{aligned} \|g(x_k + t_\xi d_k)\| &\leq \|g(x_k + t_\xi d_k) - g_k\| + \|g_k\| \\ &\leq L_g t_\xi \|d_k\| + \|g_k\| \\ &\leq L_g t_\xi \kappa_2 \|g_k\| + \|g_k\| \\ &\leq (L_g t_k^\alpha \kappa_2 + 1)\|g_k\|. \end{aligned}$$

Thus

$$f(x_k + t_k^\alpha d_k) - f(x_k + t_k^\beta d_k) \leq \kappa_2 (t_k^\alpha - t_k^\beta)(L_g t_k^\alpha \kappa_2 + 1)\|g_k\|^2. \quad (5.2.11)$$

The definition of t_k^α and t_k^β in Lemma 5.2.2 then gives that (5.2.7) and (5.2.8) both hold, and so

$$\begin{aligned} f(x_k + t_k^\alpha d_k) - f(x_k + t_k^\beta d_k) &= \alpha t_k^\alpha \langle g_k, d_k\rangle - \beta t_k^\beta \langle g_k, d_k\rangle \\ &= (\alpha t_k^\alpha - \beta t_k^\beta)\langle g_k, d_k\rangle \\ &\geq (\beta t_k^\beta - \alpha t_k^\alpha)\kappa_1 \|g_k\|^2, \end{aligned} \quad (5.2.12)$$

where we have again used the Cauchy-Schwartz inequality and (5.1.2) to deduce the last inequality. From (5.2.11), we now deduce that

$$(\beta t_k^\beta - \alpha t_k^\alpha)\kappa_1 \leq \kappa_2 (t_k^\alpha - t_k^\beta)(L_g t_k^\alpha \kappa_2 + 1). \quad (5.2.13)$$

This inequality is equivalent to

$$\kappa_2^2 L_g (t_k^\alpha)^2 + (\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta) t_k^\alpha - (\kappa_2 + \beta\kappa_1) t_k^\beta \geq 0, \quad (5.2.14)$$

and so, since $t_k^\alpha > 0$, we deduce that

$$t_k^\alpha \geq \frac{\kappa_2^2 L_g t_k^\beta - \kappa_2 - \alpha\kappa_1 + \sqrt{(\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta)^2 + 4(\kappa_2 + \beta\kappa_1)\kappa_2^2 L_g t_k^\beta}}{2\kappa_2^2 L_g}, \quad (5.2.15)$$

and therefore that

$$\begin{aligned} 2\kappa_2^2 L_g (t_k^\alpha - t_k^\beta) &\geq -(\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta) + S(t_k^\beta) \\ &= \frac{-(\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta)^2 + S(t_k^\beta)^2}{\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)} \\ &= \frac{4(\beta - \alpha)\kappa_1 \kappa_2^2 L_g t_k^\beta}{\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)}, \end{aligned} \quad (5.2.16)$$

where $S(t_k^\beta) \stackrel{\text{def}}{=} \sqrt{(\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta)^2 + 4(\kappa_2 + \beta\kappa_1)\kappa_2^2 L_g t_k^\beta}$. As a consequence, we obtain that

$$\frac{(t_k^\alpha - t_k^\beta)}{2(\beta - \alpha)\kappa_1} \geq \frac{t_k^\beta}{\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)} \stackrel{\text{def}}{=} E(t_k^\beta). \quad (5.2.17)$$

Differentiating $E(t_k^\beta)$ with respect to t_k^β then gives that

$$\begin{aligned}
E'(t_k^\beta) &= \frac{\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)}{\left[\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)\right]^2} \\
&\quad - \frac{t_k^\beta \left[\kappa_2^2 L_g + \dfrac{-(\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta)\kappa_2^2 L_g + 2(\kappa_2 + \beta\kappa_1)\kappa_2^2 L_g}{S(t_k^\beta)}\right]}{\left[\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)\right]^2} \\
&= \frac{(\kappa_2 + \alpha\kappa_1) S(t_k^\beta) + (\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta)^2}{\left[\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)\right]^2 S(t_k^\beta)} \\
&\quad + \frac{(\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta)\kappa_2^2 L_g t_k^\beta + 2(\kappa_2 + \beta\kappa_1)\kappa_2^2 L_g t_k^\beta}{\left[\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)\right]^2 S(t_k^\beta)} \\
&= \frac{(\kappa_2 + \alpha\kappa_1)[S(t_k^\beta) + \kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta] + 2(\kappa_2 + \beta\kappa_1)\kappa_2^2 L_g t_k^\beta}{\left[\kappa_2 + \alpha\kappa_1 + \kappa_2^2 L_g t_k^\beta + S(t_k^\beta)\right]^2 S(t_k^\beta)}.
\end{aligned} \tag{5.2.18}$$

It then follows that

$$E'(t_k^\beta) > 0 \text{ for all } t_k^\beta > 0$$

since

$$S(t_k^\beta) + \kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta > |\kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta| + \kappa_2 + \alpha\kappa_1 - \kappa_2^2 L_g t_k^\beta \geq 0$$

and each constant and variable in $E'(t_k^\beta)$ is positive. Thus $E(t_k^\beta)$ is increasing as a function of t_k^β, and we obtain, because of Lemma 5.2.1 and the fact that (5.1.4) holds at t_k^β by construction, that

$$E(t_k^\beta) \geq E\left(\frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2}\right),$$

and we finally deduce from (5.2.17) that

$$t_k^\alpha - t_k^\beta \geq 2(\beta - \alpha)\kappa_1 \, E\left(\frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2}\right).$$

Combining this with (5.2.10), we deduce that (5.2.9) holds with

$$\kappa_{\text{int}} \stackrel{\text{def}}{=} 2(\beta - \alpha)\kappa_1 \min\left[\frac{(1-\beta)}{\alpha L_g \kappa_2^2}, E\left(\frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2}\right)\right],$$

where this lower bound only depends on α, β, κ_1, κ_2 and L_g, as desired. □

We now turn to estimating the worst-case evaluation complexity of Algorithm 5.1.1 for the task of finding an ϵ-first-order stationary point. The following theorem completes the second part of this thesis with the final contribution of this work.

Theorem 5.2.4. Suppose that AF0-AF2 hold. Then, for any $\epsilon \in (0,1)$, Algorithm 5.1.1 needs at most

$$\left\lceil \frac{f(x_0) - \kappa_{\rm lbf}}{\kappa_{\rm decr}\epsilon^2} \right\rceil$$

function evaluations to produce an iterate x_k such that $\|g_k\| \leq \epsilon$, where $\kappa_{\rm lbf}$ is defined in AF0 and where

$$\kappa_{\rm decr} \stackrel{\rm def}{=} \alpha\kappa_1 \min\left[\frac{2(1-\beta)\kappa_1}{L_g\kappa_2^2 \max[n_1,n_2]}, \frac{t_{\rm ini}}{2(n_2+1)} \right]$$

with

$$n_1 \stackrel{\rm def}{=} \left| \log_2 \left(\frac{(1-\beta)\kappa_1}{t_{\rm ini} L_g \kappa_2^2} \right) \right| \quad \text{and} \quad n_2 \stackrel{\rm def}{=} \left| \log_2 \left(\frac{\kappa_{\rm int}}{t_{\rm ini}} \right) \right|.$$

Proof. The proof proceeds by first establishing the minimum achieved decrease in the objective function between iterate x_{k+1} and its "predecessor" $x_{\tau(k+1)}$, where

$$\tau(k+1) = k - \arg\max_{0 \leq j \leq M} f(x_{k-j}) \tag{5.2.19}$$

when using Algorithm 5.1.1.

