An active-set trust-region method for bound-constrained nonlinear optimization without derivatives applied to noisy aerodynamic design problems

Anke Tröltzsch

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Mathématiques Informatique Télécommunications

Présentée et soutenue par :
Anke TRÖLTZSCH

le : mardi 7 juin 2011

Titre :

An active-set trust-region method for bound-constrained nonlinear optimization without derivatives
applied to noisy aerodynamic design problems

Ecole doctorale :
Mathématiques Informatique Télécommunications (MITT)

Unité de recherche :
CERFACS

Directeur(s) de Thèse :
Serge Gratton
Philippe Toint

Rapporteurs :

Luis Nunes Vicente
Jean-Charles Gilbert

Membre(s) du jury :

Bijan Mohammadi, Président du jury
Matthieu Meaux, Membre

THÈSE

présentée en vue de l'obtention du titre de

DOCTEUR DE L'UNIVERSITÉ DE TOULOUSE

Spécialité : Mathématiques, Informatique et Télécommunications

par

Anke TRÖLTZSCH

CERFACS

Une méthode de région de confiance avec ensemble actif pour l'optimisation non linéaire sans dérivées avec contraintes de bornes appliquée à des problèmes aérodynamiques bruités

soutenue le 7 juin 2011 devant le jury composé de :

Serge Gratton	Directeur de thèse	IRIT et CERFACS
Phillipe L. Toint	Co-directeur de thèse	Université de Namur
Luis Nunes Vicente	Rapporteur	Université de Coimbra
Jean-Charles Gilbert	Rapporteur	INRIA Paris - Rocquencourt
Bijan Mohammadi	Examinateur	CERFACS
Matthieu Meaux	Examinateur	Airbus France

Résumé

L'optimisation sans dérivées (OSD) a connu un regain d'intérêt ces dernières années, principalement motivée par le besoin croissant de résoudre les problèmes d'optimisation définis par des fonctions dont les valeurs sont calculées par simulation (par exemple, la conception technique, la restauration d'images médicales ou de nappes phréatiques).

Ces dernières années, un certain nombre de méthodes d'optimisation sans dérivée ont été développées et en particulier des méthodes fondées sur un modèle de région de confiance se sont avérées obtenir de bons résultats.

Dans cette thèse, nous présentons un nouvel algorithme de région de confiance, basé sur l'interpolation, qui se montre efficace et globalement convergent (en ce sens que sa convergence vers un point stationnaire est garantie depuis tout point de départ arbitraire). Le nouvel algorithme repose sur la technique d'auto-correction de la géométrie proposé par Scheinberg and Toint (2010). Dans leur théorie, ils ont fait avancer la compréhension du rôle de la géométrie dans les méthodes d'OSD à base de modèles. Dans notre travail, nous avons pu améliorer considérablement l'efficacité de leur méthode, tout en maintenant ses bonnes propriétés de convergence. De plus, nous examinons l'influence de différents types de modèles d'interpolation sur les performances du nouvel algorithme.

Nous avons en outre étendu cette méthode pour prendre en compte les contraintes de borne par l'application d'une stratégie d'activation. Considérer une méthode avec ensemble actif pour l'optimisation basée sur des modèles d'interpolation donne la possibilité d'économiser une quantité importante d'évaluations de fonctions. Il permet de maintenir les ensembles d'interpolation plus petits tout en poursuivant l'optimisation dans des sous-espaces de dimension inférieure. L'algorithme résultant montre un comportement numérique très compétitif. Nous présentons des résultats sur un ensemble de problèmes-tests issu de la collection CUTEr et comparons notre méthode à des algorithmes de référence appartenant à différentes classes de méthodes d'OSD.

Pour réaliser des expériences numériques qui intègrent le bruit, nous créons un ensemble de cas-tests bruités en ajoutant des perturbations à l'ensemble des problèmes sans bruit. Le choix des problèmes bruités a été guidé par le désir d'imiter les problèmes d'optimisation basés sur la simulation. Enfin, nous présentons des résultats sur une application réelle d'un problème de conception de forme d'une aile fourni par Airbus.

Mots-clés:
optimisation sans dérivées, région de confiance, contraintes de borne, fonctions bruitées

Abstract

Derivative-free optimization (DFO) has enjoyed renewed interest over the past years, mostly motivated by the ever growing need to solve optimization problems defined by functions whose values are computed by simulation (e.g. engineering design, medical image restoration or groundwater supply).

In the last few years, a number of derivative-free optimization methods have been developed and especially model-based trust-region methods have been shown to perform well.

In this thesis, we present a new interpolation-based trust-region algorithm which shows to be efficient and globally convergent (in the sense that its convergence is guaranteed to a stationary point from arbitrary starting points). The new algorithm relies on the technique of self-correcting geometry proposed by Scheinberg and Toint [128] in 2009. In their theory, they advanced the understanding of the role of geometry in model-based DFO methods, in our work, we improve the efficiency of their method while maintaining its good theoretical convergence properties. We further examine the influence of different types of interpolation models on the performance of the new algorithm.

Furthermore, we extended this method to handle bound constraints by applying an active-set strategy. Considering an active-set method in bound-constrained model-based optimization creates the opportunity of saving a substantial amount of function evaluations. It allows to maintain smaller interpolation sets while proceeding optimization in lower dimensional subspaces. The resulting algorithm is shown to be numerically highly competitive. We present results on a test set of smooth problems from the CUTEr collection and compare to well-known state-of-the-art packages from different classes of DFO methods.

To report numerical experiments incorporating noise, we create a test set of noisy problems by adding perturbations to the set of smooth problems. The choice of noisy problems was guided by a desire to mimic simulation-based optimization problems. Finally, we will present results on a real-life application of a wing-shape design problem provided by Airbus.

Keywords:
derivative-free optimization, trust-region method, bound constraints, noisy problems

This dissertation would not have been possible without the support of many individuals. First of all, I want to thank my advisors, Serge Gratton and Philippe L. Toint, for four years of teaching, mentoring, and challenging me. I appreciated very much to work with them. I thank the referees, Luis Nunes Vicente and Jean-Charles Gilbert, and the other members of the jury, Bijan Mohammadi and Matthieu Meaux, for accepting to evaluate my work.

The people at CERFACS, and in particular the members of the Algo team, have all been very pleasant to work and learn with, and I thank all of them for a wonderful graduate experience. Especially, I want to mention Selime who is the new sunshine of the Algo team. She has always been very helpful and I wish that one day I can help her as much as she did.

I also want to thank the people from the CFD team of CERFACS, in particular Jean-Francois Boussuge and Julien Laurenceau, and people from Airbus France, in particular Pascal Larrieu and Matthieu Meaux for their support and for helpful discussions concerning the use of optimization software in aircraft design.

I would like to thank my parents, Marlies and Jörg Tröltzsch, for supporting me during all my projects and for giving me the freedom to find my own way. Moreover, I want to thank my father for proofreading this thesis.

Last but not least, I want to thank Christian wholeheartedly for his patience, support and constant encouragement.

Contents

1 Introduction **1**

2 Model-based derivative-free optimization **5**

2.1 Basic material . . . 6

2.1.1 The trust-region framework . . . 7

2.1.2 Polynomial interpolation . . . 8

2.1.3 Lagrange polynomials and Λ-poisedness . . . 8

2.1.4 Condition number as a measure of well-poisedness . . . 11

2.2 Polynomial interpolation and regression models . . . 12

2.2.1 Interpolation sets of variable size . . . 12

2.2.2 Sub-basis model . . . 13

2.2.3 Minimum ℓ_2-norm model . . . 14

2.2.4 Minimum Frobenius-norm model . . . 14

2.2.5 Least-change Frobenius-norm model . . . 15

2.2.6 Minimum ℓ_1-norm model . . . 16

2.2.7 Least-squares regression model . . . 16

2.3 A basic interpolation-based trust-region approach . . . 17

2.4 Self-correcting geometry - a recent approach . . . 20

2.4.1 The algorithm . . . 20

2.4.2 Convergence theory . . . 20

2.5 A modified algorithm with self-correcting geometry . . . 24

2.5.1 The algorithm . . . 24

2.5.2 Modifications . . . 24

2.5.3 Global convergence . . . 26

2.5.4 Outlook . . . 30

3 A bound-constrained DFO algorithm **31**

3.1 A recursive active-set trust-region algorithm . . . 32

3.1.1 Outline of the algorithm . . . 32

3.1.2 Ensuring suitability of a tentative interpolation set . . . 32

3.1.3 Recursive call in the subspace $\mathbf{S_k}$ . . . 35

3.1.4 Local solver . . . 37

3.1.5 Defining the next iterate . . . 37

3.1.6 Avoid re-entering a subspace . . . 40

3.2 Theoretical issues . . . 41

3.2.1 Global convergence . . . 41

3.3 Practical implementation issues . . . 42

3.3.1 Preparing the initial call . . . 42

3.3.2 Handling fixed variables . . . 42

3.3.3 Representation of the Lagrange polynomials . . . 43
3.3.4 Controlling the condition of the system matrix . . . 43
3.3.5 Implementation of the models . . . 47
3.3.6 The projected gradient as a stopping criterion . . . 48
3.3.7 An alternative stopping criterion . . . 49
3.4 Solving the bound-constrained trust-region subproblem in ℓ_2-norm . . . 51
3.4.1 The Moré-Sorensen algorithm . . . 52
3.4.2 The extension of the Moré-Sorensen algorithm to handle bound-constraints 54
3.5 Numerical experiments in the CUTEr testing environment . . . 57
3.5.1 Default parameters . . . 57
3.5.2 Test problems . . . 58
3.5.3 A common stopping criterion . . . 60
3.5.4 Performance profiles . . . 60
3.5.5 Comparison of different model types . . . 61
3.5.6 Comparison of local solver options . . . 62
3.5.7 Unconstrained comparisons with NEWUOA . . . 63
3.5.8 Bound-constrained comparisons with BOBYQA . . . 68
3.5.9 Unconstrained comparisons with direct-search solvers . . . 73
3.5.10 Bound-constrained comparisons with direct-search solvers . . . 77

4 Industrial application incorporating noise 83
4.1 Presentation of the optimization suite OPTaliA . . . 84
4.1.1 Optimization framework . . . 84
4.1.2 Evaluator for CFD functions . . . 85
4.1.2.1 Shape parameterization and mesh deformation . . . 85
4.1.2.2 Flow simulation . . . 86
4.1.2.3 Aerodynamic function computation . . . 87
4.1.3 Interface Matlab – OPTaliA . . . 87
4.2 Estimating noise by Hamming's difference table . . . 88
4.2.1 The idea . . . 88
4.2.2 Case study of an aerodynamic function and gradient . . . 89
4.3 Numerical experiments on noisy CUTEr test problems . . . 92
4.3.1 Noisy test problems . . . 92
4.3.2 Stopping criterion for the comparison . . . 93
4.3.3 Data profiles . . . 93
4.3.4 Comparison of different model types on unconstrained problems . . . 94
4.3.5 Comparison of different model types on bound-constrained problems . . . 97
4.4 Numerical experiments on an aerodynamical application . . . 99
4.4.1 Stopping criterion using noise estimation . . . 99
4.4.2 Reference optimizer (DOT-BFGS) . . . 101

4.4.3 Test case: Airfoil drag minimization . . . 101
4.4.4 Numerical results . . . 102
4.5 Modelling the allowed gradient noise in a gradient-based line search method . . . 103
4.5.1 Getting a descent direction in the presence of a noisy gradient . . . 104
4.5.1.1 Deterministic properties . . . 104
4.5.1.2 Statistical approach . . . 107
4.5.1.3 Numerical illustration . . . 108
4.5.2 Global convergence in the presence of a noisy gradient . . . 109
4.5.2.1 Deterministic properties . . . 110
4.5.2.2 Statistical approach . . . 112
4.5.2.3 Numerical illustration . . . 113
4.5.2.4 Numerical example of an aerodynamical test case . . . 114
4.5.3 Conclusions . . . 115
4.6 Enhancing a DFO method by inexact gradient information in the context of global optimization . . . 116
4.6.1 The algorithm SNOBFIT . . . 116
4.6.2 Modifications . . . 119
4.6.3 Experiments on academic and aerodynamic test problems . . . 120
4.6.4 Conclusions and perspectives . . . 122

5 Conclusions and future work **123**

A Test problems **127**

B Test results **131**

C French summary of the thesis **147**

Bibliography **151**

List of Figures

3.1 Minimization of a quadratic model inside the ℓ_2-norm constraint and the bounds 54
3.2 Comparison of different models in ***BCDFO+*** on unconstrained CUTEr problems 61
3.3 Comparison of local solvers in ***BCDFO+*** on unconstrained CUTEr problems . . 62
3.4 Comparison of local solvers in ***BCDFO+*** on bound-constrained CUTEr problems 63
3.5 Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (2 digits of accuracy required in final function value) 64
3.6 Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (4 digits of accuracy required in final function value) 64
3.7 Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (6 digits of accuracy required in final function value) 65
3.8 Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (8 digits of accuracy required in final function value) 65
3.9 Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (2 digits of accuracy required in final function value) 66
3.10 Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (4 digits of accuracy required in final function value) 67
3.11 Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (6 digits of accuracy required in final function value) 67
3.12 Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (8 digits of accuracy required in final function value) 68
3.13 Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (2 digits of accuracy required in final function value) 69
3.14 Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (4 digits of accuracy required in final function value) 69
3.15 Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (6 digits of accuracy required in final function value) 70
3.16 Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (8 digits of accuracy required in final function value) 70
3.17 Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (2 digits of accuracy required in final function value) 71
3.18 Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (4 digits of accuracy required in final function value) 71
3.19 Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (6 digits of accuracy required in final function value) 72
3.20 Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (8 digits of accuracy required in final function value) 72
3.21 Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (2 digits of accuracy required in final function value) 73

3.22 Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (4 digits of accuracy required in final function value) 74
3.23 Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (6 digits of accuracy required in final function value) 74
3.24 Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (8 digits of accuracy required in final function value) 75
3.25 Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (2 digits of accuracy required in final function value) 75
3.26 Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (4 digits of accuracy required in final function value) 76
3.27 Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (6 digits of accuracy required in final function value) 76
3.28 Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (8 digits of accuracy required in final function value) 77
3.29 Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (2 digits of accuracy required in final function value) 78
3.30 Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (4 digits of accuracy required in final function value) 78
3.31 Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (6 digits of accuracy required in final function value) 79
3.32 Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (8 digits of accuracy required in final function value) 79
3.33 Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (2 digits of accuracy required in final function value) 80
3.34 Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (4 digits of accuracy required in final function value) 81
3.35 Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (6 digits of accuracy required in final function value) 81
3.36 Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (8 digits of accuracy required in final function value) 82

4.1 Optimization and evaluation framework in OPTaliA 84
4.2 Illustration of the parameters β and p 85
4.3 Two-dimensional Hicks-Henne sinusoidal bump 86
4.4 Navier Stokes pressure drag objective function 90
4.5 Navier Stokes pressure drag adjoint state gradient 91

4.6 Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 200$) . . . 94
4.7 Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 200$) . . . 95
4.8 Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 15000$) . . . 96
4.9 Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 15000$) . . . 96
4.10 Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 200$) . . . 97
4.11 Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 200$) . . . 98
4.12 Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 15000$) . . . 98
4.13 Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 15000$) . . . 99
4.14 Convergence histories on 3-dim. aerodynamic test case . . . 102
4.15 Convergence histories on 9-dim. aerodynamic test case . . . 103
4.16 Regularization for Powell 2D problem with $noiseg = 10^{-3}$ . . . 106
4.17 Allowed noise level for Property 4.2 and P_Ass1=99% . . . 108
4.18 Allowed noise level for Property 4.2 and P_Ass1=99% for bigger gradient norm . 108
4.19 Allowed noise level for Property 4.2 and P_Ass1=99.99% . . . 109
4.20 Allowed noise level for Property 4.4 and P_Ass2&3=99% . . . 113
4.21 Example of CDP adjoint state gradient . . . 114
4.22 Comparison on minimal surface problem . . . 121
4.23 Comparison on a Euler CDP problem . . . 122

List of Tables

3.1 Gradient accuracy of ill-conditioned problem PALMER3C 50
3.2 Gradient accuracy of well-conditioned problem ALLINITU 50
3.3 Dimensions of considered unconstrained problems 58
3.4 Hessian structure of considered unconstrained problems 59
3.5 Dimensions of considered bound-constrained problems 59

4.1 Difference table by Hamming 88
4.2 Noise levels in CDP obtained by different strategies and sampling distances h . . 91
4.3 Difference table for the gradient of CDP with $h = 10^{-6}$ 91

A.1 Considered unconstrained CUTEr test problems 128
A.2 Considered bound-constrained CUTEr test problems 129

B.1 Results from comparison of different types of models in ***BCDFO+*** on unconstrained CUTEr problems (see Figure 3.2) 132
B.2 Results from comparison of local solvers BC-MS and TCG in ***BCDFO+*** on unconstrained CUTEr problems (see Figure 3.3) 133
B.3 Results from comparison of local solvers BC-MS and TCG in ***BCDFO+*** on bound-constrained CUTEr problems (see Figure 3.4) 134
B.4 Results from comparison of ***BCDFO+***, BC-DFO and NEWUOA on unconstrained CUTEr problems (see Figures 3.5-3.8 and Figures 3.9-3.12) 135
B.5 Results from comparison of ***BCDFO+***, BC-DFO and BOBYQA on bound-constrained CUTEr problems (see Figures 3.13-3.16 and Figures 3.17-3.20) . . . 136
B.6 Results from comparison of ***BCDFO+***, BC-DFO, SID-PSM, NOMADm and BFO on unconstrained CUTEr problems (see Figures 3.21-3.24 and Figures 3.25-3.28) 137
B.7 Results from comparison of ***BCDFO+***, BC-DFO, SID-PSM, NOMADm and BFO on bound-constrained CUTEr problems (see Figures 3.29-3.32 and Figures 3.33-3.36) 138
B.8 Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.6) 139
B.9 Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.7) 140
B.10 Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.8) 141
B.11 Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.9) 142
B.12 Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.10) 143

B.13 Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.11) 144
B.14 Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.12) . . 145
B.15 Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.13) . . 146

List of Algorithms

2.1 Improving well-poisedness via Lagrange polynomials 11
2.2 Basic DFO trust-region algorithm 18
2.3 UDFO trust-region algorithm with self-correcting geometry from [128] 21
2.4 UDFO+ modified algorithm with self-correcting geometry 25
3.1 ***BCDFO+*** $(\mathcal{S}_0, \mathcal{X}_0, x_0, \mathcal{Z}_0, \Delta_0, \epsilon_0, \epsilon)$ 33
3.2 Modified greedy algorithm for selecting a well-poised interpolation set (Inputs: $x_0, \mathcal{Z}_0$, Outputs: $\mathcal{W}_p, p$) 34
3.3 Define the next iterate 39
3.4 The l_2-norm trust-region Moré-Sorensen algorithm 53
3.5 BC-MS: Bound-constrained l_2-norm trust-region algorithm 55
4.1 SNOBFIT 118

Chapter 1

Introduction

Derivative-free optimization has enjoyed renewed interest over the past years, mostly motivated by the ever growing need to solve optimization problems defined by functions whose evaluation is computationally expensive (e.g. engineering design optimization, medical image restoration or groundwater parameter identification). These expensive optimization problems arise in science and engineering because evaluation of the function f often requires a complex deterministic simulation which is based on solving the equations that describe the underlying physical phenomena (for example ordinary or partial differential equations). The computational noise associated with these complex simulations means that obtaining derivatives is difficult and most of the time unreliable, stimulating a growing interest in derivative-free optimization.

In the last few years, a number of derivative-free optimization methods have been developed and especially model-based trust-region methods have been shown to perform quite satisfactory. These methods can be mainly classified into methods which target good practical performance and which, up to now, are only partially covered by a convergence theory and the other type of methods for which global convergence was shown but at the expense of efficiency (globally convergent in the sense that convergence is guaranteed to a stationary point from arbitrary starting points).

Many of these model-based trust-region methods construct local polynomial interpolation or regression models of the objective function and compute steps by minimizing these models inside a region using the standard trust-region methodology (see [34] for detailed information). The models are built so as to interpolate previously computed function values at past iterates or at specially constructed points. For the model to be well-defined, the interpolation points must be poised [39, 116], meaning that the geometry of this set of points has to "cover the space" sufficiently well to stay safely away from degeneracy of the interpolation conditions. To maintain a good poisedness of the set, geometry improving steps are included in many model-based DFO algorithms, but their necessity has recently been questioned by Fasano, Nocedal and Morales [58] in that a simple method not using them at all has shown surprisingly good performance. However, it has been shown by Scheinberg and Toint [128] that convergence from arbitrary starting points may then be lost, but that a new algorithm can be designed to substantially reduce the need of such geometry improving steps by exploiting a self-correcting mechanism of the interpolation set geometry in the trust-region method.

In their theory, they advance the understanding of the role of geometry in model-based DFO methods, whereas in our work, we try to improve the efficiency of their method while maintaining its good convergence properties. In this thesis, we present a new interpolation-

based trust-region algorithm for unconstrained optimization, called UDFO+, which relies on the self-correcting property from [128] to have a certain control on the poisedness of the interpolation set, but ignores geometry considerations in the trust-region management by applying a more standard trust-region management as is done in [58]. Such a trade-off in terms of geometry control seems promising and let us expect some performance improvement.

In Chapter 2, we give a short overview of existing derivative-free optimization methods and their classification. We present the general framework of trust-region methods and the particularities when applying such a method in a derivative-free context. We recall elements of multivariate polynomial interpolation theory and in particular different types of local polynomial interpolation and regression models. We state a basic model-based trust-region approach and the recent approach of Scheinberg and Toint [128] applying a self-correcting property of the interpolation set geometry before we present our new algorithm UDFO+ for which we prove global convergence to first order stationary points.

Having in mind to use this algorithm to solve real-life applications, it is crucial to consider the case where bounds are imposed on the variables, what may also correspond to restricting the domain to a region where the models are well-defined, or to provide information on the localisation of the minimizer. Hence, we extended this method to handle bound constraints which is the main contribution in this thesis and is presented in Chapter 3. The extension of such a derivative-free trust-region method to bound-constrained problems seems obvious but is in fact not as straightforward in practice as one could think. The difficulty is that the set of interpolation points may get aligned at one or more active bounds and deteriorate the quality of the interpolation set. This led to the idea of applying an active-set strategy to pursue minimization in the subspace of free variables to circumvent this difficulty. Moreover, considering an active-set method in model-based derivative-free optimization creates the opportunity of saving a considerable amount of function evaluations because such a method allows to maintain smaller interpolation sets while proceeding optimization in lower dimensional subspaces. We outline the basic framework of our algorithm, called ***BCDFO+***, and discuss its algorithmic concepts together with some practical implementation issues and consider global convergence issues.

One of the main ingredients of a trust-region method is the local solver used to find the minimizer of the trust-region subproblem at each iteration. We present the new algorithm BC-MS, to solve the bound-constrained trust-region subproblem in ℓ_2-norm. This is in general a challenging task as the intersection of a box and a ball is not a simple set to deal with. Although it is common now to apply infinity-norm trust-regions (for which an intersection with the bounds is again a box), we want to revisit the possibility of solving the subproblems in ℓ_2-norm using factorization.

To assess the performance of the software implementation of our algorithm ***BCDFO+***, we use a test set of smooth problems from the CUTEr testing collection [69]. We report numerical experiments where we first assess different types of polynomial models and compare our

new local solver BC-MS to a truncated conjugate gradient method to find the most suitable options for our trust-region algorithm. Secondly, we compare ***BCDFO+*** to NEWUOA [123] and BOBYQA [124], two state-of-the-art packages applying also a trust-region method using interpolation-based models. Thirdly, we compare our algorithm to three different software packages from the class of direct search methods.

Still having in mind to use this algorithm to solve real-life applications (as for instance an aerodynamic shape optimization problem provided by Airbus), it is important to study the impact of noise on optimization algorithms in general and to adapt our algorithm to handle noisy optimization problems. This work is presented in Chapter 4.

Aerospace industry is increasingly relying on advanced numerical flow simulation tools in the early aircraft design phase. Today's flow solvers based on the solution of the Euler and Navier-Stokes equations are able to predict aerodynamic behaviour of aircraft components under different flow conditions quite well. But numerical approximations to differential equations are often quite noisy. Adaptive methods, partial convergence, and stabilization strategies are all useful techniques in this respect, but these approaches create noise and difficulties for many optimization algorithms.

We also contribute to different aspects of studies concerning noise in general and the noisy aerodynamic application in particular. We give a short description of the optimization tool OPTaliA which is used at Airbus to perform aerodynamic shape optimization. As the used flow simulation tool provides the objective function and the adjoint gradient where the accuracy for both is unknown, we demonstrate how the level of a low-amplitude noise in a function or a gradient can be estimated using a tool which was originally developed to calculate higher order derivatives and to estimate round-off. We assess different types of interpolation and regression models inside our algorithm ***BCDFO+*** to solve noisy optimization problems from the CUTEr library and an aerodynamic shape-design problem provided by Airbus. We present a theoretical study on the allowed noise on a gradient which is used in a gradient-based line search method. Further, the derivative-free method SNOBFIT, developed by Huyer and Neumaier [86, 87], is presented in the context of global optimization and we show the performance gain by enhancing this method with inexact gradient information.

Finally, we draw some conclusions and give perspectives in Chapter 5.

Chapter 2

Model-based derivative-free optimization

Several methods have been proposed to minimize differentiable functions where derivatives are not provided. These methods can be divided into three different classes. First, there are those that compute the derivatives of the function, either by approximation, for example by finite differences (see for instance Gill et al. [65, 64], Dennis and Schnabel [49] and Nocedal and Wright [106]), or by automatic differentiation procedures (see for instance Griewank and Corliss [78], Gilbert [63] and Griewank [76, 77]) or by computing a gradient based on solving the differential equations if the problem depends on the solution of a system of differential equations (see continuous adjoint computations in [18, 66, 135]).