- Assume first that both (5.1.3) and (5.1.4) hold for $t_k = t_{\rm ini}$ (in Step 3). Then we obtain a decrease

$$f(x_{\tau(k+1)}) - f(x_{k+1}) \geq -\alpha\, t_{\rm ini} \langle g_k, d_k \rangle \geq \alpha\, t_{\rm ini} \kappa_1 \|g_k\|^2, \tag{5.2.20}$$

  because of (5.1.3) and (5.1.2), and this decrease is obtained for a single additional function evaluation.

- Assume now that (5.1.3) fails at Step 3.2, and Step 4 is therefore entered. Assume furthermore that $j_3 \geq 1$ backtracking steps are performed in Step 4.1. The j_3 is the smallest non-negative integer such that (5.1.3) holds for $t_k = t_{\rm ini} 2^{-j_3}$, which means that j_3 is the largest integer for which this inequality is violated for $t = t_{\rm ini} 2^{-j_3+1}$. Because $\alpha < \beta$, we deduce that (5.1.4) must hold for this value of t_k. Using now Lemma 5.2.1, we obtain that

$$t = 2^{-j_3+1} t_{\rm ini} \geq \frac{2(1-\beta)\kappa_1}{L_g\kappa_2^2},$$

which in turn implies that

$$j_3 \leq \left| \log_2 \left(\frac{(1-\beta)\kappa_1}{t_{\rm ini} L_g \kappa_2^2} \right) \right| \stackrel{\rm def}{=} n_1. \tag{5.2.21}$$

Step 4 therefore requires at most n_1 function evaluations. If the linesearch is terminated in Step 4.2 (i.e., branching occurs to Step 7), we obtain a decrease

$$\begin{aligned} f(x_{\tau(k+1)}) - f(x_{k+1}) &\geq -\alpha\, t_k \langle g_k, d_k \rangle \\ &\geq \alpha\, \frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2} \kappa_1 \|g_k\|^2, \end{aligned} \tag{5.2.22}$$

where we used (5.1.3), (5.1.4), (5.1.2) and Lemma 5.2.1 successively.

- If the linesearch is not terminated in Step 4, Step 6 must be entered, with a bracket $[t_{\rm low}, t_{\rm up}]$ where

$$t_{\rm low} = 2^{-j_3} t_{\rm ini} = \frac{1}{2} t_{\rm up}.$$

Thus

$$t_{\rm up} - t_{\rm low} = \frac{1}{2} 2^{-j_3+1} t_{\rm ini} = 2^{-j_3} t_{\rm ini}.$$

We know from Lemma 5.2.3 that the length of the admissible interval is at least equal to $\kappa_{\rm int} > 0$, where this constant only depends on α, β and L_g. Thus the number $j_4 \geq 1$ of bisection (and function evaluations) within Step 6 is bounded above by the smallest integer such that

$$2^{-j_4}(t_{\rm up} - t_{\rm low}) = 2^{-j_4} 2^{-j_3} t_{\rm ini} \geq \kappa_{\rm int},$$

which then yields that the total number of function evaluations in Step 4 and 6 is bounded by

$$j_3 + j_4 \leq \left| \log_2 \left(\frac{\kappa_{\rm int}}{t_{\rm ini}} \right) \right| \stackrel{\rm def}{=} n_2.$$

If we now compute the decrease obtained, we deduce, again from (5.1.3), (5.1.4) and Lemma 5.2.1, that (5.2.22) also holds in this case.

- Assume now that (5.1.4) fails in Step 3.3, and thus that Step 5 is entered. Assume furthermore that $j_2 \geq 1$ doubling of t_k (and j_2 function evaluations) occur in Step 5.1 (we know that j_2 is finite because of (5.2.4)). If the lineasearch is terminated in Step 5.2 (i.e., branching to Step 7 occurs), we obtain that the function decrease obtained is bounded below by

$$f(x_{\tau(k+1)}) - f(x_{k+1}) \geq -\alpha t_k \langle g_k, d_k \rangle \geq \alpha 2^{j_2} t_{\rm ini} \kappa_1 \|g_k\|^2.$$

- The final case is when Step 6 is entered after Step 5, in which case the initial bracket for Step 6 is given by $[t_{\rm low}, t_{\rm up}]$ where

$$t_{\rm low} = 2^{j_2-1} t_{\rm ini} = \frac{1}{2} t_{\rm up}.$$

Thus

$$t_{\rm up} - t_{\rm low} = \frac{1}{2} 2^{j_2} t_{\rm ini} = 2^{j_2-1} t_{\rm ini}.$$

Just as in the case where Step 6 is entered after Step 4, we now deduce that the number j_4 of bisections and function evaluations needed to reduce this bracket to the minimum possible value $\kappa_{\rm int}$ is limited by the inequality

$$2^{-j_4} 2^{j_2-1} t_{\rm ini} = 2^{-j_4}(t_{\rm up} - t_{\rm low}) \geq \kappa_{\rm int},$$

yielding a maximum number of bisection (and function evaluation) in Step 6 bounded by

$$j_4 \leq j_2 - 1 + \left| \log_2 \left(\frac{\kappa_{\rm int}}{t_{\rm ini}} \right) \right| = j_2 - 1 + n_2 \leq j_2(n_2 + 1).$$

In this final case, since $t_k \geq t_{\rm low} = 2^{j_2-1} t_{\rm ini}$ and (5.1.3) holds at t_k, the function decrease is bounded below by

$$f(x_{\tau(k+1)}) - f(x_{k+1}) \geq -\alpha t_k \langle g_k, d_k \rangle \geq \alpha 2^{j_2-1} t_{\rm ini} \kappa_1 \|g_k\|^2.$$

Gathering all cases together, we see that the function decrease per function evaluation is given, in the worst case, by

$$\min \left[\frac{t_{\rm ini}}{1}, \frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2 n_1}, \frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2 n_2}, \frac{2^{j_2} t_{\rm ini}}{j_2}, \frac{2^{j_2-1} t_{\rm ini}}{2 j_2 (n_2+1)} \right] \alpha \kappa_1 \|g_k\|^2, \quad (5.2.23)$$

where, by construction, n_1 and n_2 only depend on α, β, κ_1, κ_2, L_g and $t_{\rm ini}$. Noting that, for $j_2 \geq 1$,

$$\frac{2^{j_2}}{j_2} \geq 2 \quad \text{and} \quad \frac{2^{j_2-1}}{2 j_2} \geq \frac{1}{2}$$

we define

$$\kappa_{\rm decr} \stackrel{\rm def}{=} \alpha \kappa_1 \min \left[\frac{2(1-\beta)\kappa_1}{L_g \kappa_2^2 \max[n_1, n_2]}, \frac{t_{\rm ini}}{2(n_2+1)} \right].$$

Since $\kappa_{\rm decr} \|g_k\|^2$ is the function decrease per function evaluation in the worst case, we conclude that a decrease from $f(x_{\tau(k+1)})$ to $f(x_{k+1})$ requires at most

$$\left\lceil \frac{f(x_{\tau(k+1)}) - f(x_{k+1})}{\kappa_{\rm decr} \|g_k\|^2} \right\rceil \leq \left\lceil \frac{f(x_{\tau(k+1)}) - f(x_{k+1})}{\kappa_{\rm decr} \epsilon^2} \right\rceil$$

function evaluations as long as the algorithm does not terminate (i.e., as long as $\|g_k\| \geq \epsilon$). Tracing back the predecessors of iterate x_{k+1} up to x_0 and denoting the composition of j instances of the predecessor operator $\tau(\cdot)$ by $\tau^j(\cdot)$, we also deduce that at most

$$\left\lceil \frac{f(x_{\tau^{j+1}(k+1)}) - f(x_{\tau^j(k+1)})}{\kappa_{\rm decr}\epsilon^2} \right\rceil$$

function evaluations are needed to obtain a reduction from $f(x_{\tau^{j+1}(k+1)})$ to $f(x_{\tau^j(k+1)})$, for all $j = 0, \ldots, p_k$, for p_k such that $x_{\tau^{p_k}(k+1)} = x_0$ and where, by convention, $\tau^0(k+1) \stackrel{\rm def}{=} k+1$. Using the definition of $\tau(\cdot)$ in (5.2.19), we conclude that a reduction from $f(x_0)$ to $f(x_{k+1})$ requires at most

$$\left\lceil \frac{\sum_{j=0}^{p_k}[f(x_{\tau^{j+1}(k+1)}) - f(x_{\tau^j(k+1)})])}{\kappa_{\rm decr}\epsilon^2} \right\rceil = \left\lceil \frac{f(x_0) - f(x_{k+1})}{\kappa_{\rm decr}\epsilon^2} \right\rceil$$

function evaluations. Since $f(x_0) - f(x_{k+1}) \leq f(x_0) - \kappa_{\rm lbf}$ by AF0, we obtain that the total number of function evaluations in Algorithm 5.1.1 is bounded above by

$$\left\lceil \frac{f(x_0) - \kappa_{\rm lbf}}{\kappa_{\rm decr}\epsilon^2} \right\rceil.$$

□

As it can be seen in the above result, the constant $\kappa_{\rm decr}$ is proportional to κ_1 and inversely proportional to κ_2, which are positive constants used in (5.1.2) to define gradient-related search directions. In an attempt to obtain an intuitive view of this relation, assume that the directions d_k are defined as

$$d_k = -D_k \nabla f(x_k),$$

where D_k is a positive definite symmetric matrix whose eigenvalues are bounded above and bounded away from zero, i.e.

$$\kappa_1\|z\|^2 \leq \langle z, D_k z\rangle \leq \kappa_2\|z\|^2, \quad \forall z \in \mathbb{R}^n,\ k = 0, 1, \ldots \tag{5.2.24}$$

Then we have that

$$|\langle \nabla f(x_k), d_k\rangle| = |\langle \nabla f(x_k), D_k \nabla f(x_k)\rangle| \geq \kappa_1\|\nabla f(x_k)\|^2$$

and

$$\|d_k\|^2 = |\langle \nabla f(x_k), (D_k)^2 \nabla f(x_k)\rangle| \leq \kappa_2^2\|\nabla f(x_k)\|^2.$$

Because of the bounded eigenvalues condition, the directions $\{d_k\}$ defined as above are gradient-related. Returning now to the role of the constants κ_1 and κ_2 in the complexity result, it can be seen that, in this particular case, the ideal is to choose matrices D_k with eigenvalues that have bounds that are as large as possible and as small as possible at the same time. Therefore, we may

conclude that the constants κ_1 and κ_2 given by a particular method to obtain the direction d have no major influence in the complexity of the method. On the other hand, one can see that the evaluation complexity decreases as the initial step size t_{ini} increases. Regarding the parameters α and β, it can be seen that large values for α are ideal, while small values for β contribute to decrease the overall complexity. Concerning the Lipschitz constant L_g, it is difficult to conclude something about its ideal value. In the first fraction of κ_{decr}, we see that the smaller is the limit in how fast the gradient of f can change, the larger κ_{decr} will be. Nevertheless, the inverse may occur when one considers the value of L_g in the definition of n_1, where large values for L_g may be better. Note also that if L_g is too large, the fraction in n_1 approaches zero, which makes n_1 increase. Finally, it is clear that the number of function evaluations needed to find an ϵ first-order critical point also depends on the choice of the starting point x_0; the smaller its function value is, the less function evaluations are needed in the worst case. Besides, the starting point x_0 also influences the Lipschitz constant as, in practice, it suffices that the property of Lipschitz continuity hold on the level set $\mathcal{L}(x_0) = \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$.

Chapter 6

Conclusions and further research perspectives

The research work described in this thesis is concerned with the design and implementation of a new trust-region algorithm for constrained derivative-free optimization and the worst-case evaluation complexity analysis of non-monotone gradient-related algorithms for unconstrained optimization.

In Chapter 2, we have presented the fundamental principles of nonlinear optimization as well as some of the algorithms used to solve unconstrained and constrained problems. Chapters 3 and 4 have been concerned with the area of derivative-free optimization, which is the research topic of our first main contribution. In particular, Chapter 3 has introduced the important class of derivative-free trust-region methods based on polynomial interpolation models after an introduction of the basic concepts of multivariate polynomial interpolation. In Chapter 4, we have proposed our method for solving constrained nonlinear optimization problems without derivatives, discussing some implementation issues and showing numerical results from our experiments. Finally, Chapter 5 has established worst-case evaluation complexity bounds for the solution of unconstrained problems using non-monotone gradient-related algorithms.

In the next two sections, we review the development of our work on both research topics and draw some final conclusions on each one of them.

6.1 Constrained derivative-free optimization

Firstly, we have proposed a new derivative-free algorithm for equality-constrained optimization problems based on the trust-funnel approach of Gould and Toint (2010). The devised algorithm, named DEFT-FUNNEL, makes use of neither the derivatives of the objective function nor the derivatives of the constraints. It also considers both the objective and constraints as black-box

functions. It employs an ensemble of underlying techniques such as multivariate polynomial interpolation for the construction of surrogate models and a self-correcting geometry mechanism for the maintenance of the interpolation set. We have tested the proposed method on a set of 29 small-scale equality-constrained problems from the CUTEst collection and presented encouraging results. Four described variants of the new method differing by their choice of construction of the models surpassed two other trust-region methods, CDFO (Colson, 2004) and COBYLA (Powell, 1994). Among these variants, the minimum ℓ_2-norm model variant outperformed the subbasis selection and also, somewhat surprisingly, the minimum Frobenius norm approach. We then have extended the original derivative-based trust-funnel method to problems with both equality and inequality constraints and where simple bounds are also considered. When simple bounds are given, the algorithm makes use of an active-set approach to perform minimization on the subspaces defined by the active bounds. Finally, we modified DEFT-FUNNEL in order to obtain a final method that can be used to solve problems with general nonlinear constraints where the derivatives are unavailable. For the case where bound constraints are present, we have shown how the algorithm reduces the dimension of the problem by using recursion within a subspace minimization approach in order to avoid a potential degeneration of the interpolation set when the solution is approached. Numerical experiments on a set of 80 small-scale nonlinear optimization problems with general nonlinear constraints and on a set of 20 small-scale linearly constrained optimization problems were performed and showed that our method compares favorably to competing algorithms, namely: CDFO, COBYLA, and LINCOA (Powell, 2014). Finally, we note that the methods proposed in this thesis are the subject of two papers by Sampaio and Toint (2015*a*, 2015*b*).