The second class of methods are direct-search methods whose distinctive feature is that their algorithmic actions and decisions are only based on simple comparison rules of objective functions, without explicitely approximating derivatives or building models. Important examples of this class of methods include the Nelder-Mead algorithm (see Nelder and Mead [104], Kelley [89] and Singer and Singer [129]) and, more recently, pattern search and generalized pattern search methods (see Torczon [133], Lewis and Torczon [93, 94, 95], Hough, Kolda and Torczon [85], Abramson [2, 3], Gray and Kolda [75]). A further generalization of pattern search methods is the recent development of mesh adaptive direct search methods (see Audet and Dennis [10, 11], Abramson and Audet [5], Audet and Orban [12], Audet, Béchard and Le Digabel [9], Abramson, Audet, Dennis and Le Digabel [6]). Furthermore, belonging to the class of direct-search methods, a number of hybrid methods has been developed and implemented to enhance the efficiency of this type of methods. Known examples include the softwares SID-PSM [45] (developed by Custódio and Vicente) and NOMADm [4] (developed by Abramson), where in the former package minimum Frobenius-norm models are formed to speed up the direct-search run and in the latter one different types of surrogates can be used in a mesh adaptive direct search filter method.

The third class of methods, and the one we are going to explore further in this thesis, is the class of model-based methods. They have been pioneered by Winfield [141, 142] and Powell [114, 115, 117, 119, 120]. Several such methods for solving unconstrained and constrained optimization problems without derivatives are available today. In particular interpolation-based trust-region methods have been shown to be numerically efficient compared to methods from

the class of direct-search methods [102]. Trust-region methods building a model by polynomial interpolation have been developed by a number of authors (see Conn and Toint [43], Conn, Scheinberg and Toint [35, 36], Colson [27], Driessen [55], Conn, Scheinberg and Vicente [38, 39, 42], Wild [138], Fasano, Morales and Nocedal [58] and Scheinberg and Toint [128]).

Some authors developed methods to take advantage of a special problem structure in order to treat larger problems. Colson and Toint [28] exploit band structure in unconstrained derivative-free optimization problems. Further, one could make use of the sparse structure of the problem to be solved. In particular, in discretized problems, this structure is sometimes well defined, in that the sparsity pattern of the Hessian is known (see Colson and Toint 2002 [29]). Another strategy can be used when the function to be minimized is partially separable (see Colson and Toint [30], using the technique which has been introduced by Griewank and Toint [79]).

Interpolation-based methods are also widely used in practice (see, for instance, Conn, Scheinberg and Toint with their software DFO [37], Marazzi and Nocedal with their software WEDGE [97] and Powell with his software implementations UOBYQA [118], NEWUOA [121, 122, 123] and BOBYQA [124], Berghen and Bersini with their package CONDOR [15]).

Another direction to pursue in model-based derivative-free trust-region optimization was to incorporate other models than polynomials, for instance, such like radial-basis functions (RBF) (see Wild [137], Wild and Shoemaker [140], Wild, Regis and Shoemaker with their software ORBIT [139], Oeuvray [107], Oeuvray and Bierlaire [108, 109] with their software implementation BOOSTERS [110]).

Finally, we want to point out that direct-search and model-based methods are discussed extensively, in theory and practice, in the recent book by Conn, Scheinberg and Vicente [41], a comprehensive introduction to derivative-free optimization.

2.1 Basic material

Many interpolation-based trust-region methods construct local polynomial interpolation-based models of the objective function and compute steps by minimizing these models inside a region using the standard trust-region methodology (see [34] for detailed information). The models are built so as to interpolate previously computed function values at a subset of past iterates or at specially constructed points. For the model to be well-defined, the interpolation points must be poised [39, 116], meaning that the geometry of this set of points has to "span the space" sufficiently well to stay safely away from degeneracy. To provide the reader with some necessary information about used notations, we have to recall some basic material about the general trust-region framework, multivariate interpolation, Lagrange polynomials and the definition of poisedness and well-poisedness.

2.1.1 The trust-region framework

In this chapter, we consider the unconstrained optimization problem

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

where f is a nonlinear function from $\mathbb{R}^n$ into $\mathbb{R}$, which is bounded below. We are going to extend this formulation to bound constraints in Chapter 3.

We first briefly recall the general trust-region framework where derivatives of f are available before turning to the derivative-free case. At each iteration of an iterative trust-region method, a model of the form

$$m_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T H_k s \tag{2.2}$$

(where g_k and H_k are the function's gradient and Hessian, respectively) is minimized inside a trust region

$$\mathcal{B}_\infty(x_k, \Delta_k) = \{x \in \mathbb{R}^n \mid \|x - x_k\|_\infty \leq \Delta_k\}, \tag{2.3}$$

where $\|\cdot\|_\infty$ denotes the infinity norm. Note that other choices of norms are possible but that the infinity norm is especially well suited when considering bound constraints (as we do later in this thesis) because the intersection of the box representing the trust region and the bound-constraints is again a box and there exist efficient algorithms to minimize a quadratic function in a box.

This (possibly approximate) minimization yields a trial point $x_k + s_k$, which is accepted as the new iterate if the ratio

$$\rho_k \stackrel{\text{def}}{=} \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \tag{2.4}$$

is larger than a constant $\eta_1 > 0$. In this case, the model is updated and the trust-region radius is possibly increased. If $\rho_k \leq \eta_1$, the trial point is rejected and radius Δ_k is decreased. Methods of this type have long been considered for the solution of numerical optimization problems, and we refer the reader to [34] for an extensive coverage of this topic.

In our derivative-free context, the model (2.2) will be determined by interpolating known objective function values at a given set $\mathcal{Y}_k$ of *interpolation points*, meaning that the *interpolation conditions*

$$m_k(y) = f(y) \text{ for all } y \in \mathcal{Y}_k \tag{2.5}$$

must hold. The set $\mathcal{Y}_k$ is known as the *interpolation set*. The question is now under which condition on $\mathcal{Y}_k$ can an (accurate enough) interpolation model be (numerically safely) computed? The answer to this question is well-known and will be provided in Section 2.1.3 after recalling some basic concepts about multivariate interpolation. The subscript k is dropped in the following description for clarity; without loss of information since we make a focus on a given iteration of the trust-region algorithm.

2.1.2 Polynomial interpolation

Consider $\mathcal{P}_n^d$, the space of polynomials of degree $\leq d$ in $\mathbb{R}^n$. A polynomial basis $\phi(x) = \{\phi_1(x), \phi_2(x), \ldots, \phi_q(x)\}$ of $\mathcal{P}_n^d$ is a set of q polynomials of degree $\leq d$ that span $\mathcal{P}_n^d$ where we know that $q = n + 1$ for $d = 1$ and $q = \frac{1}{2}(n + 1)(n + 2)$ for $d = 2$ and $q = \binom{n}{+}dd$ in the general case. Well-known examples of such bases are the basis of monomials, also called the natural basis, and bases of Lagrange or Newton fundamental polynomials. For any basis $\phi(x)$, any polynomial $m(x) \in \mathcal{P}_n^d$ can be written uniquely as

$$m(x) = \sum_{j=1}^{q} \alpha_j \phi_j(x), \tag{2.6}$$

where α_j are real coefficients.

Given an interpolation set $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\} \subset \mathbb{R}^n$ and a polynomial $m(x)$ of degree d in $\mathbb{R}^n$ that interpolates $f(x)$ at the points of $\mathcal{Y}$, the coefficients $\alpha_1, \ldots, \alpha_q$ can be determined by solving the linear system

$$M(\phi, \mathcal{Y})\alpha_\phi = f(\mathcal{Y}), \tag{2.7}$$

where

$$M(\phi, \mathcal{Y}) = \begin{pmatrix} \phi_1(y^1) & \phi_2(y^1) & \cdots & \phi_q(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \cdots & \phi_q(y^2) \\ \vdots & \vdots & & \vdots \\ \phi_1(y^p) & \phi_2(y^p) & \cdots & \phi_q(y^p) \end{pmatrix}, \quad f(\mathcal{Y}) = \begin{pmatrix} f(y^1) \\ f(y^2) \\ \vdots \\ f(y^p) \end{pmatrix}. \tag{2.8}$$

We define the set of points $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\}$ to be *poised* for polynomial interpolation in $\mathbb{R}^n$ if the coefficient matrix $M(\phi, \mathcal{Y})$ of the system is nonsingular. How to choose this set of points is of course one of the main issues we have to address below, as not every set $\mathcal{Y}$ is suitable to ensure *poisedness*.

2.1.3 Lagrange polynomials and Λ-poisedness

If the interpolation set $\mathcal{Y}$ is poised, the basis of Lagrange polynomials $\{\ell_i(x)\}_{i=1}^p$ exists and is uniquely defined.

Definition 2.1. *Given a set of interpolation points $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\}$, a basis of p polynomials $\ell_j(x), j = 1, \ldots, p$ in $\mathcal{P}_n^d$ is called a basis of Lagrange polynomials if*

$$\ell_j(y^i) = \delta_{ij} = \begin{cases} 1 \text{ if } i = j, \\ 0 \text{ if } i \neq j. \end{cases} \tag{2.9}$$

The unique polynomial $m(x)$ which interpolates $f(x)$ on $\mathcal{Y}$ using this basis of Lagrange polynomials can be expressed as

$$m(x) = \sum_{i=1}^{p} f(y^i)\, \ell_i(x). \tag{2.10}$$

Moreover, for every poised set $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\}$, we have that

$$\sum_{i=1}^{p} \ell_i(x) = 1 \quad \text{for all} \quad x \in \mathbb{R}^n. \tag{2.11}$$

The accuracy of $m(x)$ as an approximation of the objective function f in some region $\mathcal{B} \subset \mathbb{R}^n$ can be quantified using the following notion [41]. A poised set $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\}$ is said to be Λ-poised in $\mathcal{B}$ for some $\Lambda > 0$ if and only if for the basis of Lagrange polynomials associated with $\mathcal{Y}$

$$\Lambda \geq \max_{1 \leq i \leq p} \max_{x \in \mathcal{B}} |\ell_i(x)|. \tag{2.12}$$

The right hand side of (2.12) is related to the Lebesgue constant Λ_n of the set which is defined as

$$\Lambda_n = \max_{x \in \mathcal{B}} \sum_{i=1}^{n} |\ell_i(x)|, \tag{2.13}$$

see for instance [57, 130]. Given the following relations

$$\max_{1 \leq i \leq n} |\ell_i(x)| \leq \sum_{i=1}^{n} |\ell_i(x)| \leq n \max_{1 \leq i \leq n} |\ell_i(x)|, \tag{2.14}$$

we conclude that

$$\Lambda \leq \Lambda_n \leq n\Lambda. \tag{2.15}$$

It is a measure of the accuracy of the polynomial interpolation at the set of points and also used in (2.17) below. This suggests to look for a set of interpolation points with a small Lebesgue constant. Hence, conversely, the smaller Λ, the better the geometry of the set $\mathcal{Y}$.

Importantly for our purposes, Lagrange polynomial values and Λ-poisedness can be used to bound the model function and model gradient error. In particular, it is shown in Ciarlet and Raviart [26] that for any x in the convex hull of $\mathcal{Y}$

$$\|\mathcal{D}^r f(x) - \mathcal{D}^r m(x)\| \leq \frac{\kappa_{\text{der}}}{(d+1)!} \sum_{j=1}^{p} \|y^j - x\|^{d+1} \|\mathcal{D}^r \ell_j(x)\|, \tag{2.16}$$

where $\mathcal{D}^r$ denotes the r-th derivative of a function and κ_{der} is an upper bound on $\mathcal{D}^{d+1} f(x)$ what means that this error bound requires $f(x)$ to have a bounded $(d+1)$st derivative. When $r = 0$, the bound on function values writes

$$|f(x) - m(x)| \leq \frac{\kappa_{\text{der}}}{(d+1)!} p \Lambda_\ell \Delta^{d+1}, \tag{2.17}$$

where

$$\Lambda_\ell = \max_{1 \leq i \leq p} \max_{x \in \mathcal{B}(\mathcal{Y})} |\ell_i(x)|, \tag{2.18}$$

and Δ is the diameter of $\mathcal{Y}$.

We will also make use of the following two bounds.

Lemma 2.2. *Given the sphere*

$$\mathcal{B}_2(x, \sqrt{n}\Delta) \stackrel{\text{def}}{=} \{v \in I\!R^n \mid \|v - x\|_2 \leq \sqrt{n}\Delta\},$$

a poised interpolation set $\mathcal{Y} \subset \mathcal{B}_2(x, \sqrt{n}\Delta)$ *and its associated basis of Lagrange polynomials* $\{\ell_i(x)\}_{i=1}^p$*, there exists constants* $\kappa_{ef} > 0$ *and* $\kappa_{eg} > 0$ *such that, for any interpolation polynomial* $m(x)$ *of degree one or higher of the form (2.10) and any point* $y \in \mathcal{B}_2(x, \sqrt{n}\Delta)$*, one has*

$$|f(y) - m(y)| \leq \kappa_{ef} \sum_{i=1}^{p} \|y^i - y\|_2^2 |\ell_i(y)| \tag{2.19}$$

and

$$\|\nabla_x f(y) - \nabla_x m(y)\|_2 \leq \kappa_{eg} \Lambda \Delta, \tag{2.20}$$

where $\Lambda = \max_{1 \leq i \leq p} \max_{y \in \mathcal{B}_2(x, \sqrt{n}\Delta)} |\ell_i(y)|$.

See [26] for (2.19) and Theorems 3.14 and 3.16 in Conn et al. [41] for (2.20).

Following the theory and also to have a reliable measure in practice, it is important to compute the global maximum in (2.12) relatively accurately, which can be done for linear and quadratic models using the Hebden-Moré-Sorensen algorithm (see [34], Section 7.3) in the Euclidean norm and motivates our choice of $\mathcal{B}_2(x, \sqrt{n}\Delta)$. Note that our definition of this last neighbourhood guarantees that $\mathcal{B}_\infty(x, \Delta) \subset \mathcal{B}_2(x, \sqrt{n}\Delta)$, and the error bounds (2.19)-(2.20) therefore hold in $\mathcal{B}_\infty(x, \Delta)$.

An equivalent definition of Lagrange polynomials is the following. Given a poised set $\mathcal{Y} = \{y^1, \ldots, y^p\} \subset I\!R^n$ and an $x \in I\!R^n$, we can express the vector $\phi(x)$ uniquely in terms of the vectors $\phi(y^i), i = 1, \ldots, p$, as

$$\sum_{i=1}^{p} \ell_i(x)\phi(y^i) = \phi(x) \tag{2.21}$$

or, in matrix form,

$$M(\phi, \mathcal{Y})^T \ell(x) = \phi(x), \quad \text{where } \ell(x) = [\ell_1(x), \ldots, \ell_p(x)]^T. \tag{2.22}$$

Consider now the set $\mathcal{Y}_i(x) = \mathcal{Y} \setminus \{y^i\} \cup \{x\}, i = 1, \ldots, p$. From the Cramer's rule on (2.22), we see that

$$\ell_i(x) = \frac{\det(M(\phi, \mathcal{Y}_i(x)))}{\det(M(\phi, \mathcal{Y}))}. \tag{2.23}$$

It follows that ℓ_i does not depend on the choice of ϕ as long as the polynomial space $\mathcal{P}_n^d$ is fixed. To interpret the formulation in (2.23), consider a set $\phi(\mathcal{Y}) = \{\phi(y^i)\}_{i=1}^p$ in $I\!R^p$. Let $\text{vol}(\phi(\mathcal{Y}))$ be the volume of the simplex of vertices in $\phi(\mathcal{Y})$, given by

$$\text{vol}(\phi(\mathcal{Y})) = \frac{|\det(M(\phi, \mathcal{Y}))|}{p!} \tag{2.24}$$

(Such a simplex is the p-dimensional convex hull of $\phi(\mathcal{Y})$.) Then

$$|\ell_i(x)| = \frac{\text{vol}(\phi(\mathcal{Y}_i(x)))}{\text{vol}(\phi(\mathcal{Y}))} \tag{2.25}$$

In other words, the absolute value of the i-th Lagrange polynomial at a given point x is the change in the volume of (the p-dimensional convex hull of) $\phi(\mathcal{Y})$ when y^i is replaced by x in $\mathcal{Y}$. This definition can be used to construct an algorithm which does compute a Λ-poised set from a poised set of cardinality p. This algorithm is stated as Algorithm 2.1 on page 11.

Furthermore, we have the following result from [41, Theorem 6.3]

Lemma 2.3. *For any given $\Lambda > 1$, a closed ball $\mathcal{B}$, and a fixed polynomial basis ϕ, Algorithm 2.1 terminates with a Λ-poised set $\mathcal{Y}$ after a finite number of iterations where the number of steps depends on Λ and ϕ.*

Algorithm 2.1 Improving well-poisedness via Lagrange polynomials

Initialization: Choose some constant $\Lambda > 1$. A poised set $\mathcal{Y}$ with $|\mathcal{Y}| = p$ is given. Compute the Lagrange polynomials $\ell_i(x), i = 1, \ldots, p$, associated with $\mathcal{Y}(= \mathcal{Y}_0)$. Set $k = 1$.

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

Step 2: If $\Lambda_{k-1} > \Lambda$, then let $i_k \in \{1, \ldots, p\}$ 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 maximizes $|\ell_{i_k}(x)|$ in $\mathcal{B}$. Update $\mathcal{Y}(= \mathcal{Y}_k)$ by performing the point exchange

$$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{y_*^{i_k}\} \setminus \{y^{i_k}\}.$$

Otherwise, return with a Λ-poised set $\mathcal{Y}$.

Step 3: Update all Lagrange polynomial coefficients. Go to Step 1.

2.1.4 Condition number as a measure of well-poisedness

An alternative measure of poisedness may be derived, albeit indirectly, from the matrix $M(\phi, \mathcal{Y})$. First note that the condition number of this matrix is in general not a satisfactory measure of poisedness of $\mathcal{Y}$ since it can be made arbitrarily large by changing the basis ϕ, and does not reflect the intrinsic geometry properties of $\mathcal{Y}$. However, [41] shows that a relation between the condition number of $\hat{M} = M(\bar{\phi}, \hat{\mathcal{Y}})$ and the measure of Λ-poisedness can be established when considering the basis of monomials $\bar{\phi}$ and $\hat{\mathcal{Y}}$, a shifted and scaled version of $\mathcal{Y}$. This new matrix is computed as follows. Given a sample set $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\}$, a shift of coordinates is first performed to center the interpolation set $\mathcal{Y}$ at the origin, giving $\{0, y^2 - y^1, \ldots, y^p - y^1\}$, where y^1 denotes the current best iterate which is usually the center of the interpolation. The region $\mathcal{B}$ is then fixed to be $\mathcal{B}_2(0, \Delta(\mathcal{Y}))$ and the radius

$$\Delta = \Delta(\mathcal{Y}) = \max_{2 \le i \le p} \|y^i - y^1\|_2$$

is used to scale the set, yielding

$$\hat{\mathcal{Y}} = \{0, \hat{y}^2, \ldots, \hat{y}^p\} = \{0, (y^2 - y^1)/\Delta, \ldots, (y^p - y^1)/\Delta\} = \frac{1}{\Delta}(\mathcal{Y} - y^1).$$

The resulting scaled interpolation set $\hat{\mathcal{Y}}$ is then contained in a ball of radius one in Euclidean norm centered at the origin.

The polynomial basis ϕ is fixed to be the natural basis $\bar{\phi}$ which can be described as follows (see also [42]). Let a vector $\alpha^i = (\alpha_1^i, \ldots, \alpha_n^i) \in \mathbb{N}^n$ be called a multiindex, and for any $x \in \mathbb{R}^n$, let x^{α^i} be defined by

$$x^{\alpha^i} = \prod_{j=1}^{n} x_j^{\alpha_j^i}. \tag{2.26}$$

Define also

$$|\alpha^i| = \sum_{j=1}^{n} \alpha_j^i \quad \text{and} \quad (\alpha^i)! = \prod_{j=1}^{n} (\alpha_j^i)!. \tag{2.27}$$

Then the elements of the natural basis are

$$\bar{\phi}_i(x) = \frac{1}{(\alpha^i)!} x^{\alpha^i}, \quad i = 0, \ldots, p, \quad |\alpha^i| \leq d. \tag{2.28}$$

The following result is then derived in [41, page 51].

Theorem 2.4. *If $\hat{M}$ is nonsingular and $\|\hat{M}^{-1}\|_2 \leq \Lambda$, then the set $\hat{\mathcal{Y}}$ is $\sqrt{p}\Lambda$-poised in the unit ball $\mathcal{B}(0,1)$ centered at 0. Conversely, if the set $\hat{\mathcal{Y}}$ is Λ-poised in the unit ball $\mathcal{B}(0,1)$ centered at 0, then*

$$cond(\hat{M}) \stackrel{\text{def}}{=} \|\hat{M}\|_2 \|\hat{M}^{-1}\|_2 \leq \theta p^2 \Lambda, \tag{2.29}$$

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

This means that this condition number of $\hat{M} = M(\bar{\phi}, \hat{\mathcal{Y}})$ can also be used to monitor poisedness of the interpolation set without computing Lagrange polynomials and Λ. Conversely, we can conclude that if the set $\hat{\mathcal{Y}}$ is reasonably well-poised, then $M(\bar{\phi}, \hat{\mathcal{Y}})$ is well-conditioned and the model $m(x)$ can be safely computed in (2.7) with the shifted and scaled coordinates.

One may then wonder which measure of poisedness is more appropriate. In our experience, both have their advantages. The condition number of $\hat{M}$ is cheaper to compute and suitable for checking the quality of the geometry before building the model at each iteration (see Section 3.3.4) while the measure in terms of the Lagrange polynomials is more convenient for improving poisedness of the interpolation set in a region $\mathcal{B}$ in a geometry improving step (applying Algorithm 2.1). Furthermore, Lagrange polynomials are used to estimate the highest improvement in poisedness (if any) that one can achieve when replacing a point from the interpolation set with a new trial point (see Section 3.1.5).

2.2 Polynomial interpolation and regression models

2.2.1 Interpolation sets of variable size

If we consider using interpolation models in the framework of a trust-region method for optimization, we observe that interpolation models of varying degree are possible and indeed

desirable in the course of the complete minimization. In early stages, using $p = n + 1$ function values (sufficient to build a linear model) is economical and often sufficient to make progress. In a later stage of the calculation, considering $p = \frac{1}{2}(n + 1)(n + 2)$ function values (enough to build a complete quadratic model) it is expected to achieve faster progress to a close solution.

Thinking of models of variable degree, it is natural to consider models evolving from linear to quadratic as minimization progresses as mentioned above. So, the number of interpolation conditions p that are imposed on a model $m(x)$ varies in the interval $[n+1, \frac{1}{2}(n+1)(n+2)]$ and in the case where $p < \frac{1}{2}(n+1)(n+2)$ and $q = \frac{1}{2}(n+1)(n+2)$, the matrix $M(\phi, \mathcal{Y})$ defining the interpolation conditions has more columns than rows and the interpolation polynomials defined by

$$m(y^i) = \sum_{k=1}^{q} \alpha_k \phi_k(y^i) = f(y^i), \quad i = 1, \ldots, p, \tag{2.30}$$

are no longer unique. This situation creates the need of using underdetermined quadratic interpolation models. The different approaches considered in this thesis are described in the following.

2.2.2 Sub-basis model

The easiest way one could think of to restrict (2.30) such that it has a unique solution, is to restrict $\mathcal{P}_n^d$ by taking the subset of the first p polynomial bases of ϕ. This technique keeps the interpolation system always square. We will use the notion of *sub-basis* of the basis ϕ to mean a subset of p elements of the basis ϕ. This sub-basis approach does not consider the last $q - p$ elements in the basis of ϕ so that the system to solve now writes

$$\begin{bmatrix} \phi_1(y^1) & \phi_2(y^1) & \cdots & \phi_p(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \cdots & \phi_p(y^2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(y^p) & \phi_2(y^p) & \cdots & \phi_p(y^p) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} f(y^1) \\ f(y^2) \\ \vdots \\ f(y^p) \end{bmatrix}. \tag{2.31}$$

The last $q-p$ components of α are set to zero what gives a special structure to the approximated Hessian H_k. In the course of minimization, sub-basis models become progressively "more quadratic" by considering these banded matrices H_k with increasing bandwidth. More specifically, when adding points to the system, the quadratic basis components are considered in the following order: the squared terms in $x_1^2, \ldots, x_n^2$, the quadratic terms of the first sub-diagonal in $x_1x_2, \ldots, x_{n-1}x_n$, the quadratic terms of the second sub-diagonal in $x_1x_3, \ldots, x_{n-2}x_n$, etc. depending on the size of p. Note that this "expanding band" strategy is particularly efficient if the true Hessian of the objective function is itself banded.

The drawback of this approach is that it may happen that the columns in $M(\phi, \mathcal{Y})$ are linearly dependent for the chosen sub-basis. This is usually a sign of a non-poised interpolation set and it must be repaired in this situation whereas it may have been poised when choosing another sub-basis of ϕ what is explained in the following example. Considering a two-dimensional example using the natural basis $\bar{\phi} = \{1, x_1, x_2, \frac{1}{2}x_1^2, \frac{1}{2}x_2^2, x_1x_2\}$ and a sample set

$\mathcal{Y} = \{y^1, y^2, y^3, y^4\}$ with $y^1 = (0,0), y^2 = (0,1), y^3 = (1,0), y^4 = (1,1)$. The matrix $M(\phi, \mathcal{Y})$ is given by

$$M(\phi, \mathcal{Y}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0.5 & 0 \\ 1 & 1 & 0 & 0.5 & 0 & 0 \\ 1 & 1 & 1 & 0.5 & 0.5 & 1 \end{bmatrix}.$$

Choosing now the first four columns of $M(\phi, \mathcal{Y})$, the system is determined but not well defined since the matrix is singular. We see now that the set $\mathcal{Y}$ is not poised with respect to the sub-basis $\phi = \{1, x_1, x_2, \frac{1}{2}x_1^2\}$, but if we selected the sub-basis $\phi = \{1, x_1, x_2, x_1x_2\}$, the set $\mathcal{Y}$ is well-poised and the corresponding matrix consisting of the first, the second, the third, and the sixth columns of $M(\phi, \mathcal{Y})$ is well-conditioned and a unique solution to this determined system exists.

2.2.3 Minimum ℓ_2-norm model

Another approach for getting a unique solution from the underdetermined system (2.30) is to compute its minimum ℓ_2-norm solution. The problem to solve writes here

$$\begin{aligned} &\min_\alpha \tfrac{1}{2}\|\alpha\|_2^2 \\ \text{s.t.} \quad &M(\phi, \mathcal{Y})\alpha = f(\mathcal{Y}) \end{aligned} \tag{2.32}$$

where we assume linear independence of the rows of $M(\phi, \mathcal{Y})$. Its solution is expressed as

$$\alpha = M(\phi, \mathcal{Y})^T [M(\phi, \mathcal{Y}) M(\phi, \mathcal{Y})^T]^{-1} f(\mathcal{Y}). \tag{2.33}$$

The resulting interpolating polynomial depends on the choice of ϕ but it has been observed in [42] that it is a reasonable choice to consider the minimum ℓ_2-norm underdetermined interpolant for the natural basis $\bar{\phi}$.

As we assumed that the rows of $M(\phi, \mathcal{Y})$ are linearly independent, we can equivalently use the formulation

$$M^\dagger = M^T [M M^T]^{-1}, \tag{2.34}$$

where $M^\dagger$ denotes the Moore-Penrose pseudoinverse [100, 111] of M which is also called the generalized inverse of M. A numerically stable and accurate way of solving the system (2.33) is by using the singular value decomposition of M.

2.2.4 Minimum Frobenius-norm model

It was shown in [41, Theorem 5.4] that function and gradient error bounds for underdetermined quadratic interpolation models depend on the norm of the Hessian of the model. A good idea seems therefore to build models for which the norm of the Hessian is moderate. This means, taking up the freedom in (2.30) could be achieved by minimizing the Frobenius-norm of the Hessian of the model, which is then called minimum Frobenius-norm model. To do

so, the natural basis $\bar{\phi}$ is split in its linear and quadratic parts: $\bar{\phi}_L = \{1, x_1, \ldots, x_n\}$ and $\bar{\phi}_Q = \{\frac{1}{2}x_1^2, \frac{1}{2}x_2^2, \ldots, x_1x_2, \ldots, x_{n-1}x_n\}$. The interpolation model writes now

$$m(x) = \alpha_L^T \bar{\phi}_L(x) + \alpha_Q^T \bar{\phi}_Q(x) \tag{2.35}$$

where α_L and α_Q are the corresponding parts of the coefficient vector α and are the solution to the problem

$$\begin{aligned} &\min_{\alpha_L, \alpha_Q} \tfrac{1}{2}\|\alpha_Q\|_2^2 \\ &\text{s.t. } M(\bar{\phi}_L, \mathcal{Y})\alpha_L + M(\bar{\phi}_Q, \mathcal{Y})\alpha_Q = f(\mathcal{Y}). \end{aligned} \tag{2.36}$$

Due to the choice of the natural basis $\bar{\phi}$ and the separation $\alpha = [\alpha_L \quad \alpha_Q]$, this is approximately the same as minimizing the Frobenius norm of the Hessian of the model subject to the interpolation conditions

$$\begin{aligned} &\min_{c,g,H} \tfrac{1}{4}\|H\|_F^2 \\ &\text{s.t. } c + g^T(y^i) + \tfrac{1}{2}(y^i)^T H(y^i) = f(y^i),\ i = 1, ..., p. \end{aligned} \tag{2.37}$$

To solve (2.36), and in turn (2.37), it is necessary to partition the matrix $M(\phi, \mathcal{Y})$ into linear and quadratic terms

$$M(\bar{\phi}, \mathcal{Y}) = \left[M(\bar{\phi}_L, \mathcal{Y}) \quad M(\bar{\phi}_Q, \mathcal{Y})\right] \tag{2.38}$$

and consider the matrix

$$F(\bar{\phi}, \mathcal{Y}) = \begin{bmatrix} M(\bar{\phi}_Q, \mathcal{Y})M(\bar{\phi}_Q, \mathcal{Y})^T & M(\bar{\phi}_L, \mathcal{Y}) \\ M(\bar{\phi}_L, \mathcal{Y})^T & 0 \end{bmatrix}. \tag{2.39}$$

Moreover, if $F(\bar{\phi}, \mathcal{Y})$ is nonsingular, the minimum Frobenius-norm model exists and is uniquely defined.

2.2.5 Least-change Frobenius-norm model

A variant of the minimum Frobenius-norm model is the least-change Frobenius-norm model where not the Frobenius-norm of the model Hessian is minimized to choose the solution to the system (2.30) but the Frobenius-norm of the change in the model Hessian from one iteration to the other. The problem to solve writes in this case

$$\begin{aligned} &\min_{c,g,H} \tfrac{1}{4}\|H - H^{old}\|_F^2 \\ &\text{s.t. } c + g^T(y^i) + \tfrac{1}{2}(y^i)^T H(y^i) = f(y^i),\ i = 1, ..., p \end{aligned} \tag{2.40}$$

which is similar to the formulation

$$\begin{aligned} &\min_{\alpha_L, \alpha_Q} \tfrac{1}{2}\|\alpha_Q - \alpha_Q^{old}\|_2^2 \\ &\text{s.t. } M(\bar{\phi}_L, \mathcal{Y})\alpha_L + M(\bar{\phi}_Q, \mathcal{Y})\alpha_Q = f(\mathcal{Y}), \end{aligned} \tag{2.41}$$

where again $\alpha = [\alpha_L \quad \alpha_Q]$, $\bar{\phi}_L$ contains the linear terms and $\bar{\phi}_Q$ the quadratic terms of the natural basis.

This type of interpolation model has been introduced by Powell [119, 120] and has been shown to work efficiently in several of his software implementations [121, 124]. Moreover, it has been shown [120] that if the objective f is a quadratic function then

$$\|H - \nabla^2 f\| \leq \|H^{old} - \nabla^2 f\|, \tag{2.42}$$

where $\nabla^2 f$ is the true Hessian of the objective function f.

2.2.6 Minimum ℓ_1-norm model

An approach to find the sparsest solution to the underdetermined problem (2.30) in the context of derivative-free optimization was recently presented by Bandeira, Scheinberg and Vicente [13, 14], whereas the initial idea is coming from the signal processing community for solving under- or overdetermined systems of linear equations. The type of model they suggest to construct is the sparse recovery model which is also called the minimum ℓ_1-norm model. In many problems, second order partial derivatives in the objective function f can be zero what leads to a certain sparsity pattern in the true Hessian $\nabla^2 f(x)$. Hence, the Hessian of the model $\nabla^2 m(x)$ should also be sparse. An interpolation model with a sparse Hessian could be computed by solving the minimization problem

$$\begin{aligned} &\min_{\alpha_L,\alpha_Q} \|\alpha_Q\|_0 \\ &\text{s.t. } M(\bar{\phi}_L, \mathcal{Y})\alpha_L + M(\bar{\phi}_Q, \mathcal{Y})\alpha_Q = f(\mathcal{Y}), \end{aligned} \tag{2.43}$$

where $\|x\|_0$ is oftentimes, perhaps misleadingly, called zero norm. It is defined as the number of non-zero elements of x and thus the problem (2.43) is NP-hard. Several authors (e.g. in [53, 54, 61]) have proposed to use the ℓ_1-norm instead, to approximate the sparsity of a vector as this often provides the sparsest solution to the system (2.30). This can be explained because the one-norm is the convex envelope of the function $\|x\|_0$ [84]. The problem to solve is now

$$\begin{aligned} &\min_{\alpha_L,\alpha_Q} \|\alpha_Q\|_1 \\ &\text{s.t. } M(\bar{\phi}_L, \mathcal{Y})\alpha_L + M(\bar{\phi}_Q, \mathcal{Y})\alpha_Q = f(\mathcal{Y}), \end{aligned} \tag{2.44}$$

where $\alpha = [\alpha_L \;\; \alpha_Q]$ is easier to determine as above in (2.43) as in (2.44) is only a linear program to solve.

2.2.7 Least-squares regression model

In the case we want to consider more than $p = \frac{1}{2}(n+1)(n+2)$ points to build a quadratic model, the system of interpolation conditions (2.30) is over-determined and has in general no solution. In this case, it is possible to compute the "best fit" (least-squares) solution to the system. The problem to solve writes in this case

$$\min_{\alpha} \|M(\phi, \mathcal{Y})\alpha - f(\mathcal{Y})\|^2. \tag{2.45}$$

The above system has a unique solution if the matrix

$$M(\phi, \mathcal{Y}) = \begin{bmatrix} \phi_1(y^1) & \phi_2(y^1) & \cdots & \phi_q(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \cdots & \phi_q(y^2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(y^p) & \phi_2(y^p) & \cdots & \phi_q(y^p) \end{bmatrix} \tag{2.46}$$

has full column rank. The solution to (2.45) is then expressed as

$$\alpha = [M(\phi, \mathcal{Y})^T M(\phi, \mathcal{Y})]^{-1} M(\phi, \mathcal{Y})^T f(\mathcal{Y}). \tag{2.47}$$

The relation

$$M^\dagger = [M^T M]^{-1} M^T \tag{2.48}$$

is not directly used to compute

$$\alpha = M^\dagger f(\mathcal{Y}), \tag{2.49}$$

the unique solution to the over-determined system (2.30). Solving the system (2.47) by using the singular value decomposition of M or an approach based on the QR factorization is often preferred [68]. See also [16, 17] on the topic of solving least-squares problems.

In the literature (e.g. [40, 41]), the regression model is especially recommended when there is noise in the evaluation of the true function $f(x)$. Moreover, it has been shown [41] that if the noise is random and independently and identically distributed with mean zero, then the least-squares regression of the noisy function and the one of the true function converge to each other as the number of points considered tends to infinity.

2.3 A basic interpolation-based trust-region approach

As itemized at the beginning of this chapter, different interpolation-based trust-region approaches can be found in the literature. They can be distinguished by their interpolation set update, how the geometry of the interpolation set is maintained, the support of a convergence theory, and their particular management of the trust region.

In this section, we outline one of the first derivative-free interpolation-based trust-region algorithms which proved global convergence to first-order critical points. It was developed by Conn, Scheinberg and Toint [35, 37] in the late Nineties and is depicted as Algorithm 2.2 on page 18. The algorithm was kept admittedly simplistic by its authors to study its convergence properties but additional features are suggested to enhance its efficiency. For instance, the possibility of including the point x_{k+1} in $\mathcal{Y}$ is mentioned, even if the iteration was unsuccessful; each evaluation of the objective function should indeed be exploited if possible, provided it does not deteriorate the quality of the model. An additional geometry improving step could be performed if the ratio between the predicted reduction in the model versus achieved reduction in the objective function (ρ_k from (2.4)) is very small which is an indicator that the model is not a good approximation of the objective function.

Algorithm 2.2 Basic DFO trust-region algorithm

Step 0: Initialization.
Choose constants $\epsilon_g > 0, 0 < \gamma_1 < \gamma_2 < 1 < \gamma_3, 0 < \eta_0 \leq \eta_1 < 1, \mu \geq 1$ and an initial trust-region radius $\Delta_0 > 0$. Let x_{start} and $f(x_{\text{start}})$ be given. Select an initial interpolation set $\mathcal{Y}_0$ containing x_{start} and at least another point. Then determine $x_0 \in \mathcal{Y}_0$ such that $f(x_0) = \min_{y \in \mathcal{Y}_0} f(y)$. Set $k = 0$.

Step 1: Model computation.
Build a model $m_k(x_k)$ that interpolates the function f on the interpolation set $\mathcal{Y}_k$ such that the interpolation conditions (2.5) are satisfied. Compute the model gradient $g_k = \nabla m_k(x_k)$.

If $\|g_k\| \leq \epsilon_g$ and $\mathcal{Y}_k$ is not well-poised in the region $\mathcal{B}(x_k, \mu\|g_k\|)$, then improve the geometry until $\mathcal{Y}_k$ is well-poised in $\mathcal{B}(x_k, \delta_k)$ for some $\delta_k \in (0, \mu\|g_k\|)$ and go to Step 1. If $\|g_k\| > \epsilon_g$, go to Step 2, otherwise, stop.

Step 2: Step Computation.
Compute a point $x_k + s_k$ such that

$$m_k(x_k + s_k) = \min_{x \in \mathcal{B}(x_k, \Delta_k)} m_k(x).$$

Compute $f(x_k + s_k)$ and the ratio ρ_k from (2.4).

Step 3: Interpolation set update.
Successful iteration: If $\rho_k \geq \eta_1$, insert $x_k + s_k$ in $\mathcal{Y}_k$, dropping one of the existing interpolation points if $p = \frac{1}{2}(n+1)(n+2)$.

Unsuccessful iteration: If $\rho_k < \eta_1$ and $\mathcal{Y}_k$ is inadequate in $\mathcal{B}(x_k, \Delta_k)$, improve the geometry in $\mathcal{B}(x_k, \Delta_k)$ by changing the set $\mathcal{Y}_k$.

Step 4: Trust-region radius update.
Set

$$\Delta_{k+1} = \begin{cases} [\Delta_k, \gamma_3 \Delta_k] & \text{if } \rho_k \geq \eta_1, \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if } \rho_k < \eta_1 \text{ and } \mathcal{Y}_k \text{ is adequate in } \mathcal{B}(x_k, \Delta_k), \\ \Delta_k & \text{otherwise.} \end{cases}$$

Step 5: Update the current iterate.
Determine $\hat{x}_k$ such that $f(\hat{x}_k) = \min_{y \in \mathcal{Y}_k \setminus \{x_k\}} f(y)$. Then, define the revised measure

$$\hat{\rho}_k \stackrel{\text{def}}{=} \frac{f(x_k) - f(\hat{x}_k)}{m_k(x_k) - m_k(x_k + s_k)}.$$

If $\hat{\rho}_k \geq \eta_0$ set $x_{k+1} = \hat{x}_k$, otherwise, set $x_{k+1} = x_k$. Increment k by one and go to Step 1.

They also mention that the loop in Step 1 could be seen as a geometry improvement inner iteration to ensure that the model's gradient is not too far from the true first-order information of the objective function. Such a geometry improving inner iteration can be carried out in a finite and bounded number of steps if $\nabla f(x_k) \neq 0$. This procedure can also be used as a criticality step to ensure a good quality of the model in a small neighborhood of a potential critical point (e.g. in [41, Chapter 10] and [128]). Algorithm 2.2 does not comprise a stopping test but it is proposed in [35] to stop the calculation if either the trust-region radius falls below a certain threshold, or the model's gradient becomes sufficiently small and the geometry of the interpolation set is adequate. The outlined algorithm incorporates two different regions $\mathcal{B}_k(x_k, \mu\|g_k\|)$ and $\mathcal{B}_k(x_k, \Delta_k)$, where the latter one is the trust-region to ensure convergence of the algorithm and the other one is intended to monitor the first-order part of the model. This reminds of the use of two distinctive regions in the work by Powell [117]. Other recent methods use only one region for more simplicity in algorithm and convergence analysis (eg. in [41, 74]).

Furthermore, it should be mentioned that the considered models in Algorithm 2.2 are at most quadratic as in most of the current implementations of interpolation-based trust-region methods. This is in fact not a general requirement and, for instance, Conn and Toint [43] suggest the use of models of degree exceeding two and the theory can be readily extended to account for this. Remarkably, the authors of [35] suggest to build initial models out of only two points in $\mathcal{Y}_k$ where most other practical algorithms require that the model is build using at least $n + 1$ interpolation conditions.

Many methods are augmenting the interpolation set as minimization progresses, whereas, for instance, Powell uses in his approach a fixed number of interpolation points throughout the calculation and recommends to build models from $2n + 1$ interpolation points. Moreover, he is mentioning one problem where maintaining an interpolation set of only $n+6$ points turned out to perform well in his optimization framework [121].

A practical variation of the algorithm in [35] has been presented by Weber-Mendonca [99] and both build models based on the Newton fundamental polynomial basis whereas most other known methods use models based on Lagrange polynomial bases or the basis of monomials.

Another similar approach was developed recently by Conn, Scheinberg and Vicente [42] who provided the first analysis of global convergence of derivative-free trust-region methods to second-order critical points. The trust-region management of the latter algorithm differs in the manner the trust-region radius Δ is increased in a successful iteration. In fact, they have shown that the trust-region radius needs only to be increased when it is much smaller than the second-order stationarity measure of the model. This approach also allows to accept an iterate based on simple decrease of the objective function where only the strict inequality of decrease $f(x_{k+1}) < f(x_k)$ must be true to declare an iteration as successful. This is not the case for most other interpolation-based trust-region methods which rather rely on the standard trust-region step acceptance where sufficient decrease in the objective function is required to accept the candidate as the new iterate.

2.4 Self-correcting geometry - a recent approach

2.4.1 The algorithm

The basic interpolation-based trust-region algorithm stated in the previous section, and also most algorithms in the current DFO-literature, differ from more standard trust-region schemes in that the decision to shrink the trust region depends on the quality of the interpolation model. That means, if the interpolation set is not sufficiently poised, then it may indeed turn out that the failure of the current iteration is due to the bad approximation of the function by the resulting model rather than to a too large trust region. The basic approach is therefore to improve the poisedness of the interpolation set first, before considering to shrink the trust region.

This improvement is usually carried out at special geometry improving steps as it is done in Step 1 and Step 3 of Algorithm 2.2 above. But such a procedure is expensive because additional function values at well-chosen points have to be computed. That leads naturally to the question whether this additional cost is really necessary for the algorithm to be globally convergent or not. Interestingly, Fasano et al. [58] have developed an algorithm which completely ignores any kind of geometry consideration. They suggested that it may be sufficient to replace the furthest point of the interpolation set with the new trial point at each iteration to maintain a reasonable geometry of the interpolation set. And indeed, they observed that such an algorithm may perform quite well in practice although the authors have no supporting theory or explanation for their success.

However, shortly after, it has been shown by Scheinberg and Toint [128] that it is impossible to ignore geometry consideration altogether if one wishes to maintain global convergence and they presented two counter-examples showing that such a method may converge to a non-critical point. But it was also shown that an algorithm can be designed to indeed substantially reduce the need of geometry improving steps by exploiting a self-correcting property of the interpolation set geometry. This algorithm is presented as Algorithm 2.3 on page 21. The design and convergence proof of this algorithm depends on a self-correction mechanism resulting from the combination of the trust-region framework with the polynomial interpolation setting.

2.4.2 Convergence theory

The algorithms developed in this thesis make use of the self-correcting property presented above and thus rely mainly on the convergence results obtained by Scheinberg and Toint [128]. We will therefore quote their convergence theory here for convenience and show that their proposed algorithm (stated above as Algorithm 2.3) produces a sequence of iterates $\{x_k\}$ such that the corresponding sequence of gradients of the true objective function $\{\nabla_x f(x_k)\}$ admits a subsequence converging to zero.

Algorithm 2.3 UDFO trust-region algorithm with self-correcting geometry from [128]

Step 0: Initialization.

An initial trust-region radius Δ_0, initial accuracy threshold ϵ_0 are given. An initial poised interpolation set $\mathcal{Y}_0$ that contains the starting point x_0 is known. An interpolation model m_0 around x_0 and associated Lagrange polynomials $\{l_{0,j}\}_{j=1}^p$ are computed. Constants $\eta \in (0,1), 0 < \gamma_1 \leq \gamma_2 < 1, \mu \in (0,1), \theta > 0, \beta \geq 1, \epsilon \geq 0$ and $\Lambda > 1$ are also given. Choose $v_0 \neq x_0$ where v_i is a variable introduced to keep track if the model at x_k is known to be well poised. Set $k = 0$ and $i = 0$.

Step 1: Criticality test.

Step 1a: Define $\hat{m}_i = m_k$.

Step 1b: If $\|\nabla_x \hat{m}_i(x_k)\| < \epsilon_i$, set $\epsilon_{i+1} = \mu \|\nabla_x \hat{m}_i(x_k)\|$, compute a Λ-poised model $\hat{m}_{i+1}$ in $\mathcal{B}(x_k, \epsilon_{i+1})$ and increment i by one. If $\|\nabla_x \hat{m}_i(x_k)\| < \epsilon$, then return x_k, otherwise start Step 1b again.

Step 1c: Set $m_k = \hat{m}_i, \Delta_k = \theta \|\nabla_x m_k(x_k)\|$ and define $v_i = x_k$ (what indicates that the model at x_k is well poised and Steps 4b and 4c need not to be visited in an unsuccessful iteration) if a new model has been computed.

Step 2: Compute a trial point.

Compute $x_k^+ = x_k + s_k$ such that $m_k(x_k + s_k) = \min_{x \in \mathcal{B}(x_k, \Delta_k)} m_k(x)$.

Step 3: Evaluate the objective function at the trial point.

Compute $f(x_k^+)$ and ρ_k from (2.4).

Step 4: Define the next iterate.

Step 4a: Successful iteration. If $\rho_k \geq \eta$, define $x_{k+1} = x_k^+$, choose $\Delta_{k+1} \geq \Delta_k$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point $y_{k,j}$ in $\mathcal{Y}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|_\infty^2 |l_{k,j}(x_k^+)|$.

Step 4b: Replace a far interpolation point. If $\rho_k < \eta$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \text{ such that } \|y_{k,j} - x_k\| > \beta \Delta_k \text{ and } l_{k,j}(x_k^+) \neq 0\}$$

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 the index j of any point $y_{k,j}$ in $\mathcal{F}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|_\infty^2 |l_{k,j}(x_k^+)|$.

Step 4c: Replace a close interpolation point. If $\rho_k < \eta$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k = \emptyset$ and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \text{ such that } \|y_{k,j} - x_k\| \leq \beta \Delta_k \text{ and } l_{k,j}(x_k^+) > \Lambda\}$$

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 the index j of any point $y_{k,j}$ in $\mathcal{C}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|_\infty^2 |l_{k,j}(x_k^+)|$.

Step 4d: Reduce the trust region radius. If $\rho_k < \eta$ 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.

First, the assumptions are stated.

A1: the objective function f is continuously differentiable in an open set $\mathcal{V}$ containing all iterates generated by the algorithm, and its gradient $\nabla_x f$ is Lipschitz continuous in $\mathcal{V}$ with constant L;

A2: there exists a constant κ_{low} such that $f(x) \geq \kappa_{\text{low}}$ for every $x \in \mathcal{V}$;

A3: there exists a constant $\kappa_H \geq L$ such that $1 + \|H_k\| \leq \kappa_H$ for every $k \geq 0$;

A4: $|\mathcal{Y}_k| \geq n + 1$ for every $k \geq 0$.

Note that A1 only assumes the existence of first derivatives, not that they can be computed.

Lemma 2.5. *Assume that, for some real numbers* $\{\alpha_i\}_{i=0}^t$ *with*

$$\sigma_{\text{abs}} \stackrel{\text{def}}{=} \sum_{i=0}^{t} |\alpha_i| > 2 \sum_{i=0}^{t} \alpha_i \stackrel{\text{def}}{=} 2\sigma > 0.$$

If one defines

$$i^* = \arg \max_{i=0,\ldots,t} |\alpha_i| \quad \textit{and} \quad j^* = \arg \max_{\substack{j=0,\ldots,t \\ j \neq i^*}} |\alpha_j|,$$

then

$$|\alpha_{j^*}| \geq \frac{\sigma_{\text{abs}} - 2\sigma}{2t}. \tag{2.50}$$

Now, the crucial self-correction property of Algorithm 2.3 is stated where the definition of $\mathcal{F}_k$ and $\mathcal{C}_k$ are given in Steps 4b and 4c of Algorithm 2.3.

Lemma 2.6. *Suppose that A1, A3 and A4 hold. Then, for any constant* $\Lambda > 1$*, if iteration* k *is unsuccessful and*

$$\mathcal{F}_k = \emptyset \tag{2.51}$$

and

$$\Delta_k \leq \min \left[\frac{1}{\kappa_H}, \frac{(1-\eta)\kappa_c}{2\kappa_{ef}(\beta+1)^2(p\Lambda+1)} \right] \|g_k\| \stackrel{\text{def}}{=} \kappa_\Lambda \|g\|, \tag{2.52}$$

then

$$\mathcal{C}_k \neq \emptyset. \tag{2.53}$$

This property says that, provided the trust-region radius is small enough compared to the model's gradient and all the significant interpolation points are contained in the trust region, then every unsuccessful iteration must result in an improvement of the interpolation set geometry. The geometry is therefore self-correcting at unsuccessful iterations of this type. Moreover, the value of the geometry improvement is only dependent on Λ, while the maximum size of Δ_k compared with $\|g_k\|$ depends on the problem (via κ_{ef} and κ_H), on the algorithms' parameters (via η, Λ and κ_c from the Cauchy condition (3.10)) and on the size p of the interpolation set.

It is now verified, as is usual in trust-region methods, that the step bound Δ_k cannot become arbitrarily small far away from a critical point.

Lemma 2.7. *Suppose that A1, A3 and A4 hold and assume that, for some $k_0 \geq 0$ and all $k \geq k_0$,*

$$\|g_k\| \geq \kappa_g \tag{2.54}$$

for some $\kappa_g > 0$. Then there exists a constant $\kappa_\Delta > 0$ such that, for all $k \geq k_0$,

$$\Delta_k \geq \kappa_\Delta. \tag{2.55}$$

This result allows to continue the convergence analysis in the spirit of the standard trust-region theory (see Conn et al. [34, Chapter 6]). First, the case is considered where the number of successful iterations is finite.

Lemma 2.8. *Suppose that A1, A2 and A4 hold and that there is a finite number of successful iterations. Then*

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{2.56}$$

The case when there occur infinitely many successful iterations is considered next.

Lemma 2.9. *Suppose that A1–A4 hold and that the number of successful iteration is infinite. Then*

$$\liminf_{k\to\infty} \|g_k\| = 0 \tag{2.57}$$

holds.

This shows that, eventually, the gradient of the model has to become smaller than ϵ_0. When this happens, the algorithm essentially restarts with a well-poised model in a sufficiently smaller ball. Then the same algorithm is applied, but with the value ϵ_0 replaced by the smaller ϵ_1. Applying the same argument as above we can show that eventually $\|g_k\|$ will become smaller than ϵ_1 and the process repeats. To prove that this process leads to global convergence, the following additional two technical results are needed.

Lemma 2.10. *Suppose that A1 and A3 hold. Then*

$$|f(x_k^+) - m_k(x_k^+)| \leq \|\nabla_x f(x_k) - g_k\|\Delta_k + \kappa_H \Delta_k^2. \tag{2.58}$$

Lemma 2.11. *Suppose that A1 and A3 hold, that $g_k \neq 0$, that*

$$\|\nabla_x f(x_k) - g_k\| \leq \frac{1}{2}\kappa_c(1-\eta)\|g_k\| \tag{2.59}$$

and that

$$\Delta_k \leq \frac{\kappa_c}{2\kappa_H}(1-\eta)\|g_k\|. \tag{2.60}$$

Then iteration k is successful.

The parameter κ_c is again coming from the Cauchy condition (3.10). The final result is stated in the following Theorem.

Theorem 2.12. *Suppose that A1-A4 hold. Then*

$$\liminf_{k\to\infty} \|\nabla_x f(x_k)\| = 0. \tag{2.61}$$

The respective proofs can be found in [128].