For future research, we plan to consider the possibility of having two different interpolations sets for the objective function and the constraints and analyze the performance of DEFT-FUNNEL for the cases where the constraints are linear, while the objective function is of higher degree. Another interesting subject of research would concern the subspace minimization approach described in Section 4.2.6 for handling the bounds. The idea of reducing the dimension of the problem to thwart the degeneration of the interpolation set may also be applied to more general constraints as well. In case where the constraints are linear, it is easy to see that all the interpolation points will eventually be feasible iterates generated by the algorithm. This implies that the interpolation points will become affinely dependent, and, therefore, the quality of the geometry of the interpolation set will deteriorate. A similar approach to avoid this situation can be applied by working in the subspace defined by the nullspace of the Jacobian matrix. For the case of nonlinear constraints, the same idea can be used with an additional care to ensure that the iterates in the tangent space of the constraints do not deviate too much from the feasible set. Particularly for this case, the subspace minimization approach might yield better results as the iterates get closer to the solution. We also intend to include the least

Frobenius norm and the minimum ℓ_1-norm updating approaches for building the models into DEFT-FUNNEL as they can be advantageous in many situations. Finally, convergence results for both derivative-based and -free versions of the algorithm are also of interest and are left as a development line for future work.

6.2 Worst-case evaluation complexity

We have shown that gradient-related methods using a non-monotone (and monotone) linesearch will find an ϵ-approximate first-order critical point of a smooth function with Lipschitz gradient in $O(\epsilon^{-2})$ function and gradient evaluations at most. Their worst-case behaviour is therefore, up to a factor, equivalent to that of a simple monotone pure steepest-descent algorithm, albeit their practical performance is often superior (see Toint, 1996). Moreover, it results from Cartis, Gould and Toint (2010) that this bound is sharp. We also note that the results on worst-case evaluation complexity demonstrated here are published in the paper by Cartis, Sampaio and Toint (2015).

In the same line of investigation, Cartis, Gould and Toint (2012*b*) show that the same complexity order is obtained for the steepest-descent method with exact linesearch and that it is sharp. One may expect that this result can be extended to the gradient-related algorithms analyzed in the present note, although the construction of an example illustrating the sharpness of the complexity bound is likely to be challenging without monotonicity.

List of Figures

2 Fundamentals and algorithms of nonlinear optimization

2.1 Trajectories of the iterates from Newton's method (red line) without any globalization strategy and the steepest descent method (green line) in a 2-dimensional problem. Since Newton's directions are not always descent directions, the method fails to converge to the solution x^*, while the steepest descent method achieves convergence without problems. 10

2.2 Trajectories of the iterates from Newton's method (red line) and the steepest descent method (green line) in a 2-dimensional optimization problem. As opposed to the latter, Newton's method uses curvature information to achieve convergence more rapidly. 11

2.3 Trajectories of the iterates from the CG method (red line) and the steepest descent method (green line) in a 2-dimensional optimization problem. As expected, the CG method takes no more than 2 iterations to converge. 17

2.4 Graphical representation of the projected gradient method. . . 24

2.5 Graphical representation of the filter method. The shaded area contain all the points dominated by the four pairs (f_j, v_j) in the filter, which are indicated by a dot. 27

2.6 Graphical representation of the trust-funnel method. 28

3 Introduction to derivative-free optimization

3.1 Illustration of a direct-search method in which coordinate directions are used. The level sets of the objective function are portrayed by the ellipses. Note that, at x_3, the algorithm fails at finding a descent direction in the poll step, causing reduction of the step size. The points obtained with the new step size are depicted by green dots and a red dot. The iterates are represented by hollow circles, with the exception of the minimizer $x_5 = x^*$, which is red. 30

3.1 Construction of a quadratic interpolation model using Lagrange polynomials. 38

4 A derivative-free trust-funnel method

4.1 The components of the step d_k with $\Delta_k = \Delta_k^f$. 57
4.2 Log_2-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models for the first type of comparison on a set of problems from the CUTEst collection. . . . 73
4.3 Log_2-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the first type of comparison on a set of problems from the CUTEst collection. 74
4.4 Log_2-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the first type of comparison on a set of problems from the CUTEst collection. 75
4.5 Log_2-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the second type of comparison on a set of problems from the CUTEst collection. 76
4.6 Log_2-scaled performance profiles of DEFT-FUNNEL, CDFO and COBYLA for the second type of comparison on a set of problems from the CUTEst collection. 77
4.7 Illustration of a scenario where the interpolation set becomes degenerated as the optimal solution is approached. In this example, we consider a 2-dimensional problem with the bound constraint $[x]_2 \geq 0$, which is active at the solution x^* and at the iterates close to it. 86
4.8 Log_2-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models on a set of 80 problems from the CUTEst collection. 96
4.9 Log_2-scaled performance profiles of the methods DEFT-FUNNEL and COBYLA on a set of 80 problems from the CUTEst collection. 97
4.10 Log_2-scaled performance profiles of the methods DEFT-FUNNEL and COBYLA on a set of 80 problems from the CUTEst collection. 98
4.11 Log_2-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models on a set of 20 linearly constrained problems from the CUTEst collection. 99
4.12 Log_2-scaled performance profiles of DEFT-FUNNEL and COBYLA on a set of 20 linearly constrained problems from the CUTEst collection. 100
4.13 Log_2-scaled performance profiles of DEFT-FUNNEL and COBYLA on a set of 20 linearly constrained problems from the CUTEst collection. 101

Index

A

active-set method 73, 78, 92, 96
algorithms
  for constrained optimization . 19
  for unconstrained optimization 8
Armijo's condition 10, 23

B

basis 20, 33
  monomial ..... *see* natural basis
  natural 33, 34, 38, 41, 44, 64
  polynomial 32

C

Cauchy condition . 47, 53, 56, 60, 79, 82
complexity 103, 104, 111
composite step 27, 28, 51, 78
condition number .. 18, 20, 35, 41, 65
conditions
  Goldstein ..... *see* Goldstein's conditions
  Karush-Kuhn-Tucker 8, 25
  necessary
    first-order 6, 24
    second-order 7
  optimality 6, 7, 25
  second-order sufficient 7
  Wolfe .... *see* Wolfe's conditions
conjugacy 15
conjugate gradient method 14, 19, 21, 67
  preconditioned 18
    projected 19
  truncated 54, 96
  truncated projected ... 21, 65, 69
constraint
  bound 23, 27, 31, 52, 65, 73, 80–82, 86, 95
  equality 6, 24, 27, 31, 52, 53, 72, 73, 86, 95
  inequality 6, 10, 31, 52, 95
constraint qualifications 8
convergence 18, 24, 26, 27, 71, 87, 95, 103
  global 9, 27, 31, 32, 46, 48
  linear 9
  local 9, 26
  superlinear 9
criticality 31, 48
  measure 80, 82, 87
  step 60, 71, 87, 89, 95
curvature 10, 43