2.5 A modified algorithm with self-correcting geometry

Here, we want to present our first contribution in this thesis. We present a modified version of Algorithm 2.3 (developed by Scheinberg and Toint [128] and stated as algorithm UDFO in Section 2.4 above) and prove global convergence of our modified algorithm.

As our new algorithm (and therefore also the convergence analysis) has strong similarities to Algorithm 2.3, we explain first which modifications have been made to Algorithm 2.3 and then show that the new Algorithm 2.4 still produces a sequence of iterates $\{x_k\}$ converging to a first-order critical point.

2.5.1 The algorithm

An outline of the new algorithm, which we called UDFO+, is given in Algorithm 2.4 on page 25.

2.5.2 Modifications

Following the idea of Fasano, Morales and Nocedal [58], who successfully applied a standard trust-region management in a derivative-free algorithm and showed an efficient behaviour of such a practical algorithm, we were wondering if this strategy can be integrated into a framework where it is possible to prove global convergence. Globally convergent algorithms for unconstrained derivative-free optimization have up to now only shown to be moderately efficient compared to [58] due to geometry improving steps and geometry considerations in the trust-region management (e.g. do extra calculations or wait some iterations until the geometry of the set of points is good enough to allow for shrinking the trust region).

We recall the basic statement, that an iteration might be unsuccessful due to the fact that the quality of the model is not good enough but also due to the fact that the trust region is too large. The latter possibility was not emphasized in algorithm UDFO [128] and other derivative-free algorithms which usually first ensure that the geometry is good enough to perhaps conclude that the trust-region radius must have been too large to progress (and only then shrink the trust region).

We propose an algorithm which uses the self-correcting property from [128] to have a certain control on the geometry of the interpolation set, but ignores geometry considerations in the trust-region management by applying a more standard trust-region management as was done in [58]. In our version, the algorithm is given the possibility to progress further in a smaller trust-region instead of first producing a good geometry to allow for shrinking the trust region. In particular, we suggest to decrease the trust-region radius Δ_k in each unsuccessful iteration (if $\rho < \eta_1$) as long as the trust-region radius is larger than a fixed threshold Δ_{switch}. If $\Delta_k \leq \Delta_{\text{switch}}$ is reached, we switch to the technique applied in Algorithm 2.3 [128], such that Δ_k is only reduced in an unsuccessful iteration where no point from the interpolation set can be replaced (see Step 4e of Algorithm 2.4).

Algorithm 2.4 UDFO+ modified algorithm with self-correcting geometry

Step 0: Initialization.
An initial trust-region radius Δ_0, initial accuracy threshold ϵ_0 are given. An initial poised interpolation set $\mathcal{Y}_0$ and the starting point x_0 are known. A model m_0 around x_0 and Lagrange polynomials $\{l_{0,j}\}_{j=1}^p$ are computed. Constants $\eta_1, \Delta_{\text{switch}} \in (0,1), 0 < \gamma_1 \leq \gamma_2 < 1, \mu \in (0,1), \theta > 0, \beta \geq 1, \epsilon \geq 0$, $p_{\max} \geq n+1$ and $\Lambda > 1$ are also given. Choose $v_0 \neq x_0$. Set $k = 0$ and $i = 0$.

Step 1: Criticality test.
Step 1a: Define $\hat{m}_i = m_k$.
Step 1b: If $\|\nabla_x \hat{m}_i(x_k)\| < \epsilon_i$, set $\epsilon_{i+1} = \mu \|\nabla_x \hat{m}_i(x_k)\|$, compute a Λ-poised model $\hat{m}_{i+1}$ in $\mathcal{B}(x_k, \epsilon_{i+1})$ and increment i by one. If $\|\nabla_x \hat{m}_i(x_k)\| < \epsilon$, then return x_k, otherwise go to Step 1b.
Step 1c: Set $m_k = \hat{m}_i, \Delta_k = \theta \|\nabla_x m_k(x_k)\|$ and define $v_i = x_k$ if a new model has been computed.

Step 2: Compute a trial point.
Compute $x_k^+ = x_k + s_k$ such that $m_k(x_k + s_k) = \min_{x \in \mathcal{B}(x_k, \Delta_k)} m_k(x)$.

Step 3: Evaluate the objective function at the trial point.
Compute $f(x_k^+)$ and ρ_k from (2.4).

Step 4: Define the next iterate.
Step 4a: Augment interpolation set ($\mathbf{p_k < p_{\max}}$). If $p_k < p_{\max}$, then: Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$. If $\rho_k \geq \eta_1$, then define $x_{k+1} = x_k^+$ and choose $\Delta_{k+1} \geq \Delta_k$. If $\rho_k < \eta_1$, define $x_{k+1} = x_k$ and if $\Delta_k > \Delta_{\text{switch}}$, set $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$, otherwise $\Delta_{k+1} = \Delta_k$.
Step 4b: Successful iteration. If $\rho_k \geq \eta_1$ and $p_k = p_{\max}$, define $x_{k+1} = x_k^+$, choose $\Delta_{k+1} \geq \Delta_k$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point in $\mathcal{Y}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|_\infty^2 |l_{k,j}(x_k^+)|$.
Step 4c: Replace a far interpolation point. If $\rho_k < \eta_1$, $p_k = p_{\max}$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \text{ such that } \|y_{k,j} - x_k\| > \beta \Delta_k \text{ and } l_{k,j}(x_k^+) \neq 0\}$$

is non-empty, then set $x_{k+1} = x_k$, $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$ (or set $\Delta_{k+1} = \Delta_k$ if $\Delta_k \leq \Delta_{\text{switch}}$). Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point in $\mathcal{F}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|_\infty^2 |l_{k,j}(x_k^+)|$.
Step 4d: Replace a close interpolation point. If $\rho_k < \eta_1$, $p_k = p_{\max}$, either $x_k \neq v_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k = \emptyset$ and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \text{ such that } \|y_{k,j} - x_k\| \leq \beta \Delta_k \text{ and } l_{k,j}(x_k^+) > \Lambda\}$$

is non-empty, then set $x_{k+1} = x_k$, $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$ (or set $\Delta_{k+1} = \Delta_k$ if $\Delta_k \leq \Delta_{\text{switch}}$). Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point in $\mathcal{C}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|_\infty^2 |l_{k,j}(x_k^+)|$.
Step 4e: Reduce the trust region radius. If $\rho_k < \eta_1$, $p_k = p_{\max}$ 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 $\{l_{k+1,j}\}_{j=1}^p$. Increment k by one and go to Step 1.

The efficiency of such a combined algorithm is shown in Section 3.5 below, where we present numerical experiments involving the algorithms UDFO, UDFO+, their bound-constrained extensions BC-DFO, ***BCDFO+*** and some reference software packages.

Furthermore, in our algorithm, we additionally consider the case where the interpolation (or regression) set is augmented with the new trial points and thus, the size of the set $\mathcal{Y}$ is possibly increasing (see Step 4a of Algorithm 2.4). For our theory, we need therefore another assumption, in addition to the four assumptions stated in Section 2.4.2, which ensures in the regression case that there exists a maximum number of points $p_{\max}$ considered in the set $\mathcal{Y}$ during the minimization. Please note, that now in successful iterations (in the case that $\rho \geq \eta_1$) either Step 4a or Step 4b is executed, and in unsuccessful iterations (when $\rho < \eta_1$) either Step 4a, 4c, 4d or 4e is executed. We have to consider this fact in our analysis below.

2.5.3 Global convergence

We now show that our Algorithm 2.4 produces a subsequence of gradients of the true objective function $\{g_k\}$ converging to zero.

We start by stating our assumptions.

A1: the objective function f is continuously differentiable in the feasible set $\mathcal{F}$, and its gradient $\nabla_x f$ is Lipschitz continuous in $\mathcal{F}$ with constant L;

A2: there exists a constant κ_{low} such that $f(x) \geq \kappa_{\text{low}}$ for every $x \in \mathcal{F}$;

A3: there exists a constant $\kappa_H \geq L$ such that $1 + \|H_k\| \leq \kappa_H$ for every $k \geq 0$;

A4: $|\mathcal{Y}_k| \geq n + 1$ for every $k \geq 0$;

Note that A3 implies that the Hessian of the model H_k remains bounded which cannot be guaranteed by Algorithm 3.1 during successful iterations. However, in practice, a safeguard (as described below in Section 3.3) is applied in situations where the poisedness of the interpolation set deteriorates, by checking the condition number of the matrix $M(\bar{\phi}, \hat{\mathcal{Y}})$. This strategy guarantees a bounded model Hessian matrix in practice.

In addition, we assume that the algorithmic parameter ϵ is set to $\epsilon = 0$ for the purposes of the convergence theory.

Lemma 2.13. *Suppose that A1, A3 and A4 hold. Then, for any constant $\Lambda > 1$, if iteration k is unsuccessful, $p = p_k = p_{\max}$,*

$$\mathcal{F}_k = \emptyset \tag{2.62}$$

and

$$\Delta_k \leq \min\left[\frac{1}{\kappa_H}, \frac{(1-\eta_1)\kappa_c}{2\kappa_{ef}(\beta+1)^2(p_k\Lambda+1)}\right] \|g_k\| \stackrel{\text{def}}{=} \kappa_\Lambda \|g_k\|, \tag{2.63}$$

then

$$\mathcal{C}_k \neq \emptyset. \tag{2.64}$$

As we assume that the maximum number of points in the set $p_k = p_{\max}$ is reached, the proof of the lemma is the same as in [128, Lemma 5.2].

This means that the self-correcting property introduced by [128] also holds in our modified unconstrained algorithm and that therefore, provided the maximum number of points $p_{\max}$ in the interpolation/regression set is reached, the trust-region radius is small compared to the model's gradient and if all points of $\mathcal{Y}$ are contained in the trust region, then every unsuccessful iteration must result in an improvement of the interpolation/regression set geometry.

The next step is to verify that the step bound Δ_k cannot become arbitrarily small far away from a critical point.

Lemma 2.14. *Suppose that A1, A3 and A4 hold and assume that, for some $k_0 \geq 0$ and all $k \geq k_0$,*

$$\|g_k\| \geq \kappa_g \tag{2.65}$$

for some $\kappa_g > 0$. Then there exists a constant $\kappa_\Delta > 0$ such that, for all $k \geq k_0$,

$$\Delta_k \geq \kappa_\Delta. \tag{2.66}$$

Proof. Assume that, for some $k \geq 0$,

$$\Delta_k < \min(\kappa_\Lambda \kappa_g, \mu\kappa_g, \Delta_{\text{switch}}). \tag{2.67}$$

If iteration k is successful (i.e. $\rho \geq \eta_1$), then we have that $\Delta_{k+1} \geq \Delta_k$. On the other hand, if $\rho < \eta_1$, then we show that only three cases may occur. The first case is when $p_k < p_{\max}$ and Step 4a of Algorithm 2.4 is executed. Observe now that (2.67) ensures that $\Delta_k < \Delta_{\text{switch}}$ and therefore, $\Delta_{k+1} = \Delta_k$. The second case is when $p_k = p_{\max}$ and $\mathcal{F}_k \neq \emptyset$. If now $i > 0$, then (2.65) and (2.67) ensure that

$$\Delta_k < \mu\|g_{k_i}\| = \epsilon_i, \tag{2.68}$$

where k_i is the index of the last iteration before k where a new Λ-poised model has been recomputed in the criticality test (Step 1) of Algorithm 2.4. Step 4c is therefore executed and with $\Delta_k < \Delta_{\text{switch}}$, we have that $\Delta_{k+1} = \Delta_k$. The third case is when $p_k = p_{\max}$ and $\mathcal{F}_k = \emptyset$ in which case (2.67) and Lemma 2.13 guarantee that $\mathcal{C}_k \neq \emptyset$. Since (2.68) also holds in this case, Step 4d is executed and $\Delta_{k+1} = \Delta_k$ because $\Delta_k < \Delta_{\text{switch}}$. As a consequence, the trust-region radius may only be decreased if

$$\Delta_k \geq \min(\kappa_\Lambda \kappa_g, \mu\kappa_g, \Delta_{\text{switch}}),$$

and the mechanism of the algorithm then implies the desired result with

$$\kappa_\Delta = \min[\Delta_0, \gamma_1 \min(\kappa_\Lambda \kappa_g, \mu\kappa_g, \Delta_{\text{switch}})].$$

□

We continue our analysis in the spirit of a standard trust-region theory (see [34, Chapter 6]). We first consider the case where the number of successful iterations is finite.

Lemma 2.15. *Suppose that A1, A2 and A4 hold and that there is a finite number of successful iterations. Then*

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{2.69}$$

Proof. We observe first that, since every iteration is eventually unsuccessful (i.e. $\rho < \eta_1$), $x_k = x_*$ for some x_* and all k sufficiently large. Assume further, for the purpose of deriving a contradiction, that (2.65) holds for some $\kappa_g > 0$ and all k. Then, by Lemma 2.14, we have that $\Delta_k > \kappa_\Delta > 0$ on all iterations. Since the number of successful iterations is finite, eventually all iterations are unsuccessful (of type 4a, 4c, 4d or 4e, regarding the steps of Algorithm 2.4). As a consequence, the sequence $\{\Delta_k\}$ is non-increasing and bounded below, and therefore convergent. Let $\Delta_\infty \stackrel{\text{def}}{=} \lim_{k\to\infty} \Delta_k \geq \kappa_\Delta$. Unsuccessful iterations of type 4a, 4c and 4d, in the case where points are replaced and Δ_k is decreased at the same time, cannot happen infinitely often because $\Delta_{\text{switch}} > \kappa_\Delta$. Iterations of type 4e cannot happen infinitely often because Δ_k is bounded below by Δ_∞ and $\gamma_2 < 1$. Thus, $\Delta_k = \Delta_\infty < \Delta_{\text{switch}}$ for all k sufficienty large, and all iterations are eventually of type 4d since $p_k = p_{\max}$ must eventually be reached and at most $p_{\max}$ iterations of type 4c can possibly be necessary to ensure that all interpolation points belong to $\mathcal{B}(x_*, \beta\Delta_\infty)$. Note that during an iteration of type 4d the new trial point replaces some interpolation point from the set $\mathcal{C}_k$. From the definition of $\mathcal{C}_k$ it follows that, for all k large enough, the trial point x_k^+ replaces a previous interpolation point $y_{k,j}$ such that $|\ell_{k,j}(x_k^+)| \geq \Lambda$. But this is impossible in view of Lemma 2.3 what leads to the desired contradiction. □

Now, the case with infinitely many successful iterations is considered.

Lemma 2.16. *Suppose that A1–A4 hold and that the number of successful iterations is infinite. Then*

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{2.70}$$

As the lemma wasn't modified, the proof remains as in [128, Lemma 5.5].

The result of the last two lemmas is that the model gradient g_k has eventually to become smaller than ϵ_0, whereafter the algorithm restarts with a well-poised set in a sufficiently smaller ball. The same algorithm is applied but with the value ϵ_0 replaced by the smaller value ϵ_1. Applying the same argument as above, $\|g_k\|$ will become smaller than ϵ_1 and the process repeats.

Now, we present the final result.

Theorem 2.17. *Suppose that A1-A4 hold. Then*

$$\liminf_{k\to\infty} \|\nabla_x f(x_k)\|_\infty = 0. \tag{2.71}$$

Proof. Assume, that $p_k = p_{\max}$ for all k sufficiently large. Assume further, by contradiction that there exists a $\kappa_g > 0$ such that

$$\|\nabla_x f(x_k)\|_\infty \geq \kappa_g \tag{2.72}$$

for all k sufficiently large. Lemmas 2.15 and 2.16 show that, for any $\epsilon_i \in (0, 1)$, Algorithm 2.4 will generate an iterate k_i such that $\|g_{k_i}\| \leq \epsilon_i$ (at the beginning of Step 1). The mechanism of

Step 1 then implies that the sequence $\{k_i\}$ is infinite and that $\{\epsilon_i\}$ converges to zero. Regarding now the case where i is sufficiently large to ensure that

$$\epsilon_i \leq \frac{1}{2}\min\left[\frac{\kappa_c(1-\eta_1)}{\kappa_{eg}\Lambda}, \gamma_1\theta, \frac{\gamma_1\kappa_c(1-\eta_1)}{2\kappa_H}\right]\kappa_g. \tag{2.73}$$

Then Lemma 2.2 ensures that after Step 1 of Algorithm 2.4 is executed at iteration k_i,

$$\|\nabla_x f(x_{k_i}) - g_{k_i}\| \leq \kappa_{eg}\Lambda\epsilon_i \leq \frac{1}{2}\kappa_c(1-\eta_1)\|\nabla_x f(x_{k_i})\| \leq \frac{1}{2}\|\nabla_x f(x_{k_i})\|, \tag{2.74}$$

where we used (2.72) and (2.73). Thus, after Step 1 is executed at iteration k_i for i sufficiently large, using (2.72) and (2.74), one has that

$$\|g_{k_i}\|_\infty \geq \|\nabla_x f(x_{k_i})\|_\infty - \|\nabla_x f(x_{k_i}) - g_{k_i}\|_2 \geq \frac{1}{2}\kappa_g. \tag{2.75}$$

As a consequence, no loop occurs within Step 1 for large i and we have that

$$\Delta_{k_i} = \theta\|g_{k_i}\| \geq \frac{1}{2}\theta\kappa_g \geq \frac{\epsilon_i}{\gamma_1} > \epsilon_i, \tag{2.76}$$

using (2.73). Moreover, after applying a criticality step, we have that $v_i = x_{k_i}$ at all iterations between k_i and the next successful iteration. This means, together with (2.76), that no unsuccessful iteration of type 4c or 4d may occur before the next successful iteration or before the trust-region radius becomes smaller than ϵ_i. Iterations of type 4a can not occur as $p_k = p_{\max}$, so either a successful iteration (type 4b) occurs or the trust-region radius is decreased in Step 4e without altering the model. Iteration of type 4e may happen for $j \geq 0$ as long as

$$\Delta_{k_i+j} > \frac{\kappa_c(1-\eta_1)\kappa_g}{4\kappa_H}, \tag{2.77}$$

but Lemma 2.11 (which is a technical result based on the Cauchy condition and still holds with the new algorithm) and (2.75) imply that a successful iteration of type 4b must occur as soon as (2.77) is violated. It follows that a successful iteration $k_i + j_s$ must occur with

$$\Delta_{k_i+j_s} > \frac{\gamma_1\kappa_c(1-\eta_1)\kappa_g}{4\kappa_H} \stackrel{\text{def}}{=} \Delta_{\min} \geq \epsilon_i, \tag{2.78}$$

where (2.73) was used to get the last inequality. Since the model has not been recomputed between iterations k_i and $k_i + j_s$, we have from (2.75) that

$$\|g_{k_i+j_s}\| = \|g_{k_i}\| \geq \frac{1}{2}\kappa_g. \tag{2.79}$$

Using the Cauchy condition (3.10) and inserting (2.78) and (2.79) for the successful iteration $k_i + j_s$ yields, using the definition of ρ (2.4) and $\rho_{k_i+j_s} \geq \eta_1$, that

$$f(x_{k_i+j_s}) - f(x_{k_i+j_s+1}) \geq \frac{1}{2}\eta_1\kappa_c\kappa_g\min\left[\frac{\frac{1}{2}\kappa_g}{\kappa_H}, \Delta_{\min}\right] > 0. \tag{2.80}$$

Therefore, we see that the objective function must be unbounded below because the bound in (2.80) is satisfied for all i large enough to ensure (2.73). This is impossible in view of Assumption A2 and thus our assumption that (2.72) holds for all k sufficiently large is also impossible, and (2.71) follows which concludes the proof. □

2.5.4 Outlook

The extension of such a self-correcting derivative-free trust-region method to bound-constrained problems seems obvious but is in fact not as straightforward in practice as one could think. The difficulty is that the set of interpolation points may get aligned at one or more active bounds and deteriorate the geometry of the interpolation set. This led to the idea, which is developed in this thesis, to apply an active-set strategy to pursue minimization in the subspace of free variables to circumvent this difficulty. A detailed description of the algorithm can also be found in [74]. Numerical experiments performed with a practical implementation of algorithm UDFO (which was initiated by Philippe L. Toint who was also involved as well as Serge Gratton in different phases of its extension to handle bound-constrained problems) are reported in this thesis (and in [74]). This software implementation is called BC-DFO which is the acronym for Bound-Constrained Derivative-Free Optimization. Moreover, the main contribution in this thesis is the new algorithm ***BCDFO+*** which is presented in Chapter 3 of this thesis. It is an extension of Algorithm 2.4 (UDFO+) to handle bound-constraints using the mentioned active-set approach.

Chapter 3

A bound-constrained DFO algorithm

We consider the bound-constrained optimization problem

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x), \\ &\text{subject to} \quad l_i \le x_i \le u_i,\ i = 1,\dots,n \end{aligned} \tag{3.1}$$

where f is a nonlinear function from $\mathbb{R}^n$ into $\mathbb{R}$, which is bounded below, and where l and u are vectors of (possibly infinite) lower and upper bounds on x. We denote the feasible domain of this problem by $\mathcal{F}$.

Our approach uses an iterative trust-region method. As mentioned in Chapter 2, at each iteration of such a method, a model of the form

$$m_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T H_k s \tag{3.2}$$

(where g_k and H_k are the function's gradient and Hessian, respectively) is minimized inside a trust region $\mathcal{B}_\infty(x_k, \Delta_k)$. As derivatives are not given, g_k and H_k are approximated by determining its coefficients (here represented by the vector α) from the interpolation conditions

$$\big(m(y^i) =\big)\ \sum_{j=1}^{p} \alpha_j \phi_j(y^j) = f(y^i), i = 1,\dots,p. \tag{3.3}$$

The points $y^1,\dots,y^p$ considered in the interpolation conditions (3.3) form the interpolation set $\mathcal{Y}_k$. The set $\mathcal{Y}_k$ contains in our case at least $n+1$ points and is chosen as a subset of $\mathcal{X}_k$, the set of all points where the value of the objective function f is known. How to choose this interpolation set is of course one of the main issues we have to address below, as not every set $\mathcal{Y}_k$ is suitable because of poisedness issues.

We propose to handle the bound constraints by an "active-set" approach where a bound l_i or u_i is called active at x if $l_i = x_i$ or $x_i = u_i$, respectively. The bound constraints l_i and u_i are inactive if $l_i < x_i < u_i$ at x. A standard active-set method anticipates to update the set of active constraints while adding and/or removing constraints at each iteration whereas our approach allows only for adding constraints to the set of active constraints. Furthermore, our method considers all bound constraints which are active and in addition also those which are (presumably) nearly-active at the current iterate x_k. It then performs minimization in the subspace of the remaining inactive (also called free) variables.

This chapter is organized as follows. We outline the basic framework of our algorithm and discuss algorithmic concepts in Section 3.1. We present theoretical issues in Section 3.2, while Section 3.3 is concerned with practical implementation issues. In Section 3.4, the new algorithm BC-MS is presented to solve the trust-region subproblem in ℓ_2-norm where bound constraints are considered. Section 3.5 reports numerical experiments with our new algorithm ***BCDFO+*** where we first assess different types of models and two different local solvers to find the most suitable for our algorithm. Secondly, we will compare ***BCDFO+*** with the best model option to NEWUOA [123] and BOBYQA [124], two state-of-the-art packages applying also an interpolation-based trust-region method, and to its predecessor BC-DFO. Thirdly, we compare our new algorithm to different existing implementations from the class of direct-search methods.

3.1 A recursive active-set trust-region algorithm

3.1.1 Outline of the algorithm

An outline of our new bound-constrained algorithm is given in Algorithm 3.1 on page 33. This outline is purposely schematic and many of its details needs to be discussed. This discussion constitutes the body of this section.

3.1.2 Ensuring suitability of a tentative interpolation set

To safely build an initial interpolation model m_0 in Step 1 of Algorithm 3.1, we have to ensure that the used interpolation set $\mathcal{Y}_0$ is suitable in the sense that it is (sufficiently well) poised. At this stage, we are only given the tentative interpolation set $\mathcal{Z}_0$ and we target to obtain the set $\mathcal{Y}_0$ using $\mathcal{Z}_0$ as much as possible. We now describe the procedure used in our algorithm to modify $\mathcal{Z}_0$, if necessary, to form the set $\mathcal{Y}_0$. This procedure distinguishes two cases, depending on whether or not $\mathcal{Z}_0$ contains more than a single point.

If $|\mathcal{Z}_0| = 1$, our objective is then to build a poised interpolation set $\mathcal{Y}_0$ containing $\{x_0\} = \mathcal{Z}_0$ and contained in the initial trust region $\mathcal{B}_\infty(x_0, \Delta_0)$. This is achieved by choosing the interpolation points at the vertices of an n-dimensional simplex, as given by the formula

$$y_{i+1} = x_0 \pm \Delta_0 e_i, i = 1, 2, ..., n$$

where e_i is the i-th coordinate vector in $\mathbb{R}^n$ and the sign is negative for the initial computation of an interpolation set but, in an attempt to diversify the set of interpolation points, alternates whenever applied again during the minimization process.

If $|\mathcal{Z}_0| > 1$, an obvious choice would be to search first for the set $\mathcal{Y}_0 \subseteq \mathcal{Z}_0$ of size $p \leq n+1$ for which the condition number of the shifted and scaled system matrix $M(\bar{\phi}, \hat{\mathcal{Y}}_0)$ is the smallest out of all matrices associated with subsets of $\mathcal{Z}_0$ consisting of at most $\min(n+1, |\mathcal{Z}_0|)$ points. However, this procedure is quite costly due to its combinatorial nature, and we have decided to use a cheaper technique adapted from [20] which we explain now.

Algorithm 3.1 ***BCDFO+*** $(\mathcal{S}_0, \mathcal{X}_0, x_0, \mathcal{Z}_0, \Delta_0, \epsilon_0, \epsilon)$

Step 0: Initialization. A trust-region radius Δ_0, an accuracy threshold ϵ and $\epsilon_0 \geq \epsilon$ are given. An initial subspace $\mathcal{S}_0$, $\mathcal{X}_0$, the set of all points, and a tentative interpolation set $\mathcal{Z}_0$ which contains x_0 are also given. Parameters $\eta_1, \mu, \gamma_4 \in (0,1), 0 < \gamma_1 < \gamma_2 < 1, \theta > 0, \Lambda > 1$ and $p_{\max} \geq n+1$, the maximum number of points considered, are defined. Choose $v_0 \neq x_0$. Set $k = 0$ and $i = 0$.