D

derivative-free
  method 29, 86
  optimization 29, 51, 73, 86
direct-search method 29, 30
direction
  Cauchy 9
    projected 79, 82, 92
  coordinate 29
  descent 9
    steepest 92
  search 8
  tangent 81
dominance 26

E

eigenvalues 18

F

feasibility 26, 28, 51, 52, 57–59, 71, 85
feasible set 6, 23, 95
filter method 26, 27, 71
finite differences 29, 31, 104

G

Gauss-Newton method 53
geometry 31, 46–48, 50, 86
    self-correcting . 32, 47, 48, 50–52, 60, 65
Goldstein's conditions 10
gradient projection method ..... *see* projected gradient method

I

interpolation set ..... *see* sample set

L

Lagrange multipliers 8, 54, 55, 57, 59, 60, 67, 81, 83, 88, 92
Lagrange polynomials 31, 35–37, 39–42, 44, 46–49, 60, 65
Lagrangian function 7, 24, 54, 80
least-squares
    bounded linear 78, 79, 83, 92, 95
    nonlinear 12
    problem 55
    regression 45
Levenberg method 12
Levenberg-Morrison-Marquardt method 13
linesearch method 8, 21, 103, 104, 112
    backtracking 11, 23, 92
      non-monotone 12
    non-monotone 105
      gradient-related 105
    SQP 26

M

merit function 26, 51
minimizer
    global 6
    local 6
      strict 6
      weak 6
model
    Gauss-Newton 53, 79, 80
    improvement steps 46, 47
    interpolation .. 29, 31–34, 41, 43, 44, 47, 64, 86–88
      linear 31, 43, 44, 64, 72
      quadratic 31, 44, 64, 96
    quadratic 13, 24, 33, 54, 55
    regression 45, 65
    SQP 80
    surrogate 33, 51
Moré-Sorensen algorithm 54, 65
multiobjective optimization 26

N

Nelder-Mead method 30
Newton's method 9, 13, 24, 25
normal step 27, 28, 51, 52, 54, 57, 58, 61, 78, 79, 81–83, 85, 89, 92
null space 20, 55, 66, 82

O

optimality . 6–8, 25–27, 51–53, 56–58, 71, 78
orthogonality 16, 104

P

pattern-search method
    generalized 31
penalty method 26, 27
performance profiles 72, 95
point
    Cauchy 56, 82
    dummy 87
    feasible 6, 96
    KKT 8
    regular 8
    stationary
      first-order . 6, 8, 23, 31, 32, 60, 80
points
    stationary

first-order 58
poisedness ..34–37, 39, 40, 42, 46, 64
Λ-poisedness 40, 41, 67
well 35, 38–43, 46, 65, 67, 71
poll step 29
polynomial interpolation ..31, 32, 36, 40, 43, 45, 51, 52
preconditioning 18–21
projected gradient method 22
non-monotone spectral 92
projection . 19, 22, 23, 65, 80, 82, 87, 88, 95

Q

quasi-Newton method 9, 13, 31

S

sample set . 29, 31, 33, 35, 38, 39, 41, 43, 46, 48, 50, 51, 60, 64, 65, 67, 71, 78, 86–89, 91
search step 29
secant method .....*see* quasi-Newton method
sequential quadratic programming method 24, 25, 27, 51
simplex search method .....*see* Nelder-Mead method
solution . 9, 13, 15, 20, 23, 24, 26, 31, 43, 78, 80, 86, 87
global .....*see* global minimizer
local .....*see* local minimizer
SQP method *see* sequential quadratic programming method
steepest descent method 9, 12, 17, 23, 103, 104
Steihaug-Toint CG method *see* truncated projected CG method
steplength ...8, 10, 13, 16, 21–23, 26, 29, 92, 103
subbasis 43, 71, 72, 95, 96
subspace minimization 73, 78, 88
sufficient decrease 10, 13, 31, 55

T

tangent step 27, 28, 51, 52, 54–58, 61, 78, 80, 82, 83, 89, 92
trust-funnel method ... 27, 51, 72, 86
trust-region method ...12, 27, 31, 46, 71, 96, 103
derivative-free 31, 32
SQP 26

V

variable
dual 8, 24, 25
primal 8, 24, 25
slack 81

W

Wolfe's conditions 10

Z

zero-order method ..*see* direct-search method

Appendix

Appendix A

Detailed numerical results

We report here the number of function evaluations required by DEFT-FUNNEL and other derivative-free methods to converge on a set of problems from the CUTEST collection. The first two tables concern the tests with the first version of our method on a set of equality-constrained problems, while the remaining are related to problems with general nonlinear constraints. For the latter group, we denote by m_{B}, m_{EQ} and m_{IQ} the number of constraints in the problem of the type bound, equality and inequality, respectively. The limit of function evaluations imposed in our experiments was $500 \times n$, where n is the number of variables in the problem. We indicate by "NaN" (standing for not a number) the problems where no convergence to a solution was achieved within this limit.

Table A.1: Number of function evaluations required by the first version of DEFT-FUNNEL, CDFO and COBYLA methods to converge in the first type of comparison on a set of 29 equality-constrained problems.

Problem	n	m	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	CDFO	COBYLA
BT1	2	1	46	65	453	479	105	53
BT2	3	1	439	131	87	153	191	249
BT3	5	3	65	55	41	43	237	135
BT4	3	2	34	34	32	34	42	64
BT5	3	2	40	30	30	22	53	78
BT6	5	2	134	131	111	252	912	91
BT7	5	3	99	113	97	104	676	176
BT8	5	2	70	66	69	51	499	95
BT9	4	2	52	55	55	73	NaN	115
BT10	2	2	48	51	51	27	101	23
BT11	5	3	184	101	107	171	466	152
BT12	5	3	44	45	63	44	1098	601
HS6	2	1	28	23	23	24	16	40
HS7	2	1	32	24	23	47	685	51
HS8	2	2	19	18	13	15	68	17
HS9	2	1	52	52	142	488	23	33
HS26	3	1	90	76	78	101	NaN	71
HS27	3	1	95	57	57	80	548	939
HS28	3	1	35	34	31	33	19	60
HS39	4	2	52	55	55	73	179	46
HS40	4	3	49	54	46	33	1374	78
HS42	4	2	39	45	45	48	54	72
HS46	5	2	149	421	197	242	302	751
HS48	5	2	46	76	65	45	37	87
HS49	5	2	235	NaN	250	620	281	1096
HS50	5	3	264	140	141	345	40	148
HS51	5	3	78	77	85	123	37	84
HS61	3	2	844	700	899	NaN	NaN	NaN
HS100LNP	7	2	253	282	265	436	905	195

Table A.2: Number of function evaluations required the first version of DEFT-FUNNEL, CDFO and COBYLA methods to converge in the second type of comparison on a set of 29 equality-constrained problems.