Step 1: Ensure the suitability of $\mathcal{Z}_0$ and build the initial model. Update $\mathcal{Z}_0$ to obtain an interpolation set $\mathcal{Y}_0$ suitable for building an interpolation model with $|\mathcal{Y}_0| \geq \dim(\mathcal{S}_0)+1$. Then build the corresponding interpolation model m_0.

Step 2: Possibly restrict minimization to a subspace $\mathcal{S}_k$.

Step 2.1: Check for (nearly) active bounds. Determine (nearly) active bounds and the corresponding subspace $\mathcal{S}_k$ spanned by the remaining free variables. If there is no (nearly) active bound or if $\mathcal{S}_k$ has already been explored, go to Step 3. If all bounds are active, go to Step 2.5.

Step 2.2: Project information on the subspace of free variables. Project points in $\mathcal{X}_k$ which lie close to the (nearly) active bounds on $\mathcal{S}_k$ and associate with them suitable function value estimates. If the current iterate x_k is projected onto $\mathcal{S}_k$, compute $f(P_{S_k}(x_k))$ and if then $f(P_{S_k}(x_k)) > f(x_k)$, go to Step 4, otherwise set $x_k = P_{S_k}(x_k)$.

Step 2.3: Build a tentative interpolation set in the subspace. Build a new tentative interpolation set $\mathcal{Z}_k$ in $\mathcal{S}_k$ including the projected points, if any.

Step 2.4: Solve in $\mathcal{S}_k$ by a recursive call. Call

$$\boldsymbol{BCDFO+}(\mathcal{S}_k, \mathcal{X}_k, x_k, \mathcal{Z}_k, \Delta_k, \epsilon_i, \epsilon),$$

yielding a solution $x^*_{\mathcal{S}}$ of the subspace problem.

Step 2.5: Return to the full space. If $\dim(\mathcal{S}_k) < n$, return $x^*_{\mathcal{S}}$. Otherwise, redefine $x_k = x^*_{\mathcal{S}}$, construct a new interpolation set $\mathcal{Y}_k$ around x_k and build the corresponding model m_k. Go to Step 4.

Step 3: Avoid re-entering a subspace. If $\mathcal{S}_k$ has already been explored at x_k, then set $x_{k+1} = x_k$, reduce the trust-region radius $\Delta_{k+1} = \gamma_4 \Delta_k$ and build a new poised set $\mathcal{Y}_{k+1}$ in Δ_{k+1}. Compute m_{k+1} and increment k by one.

Step 4: Criticality test. Define $\hat{m}_i = m_k$.

Step 4.1: If $\|P_{\mathcal{F}}(x_k - \nabla_x \hat{m}_i(x_k)) - x_k\|_\infty < \epsilon_i$, set $\epsilon_{i+1} = \max(\epsilon, \mu \|P_{\mathcal{F}}(x_k - \nabla_x \hat{m}_i(x_k)) - x_k\|_\infty)$, compute a Λ-poised model $\hat{m}_{i+1}$ in $\mathcal{B}(x_k, \epsilon_{i+1})$ and increment i by one.
If $\|P_{\mathcal{F}}(x_k - \nabla_x \hat{m}_i(x_k)) - x_k\|_\infty \leq \epsilon$, return x_k, otherwise start Step 4.1 again.

Step 4.2: Set $m_k = \hat{m}_i, \Delta_k = \theta \|P_{\mathcal{F}}(x_k - \nabla_x m_k(x_k)) - x_k\|_\infty$ and define $v_i = x_k$ if a new model has been computed.

Step 5: Compute a trial point and evaluate the objective function. Compute $x_k^+ = x_k + s_k$ such that $m_k(x_k + s_k) = \min_{x \in \mathcal{B}(x_k, \Delta_k)} m_k(x)$. Evaluate f at x_k^+ and compute the ratio ρ_k from (2.4).

Step 6: Define the next iterate and update the trust-region radius. Decide how to possibly incorporate the current trial point x_k^+ into the set $\mathcal{Y}_{k+1}$, define x_{k+1} and Δ_{k+1} by applying Algorithm 3.3.

Step 7: Update the model. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation model m_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$. Update $\mathcal{X}_{k+1} = \mathcal{X}_k \cup \{x_{k+1}\}$. Increment k by one and go to Step 2.

Suppose that there exists a subset of points $\mathcal{W}_p = \{x^1, x^2, ..., x^p\}$ in $\mathcal{Z}_0$ spanning a $(p-1)$-dimensional linear manifold L. Our selection problem in $\mathcal{Z}_0$ can be seen as an optimal basis problem in $\mathcal{W}_p$ where we have to find $p-1$ vectors (of the form $x^i - x^j$) out of $\mathcal{Z}_0$ which are "as linearly independent as possible". This problem can be formalized by regarding the points in $\mathcal{W}_p$ as nodes of a graph and the vectors $x^i - x^j$ as edges e_{ij} in this graph. It can then be shown [21] that any set of $p-1$ linearly independent vectors of the form $x^i - x^j$ that generate L corresponds to a tree spanning all nodes of $\mathcal{W}_p$, and conversely. In addition, [20] shows that the optimal basis problem can be reduced to finding the spanning tree t which minimizes the functional

$$\phi(t) = \sum_{e_{ij} \in t} \|x^i - x^j\|_2. \tag{3.4}$$

Burdakov proposes a greedy algorithm for the solution of this minimization problem [21], in which the measure of linear independence given by $\Gamma(\{x^1, \ldots, x^p\}) = \det(A^T A)$ is exploited, where

$$A = \left[\frac{x^1 - x^2}{\|x^1 - x^2\|_2}, \cdots, \frac{x^{p-1} - x^p}{\|x^{p-1} - x^p\|_2} \right] \in R^{n \times p-1}.$$

It can be shown that Γ is a scaling invariant measure of linear independence of the columns of A and thus also measures the linear dependence of $\{x^1, \ldots, x^p\}$. It is always included in the interval $[0, 1]$ and takes the value 0 and 1 for linearly dependent and orthogonal columns, respectively, and the larger the value $\Gamma(\{x^1, \ldots, x^p\})$, the more linearly independent are the

---

Algorithm 3.2 Modified greedy algorithm for selecting a well-poised interpolation set (Inputs: $x_0, \mathcal{Z}_0$, Outputs: $\mathcal{W}_p, p$)

Step 1: Compute distances $\|x^i - x^j\|_2$ for $i, j = 1, \ldots, |\mathcal{Z}_0|$.

Step 2: Define $p = 1$, $\mathcal{W}_1 = \{x_0\}$ and $\mathcal{T}_0 = \emptyset$. Set $\Gamma(\mathcal{W}_1) = 1$ and $k = 0$.

while $(p < n + 1)$ and $(k < |\mathcal{Z}_0|)$ **do**

Step 3: Find $x^i \in \mathcal{W}_p$ and $x^j \in \mathcal{Z}_0 \setminus (\mathcal{W}_p \cup \mathcal{T}_k)$, such that $\|x^i - x^j\|_2$ is minimal.

Step 4: Compute the measure of degeneracy

$$\Gamma(\mathcal{W}_p \cup \{x^j\}) = \Gamma(\mathcal{W}_p) \frac{\|x^j_\perp\|_2^2}{\|x^i - x^j\|_2^2}$$

where $x^j_\perp = x^j - P_p x^j$, and P_p is the orthogonal projector on the linear manifold spanned by $\{x^i\}_1^p$.

Step 5: If $\Gamma(\mathcal{W}_p \cup \{x^j\}) \geq \kappa_{th}$, then set $\mathcal{W}_{p+1} = \mathcal{W}_p \cup \{x^j\}$, $\mathcal{T}_{k+1} = \mathcal{T}_k$, and increment p by one, else set $\mathcal{T}_{k+1} = \mathcal{T}_k \cup \{x^j\}$.

Step 6: Increment k by one.

end while

---

vectors $\{x^1, \ldots, x^p\}$ [20]. For a given threshold $\kappa_{th} \in (0, 1)$, we thus consider as sufficiently well-poised those sets of points, for which $\Gamma(\{x^1, \ldots, x^p\}) \geq \kappa_{th}$. It has also been proved in

[21] that $\Gamma(\{x^1, \ldots, x^p\})$ can be updated to $\Gamma(\{x^1, \ldots, x^p, x^{p+1}\})$ by a simple algebraic formula (used in Step 4 of Algorithm 3.2), thereby avoiding the repetitive computation of determinants.

As we do not know a subset of $\mathcal{Z}_0$ containing linearly independent points and not even the final number p of linearly independent points in $\mathcal{W}_p$, a modified version of Burdakov's greedy algorithm is proposed. In our version, the desired set is built incrementally in a sequence $\mathcal{W}_1$, $\ldots$, $\mathcal{W}_p$, where $\mathcal{W}_{p+1}$ is chosen over all sets of the form $\mathcal{W}_p \cup \{y\}$ for $y \in \mathcal{Z}_0 \setminus (\mathcal{W}_p \cup \mathcal{T}_k)$, where $\mathcal{T}_k$ contains the points which were tried but could not be included in $\mathcal{W}_p$ while keeping Γ sufficiently large. The algorithm is formalized as Algorithm 3.2 on page 34.

Note that p, the number of points selected by Algorithm 3.2 from $\mathcal{Z}_0$ may be smaller than $n+1$. In this case, we propose to complete $\mathcal{W}_p$ by $n+1-|\mathcal{W}_p|$ points selected randomly in the trust region to form the final interpolation set $\mathcal{Y}_0$. To ensure a good geometry of $\mathcal{Y}_0$, these random points $\{y^{p+1}, \ldots, y^{n+1}\}$ are then successively improved using the observation that replacing an interpolation point by the maximum of its associated Lagrange polynomial in the trust region ameliorates poisedness of the interpolation set (see [128], for instance). More precisely, for each $j = p+1, \ldots, n+1$, the absolute value of the Lagrange polynomial $\ell_j(x)$ associated with y^j is maximized inside $\mathcal{B}(x_k, \Delta_k)$ and y^j is then replaced by the computed maximizer $\tilde{y}^j$ (see also Algorithm 2.1). This finally yields the "optimized" interpolation set $\mathcal{Y}_0 = \mathcal{W}_p \cup \{\tilde{y}^{p+1}\} \cup \ldots \cup \{\tilde{y}^{n+1}\}$.

3.1.3 Recursive call in the subspace $\mathbf{S_k}$

As we have mentioned above, our algorithm is of the active-set type and proceeds by exploring the subspace $\mathcal{S}_k$ defined by fixing active or nearly active variables at their bounds (see Step 2 of Algorithm 3.1). This choice is intended to prevent the interpolation set from degenerating as would happen when points belonging to such a subspace are included in $\mathcal{Y}$. This section is devoted to the description of the mechanism for selecting (nearly) active bounds and then restarting the minimization in the associated subspace.

Before going into further details, we need to introduce some notation. We will drop the iteration counter k here for clarity. We will use the projection operator on the feasible set $\mathcal{F}$ which is defined componentwise by

$$[P_{\mathcal{F}}(x)]_i = \begin{cases} l_i & \text{if } x_i \le l_i, \\ u_i & \text{if } x_i \ge u_i, \\ x_i & \text{otherwise,} \end{cases} \tag{3.5}$$

where $i = 1, \ldots, n$. This operator projects the point x on the convex set $[l, u]$.

The lower and upper (nearly) active bounds at the point x are defined by those whose index is in one of the sets

$$\mathcal{L} = \{\, i \mid \; x_i - \nabla m_i < l_i \text{ and } x_i - l_i \le \epsilon_b \},$$
$$\mathcal{U} = \{\, i \mid \; x_i - \nabla m_i > u_i \text{ and } u_i - x_i \le \epsilon_b \},$$

where $\epsilon_b = \min\{\epsilon, |[P_{\mathcal{F}}(x - \nabla m) - x]_i|\}$, being the minimum of the required accuracy for termination ϵ and the absolute value of the appropriate projected model gradient component. Considering the combined measure ϵ_b on the bounds l and u indeed enables us to define not only currently active bounds but also "nearly-active" bounds which are presumed to become active in the next local minimization problem. If the set $\mathcal{L} \cup \mathcal{U}$ is non-empty, the minimization is then restricted to the affine subspace

$$\mathcal{S} = \{x \in \mathcal{F} \mid x_i = l_i \ \text{ for } \ i \in \mathcal{L} \ \text{ and } \ x_i = u_i \ \text{ for } \ i \in \mathcal{U}\}, \tag{3.6}$$

and the number of free variables consequently reduces to $n_{free} = n - |\mathcal{L} \cup \mathcal{U}|$.

Considering now iteration k, to pursue minimization in the subspace $\mathcal{S}_k$, a new linear model has to be built to initiate the computation of the first step, and the interpolation set for this model must consist of points lying exactly in this subspace. In an attempt to use all the available information when entering the subspace, all points of $\mathcal{X}_k$ lying inside a $\pm\epsilon_b$-region of the active bounds are projected on $\mathcal{S}_k$ (see Step 2.2 of Algorithm 3.1). To save function evaluations, function values corresponding to these projected points are not recomputed but replaced by model values, giving rise to points with approximate function values that we call "dummy" points. Specifically, we define

$$\mathcal{A}_k = \{y \in \mathcal{X}_k \mid 0 \leq |y_i - l_i| \leq \epsilon_b, \forall i \in \mathcal{L}_k \text{ and } 0 \leq |u_i - y_i| \leq \epsilon_b, \forall i \in \mathcal{U}_k\}, \tag{3.7}$$

where for at least one i, the strict inequality $0 < |y_i - l_i|, i \in \mathcal{L}_k$ or $0 < |u_i - y_i|, i \in \mathcal{U}_k$ must hold. The set $\mathcal{A}_k$ contains those points which are close to the active bounds but not on these. All points $y \in \mathcal{A}_k$ are then projected on $\mathcal{S}_k$, yielding $y_s = P_{\mathcal{S}_k}(y)$, and these "dummy" points $\{y_s\}$ are included in $\mathcal{X}_k$ with associated function values given by the model values $\{m_k(y_s)\}$. An exception is made when the current best point x_k belongs to $\mathcal{A}_k$ and is thus projected onto $\mathcal{S}_k$: the objective function is then evaluated at the projected point $P_{\mathcal{S}_k}(x_k)$, rather than m_k. If the new function value is such that the projected point is not the new best point, we refrain from exploring the newly defined subspace as the model gradient information at x_k turns out to be not correct. Minimization is pursued in the subspace $\mathcal{S}_{k-1}$, which we just left, instead. In the case where $f(P_{\mathcal{S}_k}(x_k)) < f(x_k)$, the current best point consequently changes to the projected point in $\mathcal{S}_k$. All dummy points are also appended to the set $\mathcal{D}um_k$, the set of all dummy points.

In Step 2.3 of Algorithm 3.1, a new tentative interpolation set $\mathcal{Z}_k = \mathcal{X}_k \cap \mathcal{S}_k$ is built and the algorithm then proceeds by recursively calling itself in the subspace $\mathcal{S}_k$, as indicated in Step 2.4 of the same algorithm.

While minimizing in $\mathcal{S}_k$, dummy points are successively replaced by real points with high priority (see Section 3.1.5 below). Moreover, we ensure that there is no dummy point in the interpolation set at a potential subspace solution: if the interpolation set still contains dummy points at this stage, the true function values are computed to ensure that convergence in $\mathcal{S}_k$ is solely based on real function values.

Once the algorithm has converged to an approximate solution of the problem restricted to $\mathcal{S}_k$, it must be verified whether it is also an approximate solution for the full-dimensional problem

(after adding the fixed components). Therefore, as mentioned in Step 2.5 of Algorithm 3.1, a safely non-degenerate full-space interpolation set of degree $n+1$ in an ϵ-neighbourhood around the suspected solution x^* is constructed following the technique described in Section 3.1.2. After computing the associated model, its gradient is checked for convergence (see Step 4 of Algorithm 3.1). If convergence can not be declared, minimization is continued in $\mathbb{R}^n$.

3.1.4 Local solver

To minimize the interpolation model m_k inside the intersection of the trust region and the bounds in Step 5 of Algorithm 3.1, a simple projected truncated conjugate-gradient algorithm is used, as in the LANCELOT package [33]. As is standard for such techniques, the set of active bounds is never reduced and a piece-wise linesearch is performed on the path defined by the projection of the current search direction onto the feasible set $\mathcal{F}$ if a new bound is hit in the course of the conjugate-gradient calculation. The computation is stopped as soon as the iterates leave the trust region or all bounds are active. Further, in our experiments, the computation is also terminated if the size of the projected gradient falls below a threshold of 10^{-9} or the number of conjugate-gradient iterations exceeds the limit of 1500.

Remark: In our method, as in standard trust-region methods, it is imposed that $m_k(x_k^+)$ must be "sufficiently small" compared to $m_k(x_k)$. To satisfy this model decrease condition, m_k is usually minimized (as in our case), but the theory just requires the condition that

$$m_k(x_k) - m_k(x_k^+) \geq \kappa_c \|P_{\mathcal{F}}(x_k - \nabla_x m(x_k)) - x_k\| \min\left[\frac{\|P_{\mathcal{F}}(x_k - \nabla_x m(x_k)) - x_k\|}{\beta_k}, \Delta_k, 1\right], \tag{3.8}$$

where β_k is an upper bound on the condition number of the model Hessian defined as

$$\beta_k \stackrel{\text{def}}{=} 1 + \|H_k\| \tag{3.9}$$

and a constant $\kappa_c \in (0,1)$ (see [31] and [103]) This condition is well-known in trust-region analysis under the name of "Cauchy condition", and indicates that the model reduction must be at least a fraction of that achievable along the steepest descent direction while remaining in the trust region. In the unconstrained case, (3.8) reduces to

$$m_k(x_k) - m_k(x_k^+) \geq \kappa_c \|g_k(x_k)\| \min\left[\frac{\|g_k(x_k)\|}{1 + \|H_k\|}, \Delta_k\right]. \tag{3.10}$$

3.1.5 Defining the next iterate

At each iteration of a classical trust-region method, a new trial point x_k^+ is computed by minimizing the interpolation model m_k inside the trust-region Δ_k. The point x_k^+ is accepted to be the new iterate x_{k+1} if the ratio ρ between achieved and predicted reduction (2.4) exceeds a constant η_1. In this case, the iteration is declared successful. Otherwise, the iteration is unsuccessful.

Following [128], we always try to improve the geometry by replacing an appropriate point from the set in unsuccessful iterations and if we cannot find such a point, the trust-region radius is decreased. But contrary to [128], we also decrease the trust-region radius Δ in unsuccessful iterations in which a point was replaced (as long as the trust-region radius is larger than a fixed threshold Δ_{switch}) as in the unconstrained algorithm UDFO+ described in Section 2.5. This is a slightly more aggressive strategy as applied in [128] because we do not wait until the geometry is good enough to conclude that the trust-region radius must have been too large to progress. Finally, this resembles the trust-region management of a standard trust-region method and the algorithm is given the possibility to progress in a smaller trust-region where the model might be more suitable. We now describe the details of the complete replacement/updating procedure. An algorithmic outline of this routine is given in Algorithm 3.3 which is a detailed version of Step 6 of Algorithm 3.1.

The first step is to check whether the current model is already quadratic (see Step 6a). If $p_k < p_{\max}$, the size of the interpolation set is augmented by the new trial point. If the iteration is successful, we update the iterate by $x_{k+1} = x_k^+$ and the trust-region radius by

$$\Delta_{k+1} = \min(\max(\gamma_3 \|s_k\|_\infty, \Delta_k), \Delta_{\max}). \tag{3.11}$$

In an unsuccessful iteration, we define $x_{k+1} = x_k$ and if $\Delta_k > \Delta_{\text{switch}}$, we set $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$, otherwise $\Delta_{k+1} = \Delta_k$ is kept.

When the model is quadratic, we try to replace the dummy points in the current interpolation set to avoid keeping approximate information in the model for too long (see Step 6b). If there are any dummy points in the current interpolation set (what means that $\mathcal{D}um_k \cap \mathcal{Y}_k \neq \emptyset$) for which $\ell_{k,j}(x_k^+)$ (the value of the associated Lagrange polynomial evaluated at the trial point) is nonzero, the dummy point for which $\ell_{k,j}(x_k^+)$ is largest in absolute value is replaced by x_k^+. In an unsuccessful iteration, the trust-region radius is decreased if $\Delta_k > \Delta_{\text{switch}}$. If the current iteration is successful, we have to update the trust region as in (3.11) and the current best iterate and thus the center of the interpolation set is also updated.

If the trial point could not yet be included in the interpolation set, we apply a strategy close to the one described in Algorithm 2.3 in Section 2.4 for the unconstrained case (see also [128, Algorithm 2]). If the iteration is successful, we define, as above, $x_{k+1} = x_k^+$ and update the radius by (3.11) in Step 6c. In the interpolation set, one point $y_{k,r}$ is then replaced by the trial point $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ for

$$r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.12}$$

In the unsuccessful case, i.e. when $\rho < \eta_1$, we still attempt to include the trial point in the interpolation set to improve its geometry. To do so, a point from $\mathcal{Y}_k \setminus \{x_k\}$ has to be replaced by x_k^+ and we first attempt to replace a far interpolation point (see Step 6d). If the set

$$\mathcal{F}_k \overset{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \ \mid \ \|y_{k,j} - x_k\|_\infty > \beta \Delta_k \text{ and } \ell_{k,j}(x_k^+) \neq 0\} \tag{3.13}$$

is non-empty, where $\beta \geq 1$, then we set $x_{k+1} = x_k$, $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$ if $\Delta_k > \Delta_{\text{switch}}$, otherwise $\Delta_{k+1} = \Delta_k$. We define the new interpolation set by $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where

Algorithm 3.3 Define the next iterate

Step 6a: Augment interpolation set ($\mathbf{p_k < p_{\max}}$). If $p_k < p_{\max}$, then: Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$. If $\rho_k \geq \eta_1$, then define $x_{k+1} = x_k^+$ and choose $\Delta_{k+1} \geq \Delta_k$ as in (3.11). If $\rho_k < \eta_1$, define $x_{k+1} = x_k$ and if $\Delta_k > \Delta_{\text{switch}}$, set $\Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$, otherwise $\Delta_{k+1} = \Delta_k$.

Step 6b: Replace a dummy interpolation point. If $p_k = p_{\max}$ and the set

$$\mathcal{D}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{D}um_k \cap \mathcal{Y}_k\}$$

is non-empty, then: Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where $\ell_{k,r}(x_k^+) \neq 0$ and r is an index of any point in $\mathcal{D}_k$, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|$. If $\rho_k \geq \eta_1$, then define $x_{k+1} = x_k^+$ and choose $\Delta_{k+1} \geq \Delta_k$ as in (3.11). If $\rho_k < \eta_1$, then define $x_{k+1} = x_k$ and if $\Delta_k > \Delta_{\text{switch}}$, update $\Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$, otherwise we set $\Delta_{k+1} = \Delta_k$.

Step 6c: Successful iteration. If $p_k = p_{\max}$, $\rho_k \geq \eta_1$ and the set $\mathcal{D}_k = \emptyset$, then: Define $x_{k+1} = x_k^+$, choose $\Delta_{k+1} \geq \Delta_k$ as in (3.11). Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point $y_{k,j}$ in $\mathcal{Y}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|$.

Step 6d: Replace a far interpolation point. If $p_k = p_{\max}$, $\rho_k < \eta_1$, $[x_k \neq v_i$ or $\Delta_k \leq \epsilon_i]$, the set $\mathcal{D}_k = \emptyset$ and the set

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

is non-empty, then: Set $x_{k+1} = x_k$, $\Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$ (or set $\Delta_{k+1} = \Delta_k$ if $\Delta_k \leq \Delta_{\text{switch}}$). Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point $y_{k,j}$ in $\mathcal{F}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|$.

Step 6e: Replace a close interpolation point. If $p_k = p_{\max}$, $\rho_k < \eta_1$, $[x_k \neq v_i$ or $\Delta_k \leq \epsilon_i]$, the set $\mathcal{D}_k \cup \mathcal{F}_k = \emptyset$ and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \text{ such that } \|y_{k,j} - x_k\| \leq \beta\Delta_k \text{ and } \ell_{k,j}(x_k^+) > \Lambda\}$$

is non-empty, then: Set $x_{k+1} = x_k, \Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$ (or set $\Delta_{k+1} = \Delta_k$ if $\Delta_k \leq \Delta_{\text{switch}}$). Define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point $y_{k,j}$ in $\mathcal{C}_k$, for instance, such that $r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|$.

Step 6f: Reduce the trust region radius. If $p_k = p_{\max}$, $\rho_k < \eta_1$ and either $[x_k = v_i$ and $\Delta_k > \epsilon_i]$ or $\mathcal{D}_k \cup \mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then: Set $x_{k+1} = x_k, \Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$ and define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

r is the index j of any point $y_{k,j}$ in $\mathcal{F}_k$, for instance such that

$$r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)| \tag{3.14}$$

or

$$r = \arg\max_j \|y_{k,j} - x_k^+\|^2. \tag{3.15}$$

If the set $\mathcal{F}_k$ is empty and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y_{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \;\mid\; \|y_{k,j} - x_k\|_\infty \leq \beta\Delta_k \text{ and } \ell_{k,j}(x_k^+) > \Lambda_C\} \tag{3.16}$$

is non-empty, where $\Lambda_C > 1$ is defined by the user, we then set $x_{k+1} = x_k$, $\Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$ if $\Delta_k > \Delta_{\text{switch}}$, otherwise $\Delta_{k+1} = \Delta_k$ (see Step 6e). The new interpolation set is defined by $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index j of any point $y_{k,j}$ in $\mathcal{C}_k$, for instance such that

$$r = \arg\max_j \|y_{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)| \tag{3.17}$$

or

$$r = \arg\max_j |\ell_{k,j}(x_k^+)|. \tag{3.18}$$

(The current default in our algorithm, based on our numerical experience, is to choose (3.15) and (3.18). This seems also natural as the target is to first remove far points to improve the locality of the model and then to improve the poisedness of the set when all points are rather close to the current iterate.)

If the trial point could not be included into the interpolation set under the above conditions, it implies that the interpolation set must be reasonably poised, as otherwise we could have improved it. As a consequence, we set $x_{k+1} = x_k, \mathcal{Y}_{k+1} = \mathcal{Y}_k$ and $\Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$ in Step 6f.