Problem	n	m	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	CDFO	COBYLA
BT1	2	1	41	60	99	111	71	18
BT2	3	1	393	89	54	97	220	223
BT3	5	3	16	17	20	20	216	74
BT4	3	2	13	11	11	11	42	31
BT5	3	2	11	11	10	10	369	NaN
BT6	5	2	92	64	55	106	525	51
BT7	5	3	40	209	38	59	426	139
BT8	5	2	50	25	47	26	138	56
BT9	4	2	21	22	25	48	480	77
BT10	2	2	7	10	10	18	68	17
BT11	5	3	125	114	38	128	322	118
BT12	5	3	23	24	22	22	860	563
HS6	2	1	11	12	12	12	12	24
HS7	2	1	21	13	18	24	413	32
HS8	2	2	14	13	8	8	63	18
HS9	2	1	30	30	51	34	NaN	8
HS26	3	1	31	23	24	45	NaN	30
HS27	3	1	84	22	26	57	NaN	NaN
HS28	3	1	16	15	12	12	10	30
HS39	4	2	21	22	25	48	133	38
HS40	4	3	20	26	NaN	NaN	248	47
HS42	4	2	NaN	61	NaN	NaN	53	46
HS46	5	2	50	73	55	91	242	37
HS48	5	2	18	16	20	20	15	32
HS49	5	2	61	59	102	117	120	51
HS50	5	3	40	43	30	97	17	60
HS51	5	3	16	12	18	18	17	32
HS61	3	2	80	79	370	NaN	NaN	NaN
HS100LNP	7	2	202	135	112	191	763	133

Table A.3: Number of function evaluations required by the extended DEFT-FUNNEL and COBYLA methods to converge on a set of 80 problems with general nonlinear constraints.

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	COBYLA
HS6	2	0	1	0	21	27	27	32	24
HS7	2	0	1	0	31	23	25	49	32
HS8	2	0	2	0	13	13	13	15	18
HS9	2	0	1	0	24	24	57	45	8
HS10	2	0	0	1	34	34	32	25	61
HS11	2	0	0	1	26	26	26	18	53
HS12	2	0	0	1	64	59	64	47	35
HS13	2	2	0	1	36	47	32	48	50
HS14	2	0	1	1	18	17	17	18	18
HS15	2	1	0	2	40	39	28	99	122
HS16	2	3	0	2	13	13	9	9	17
HS17	2	3	0	2	34	32	51	70	30
HS18	2	4	0	2	15	77	15	95	51
HS19	2	4	0	2	17	21	17	23	28
HS20	2	2	0	3	17	17	12	15	18
HS21	2	4	0	1	14	19	61	33	25
HS22	2	0	0	2	34	28	19	16	18
HS23	2	4	0	5	36	36	43	16	18
HS24	2	2	0	3	14	14	18	21	14
HS26	3	0	1	0	109	85	98	140	30
HS27	3	0	1	0	61	79	57	116	NaN
HS28	3	0	1	0	31	29	30	31	30
HS29	3	0	0	1	59	68	77	77	69
HS30	3	6	0	1	14	16	34	46	60
HS31	3	6	0	1	63	67	63	40	56
HS32	3	3	1	1	55	36	28	28	21
HS33	3	4	0	2	17	21	40	40	25
HS34	3	6	0	2	38	38	36	40	NaN
HS35	3	3	0	1	84	101	82	71	51
HS36	3	6	0	1	18	27	20	20	31

Table A.4: Number of function evaluations required by the extended DEFT-FUNNEL and COBYLA methods to converge on a set of 80 problems with general nonlinear constraints. (Continued)

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	COBYLA
HS37	3	6	0	2	76	87	66	117	67
HS39	4	0	2	0	58	63	59	87	38
HS40	4	0	3	0	50	43	42	43	47
HS41	4	8	1	0	66	71	35	38	NaN
HS42	4	0	2	0	32	31	31	33	46
HS43	4	0	0	3	79	94	77	36	86
HS44	4	4	0	6	54	60	33	33	33
HS46	5	0	2	0	164	122	197	275	37
HS47	5	0	3	0	125	103	169	211	266
HS48	5	0	2	0	29	47	80	81	32
HS49	5	0	2	0	272	215	208	421	51
HS50	5	0	3	0	122	114	111	202	60
HS51	5	0	3	0	65	43	53	53	32
HS52	5	0	3	0	44	43	34	34	142
HS53	5	10	3	0	29	43	58	58	85
HS55	6	8	6	0	134	115	178	160	NaN
HS56	7	0	4	0	148	224	221	349	262
HS60	3	6	1	0	81	88	86	85	50
HS61	3	0	2	0	23	22	65	47	NaN
HS62	3	6	1	0	82	96	NaN	NaN	NaN
HS63	3	3	2	0	30	21	86	88	57
HS64	3	3	0	1	160	153	174	274	355
HS65	3	6	0	1	120	110	111	120	66
HS66	3	6	0	2	60	45	97	126	58
HS71	4	8	1	1	104	99	73	109	62
HS73	4	4	1	2	35	40	80	180	41
HS76	4	4	0	3	45	50	34	39	57
HS77	5	0	2	0	166	141	168	171	132
HS78	5	0	3	0	66	67	62	62	102
HS79	5	0	3	0	106	76	74	149	93

Table A.5: Number of function evaluations required by the extended DEFT-FUNNEL and COBYLA methods to converge on a set of 80 problems with general nonlinear constraints. (Continued)

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	COBYLA
HS80	5	10	3	0	59	67	91	113	100
HS81	5	10	3	0	116	257	342	NaN	156
HS93	6	6	0	2	302	313	240	284	834
HS100LNP	7	0	2	0	174	236	NaN	NaN	133
HS106	8	16	0	6	NaN	NaN	NaN	985	NaN
HS108	9	1	0	13	146	176	200	201	161
HS113	10	0	0	8	225	342	323	257	203
BT1	2	0	1	0	NaN	86	NaN	NaN	18
BT2	3	0	1	0	118	124	129	198	223
BT3	5	0	3	0	40	86	62	63	74
BT4	3	0	2	0	32	31	22	49	31
BT5	3	0	2	0	30	21	20	21	NaN
BT6	5	0	2	0	203	163	176	238	51
BT7	5	0	3	0	86	86	83	166	139
BT8	5	0	2	0	63	88	69	106	56
BT9	4	0	2	0	58	63	59	87	77
BT10	2	0	2	0	7	7	19	24	17
BT11	5	0	3	0	89	92	127	220	118
BT12	5	0	3	0	65	115	129	48	563
BT13	5	1	1	0	NaN	161	158	40	637

Table A.6: Number of function evaluations required by the extended DEFT-FUNNEL and LINCOA methods to converge on a set of 20 linearly constrained problems.

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	LINCOA
HS9	2	0	1	0	24	24	57	45	31
HS21	2	4	0	1	13	16	13	15	25
HS24	2	2	0	3	14	14	18	21	22
HS28	3	0	1	0	31	29	30	31	43
HS35	3	3	0	1	84	101	82	71	32
HS36	3	6	0	1	18	27	20	20	33
HS37	3	6	0	2	76	87	66	117	44
HS44	4	4	0	6	54	60	33	33	39
HS48	5	0	2	0	29	47	80	81	45
HS49	5	0	2	0	272	215	208	421	152
HS50	5	0	3	0	122	114	111	202	117
HS51	5	0	3	0	65	43	53	53	53
HS52	5	0	3	0	56	88	34	34	35
HS53	5	10	3	0	40	43	31	31	36
HS62	3	6	1	0	82	96	NaN	109	53
HS76	4	4	0	3	45	50	34	39	38
BT3	5	0	3	0	40	43	31	31	36
HATFLDH	4	8	0	13	34	46	35	35	38
STANCMIN	3	3	0	2	47	34	40	33	32
SIMPLLPA	2	2	0	2	15	15	9	9	25

Bibliography

C. Audet and J. E. Dennis. Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, **11**(3), 573–594, 2000.

C. Audet and J. E. Dennis. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, **14**(4), 980–1010, 2004.

C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, **17**(1), 188–217, 2006.

Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, **1**(1), 1–23, 1993.

A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. *Mathematical Programming*, **134**(1), 223–257, 2012.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies – A comprehensive introduction. *Natural Computing*, **1**(1), 3–52, 2002.

L. T. Biegler and A. Wächter. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, **16**(1), 1–31, 2005.

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.

P. T. Boggs and J. W. Tolle. A strategy for global convergence in a sequential quadratic programming algorithm. *SIAM Journal on Numerical Analysis*, **26**(3), 600–623, 1989.

G. E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **6**(2), 81–101, 1957.

G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, **13**(1), 1–45, 1951.

A. Bürmen, J. Puhan, and T. Tuma. Grid restrained Nelder-Mead algorithm. *Computational Optimization and Applications*, **34**, 359–375, 2006.

R. H. Byrd, J. Nocedal, and R. A. Waltz. Knitro: An integrated package for nonlinear optimization. *in* G. Di Pillo and M. Roma, eds, 'Large-Scale Nonlinear Optimization', Vol. 83 of *Nonconvex Optimization and Its Applications*, pp. 35–59. Springer US, 2006.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, **20**(6), 2833–2852, 2010.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. part ii: worst-case function- and derivative-evaluation complexity. *Mathematical Programming*, **130**(2), 295–319, 2011*a*.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. On the evaluation complexity of composite function minimization with applications to nonconvex nonlinear programming. *SIAM Journal on Optimization*, **21**(4), 1721–1739, 2011*b*.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. An adaptive cubic regularization algorithm for nonconvex optimization with convex constraints and its function-evaluation complexity. *IMA Journal of Numerical Analysis*, **32**(4), 1662–1695, 2012*a*.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. On the complexity of the steepest-descent with exact linesearches. Technical report, Namur Centre for Complex Systems (naXys), University of Namur, Namur, Belgium, 2012*b*.

C. Cartis, N. I. M. Gould, and Ph. L. Toint. On the complexity of finding first-order critical points in constrained nonlinear optimization. *Mathematical Programming*, **144**(1-2), 93–106, 2014.

C. Cartis, Ph.R. Sampaio, and Ph.L. Toint. Worst-case evaluation complexity of non-monotone gradient-related algorithms for unconstrained optimization. *Optimization*, **64**(5), 1349–1361, 2015.

A.-L. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte Rendu des Séances de L'Académie des Sciences XXV*, **Série A**(25), 536–538, 1847.

P. G. Ciarlet and P. A. Raviart. General Lagrange and Hermite interpolation in $\mathbb{R}^n$ with applications to finite element methods. *Arch. Ration. Mech. Anal.*, **46**, 177–199, 1972.

B. Colson. *Trust-Region Algorithms for Derivative-Free Optimization and Nonlinear Bilevel Programming.* PhD thesis, Department of Mathematics, FUNDP - University of Namur, Namur, Belgium, 2004.

A. R. Conn. Linear programming via a non-differentiable penalty function. *SIAM Journal on Numerical Analysis*, **13**, 145–154, 1976.

A. R. Conn and T. F. Coleman. Nonlinear programming via an exact penalty function: Global analysis. *Mathematical Programming*, **24**(1), 137–161, 1982.

A. R. Conn and T. Pietrzykowski. A penalty function method converging directly to a constrained optimum. *SIAM Journal on Numerical Analysis*, **14**(2), 348–375, 1977.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods.* MOS-SIAM Series on Optimization, Philadelphia, 2000.

A. R. Conn, K. Scheinberg, and Ph.L. Toint. A derivative-free optimization algorithm in practice. *in* '7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization', St. Louis, MO, 1998.

A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. *in* A. Iserles and M. Buhmann, eds, 'Approximation Theory and Optimization: Tributes to M. J. D. Powell', pp. 83–108, Cambridge, England, 1997. Cambridge University Press.

A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of interpolation sets in derivative free optimization. *Mathematical Programming, Series B*, **111**(1-2), 141–172, 2008*a*.

A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of sample sets in derivative free optimization: polynomial regression and underdetermined interpolation. *IMA Journal of Numerical Analysis*, **28**(4), 721–748, 2008*b*.

A. R. Conn, K. Scheinberg, and L. N. Vicente. Global convergence of general derivative-free trust-region algorithms to first and second order critical points. *SIAM Journal on Optimization*, **20**(1), 387–415, 2009*a*.

A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization.* MPS-SIAM Book Series on Optimization, Philadelphia, 2009*b*.

A. R. Conn, K. Scheinberg, and H. Zang. A derivative-free algorithm for least-squares minimization. *SIAM Journal on Optimization*, **20**(6), 3555–3576, 2010.

H. B. Curry. The method of steepest descent for nonlinear minimization problems. *Quaterly of Applied Mathematics*, **2**, 258–261, 1944.

F. Curtis, N. I. M. Gould, D. Robinson, and Ph. L. Toint. An interior-point trust-funnel algorithm for nonlinear optimization. Technical Report naXys-02-2014, Department of Mathematics, University of Namur, 2014.

W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, **1**, 1–17, 1991.

E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–13, 2002.

G. Fasano, J. Nocedal, and J.-L. Morales. On the geometry phase in model-based algorithms for derivative-free optimization. *Optimization Methods and Software*, **24**(1), 145–154, 2009.

R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, **91**(2), 239–269, 2002.

R. Fletcher, N. I. M. Gould, S. Leyffer, Ph. L. Toint, and A. Wächter. Global convergence of a trust-region SQP-filter algorithm for general nonlinear programming. *SIAM Journal on Optimization*, **13**(3), 635–659, 2002*a*.

R. Fletcher, S. Leyffer, and Ph. L. Toint. On the global convergence of a filter–SQP algorithm. *SIAM Journal on Optimization*, **13**(1), 44–59, 2002*b*.

R. Fletcher, S. Leyffer, D. Ralph, and S. Scholtes. Local convergence of SQP methods for mathematical programs with equilibrium constraints. *SIAM Journal on Optimization*, **17**(1), 259–286, 2006.

Ph. E. Gill and E. Wong. Sequential quadratic programming methods. *in* J. Lee and S. Leyffer, eds, 'Mixed Integer Nonlinear Programming', Vol. 154 of *The IMA Volumes in Mathematics and its Applications*, pp. 147–224. Springer New York, 2012.

Ph. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, **47**(1), 99–131, 2005.

Ph. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for npsol 5.0: A fortran package for nonlinear programming. Technical report, Systems Optimization Laboratory, Stanford University, USA, 2001.

N. I. M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. *in* M. Powell and S. Scholtes, eds, 'System Modelling and Optimization', Vol. 46 of *IFIP – The International Federation for Information Processing*, pp. 149–178. Springer US, 2000.

N. I. M. Gould and Ph. L. Toint. Nonlinear programming without a penalty function or a filter. *Mathematical Programming*, **122**(1), 155–196, 2010.

N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM J. Sci. Comput.*, **23**(4), 1376–1395, April 2001.

N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr, a constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, **29**(4), 373–394, 2003*a*.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD, a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Softw.*, **29**(4), 353–372, December 2003*b*.

N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, pp. 1–13, 2014.

S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. Technical report, Department of Mathematics, University of Coimbra, Portugal, 2014.

S. Gratton, A. Sartenaer, and Ph. L. Toint. Recursive trust-region methods for multiscale nonlinear optimization. *SIAM Journal on Optimization*, **19**(1), 414–444, 2008.

S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for bound-constrained nonlinear optimization without derivatives. *Optimization Methods and Software*, **26**(4-5), 875–896, 2011.

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.

L. Grippo, F. Lampariello, and S. Lucidi. A truncated newton method with nonmonotone line search for unconstrained optimization. *Journal of Optimization Theory and Applications*, **60**(3), 401–419, 1989.

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.

W. Hock and K. Schittkowski. Test example for nonlinear programming codes. *Journal of Optimization Theory and Applications*, **30**(1), 127–129, 1980.

J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM*, **8**, 212–229, 1961.

J. Kennedy and R. Eberhart. Particle swarm optimization. *in* 'Neural Networks, 1995. Proceedings., IEEE International Conference on', Vol. 4, pp. 1942–1948, 1995.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, **220**(4598), 671–680, 1983.

J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, **9**, 112–147, 1998.

K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, **2**, 164–168, 1944.

R. M. Lewis and V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, **9**, 1082–1099, 1999.

R. M. Lewis and V. Torczon. Pattern search algorithms for linearly constrained minimization. *SIAM Journal on Optimization*, **10**, 917–941, 2000.

R. M. Lewis and V. Torczon. A globally convergent augmented langragian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, **12**(4), 1075–1089, 2002.

R. M. Lewis and V. Torczon. Active set identification for linearly constrained minimization without explicit derivatives. *SIAM Journal on Optimization*, **20**(3), 1378–1405, 2009.

R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methos: then and now. *Journal of Computational and Applied Mathematics*, **124**(1-2), 191–207, 2000.

S. Lucidi, M. Sciandrone, and P. Tseng. Objective-derivative-free methods for constrained optimization. *Mathematical Programming, Series A*, **92**, 37–59, 2002.

M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming, Series A*, **91**(2), 289–300, 2002.

D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, **11**(2), 431–441, 1963.

D. D. Morrison. Methods for nonlinear least squares problems and convergence proofs. *in* J. Lorell and F. Yagi, eds, 'Proceedings of the Seminar on Tracking Programs and Orbit Determination', pp. 1–9, Pasadena, USA, 1960. Jet Propulsion Laboratory.

J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, **4**(3), 553–572, 1983.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, **7**, 308–313, 1965.

Y. Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

Y. Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, **108**(1), 177–205, 2006.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Series in Operations Research. Springer Verlag, Heidelberg, Berlin, New York, 1999.

E. O. Omojokun. *Trust region algorithms for optimization with nonlinear equality and inequality constraints*. PhD thesis, University of Colorado, Boulder, Colorado, USA, 1989.

J. Ortega and Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, 2000.

M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *in* 'Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico', Vol. 275, pp. 51–67, Dordrecht, The Netherlands, 1994. Kluwer Academic Publishers.

M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, **7**, 287–336, 1998.

M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic interpolation. *Mathematical Programming, Series A*, **92**, 555–582, 2002.

M. J .D. Powell. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming, Series B*, **100**(1), 183–215, 2004.

M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. *in* G. Pillo, M. Roma and P. Pardalos, eds, 'Large-Scale Nonlinear Optimization', Vol. 83 of *Nonconvex Optimization and Its Applications*, pp. 255–297. Springer US, 2006.

M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA Journal of Numerical Analysis*, **28**(4), 649–664, 2008.

M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2009.

M. J. D. Powell. On fast trust region methods for quadratic models with linear constraints. Technical Report DAMTP 2014/NA02, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2014.

C. J. Price, I. D. Coope, and D. Byatt. A convergent variant of the Nelder-Mead algorithm. *Journal of Optimization Theory and Applications*, **113**(1), 5–19, 2002.

Ph. R. Sampaio and Ph. L. Toint. A derivative-free trust-funnel method for equality-constrained nonlinear optimization. *Computational Optimization and Applications*, **61**(1), 25–49, 2015*a*.

Ph. R. Sampaio and Ph. L. Toint. Numerical experience with a derivative-free trust-funnel method for nonlinear optimization problems with general nonlinear constraints. Technical report, Namur Centre for Complex Systems (naXys), University of Namur, Namur, Belgium, 2015*b*.

Th. Sauer and X. Yuan. On multivariate Lagrange interpolation. *Mathematics of Computation*, **64**, 1147–1170, 1995.

K. Scheinberg and Ph. L. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM Journal on Optimization*, **20**(6), 3512–3532, 2010.

K. Schittkowski. NLPQL: A fortran subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, **5**(2), 485–500, 1986.

S. Singer and S. Singer. Complexity analysis of Nelder-Mead search iterations. *in* M. Rogina, V. Hari, N. Limić and Z. Tutek, eds, 'Proceedings of the 1st. Conference on Applied Mathematics and Computation', pp. 185–196, Dubrovnik, Croatia, 2001.

W. Spendley, G.R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, **4**, 441–461, 1962.

T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, **20**(3), 626–637, 1983.

Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. *in* I. S. Duff, ed., 'Sparse Matrices and Their Uses', pp. 57–88, London, 1981. Academic Press.

Ph. L. Toint. An assessment of nonmonotone linesearch techniques for unconstrained optimization. *SIAM Journal on Scientific Computing*, **17**(3), 725–739, 1996.

V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, **7**(1), 1–25, 1997.

A. Tröltzsch. *An active-set trust-region method for bound-constrained nonlinear optimization without derivatives applied to noisy aerodynamic design problems*. PhD thesis, University of Toulouse, France, 2011.

M. Ulbrich, S. Ulbrich, and L. N. Vicente. A globally convergent primal-dual interior-point filter method for nonlinear programming. *Mathematical Programming*, **100**(2), 379–410, 2004.

R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, **12**, 451–484, 1999.

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.

R. B. Wilson. *A simplicial algorithm for concave programming*. PhD thesis, Harvard University, Massachusetts, USA, 1963.

D. Winfield. *Function and functional optimization by interpolation in data tables*. PhD thesis, Harvard University, Cambridge, USA, 1969.

D. Winfield. Function minimization by interpolation in a data table. *Journal of the Institute of Mathematics and its Applications*, **12**, 339–347, 1973.

W.-C. Yu. The convergent property of the simplex evolutionary techniques. *Scientia Sinica, Special Issue of Mathematics*, **1**, 68–77, 1979*a*.

W.-C. Yu. Positive basis and a class of direct search techniques. *Scientia Sinica, Special Issue of Mathematics*, **1**, 53–67, 1979*b*.

W.-C. Yu and Y.-X. Li. A direct search method by the local positive basis for the nonlinearly constrained optimization. *Chinese Annals of Mathematics*, **2**, 269–280, 1981.