3.1.6 Avoid re-entering a subspace

We have stated in Step 2.1 of Algorithm 3.1 that we never re-enter a subspace $\mathcal{S}_k$ which has already been explored at the current iterate x_k as this is of no interest anymore. Instead, we proceed in this situation with a criticality step in a reduced trust region (Step 3 of Algorithm 3.1). We now justify that feature.

Imagine the following situation: convergence is declared in subspace $\mathcal{S}_k$ and a new model of degree one is built at x^* in $\mathbb{R}^n$. Assume also that x^* is a solution of the full-space problem. It may then happen that the model gradient $\nabla m_k(x^*)$ of the linear model m_k is too large to declare convergence in $\mathbb{R}^n$, because m_k is not a sufficiently accurate model even if the interpolation set is well poised.

Indeed, we know (Theorem 2.11, p. 29, in [41]) that, for linear models,

$$\|\nabla_x f(y) - \nabla_x m(y)\|_2 \leq \kappa_{eg}\Delta, \tag{3.19}$$

and since a projection onto a convex set is Lipschitz continuous in 2-norm we have that

$$\|P_{\mathcal{F}}(y - \nabla_x f(y)) - y - [P_{\mathcal{F}}(y - \nabla_x m(y)) - y]\|_2 \leq \|\nabla_x f(y) - \nabla_x m(y)\|_2 \leq \kappa_{eg}\Delta, \tag{3.20}$$

where κ_{eg} is given by

$$\kappa_{eg} = \nu(1 + n^{\frac{1}{2}}\|\hat{L}^{-1}\|_2/2), \tag{3.21}$$

where $\hat{L} = \frac{L}{\Delta} = \frac{1}{\Delta}[y_2 - y_1, ..., y_n - y_1]$ and ∇f is Lipschitz continuous with constant $\nu > 0$. As a consequence, a big difference between ∇m_k and ∇f can only occur either because κ_{eg} or the

trust-region radius are too large. As the size of the Lipschitz constant is beyond our control, building a new well-poised set in a reduced trust-region radius must solve the problem because (3.19) implies that the model gradient will converge to the true one. This procedure can be seen as a criticality step and either convergence will be declared (in Step 4 of Algorithm 3.1) or the algorithm may proceed with a more accurate model.

3.2 Theoretical issues

In this section, we want to talk about the convergence of our bound-constrained algorithm ***BCDFO+*** presented as Algorithm 3.1 in the previous section.

3.2.1 Global convergence

After having presented a bound-constrained algorithm, which successfully combines techniques from different fields of optimization, it turned out that proving global convergence of our recursive active-set trust-region algorithm is rather difficult.

In fact, active-set methods are widely used in the context of gradient-based optimization [22, 31, 34, 80, 106] to efficiently handle bound constraints. Due to the availability of first-order information, these methods check the validity of the active set at every iteration. In our derivative-free context we decided, to avoid degeneration of the sample set, to pursue minimization in the subspace, where active bounds are allowed to be added but not to be released, until convergence is declared.

We had the idea, that such a recursive subspace-step can be regarded as a magical step, as described in [44] and [34, Chapter 10.4.1]. A magical step s_k^{MA} is an additional step which is performed, once the model-decreasing step s_k has been determined. Such a composite step is better than s_k in the sense that $f(x_k + s_k + s_k^{MA}) \leq f(x_k + s_k)$. In our context, a subspace step could be seen as such a magical step and would contribute to the minimization process of the full-space problem with a "very successful" step. One problem is that our algorithm does not obey the rules for the trust-region management of a method including magical steps as this would be, for instance, to apply a strict trust-region management in the full-space. This would mean to expand or contract the trust-region radius in the full-space after a subspace step according to its value before the subspace step what is not the case in our algorithm. Furthermore, and more importantly, the convergence theory of an algorithm with magical steps is still based on the decrease achieved at the step s_k. This is different in our approach because convergence may be declared inside a subspace in the bound-constrained case. This implies that we may have converged after a magical step and can not ensure to produce infinitely many steps in the full-space what would be essential to prove global convergence of our method.

Hence, magical steps do not apply naturally in our context; we could think that it might be more appropriate to apply the theory of a multi-level-type algorithm as this keeps track of what happens in the subspace. But in multi-level trust-region methods (as e.g. in [71, 73, 103]),

the minimization in the subspace (coarser level) has to be proceeded inside the trust-region of the full-space (fine level), or a bounded violation of it. Thus, this theory does not apply either to our algorithm. Another point is that the decision on whether to proceed minimization in a subspace or not is taken by ensuring that the reduction in the subspace at the first successful iteration there gives a Cauchy decrease which is a fraction of that which would be obtained in the full-space. Such a descent condition is not ensured by our algorithm because the decision is not taken based on the sufficient decrease in the subspace compared to the full-space but to prevent degeneration of the interpolation set.

Nevertheless, we still believe that it is possible to prove global convergence of our algorithm without compromising its efficiency but this seems to be a bit more complex and is left as an open task to deal with in the near future.

3.3 Practical implementation issues

The description of our algorithm ***BCDFO+*** in the previous section leaves a number of practical questions open. In this section, we briefly report some further details about the particular implementation of the algorithm whose numerical performance is reported below in Section 3.5.

3.3.1 Preparing the initial call

At the first call to ***BCDFO+***, it is assumed that

$$\Delta_0 \leq \frac{1}{2} \min_{i=1,\ldots,n} (u_i - l_i) \quad \text{and} \quad l_i + \Delta_0 \leq x_{0i} \leq u_i - \Delta_0, \tag{3.22}$$

where x_i denotes the i-th component of the vector x.

Further, the initial call is performed with $\mathcal{S}_0 = \mathbb{R}^n$, which means that the initial subspace is the n-dimensional full space. Both, the set of all points for which the function values have been computed and the initial tentative interpolation set contain only the starting point x_0 (i.e. $\mathcal{X}_0 = \mathcal{Z}_0 = \{x_0\}$).

We finally note that in our implementation, only a matrix containing the set $\mathcal{X}_k$ and a vector of the associated function values $f_{\mathcal{X}_k}$ are stored. The points contained in the subsets $\mathcal{Z}_k$ and $\mathcal{Y}_k$, and their function values respectively, are held as integer pointers to $\mathcal{X}_k$ and $f_{\mathcal{X}_k}$.

3.3.2 Handling fixed variables

In practice, it is often very convenient for users of an optimization package, to be able to fix the value of certain variables. Hence, we have that

$$l_i = x_{0i} = u_i.$$

In order to handle such a case, we check for variables where $u_i - l_i = 0$. The corresponding indices i together with their fixed values x_{0i} are then stored in a vector for use when evaluating

the objective function throughout the calculation, but the associated variables are otherwise excluded from the minimization process.

3.3.3 Representation of the Lagrange polynomials

As $\mathcal{Y}$ varies, the code computes a QR factorization

$$M(\phi, \mathcal{Y})^T = Q_{\mathcal{Y}} R_{\mathcal{Y}}$$

of the matrix of the system (2.8) (or of its shifted version $\hat{M}$ if appropriate), where the basis ϕ is that of the monomials. If the vector ℓ_j contains the coefficients of the Lagrange polynomial $\ell_j(x)$ associated with $\mathcal{Y}$, their definition implies that they have to satisfy $M(\phi, \mathcal{Y})\ell_j = e_j$ and hence may be retrieved from the formula $\ell_j = Q_{\mathcal{Y}} R_{\mathcal{Y}}^{-T} e_j$.

3.3.4 Controlling the condition of the system matrix

We have shown that Algorithm 2.3 and Algorithm 3.1 are globally convergent algorithms which apply a self-correcting property. We repeat here that the self-correcting property in Lemma 2.13 states that the geometry of the interpolation set $\mathcal{Y}$ improves in unsuccessful iterations when the trust region is small (relative to the model's gradient) and the interpolation points are contained in the trust region. For replacing a point in a unsuccessful iteration in the case, there are points lying outside of the trust region, the theory in Section 3.2 says that if $|\ell_j(x^+)| \neq 0$, hence the absolute value of the corresponding Lagrange polynomial evaluated at the new point is not zero and thus the system matrix $M(\bar{\phi}, \mathcal{Y})$ can not become singular. Whereas no care of the poisedness of the interpolation set is taken when a successful point is included in the set.

In practice, even shifting and scaling of the interpolation set to work with $M(\bar{\phi}, \hat{\mathcal{Y}})$ doesn't prevent the condition number of $M(\bar{\phi}, \hat{\mathcal{Y}})$ from growing. In the context of finite precision arithmetic, we may for instance encounter numerical difficulties in the following three situations:

- **The gap between two geometry improving steps is too big.** Geometry improving steps are invoked when the gradient falls below a certain threshold ϵ_i. This threshold is adapted after each such step by multiplying ϵ_i with a constant $\mu \in [0, 1)$ to be the new threshold for the next geometry improving step. The size of μ and thus the frequency of the improving steps is user-defined and if it is chosen too small, the geometry may deteriorate before such a step is performed. For instance, imagine, a lot of progress was made towards the solution and many points of the set are far from the current iterate whereas, at the same time, the algorithm starts to sample points in the close neighbourhood of the current iterate. As a consequence, the algorithm is busy with replacing far points without taking care of the poisedness of the set of points. This slow deterioration of the sample set could be prevented by asking for a big enough absolute value of the Lagrange polynomial associated with the far point y_j to replace and evaluated at the new trial point x^+ and test the condition $|\ell_j(x^+)| \geq \kappa_\ell$ to be true before including the new point. But from

our experience we know that it is nearly impossible to find a good default value for κ_ℓ. Because such a threshold must not be too small to get into troubles in any test case of any test set and it should also not be too big to reject unnecessarily many points in other test cases as this has a bad influence on the performance of the method which relies on the constant replacement of points by the new trial points.

- **The real problem Hessian is ill-conditioned.** A good example is the unconstrained quadratic problem NASTY from the CUTEr collection. For instance, the solvers NEWUOA and BOBYQA (with the default parameters) are not able to solve this problem. The problem is two-dimensional and the Hessian of the problem writes

$$H_0 = \begin{bmatrix} 10^{20} & 0 \\ 0 & 1 \end{bmatrix}.$$

  where the condition number of H_0 is 10^{20}. This provokes the difficulty that the model's gradient is quite big (due to the nature of the problem) and that convergence is slow. Moreover, the structure of the problem causes the trial points to lie more or less all in one direction what is also not desirable for the interpolation set geometry. As mentioned above, a too big gradient prevents the algorithm from applying a geometry improving step and thus the condition number of the system matrix is in parts strictly increasing and the set of points may deteriorate at some point.

- **The degree of the model is between linear and quadratic.** According to the proposed algorithm, if the set of points for quadratic interpolation contains less than $\frac{1}{2}(n+1)(n+2)$ points, every new trial point is added to the set. As there's not a point to replace in this case, checking Lagrange polynomial values is not an option in this case. Nevertheless, the poisedness of $\mathcal{Y}_k$ may deteriorate when appending new points.

As a solution to all of these practical issues and thus to be sure to have a reliable algorithm, the condition number of the shifted and scaled system matrix $M(\bar{\phi}, \hat{\mathcal{Y}})$ (or $F(\bar{\phi}, \hat{\mathcal{Y}})$ for the minimum Frobenius-norm model) is computed at each iteration before building the polynomial model. The threshold κ_{illcond} for declaring $M(\bar{\phi}, \hat{\mathcal{Y}})$ or $F(\bar{\phi}, \hat{\mathcal{Y}})$ as badly conditioned is user-defined in the software. In the case $\text{cond}(\hat{M}) > \kappa_{\text{illcond}}$, the singular value decomposition of this matrix is determined and all singular values smaller than a threshold δ are replaced by this threshold. Then the corresponding system which computes the model is solved. If we want to guarantee that the error bound on the gradient for (at least) linear interpolation is not violated, we have to determine the maximum size of such a δ. In the following, we want to establish an error bound on the model gradient $\nabla m(\tilde{y})$ of the perturbed model. The general assumption is the following.

Assumption 3.1. *We assume that $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\} \subset \mathbb{R}^n$ is a poised set of sample points (in the linear interpolation sense) contained in the ball $\mathcal{B}(y^1, \Delta(\mathcal{Y}))$ of radius $\Delta = \Delta(\mathcal{Y})$. Further, we assume that the function f is continuously differentiable in an open domain Ω containing $\mathcal{B}(y^1, \Delta)$ and ∇f is Lipschitz continuous in Ω with constant $\nu > 0$.*

Another assumption needed for our result is stated below.

Assumption 3.2. *We assume that each entry of the interpolation set* $\mathcal{Y} = \{y^1, y^2, \ldots, y^p\}$ *is perturbed so that we have a set of points* $\tilde{\mathcal{Y}} = \{\tilde{y}^1, \tilde{y}^2, \ldots, \tilde{y}^p\}$ *with* $\tilde{y}_i^j = y_i^j + \delta_i^j$ *for* $i = 1, ..., n, j = 1, ..., p$ *where*

$$\delta_i^j \leq \delta = \frac{\nu}{4} \frac{\Delta^2}{\|\nabla m(\tilde{y})\|} \quad \text{for} \quad i = 1, ..., n, j = 1, ..., p. \tag{3.23}$$

Furthermore, we assume that

$$\|\delta^j - \delta^1\| \leq \delta, \tag{3.24}$$

where $\delta^j = (\delta_i^j)_{i=1}^n$.

As in [42, Theorem 2.11], we want to establish our result using the scaled matrix

$$\hat{L} = \frac{1}{\Delta} L = \frac{1}{\Delta} \begin{bmatrix} y^2 - y^1 & \ldots & y^p - y^1 \end{bmatrix}^T = \begin{bmatrix} \frac{y_1^2 - y_1^1}{\Delta} & \ldots & \frac{y_n^2 - y_n^1}{\Delta} \\ \vdots & \vdots & \vdots \\ \frac{y_1^p - y_1^1}{\Delta} & \ldots & \frac{y_n^p - y_n^1}{\Delta} \end{bmatrix},$$

where the matrix

$$\begin{bmatrix} 1 & {y^1}^T \\ 0 & L \end{bmatrix}$$

is a block expression of the matrix $M = M(\bar{\phi}, \mathcal{Y})$ after performing one step of Gaussian elimination to M. This leads to the fact that the matrix L is nonsingular if and only if M is nonsingular, since $\det(L) = \det(M)$.

We can establish the following result in terms of the perturbed set $\tilde{\mathcal{Y}}$:

Theorem 3.1. *Let Assumption 3.1 and Assumption 3.2 hold. The gradient of the linear interpolation model satisfies, for all points* y *in* $\mathcal{B}(y^1, \Delta)$ *and their corresponding perturbed versions* $\tilde{y}$*, an error bound of the form*

$$\|\nabla f(y) - \nabla m(\tilde{y})\| \leq \kappa_{egp} \Delta, \tag{3.25}$$

where $\kappa_{egp} = \nu(1 + \sqrt{p-1}\|\hat{L}^{-1}\|)$ *and* $\hat{L} = L/\Delta$.

Proof. We assumed that the set $\mathcal{Y}$ is poised, thus we have that M and L are nonsingular. We consider the perturbed system matrix

$$M(\bar{\phi}, \tilde{\mathcal{Y}}) = \begin{bmatrix} 1 & y_1^1 + \delta_1^1 & \ldots & y_n^1 + \delta_n^1 \\ \vdots & \vdots & & \vdots \\ 1 & y_1^p + \delta_1^p & \ldots & y_n^p + \delta_n^p \end{bmatrix}. \tag{3.26}$$

Now, we look at the gradient of f at the point y^1. Substracting the first interpolating condition from the remaining $p - 1$, we obtain

$$\left[(y^j + \delta^j) - (y^1 + \delta^1) \right]^T \nabla m(\tilde{y}) = f(y^j) - f(y^1), \quad j = 2, \ldots, p$$

and thus we get

$$(y^j - y^1)^T \nabla m(\tilde{y}) = f(y^j) - f(y^1) - (\delta^j - \delta^1)^T \nabla m(\tilde{y}), \quad j = 2, \ldots, p.$$

Then, using the integral form of the mean value theorem

$$f(y^j) - f(y^1) = \int_0^1 (y^j - y^1)^T \nabla f(y^1 + t(y^j - y^1))\, \mathrm{d}t,$$

we obtain

$$(y^j - y^1)^T (\nabla f(y^1) - \nabla m(\tilde{y})) \leq \int_0^1 (y^j - y^1)^T [\nabla f(y^1) - \nabla f(y^1 + t(y^j - y^1))]\, \mathrm{d}t - (\delta^j - \delta^1)^T \nabla m(\tilde{y}),$$

for $j = 2, \ldots, p$. From the Lipschitz continuity of ∇f, we get that

$$\|(y^j - y^1)^T (\nabla f(y^1) - \nabla m(\tilde{y}))\| \leq \frac{\nu}{2} \|y^j - y^1\|_2^2 + \|\delta^j - \delta^1\|_2 \|\nabla m(\tilde{y})\|_2, \quad j = 2, \ldots, p.$$

Then, from these last $p-1$ inequalities and (3.24), it can be derived

$$\|L(\nabla f(y^1) - \nabla m(\tilde{y}))\|_2 \leq \sqrt{p-1}\, \frac{\nu}{2}\, \Delta^2 + 2\sqrt{p-1}\, \delta\, \|\nabla m(\tilde{y})\|_2,$$

from which it can be concluded that

$$\|\nabla f(y^1) - \nabla m(\tilde{y})\|_2 \leq \sqrt{p-1}\, \|\hat{L}^{-1}\| (\frac{\nu}{2} \Delta + \frac{2\delta\, \|\nabla m(\tilde{y})\|_2}{\Delta}).$$

Applying (3.23) gives

$$\|\nabla f(y^1) - \nabla m(\tilde{y})\|_2 \leq \sqrt{p-1}\, \|\hat{L}^{-1}\| \nu \Delta.$$

Now, the error bound for any point y in the ball $\mathcal{B}(y^1, \Delta)$ and its corresponding perturbed point $\tilde{y}$ can be derived from the Lipschitz continuity of the gradient of f:

$$\|\nabla f(y) - \nabla m(\tilde{y})\|_2 \leq \|\nabla f(y) - \nabla f(y^1)\|_2 + \|\nabla f(y^1) - \nabla m(\tilde{y})\|_2 \leq \nu (1 + \sqrt{p-1}\, \|\hat{L}^{-1}\|) \Delta.$$

□

This result allows for replacing small singular values of $\hat{M}$ by a value δ to safeguard the computation of the model. More precisely, in case the condition number of $\hat{M}$ passes the threshold κ_{illcond}, δ is computed from (3.23). As the computation of δ involves the gradient of the perturbed model which is unknown when δ is computed, its value is determined by using the model gradient of the last iteration. This approximation has shown to be sufficient in our experiments.

Please note that the above described value of δ is valid for linear interpolation but it is at the moment also applied when working with quadratic models and also in the regression case. The described strategy works well in our general algorithmic context but may not guarantee error bounds for interpolation when in addition quadratic polynomial bases are considered to build a model or when regression models are considered. For this reason, we plan to extend our theory above to the mentioned cases in a future project.

3.3.5 Implementation of the models

In this section, we describe our particular implementation in Matlab [1] of the different model types considered in this work, as there are the sub-basis model, the minimum ℓ_2-norm model, the minimum Frobenius-norm model and the least-squares regression model.

In any case, if $p_k = |\mathcal{Y}_k| = n + 1$, a linear model and not an underdetermined quadratic model is built. This stems from the different theoretical model gradient errors of this two kinds of models. In fact, we know from [41, 45] that the error bound for the gradient approximation in the linear case is written as

$$\|\nabla f(y) - \nabla m(y)\| \leq \nu(1 + \sqrt{n}\|\hat{L}^{-1}\|/2)\Delta, \tag{3.27}$$

and the error bound for the gradient of an underdetermined quadratic model involves the model Hessian norm and is written as

$$\|\nabla f(y) - \nabla m(y)\| \leq \frac{5}{2}\sqrt{p}\|\hat{L}^{\dagger}\|(\nu + \|H\|)\Delta, \tag{3.28}$$

where $\hat{L}^{\dagger}$ denotes again the Moore-Penrose generalized inverse of the matrix $\hat{L}$. The scaled matrix $\hat{L}$ (see Section 3.3.4 for the definition) corresponds to a scaled sample set contained in a ball of radius 1 centered at y^1/Δ, i.e.,

$$\hat{Y} = \{y^1/\Delta, y^2/\Delta, \ldots, y^p/\Delta\} \subset \mathcal{B}(y^1/\Delta, 1).$$

From this we can see that the model gradient of a linear polynomial might be more accurate than the one of an underdetermined quadratic polynomial using $n+1$ interpolation conditions. This is a very crucial point in our work because our algorithm relies on the accuracy of the model gradient as a stopping criterion. Especially in a bound-constrained calculation when coming back from exploring a subspace and checking convergence in the full-space with a sample set containing only $n + 1$ points, we have to take care of the quality of the model gradient to be able to declare convergence.

Furthermore, our implementation of ***BCDFO+*** is using the function `pinv()` when computing the least-squares solution in the under- and overdetermined case, meaning that the minimum ℓ_2-norm and regression models are built by computing the Moore-Penrose pseudo-inverse instead of using the simpler (and faster) backslash-operator. The reason is that the solution to the least-squares system using the backslash-operator in Matlab (version 7.1.0.183 with service pack 3) gives the solution with the least number of non-zero entries and not the least-norm solution.

A particularity of our implementation of the minimum Frobenius-norm model is, to solve the problem $M(\bar{\phi}, \mathcal{Y})\alpha = f(\mathcal{Y})$ when quadratic degree is reached rather than continuing to solve $F(\bar{\phi}, \mathcal{Y})\ [\lambda,\ \mu]^T = [f(\mathcal{Y}),\ 0]^T$. In the latter case, each iteration involves a matrix-matrix-product to establish $F(\bar{\phi}, \mathcal{Y})$ which is not expensive compared to a function evaluation in some industrial test cases as those considered in Chapter 4 of this thesis. The main reason, to work with $M(\bar{\phi}, \mathcal{Y})$ rather than with $F(\bar{\phi}, \mathcal{Y})$ in the complete quadratic case is that the condition

number of the matrix $F(\bar{\phi}, \mathcal{Y})$ may be of the order of the square of the condition number of $M(\bar{\phi}, \mathcal{Y})$ due to its construction which is stated here again for convenience (see also Section 2.2.4)

$$F(\bar{\phi}, \mathcal{Y}) = \begin{bmatrix} M(\bar{\phi}_Q, \mathcal{Y})M(\bar{\phi}_Q, \mathcal{Y})^T & M(\bar{\phi}_L, \mathcal{Y}) \\ M(\bar{\phi}_L, \mathcal{Y})^T & 0 \end{bmatrix}. \tag{3.29}$$

Concerning the least-squares regression approach, the minimum ℓ_2-norm model is build as long as the interpolation system matrix is underdetermined. If more than $\frac{1}{2}(n+1)(n+2)$ points are avilable, the mentioned Matlab-function `pinv()` is used. A maximum of $(n+1)(n+2)$ interpolation conditions is used to build a regression model as it seems not to be desirable [40] to use all available points. This also seems logic as too many far points would then be considered which are not necessarily relevant to build up a local model around the current iterate.

3.3.6 The projected gradient as a stopping criterion

As indicated above, the model gradient $\nabla_x m_k(x)$ is used to check convergence to a first-order critical point, in the sense that we verify the inequality

$$\|P_{\mathcal{F}}(x_k - \nabla m_k(x_k)) - x_k\|_\infty \leq \epsilon, \tag{3.30}$$

which [72] have shown to correspond to a suitable measure of backward error for bound-constrained problems. Moreover, we have that

$$\begin{aligned} &\|P_{\mathcal{F}}(x_k - \nabla f(x_k)) - x_k\|_\infty \\ &\qquad \leq \|P_{\mathcal{F}}(x_k - \nabla m_k(x_k)) - x_k\|_\infty + \|\nabla m_k(x_k) - \nabla f(x_k)\|_\infty \\ &\qquad \leq \|P_{\mathcal{F}}(x_k - \nabla m_k(x_k)) - x_k\|_\infty + \|\nabla m_k(x_k) - \nabla f(x_k)\|_2, \end{aligned} \tag{3.31}$$

and, using (2.20), we deduce that the left-hand side of this inequality can be made small if (3.30) holds and Λ and Δ_k are sufficiently small. In practice, we require the interpolation points $y_i, i = 1, ..., p$ used to build $m_k(x)$ to be contained in the ball $\mathcal{B}_2(x_k, \epsilon)$ and $\mathcal{Y}_k$ is poised enough to ensure $\kappa_{eg}\Lambda\Delta_k \leq \epsilon$ for some user-defined constant $\kappa_{eg} > 0$.

Here, we give a proof of the inequality (3.31) to justify the use of the projected model gradient as a stopping criterion in our algorithm.

Lemma 3.2. *Suppose that A1 and A3 hold. Then*

$$\|P_{\mathcal{F}}(x_k - \nabla_x f(x_k)) - x_k\|_\infty \leq \|\pi_k\|_\infty + \|\nabla_x f(x_k) - g_k\|_2. \tag{3.32}$$

Proof. The feasible region $\mathcal{F}$ can be written as

$$\mathcal{F} = \mathcal{F}_1 \times \ldots \times \mathcal{F}_n \tag{3.33}$$

where $\mathcal{F}_i = [l_i, u_i]$ is the interval of the i-th variable.

We also use the fact that a projection on a box means to project in each dimension. We have by definition that $\forall v$

$$P_{\mathcal{F}}(v) = [P_{\mathcal{F}_i}(v_i)]_{i=1,\ldots,n}. \tag{3.34}$$

Since a projection on a convex set is Lipschitz-continuous in each single direction, we have that

$$|P_{\mathcal{F}_i}(v_i + w_i) - P_{\mathcal{F}_i}(v_i)| \leq |v_i + w_i - v_i|.$$

Using (3.34) and setting $z = P_{\mathcal{F}}(v + w) - P_{\mathcal{F}}(v)$, we get

$$|z_i| \leq |w_i|.$$

As this expression is true for all i, so also for

$$\max |z_i| \leq \max |w_i|$$

what in turn gives

$$\|z\|_\infty \leq \|w\|_\infty.$$

After replacing z again and including $+x - x$ in the left-hand side term we have

$$\|P_{\mathcal{F}}(v + w) - x - [P_{\mathcal{F}}(v) - x]\|_\infty \leq \|w\|_\infty.$$

Now, we substitute $v = x_k - g_k$ and $w = \nabla_x f(x_k) - g_k$ and we get

$$\|P_{\mathcal{F}}(x_k - \nabla_x f(x_k)) - x_k - [P_{\mathcal{F}}(x_k - g_k) - x_k]\|_\infty \leq \|\nabla_x f(x_k) - g_k\|_\infty.$$

Applying the triangle inequality gives

$$\|P_{\mathcal{F}}(x_k - \nabla_x f(x_k)) - x_k\|_\infty - \|\pi_k\|_\infty \leq \|\nabla_x f(x_k) - g_k\|_\infty,$$

which yields

$$\|P_{\mathcal{F}}(x_k - \nabla_x f(x_k)) - x_k\|_\infty \leq \|\pi_k\|_\infty + \|\nabla_x f(x_k) - g_k\|_\infty$$

and therefore also gives

$$\|P_{\mathcal{F}}(x_k - \nabla_x f(x_k)) - x_k\|_\infty \leq \|\pi_k\|_\infty + \|\nabla_x f(x_k) - g_k\|_2,$$

what concludes the proof. □

3.3.7 An alternative stopping criterion

To declare convergence to a critical point, we decided to ask for a sufficiently small model gradient where the model was built from a well-poised set of points in the ball $\mathcal{B}(x_k, \Delta_k)$. To reach this goal, it is sometimes necessary to decrease Δ_k to a level where the (at most quadratic but sometimes only linear) model approximates a possibly highly nonlinear function sufficiently well (see also the criticality step in Step 4 of Algorithm 3.1). But in practice, this strategy is not always successful in the context of derivative-free optimization, in the sense that in some cases, a radius Δ_k tending to zero does not yield a model gradient of required size ϵ. This difficulty can be explained by the approximative nature of the model gradient which could also be seen as a gradient computed by a kind of finite differences with $h = \Delta_k$. To show the

size of Δ	$\|\nabla f(x^*)\|_\infty$	$\|\nabla m(x^*)\|_\infty$	$\|\nabla f(x^*) - \nabla m(x^*)\|_2$
1.000000e-03	5.074018e-08	5.074108e-08	1.371287e-10
1.000000e-04	5.074018e-08	5.074800e-08	3.609084e-10
1.000000e-05	5.074018e-08	5.081872e-08	3.245846e-10
1.000000e-06	5.074018e-08	5.235049e-08	2.117399e-09
1.000000e-07	5.074018e-08	5.651729e-08	1.609209e-08
1.000000e-08	5.074018e-08	2.161465e-07	2.334213e-07
1.000000e-09	5.074018e-08	1.052973e-06	1.493022e-06
1.000000e-10	5.074018e-08	1.806191e-04	2.455110e-04
1.000000e-11	5.074018e-08	1.385372e-03	1.803735e-03
1.000000e-12	5.074018e-08	4.677834e-03	8.378769e-03

Table 3.1: Gradient accuracy of ill-conditioned problem PALMER3C

behaviour of an interpolation model gradient for decreasing radii Δ, we did some experiments on different test problems. Here, a complete quadratic model using $p = \frac{1}{2}(n+1)(n+2)$ points was computed by sampling points at the vertices of a simplex with sides along the coordinate axes plus their mid-points around the approximate solution x^*. We examined an ill-conditioned problem (PALMER3C, for which $\text{cond}(\nabla^2 f(x^*)) = 2.689 \cdot 10^{11}$) and also a well-conditioned problem (ALLINITU, for which $\text{cond}(\nabla^2 f(x^*)) = 2.524$) from the CUTEr testing environment. The results are depicted in Table 3.1 and Table 3.2 and we can see that the gradient accuracy

size of Δ	$\|\nabla f(x^*)\|_\infty$	$\|\nabla m(x^*)\|_\infty$	$\|\nabla f(x^*) - \nabla m(x^*)\|_2$
1.000000e-03	1.688089e-07	2.650513e-06	2.723041e-06
1.000000e-04	1.688089e-07	1.688116e-07	2.722211e-08
1.000000e-05	1.688089e-07	1.687983e-07	2.878566e-10
1.000000e-06	1.688089e-07	1.683098e-07	1.457556e-09
1.000000e-07	1.688089e-07	1.731948e-07	1.261984e-08
1.000000e-08	1.688089e-07	7.993606e-07	1.187783e-06
1.000000e-09	1.688089e-07	4.884981e-06	6.979543e-06
1.000000e-10	1.688089e-07	1.776357e-05	2.792555e-05
1.000000e-11	1.688089e-07	2.220446e-04	2.663875e-04
1.000000e-12	1.688089e-07	1.776199e-03	2.664249e-03

Table 3.2: Gradient accuracy of well-conditioned problem ALLINITU

strongly depends on the size of the radius Δ where for very small Δ the size of the model gradient is increasing and getting more inaccurate with respect to the true gradient $\nabla f(x^*)$.

We therefore decided to terminate a run when the radius Δ_k becomes smaller than a threshold $\Delta_{\min}$ where $\Delta_{\min} \approx \sqrt{\epsilon_{\text{machine}}}$ is a reasonable choice as is used when working with finite

differences (see for instance [49, 132]). We do not declare convergence in this case but print out a message that we suspect convergence but that the trust region is too small to continue the minimization process. In our numerical experiments, we observed that in all cases where the run was terminated due to a too small trust region, a critical point was found by the algorithm.

3.4 Solving the bound-constrained trust-region subproblem in ℓ_2-norm

In this section, we want to introduce another contribution in this thesis which is to propose an algorithm to solve the bound-constrained trust-region subproblem in ℓ_2-norm based on the Moré-Sorensen algorithm.

Given the vector $g \in \mathbb{R}^n$, the symmetric matrix $H \in \mathbb{R}^{n\times n}$ and the scalar Δ, we solve the following problem

$$\begin{aligned} \min_{s\in\mathbb{R}^n} m(x+s) = q(s) = g^T s + \tfrac{1}{2} s^T H s \\ \text{subject to } \|s\|_2 \leq \Delta, \\ \text{and subject to the bounds } l \leq x+s \leq u. \end{aligned} \tag{3.35}$$

In most trust-region algorithms which consider bound constraints, the infinity-norm trust region is used because it fits the geometry of the domain of the problem (see e.g. [32, 60]). The reason is that the intersection of the feasible box and a box-shaped trust region is again a box and thus easier to handle than the intersection of a box and a ball which is not such a simple set. Nevertheless, Euclidean norm trust-regions have also been considered in bound-constrained trust-region methods (see e.g. [8, 48]). Whereas in [48] a conjugate-gradient method is used to solve the QP problem, in [8], the trust-region subproblem is solved by applying a Moré-Sorensen-type algorithm which was particularly developed to find the global solution to a QP problem in an ℓ_2-norm constraint (see Section 3.4.1 below). As one can imagine, a global solution can not be ensured anymore when considering bound constraints as the problem gets NP-hard due to the consideration of the bounds. But contrary to [8], where simple truncation is used to stay feasible, in this thesis, we want to go a step further and try to find a "more global" solution to problem (3.35). For this purpose, we developed an iterative framework which relies on the successive application of the Moré-Sorensen algorithm. In our extension, the bounds are handled by applying an active-set strategy where at each iteration at least one bound is added to the set of active bounds until convergence inside the inactive bounds is declared or the current iterate is a vertex. In the next section, we recall the technique used in the Moré-Sorensen method before we describe our extension of the method to handle bound constraints.

3.4.1 The Moré-Sorensen algorithm

Pioneered and further developed by Hebden [82], Gay [62], Sorensen [131], Moré and Sorensen [101] and reassessed by Dollar, Gould and Robinson [70], such a method obtains the solution of the trust-region subproblem in ℓ_2-norm by factorization of a sequence of parametrized linear systems. When solving the locally constrained problem

$$\begin{aligned} \min_{s\in\mathbb{R}^n} q(s) = g^T s + \tfrac{1}{2} s^T H s \\ \text{subject to } \|s\|_2 \leq \Delta, \end{aligned} \tag{3.36}$$

such a method seeks to find the model minimizer s^M which is the point that makes the model as small as possible in $\mathcal{B}$, or a close approximation to it. The solution to (3.36) lies either interior to the trust region, that is $\|s^M\|_2 < \Delta$, or on the boundary, $\|s^M\|_2 = \Delta$. This suggests to solve the trust-region subproblem to find the unconstrained minimizer of the model. In the case, the model is unbounded below or the model minimizer lies outside the trust-region, the model minimizer must occur on the boundary. It can then be found as the global minimizer of $q(s)$ subject to the equality constraint $\|s\|_2 = \Delta$. We know (e.g. [67], [34, Corollary 7.2.2]) that any global minimizer of $q(s)$ subject to $\|s\|_2 \leq \Delta$ satisfies the equation

$$(H + \lambda^M I)s^M = -g, \tag{3.37}$$

where $H + \lambda^M I$ is positive semidefinite, $\lambda^M \geq 0$, and $\lambda^M(\|s^M\|_2 - \Delta) = 0$. If $H + \lambda I$ is positive definite then s is the only solution to (3.36). Moreover, if H is positive definite and $\|H^{-1}g\| < \Delta$ then (3.36) has a solution with $\|s\| < \Delta$, in the interior. We now assume that (3.36) has a solution on the boundary. In most cases, the nonlinear equation $\|s_\lambda\| = \Delta$ where

$$s_\lambda = -(H + \lambda I)^{-1} g \tag{3.38}$$

has a solution $\lambda \geq 0$ in $(-\lambda_1, \infty)$ and s_λ is the solution of problem (3.36). However, in the "hard case", $\|s_\lambda\| = \Delta$ has no solutions in $(-\lambda_1, \infty)$ what may happen if g is orthogonal to the space of eigenvectors corresponding to λ_1, the most negative eigenvalue of H and especially in the case where $g = 0$. The difficulty in the hard case is that $\|s_\lambda\| < \Delta$ whenever $H + \lambda I$ is positive definite with $\lambda > 0$. In this case, a solution to (3.36) can be obtained by solving

$$(H - \lambda_1 I)s = -g \tag{3.39}$$

for s with $\|s\| \leq \Delta$ and by determining an eigenvector z corresponding to the eigenvalue λ_1. Then

$$(H - \lambda_1 I)(s + \tau z) = -g \tag{3.40}$$

and thus

$$s = -(H - \lambda_1 I)^\dagger g + \tau z, \tag{3.41}$$

where the superscript † denotes the Moore-Penrose generalized inverse and τ is chosen so that $\|s\| = \Delta$ in (3.41).

Turning back to the case where H is not singular, the question is how to find the root of $\psi(\lambda) = \|s_\lambda\|_2 - \Delta = 0$. In fact, Reinsch [125, 126] and Hebden [82] observed that great advantage could be taken of the fact that the function $\|s_\lambda\|^2$ is a rational function in λ with second order poles on a subset of the negatives of the eigenvalues of H. This means that the function $1/\psi(\lambda)$ has zeros but no (finite) poles and the zeros occur at the negatives of the eigenvalues of H. The better behaviour of $1/\psi(\lambda)$ may be exploited by applying Newton's method to the zero finding problem

$$\phi(\lambda) \stackrel{\text{def}}{=} \frac{1}{\|s_\lambda\|_2} - \frac{1}{\Delta} = 0, \tag{3.42}$$

where (3.42) is also called the secular equation. Newton's method should be very efficient when applied to (3.42) as the curve of $\phi(\lambda)$ is nearly linear in the region of interest, that is $\lambda > \lambda_1$.

Algorithm 3.4 The l_2-norm trust-region Moré-Sorensen algorithm

Initialization: Let $\lambda^L, \lambda^U, \lambda_S$ and $\Delta > 0$ be given.

Step 1: Safeguard λ by setting $\lambda := \min(\max(\lambda, \lambda^L), \lambda^U)$.
If $\lambda \leq \lambda_S$, then $\lambda := \max(\tau_\lambda \lambda^U, \sqrt{\lambda^L \lambda^U})$.
Step 2: Factor $B + \lambda I = R^T R$. If factorization not successful, go to Step 6.

Step 3: Solve $R^T R s = -g$. If ($\lambda \leq \epsilon$ and $\|s\|_2 < \Delta$) or $\|s\|_2 \approx \Delta$, RETURN.

Step 4: Solve $R^T w = s$.

Step 5: Let $\lambda := \lambda + \left(\frac{\|s\|}{\|w\|}\right)^2 \left(\frac{\|s\| - \Delta}{\Delta}\right)$.

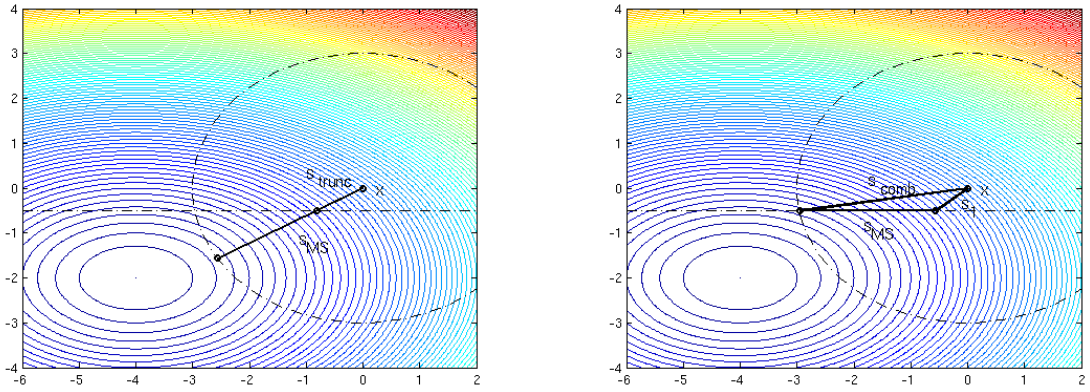
Step 6: Update the uncertainty interval: If $\lambda \in (-\lambda_1, \infty)$ and $\phi(\lambda) < 0$ then set $\Lambda^U := \min(\lambda^U, \lambda)$, otherwise set $\Lambda^L := \max(\lambda^L, \lambda)$. Set $\lambda_S : \max(\lambda_S, \lambda - \|R\hat{z}\|^2)$. Reset $\lambda^L := \max(\lambda^L, \lambda_S)$. Go to Step 1.

Algorithm 3.4 on page 53 shows how to apply Newton's method to (3.42) to update λ where we mainly use the gathered ideas from [101]. But as Newton's method alone does not offer the guarantee of convergence, the method must be safeguarded by finding appropriate lower and upper bounds on λ and ensuring that they coincide in the worst case. The safeguarding depends on the fact that $\psi(\lambda)$ is convex and strictly decreasing on $(-\lambda_1, \infty)$. This implies that $\phi(\lambda)$ is concave and that Newton's method, started from $\lambda \in (-\lambda_1, \infty)$ with $\phi(\lambda) > 0$, produces a monotonically increasing sequence converging to the solution of $\phi(\lambda) = 0$. In order to safeguard the Newton iteration, an interval of uncertainty $[\lambda^L, \lambda^U]$ is constructed in which the solution λ^M is known to occur and in which the current estimate λ is forced to lie. Furthermore, λ_S, a lower bound on $-\lambda_1$ is needed. How to compute initial bounds λ^L, λ^U and λ_S, and many other details, can be found, for instance, in [34], [62] and [101].

3.4.2 The extension of the Moré-Sorensen algorithm to handle bound-constraints

The outline of the new bound-constrained algorithm is given as Algorithm 3.5 on page 55. In simple terms, this iterative algorithm computes a step s_{MS} inside the Euclidean-norm trust region and checks if this step is violating any bounds (Step 2). If so, a step s_λ with $\|s_\lambda\| < \|s_{MS}\|$ and inside the bounds, but activating one or more of the violated bounds, is found (Step 3). The corresponding step component(s) are fixed and the search space is reduced to the remaining free variables (Step 4) as in a typical active-set method. Then, a new step s_{MS} is computed in the reduced space until convergence is declared in Step 2 or there are no free variables left.

The simple example in Figure 3.1 illustrates the fact that with our active-set strategy, a better point may be found than with just truncating the computed step s_{MS} at the bound (as is done in [8]). We see on the left-handside of Figure 3.1, the Moré-Sorensen algorithm finds the minimum at the trust region with the step $s_{MS} = [-2.5615, -1.5615]$ where the step provides a model decrease of $q(s) = -5.3866$. But this step lies clearly outside of the feasible region and is not a solution to (3.35). By truncating the step at the bound, the step



(a) MS-step and truncated step

(b) Combination of two MS-steps

Figure 3.1: Minimization of a quadratic model inside the ℓ_2-norm constraint and the bounds

$s_{\text{trunc}} = [-0.82, -0.5]$ would be returned by the solver and the corresponding model decrease would give $q(s) = -0.3469$ whereas on the right-handside of Figure 3.1, we see that the active-set technique which combines multiple step-computations returns the combined step $s_{\text{comb}} = s_2 = s_1 + s_{MS} = [-2.9580, -0.5]$ (where s_{MS} is computed in the one-dimensional subspace in the second iteration of Algorithm 3.5) as a solution which provides a model decrease of $q(s) = -4.6036$. And assumed, the model is a good approximation of the function, this technique provides a better trial point which gives rise for a larger decrease in the objective function f.

Algorithm 3.5 BC-MS: Bound-constrained l_2-norm trust-region algorithm

Step 1: Initialization.

The required accuracy for a boundary solution $\epsilon_b \geq 0$, the trust-region radius Δ_0, $g_0 = \nabla m(x)$ and $H = \nabla^2 m(x)$ are given. Define the initial set of active bounds $\mathcal{I}_{act}$ as:

$$\mathcal{I}_{act} = \{i | (u(i) - x(i) = 0 \text{ and } g_0(i) < 0) \text{ or } (x(i) - l(i) = 0 \text{ and } g_0(i) > 0)\},\ i = 1, ..., n.$$

Define the set of free indices $\{i_{free}\} = i \notin \mathcal{I}_{act}$. Set $n_{free} = |\{i_{free}\}|$, the number of remaining free variables. Reduce dimension $g_{\text{red}} = g_0(i_{free}), H_{\text{red}} = H(i_{free}, i_{free})$. Initialize $k = 0$ and $s_0 = 0$.

while $n_{free} > 0$ **do**

Step 2: Compute a Moré-Sorensen step $\mathbf{s_{MS}}$ in $\mathbf{\Delta_k}$.

Step 2.1: Apply Algorithm 3.4 to solve the constrained problem

$$\min\{\psi(s_{MS}) : \|s_{MS}\|_2 \leq \Delta_k\},$$

where $\psi(s_{MS}) = g_{\text{red}}^T s_{MS} + \frac{1}{2} s_{MS}^T H_{\text{red}} s_{MS}$ represents a local model of the objective function.

Step 2.2: If $l \leq x + s_k + s_{MS} \leq u$, then set $s_{k+1} = s_k + s_{MS}$, increment k by one and go to Step 5.

Step 3: Compute a step $\mathbf{s_\lambda}$ (with $\mathbf{s_\lambda < s_{MS}}$) in $\mathbf{\Delta_k}$ and the bounds.

Step 3.0: Let $\lambda \geq 0$ be given from computing the MS-step in Step 2.1. Set $\lambda^L = \lambda$ and compute an appropriate upper bound λ^U on λ. Determine the sets of violated upper and lower bounds $\mathcal{V}_U = \{i |\ u(i) < x(i) + s_k(i) + s_{MS}(i)\}$ and $\mathcal{V}_L = \{i |\ x(i) + s_k(i) + s_{MS}(i) < l(i)\}$.

Step 3.1: If $H_{\text{red}} + \lambda I$ not positive definite, then update $\lambda = \frac{1}{2}(\lambda^L + \lambda^U)$ and go to Step 3.9.

Step 3.2: Factorize $H_{\text{red}} + \lambda I = R^T R$.

Step 3.3: Compute the new step s_λ by solving the system $R^T R s_\lambda = -g_{\text{red}}$.

Step 3.4: Define the set of new active bounds:

$$\mathcal{I}_{act}^+ = \{i |\ (u(i) - \epsilon_b \leq x(i) + s_k + s_\lambda(i) \leq u(i),\ i \in \mathcal{V}_U) \text{ or } (l(i) \leq x(i) + s_k + s_\lambda(i) \leq l(i) + \epsilon_b,\ i \in \mathcal{V}_L)\}.$$

If $\mathcal{I}_{act}^+ \neq \emptyset$, then go to Step 4.

Step 3.5: Newton's iteration on the secular equation: compute w by solving $R^T w = s_\lambda$.

Step 3.6: Choose an appropriate $\bar{i}$ from the set $\{\mathcal{V}_U \cup \mathcal{V}_L\}$. If $\bar{i} \in \mathcal{V}_U$, set $\Delta_b(\bar{i}) = |u(\bar{i}) - x(\bar{i}) - s_k(\bar{i})|$, else set $\Delta_b(\bar{i}) = |l(\bar{i}) + x(\bar{i}) + s_k(\bar{i})|$.

Step 3.7: Update $\lambda := \lambda + \left(\frac{s_\lambda(\bar{i})}{\|w\|}\right)^2 \left(\frac{s_\lambda(\bar{i}) - \Delta_b(\bar{i})}{\Delta_b(\bar{i})}\right)$.

Step 3.8: If $l \leq x + s_k + s_\lambda \leq u$, then $\lambda^U = \lambda$, else $\lambda^L = \lambda$.

Step 3.9: Check value of λ and safeguard if necessary. Go to Step 3.1.

Step 4: Update values.

Set $\{i_{free}\} = \{i_{free}\} \setminus \{\mathcal{I}_{act}^+\}, n_{free} = |\{i_{free}\}|, g_{k+1} = g_k + H s_\lambda$. Update $g_{\text{red}} = g_k(i_{free}), H_{\text{red}} = H(i_{free}, i_{free})$. Set $s_{k+1} = s_k + s_\lambda$ and $\Delta_{k+1} = \Delta_0 - \|s_{k+1}\|$. Increment k by one.

end while

Step 5: Establish new iterate.

Compute $x^+ = x + s_k$. RETURN.

The question is now which strategy should be applied to find the step s_λ with $\|s_\lambda\| < \|s_{MS}\|$ which provides a point $x + s_k + s_\lambda$ at the boundary. Projecting the step s_{MS} onto the feasible set or truncating it at the boundary could be suitable options. But we decided for a somewhat more sophisticated approach which finds a presumably better feasible step than projection and truncation. To do so, we compute in Step 3 of Algorithm 3.5 the step $s_\lambda = (H + \lambda I)^{-1} g$ by applying successive trial values of λ until the set $\mathcal{I}_{act}^+$ is non-empty (see Step 3.4). Then s_λ is inside the feasible region and activates at least one of the before violated bounds from the set $\{\mathcal{V}_U \cup \mathcal{V}_L\}$.

We will now describe some steps of Algorithm 3.5 a bit more in detail. In Step 3.0, the appropriate upper bound λ^U on λ is computed by the formula

$$\lambda^U = \frac{\|g\|}{\Delta_{vb}} + \|H\|_1 \tag{3.43}$$

which has been proposed in [62] and [101] to compute the upper bound on λ for computing the standard Moré-Sorensen step. The difference to the standard case is that we do not consider the trust-region radius Δ but the distance Δ_{vb} from the point $x + s_k$ to the closest violated bound. We chose to use the closest one to obtain a safe upper bound.

To efficiently update λ at each iteration, we have to choose an appropriate bound from the set of violated bounds $\{\mathcal{V}_U \cup \mathcal{V}_L\}$ in Step 3.6. There, we have to distinct two cases where the first one applies when the point $x + s_k + s_\lambda$ is outside of the feasible box and λ has to be increased to obtain a shorter step s_λ. The second case applies when the point $x + s_k + s_\lambda$ lies inside the bounds but not inside the ϵ_b-environment of the bound. In this case, λ has to be decreased to get a longer step s_λ. This reminds the Newton's iteration on the secular equation described above but instead of applying the trust-region radius Δ as a constraint, we use Δ_b, the distance between a bound (with index $\bar{i}$ from the index set $\{\mathcal{V}_U \cup \mathcal{V}_L\}$) and the corresponding component of the point $x + s_k$. When $x + s_k + s_\lambda$ lies outside of the bound, we choose the index $\bar{i}$ which corresponds to the largest violation of a bound of one of the step components of $x + s_k + s_\lambda$. Taking the step component with the largest violation guarantees that one of the next steps s_λ will be inside the bounds. In the second case, where the step is inside the feasible box and must be increased, the step component $\bar{i}$ closest to its bound from the set $\{\mathcal{V}_U \cup \mathcal{V}_L\}$ is found.

Then, λ is updated in Step 3.7 using the formula

$$\lambda := \lambda + \left(\frac{s_\lambda(\bar{i})}{\|w\|}\right)^2 \left(\frac{s_\lambda(\bar{i}) - \Delta_b(\bar{i})}{\Delta_b(\bar{i})}\right),$$

where we apply a modified version of the updating formula from Step 5 of the Moré-Sorensen algorithm (Algorithm 3.4). Here, $s_\lambda(\bar{i})$ is the step component which was chosen (in Step 3.6) to be the one which is supposed to activate its bound and thus has to equal the distance $\Delta_b(\bar{i})$ to its formerly violated bound $l(\bar{i})$ or $u(\bar{i})$.

Note that if Algorithm 3.5 is applied to unconstrained optimization problems, it reduces to a standard Moré-Sorensen algorithm. Algorithm BC-MS is implemented as an option to be used

as local solver in our trust-region algorithm ***BCDFO+***. Numerical experiments comparing BC-MS to the standard option of a truncated conjugate gradient algorithm will be presented in the next Section.

3.5 Numerical experiments in the CUTEr testing environment

The described algorithm ***BCDFO+*** has been implemented in Matlab and all numerical experiments reported below were run on a single processor workstation. As the time to compute the objective function values in derivative-free optimization typically dominates other costs of the algorithm, our results will be presented in terms of number of function evaluations.

In what follows, ***BCDFO+*** is compared to its predecessor BC-DFO [74] developed by S. Gratton, Ph.L. Toint and A. Tröltzsch, to the packages NEWUOA [123] and BOBYQA [124] developed by M.J.D. Powell, SID-PSM [46, 45] developed by A.L. Custódio and L.N. Vicente, NOMADm [4] by M.A. Abramson, BFO [113] developed by Ph.L. Toint and M. Porcelli. Powell's codes are trust-region algorithms using a quadratic model where the remaining degrees of freedom in the interpolation, when using less than $\frac{1}{2}(n+1)(n+2)$ points, are determined by minimizing the change to the Hessian of the model between two consecutive iterations. We use BOBYQA for the comparison on bound-constrained problems and NEWUOA for the comparison on unconstrained problems because it outperforms BOBYQA from our experience in solving unconstrained problems. SID-PSM is a pattern-search method with the poll step guided by simplex derivatives. The search step relies on the optimization of quadratic Minimum Frobenius-norm interpolation models. The package was mainly developed and tuned for unconstrained optimization but is also able to handle bounds and general constraints. SID-PSM has shown to be [127] one of the most competitive algorithms in the class of pattern search methods. NOMADm is a Matlab implementation of the commercial software NOMAD. The considered method is a mesh adaptive direct-search method which is a generalization of the class of pattern search methods. BFO (which stands for Brute-Force Optimizer) is another implementation of a mesh adaptive direct-search method based on elementary refining grid search. It is able to handle bound-constraints and problems containing continuous and/or discrete variables.

3.5.1 Default parameters

In ***BCDFO+*** as well as in BC-DFO, we fixed the trust-region parameters to $\eta_1 = 0.0001, \gamma_1 = 0.01, \gamma_2 = 0.5$ and $\gamma_4 = 0.1$. The initial trust-region radius Δ_0 is set to 1, as suggested in Section 17.2 of [34]. We apply a maximum trust-region radius of $\Delta_{\max} = 10000$ and a minimum trust-region radius of $\Delta_{\min} = 10^{-10}$. The parameter for switching to the non-decreasing trust-region strategy when replacing points is set to $\Delta_{\text{switch}} = 10^{-7}$. To build a sufficiently well-poised set in the modified greedy algorithm, we set the threshold $\kappa_{th} = 0.005$. After appending

a point to an incomplete interpolation set, we check the condition of the shifted and scaled system matrix $\hat{M}$ to be smaller than $\kappa_{\text{illcond}} = 10^{15}$. To divide the interpolation set into far and close points when incorporating the new trial point, we set $\beta = 1$. When replacing a close interpolation point, we use the parameter $\Lambda_C = 1.2$ to ensure an improvement of the interpolation set geometry. For declaring convergence, the desired accuracy on the projected model gradient norm and the tolerated error on the gradient is set to $\epsilon = 10^{-5}$ while parameter κ_{eg} is set to 0.1. We set $\epsilon_0 = \epsilon$ and $\mu = 0$ to skip the loop in the criticality step. The only difference in parametrisation of the two algorithms is that ***BCDFO+*** uses $\gamma_3 = 2$ and in BC-DFO the trust-region parameter is fixed to $\gamma_3 = 1.5$. The same parameters of BC-DFO were used to obtain the results published in [74].

We always use the default parameters for the codes NEWUOA, BOBYQA, SID-PSM, NOMADm and BFO. We run BOBYQA with a number of $m = 2n + 1$ interpolation points using the Frobenius norm approach and NEWUOA with a full quadratic model, as these two options give the best results for these solvers, out of the choice $m \in \{n + 1, 2n + 1, \frac{1}{2}(n + 1)(n + 2)\}$.

3.5.2 Test problems

The CUTEr testing environment [69] is used in our experiments. To compare ***BCDFO+*** to the other software packages on unconstrained problems, we chose to use the test problems from the CUTEr test collection which were selected in [58]. Two problems[1] were excluded from the test set because they contain fixed variables and NEWUOA does not provide facilities to handle such cases and one listed problem[2] contains bounds. After running all problems in this test set, three problems[3] were removed because the solvers converged to different solutions, making a comparison meaningless. A total of 54 unconstrained problems were thus considered. The distribution of the dimension n among the 54 problems is shown in Table 3.3. We also want to

n	2	3	4	5	6	8	10	11	15
nbr of problems	17	7	5	1	2	3	13	1	5

Table 3.3: Dimensions of considered unconstrained problems

give some information about the structure of the true problem Hessians in the considered test set (see Table 3.4). As interpolation-based method attempt to approximate this information, it might be helpful to conclude why some methods or options perform better than others on specific problems.

Considering bound-constrained problems, we took all bound-constrained problems provided by the CUTEr collection with a size of at most 30 variables. Problems with more variables can be successfully solved with the considered algorithm. However, as in [58, 45], we restricted this

[1] BIGGS3, BOX2
[2] CHEBYQAD
[3] ENGVAL2, HATFLDD, HATFLDE

Hessian structure	problem dimension	nbr. of problems
sparse	2, 10, 15	6
diagonal	2, 3, 10	4
arrow-head	15	1
tridiagonal	3, 5, 10	6
5-diagonal	10	1
dense	2, 3	21
dense	4, 6, 8, 10, 11	16

Table 3.4: Hessian structure of considered unconstrained problems

problem size to make the performance comparison on a large number of test cases possible with our computing facilities.

We could not consider problems containing fixed variables because not all solvers do provide the required facilities. Furthermore, in order to avoid too many problems of the same kind, we chose randomly four of the 26 bound-constrained PALMER problems provided in CUTEr. After running all solvers considered in this comparison on these 53 remaining problems, thirteen of them[4] had to be excluded from our comparison because the considered algorithms didn't converge to the same solution.

This is to explain with the existence of multiple minima, but is also due to the fact that many solvers usually only check first-order optimality. Also BC-DFO and ***BCDFO+*** sometimes check convergence in the full-space without taking second order information into account (after having converged inside a subspace). This creates the possibility to declare convergence at a saddle point which is a minimum in the explored subspace. Such a situation can be circumvented (at some cost) by requiring that a full quadratic model is built before declaring termination (which is an option in our implementation).

The final test set of bound-constrained problems contains 40 problems. The distribution of the dimension n among these problems is shown in Table 3.5.

n	1	2	3	4	5	6	8	9	10	12	18	19	25	30
nbr of problems	1	10	2	7	1	2	1	1	8	1	1	1	3	1

Table 3.5: Dimensions of considered bound-constrained problems

The detailed list of all considered bound- and unconstrained problems and their characteristics is provided in Table A.2 and Table A.1 in the Appendix of this thesis.

[4]EG1, EXPLIN, EXPLIN2, EXPQUAD, HART6, HS2, KOEBHELB, MAXLIKA, PALMER3E, PROBPENL, S368, SINEALI, WEEDS

3.5.3 A common stopping criterion

As BOBYQA, NEWUOA, SID-PSM, NOMADm and BFO use different stopping criteria from the one of BC-DFO and ***BCDFO+***, an independent criterion needs to be applied for the comparison. For this reason, we use the optimal objective function value computed by the TRON package [96] (using first and second derivatives) as a reference for our bound-constrained experiments. In the experiments with unconstrained problems we take the optimal objective function value computed by the KNITRO package [23] used in the paper of Fasano, Morales and Nocedal [58]. We take the number of function evaluations needed until a prescribed number of correct significant figures in the objective value was attained.

To provide a fair comparison, we followed the testing framework proposed by Dolan, Moré, and Munson in [52]. In this framework, the solvers are run first with their own default stopping criterion. If, for a given problem, convergence of one of the solvers to the common stopping criterion can't be declared with this configuration, the stopping criterion for this solver is strenghtened and the run repeated using the more stringent criterion. For a few test problems, some solvers were run several times while decreasing its own stopping criterion after each run, trying to attain the commonly required accuracy in the objective function value. This procedure was successful for a subset of the problems, for others the limit of function evaluations (15000) was reached. No time limitation was set.

3.5.4 Performance profiles

We now report our results using performance profiles (see [51]). Given a test set $\mathcal{P}$ containing n_p problems and n_s solvers, these profiles provide a way to graphically present the comparison of quantities $t_{p,s}$ (such as required computing time or number of function evaluations to solve problem p by solver s) obtained for each problem and each solver. For this, the performance ratio for a problem p and a solver s is defined as

$$r_{p,s} := \frac{t_{p,s}}{\min\{t_{p,s} : 1 \leq s \leq n_s\}}. \tag{3.44}$$

If solver s for problem p leads to a failure $r_{p,s} := 2 \cdot \max\{t_{p,s} : 1 \leq s \leq n_s\}$ is defined. Then,

$$\rho_s(\tau) := \frac{1}{n_p}\text{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\} \tag{3.45}$$

is the fraction of the test problems which were solved by solver s within a factor $\tau \geq 1$ of the performance obtained by the best solver. The function ρ_s is the (cumulative) distribution function for the performance ratio. The performance plots present ρ_s for each solver s as a function of τ. In this work, a logarithmic scale is used for the τ-axis. That means,

$$\rho_s(\tau) := \frac{1}{n_p}\text{size}\{p \in \mathcal{P} : \log_2(r_{p,s}) \leq \tau\} \tag{3.46}$$

is plotted. In our work, the presented performance profiles compare the number of function evaluations needed by each solver to achieve the desired accuracy in the objective function value. We use four different levels of accuracy: 2, 4, 6 and 8 significant figures in $f(x^*)$.

3.5.5 Comparison of different model types

In this section, we compare different model options implemented in algorithm ***BCDFO+***, namely, sub-basis model (see Section 2.2.2), minimum ℓ_2-norm model (see Section 2.2.3), minimum Frobenius-norm model (see Section 2.2.4) and regression model (see Section 2.2.7). The experiments are performed using default parameter values and the standard truncated CG algorithm to solve the trust-region subproblems. Applying the algorithm BC-MS as local solver gave similar results. Regarding Figure 3.2, we can see a clear advantage in efficiency for the

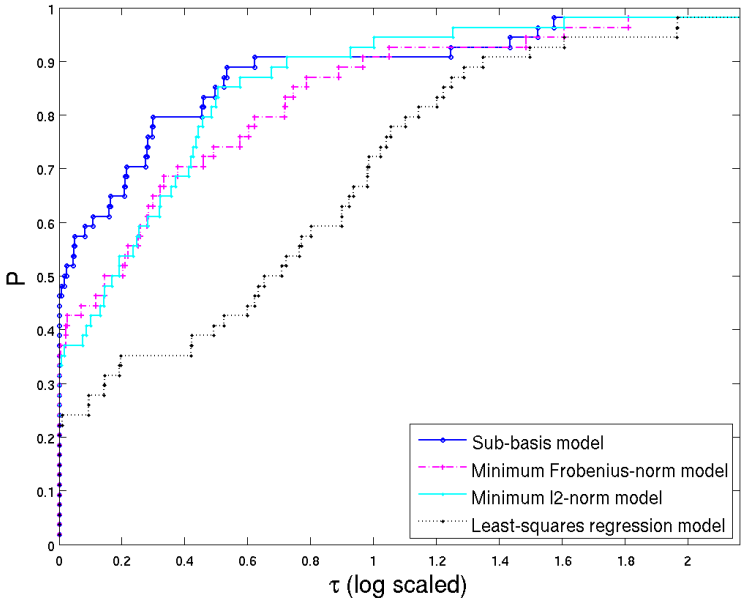


Figure 3.2: Comparison of different models in ***BCDFO+*** on unconstrained CUTEr problems

sub-basis model. Applying this option to the algorithm solves 46% of the test problems faster than the other model-options. The two minimum norm models perform equally well what means that it makes not a big difference whether using a minimum ℓ_2 norm or a minimum Frobenius-norm approach when the interpolation set is incomplete. In particular, the version with the minimum Frobenius-norm model solves 35% and the version with the minimum ℓ_2-norm model solves 33% of the test problems faster than the other versions. The use of more than $\frac{1}{2}(n+1)(n+2)$ points and building a least-squares regression model seems not an appropriate idea in this context. This version solves only 22% of the problems fastest. Regarding robustness, all model options perform equally whereas none of them was able to solve the problem PALMER1C to the required accuracy in the given budget of function evaluations.

Looking at these results, one could suppose that the test set might have been in favour of using the sub-basis model as it has an advantage when approximating a banded Hessian matrix. But in fact, this is not the case. We have stated the Hessian structure of the problems in the

test set in Section 3.5.2 above and see that there is a balance between sparse problems and dense problems with dimension higher than 3 (we separated the dense problems of size two and three, as the approximation technique shouldn't play an important role in these cases).

The reason for the regression model to not perform as well as the interpolation models may come from our choice of $p_{\max} = (n+1)(n+2)$, the maximum number of interpolation conditions considered to solve the overdetermined system. At the moment, no special care is taken in our implementation to ensure a certain "locality" of the model and we believe that there is space for improvement. In other implementations, the use of 80% close points and 20% of points further away from the current iterate has been proposed [45]. Another idea could be to adapt the number of considered points dependend on their distance to the current iterate in relation to the size of the current trust-region radius.

From these experiments, we conclude that it would be best to perform the comparisons to other software packages by using the sub-basis model option in our algorithm. The test results can be found in Table B.1 in the Appendix.

3.5.6 Comparison of local solver options

In this section, we want to compare the two local solvers implemented in our algorithm ***BCDFO+***, which are the new algorithm BC-MS, presented as Algorithm 3.5 in Section 3.4 and the standard truncated conjugate gradient algorithm described in Section 3.1.4. BC-MS operates in an ℓ_2-norm trust region whereas if TCG is applied, the outer trust-region algorithm is working with an infinity-norm trust region. Default parameters and the sub-basis model are used in

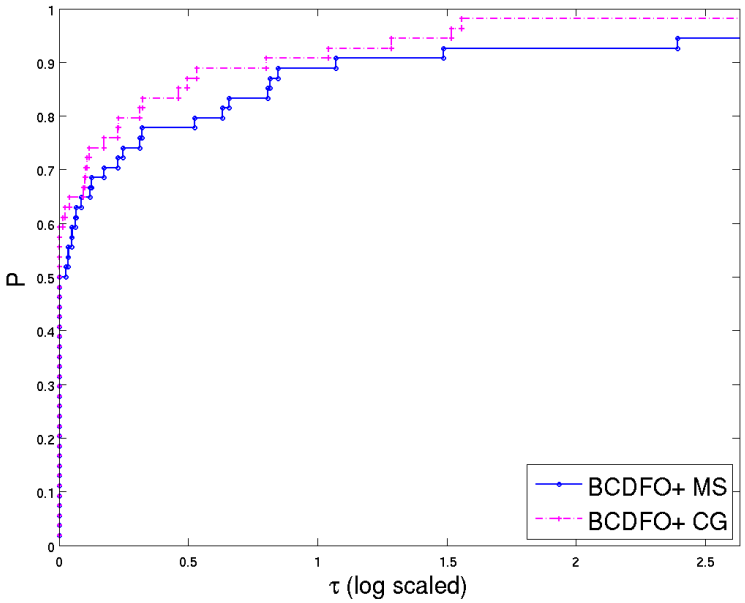


Figure 3.3: Comparison of local solvers in ***BCDFO+*** on unconstrained CUTEr problems

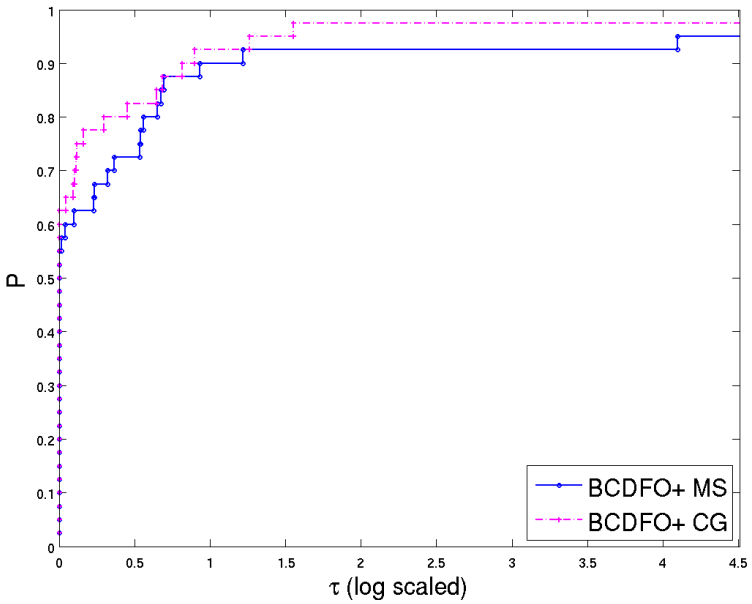


Figure 3.4: Comparison of local solvers in ***BCDFO+*** on bound-constrained CUTEr problems

this comparison. Regarding the results in Figure 3.3 and Figure 3.4, we can see that the CG option performs better than applying BC-MS. In the unconstrained case, the MS option solves exactly 50% of the test problems faster than CG but the CG option solves 59% of the problems faster. In terms of robustness, 95% of the problems are managed to be solved apllying the MS local solver and 98% if applying the CG option. The robustness is the same for bound-constrained problems, but in terms of number of function evaluations, the distance between the two solver options is slightly smaller. Applying BC-MS, 55% of the test set is solved faster against 62% when applying the conjugate gradient option. The results of this testing can be found in Table B.2 for the unconstrained and in Table B.3 for the bound-constrained test cases.

3.5.7 Unconstrained comparisons with NEWUOA

First, we want to report results obtained with BC-DFO, a bound-constrained implementation of Algorithm 2.3, the predecessor of ***BCDFO+***. We want to recall here that the difference of the two softwares is the different handling of the trust-region where BC-DFO is using a typical DFO trust-region management, shrinking the trust-region only when the quality of the model is assured, and ***BCDFO+*** uses a more standard one which is shrinking the trust-region at every unsuccessful iteration until a threshold $\Delta_{\min}$ is reached. Details about the particular implementation and also the here stated numerical results of BC-DFO can be found in [74].

Let's now turn to the numerical results obtained by comparing the solver BC-DFO to NEWUOA on our set of 53 unconstrained test problems from the CUTEr library. Figures 3.5-3.8 show that BC-DFO appears to be more robust but less efficient than NEWUOA,

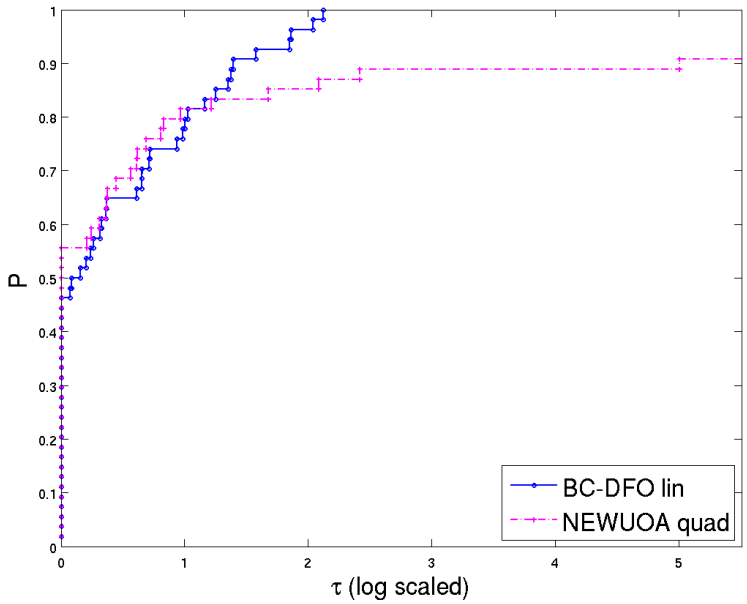


Figure 3.5: Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (2 digits of accuracy required in final function value)

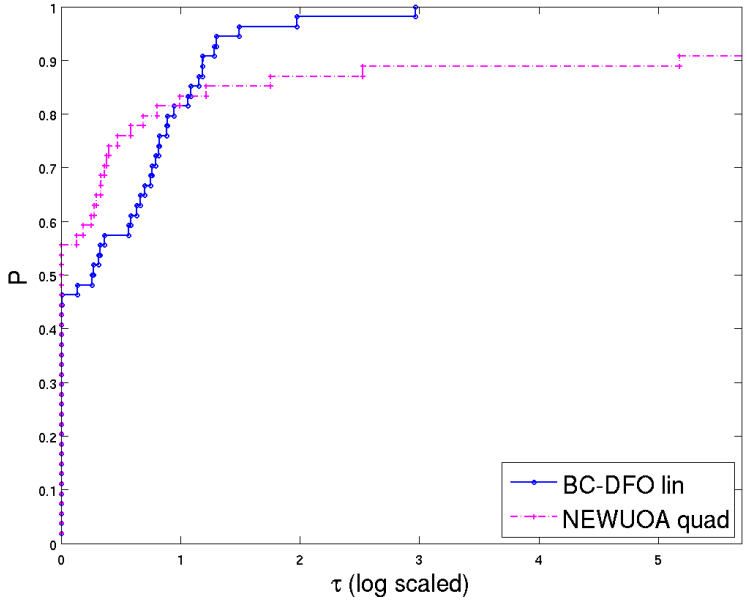


Figure 3.6: Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (4 digits of accuracy required in final function value)

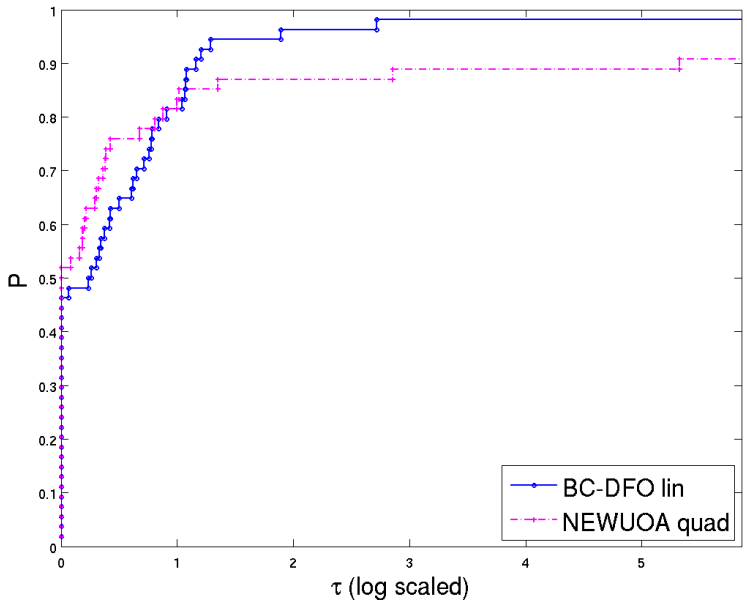


Figure 3.7: Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (6 digits of accuracy required in final function value)

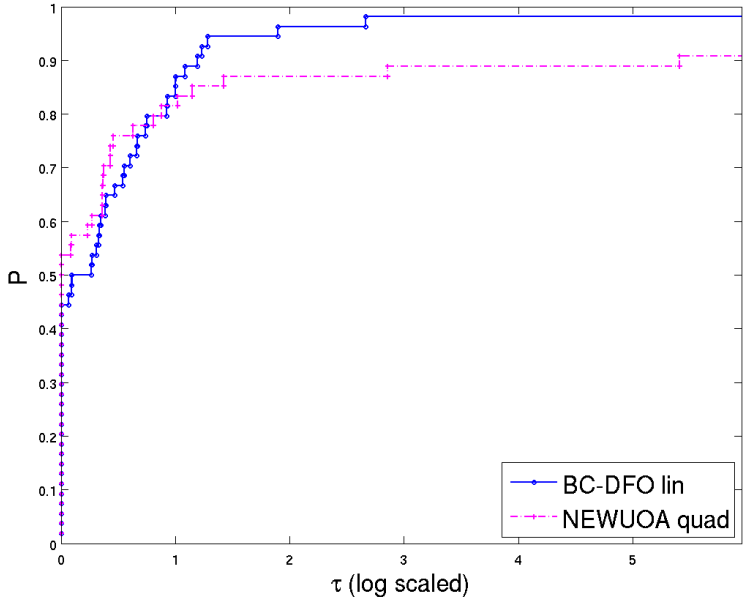


Figure 3.8: Comparison of BC-DFO and NEWUOA on unconstrained CUTEr problems (8 digits of accuracy required in final function value)

irrespective of the accuracy required. For instance, NEWUOA solves 52% of the problems faster and BC-DFO solves 46% of the test cases faster, when 6 digits of accuracy are requested. The gap is slightly bigger when 2, 4 or 8 significant digits in f are required. BC-DFO is able to solve all test problems up to 4 significant accurate digits required in f where it is not able to solve the problem PALMER1C to a more accurate level. NEWUOA has some difficulties to solve some of the PALMER problems in the given limit of function evaluations and it is not able to solve the highly ill-conditioned problem NASTY.

Considering in addition also our new algorithm ***BCDFO+***, we can see in Figures 3.9-3.12 that ***BCDFO+*** does perform considerably better than BC-FO and NEWUOA, irrespective of the accuracy required. For instance, when 6 correct digits in f are requested, ***BCDFO+*** solves 76% of the test cases faster than the other two, NEWUOA solves 24% of the problems fastest and BC-DFO solves 20% of the test cases fastest. To explain why the percentages do not sum up to 100%, we should note that equal results for two or more solvers count for each of them. Regarding robustness, ***BCDFO+*** has the same difficulty as BC-DFO to solve the problem PALMER1C in the limit of 15000 function evaluations to a highly accurate level.

Table B.4 in the appendix contains the detailed results for each of the three solvers for the various accuracy levels.

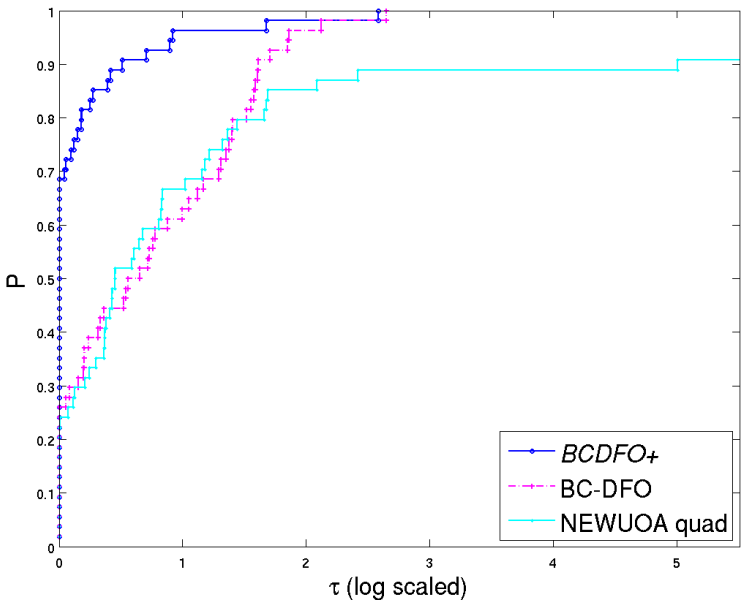


Figure 3.9: Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (2 digits of accuracy required in final function value)

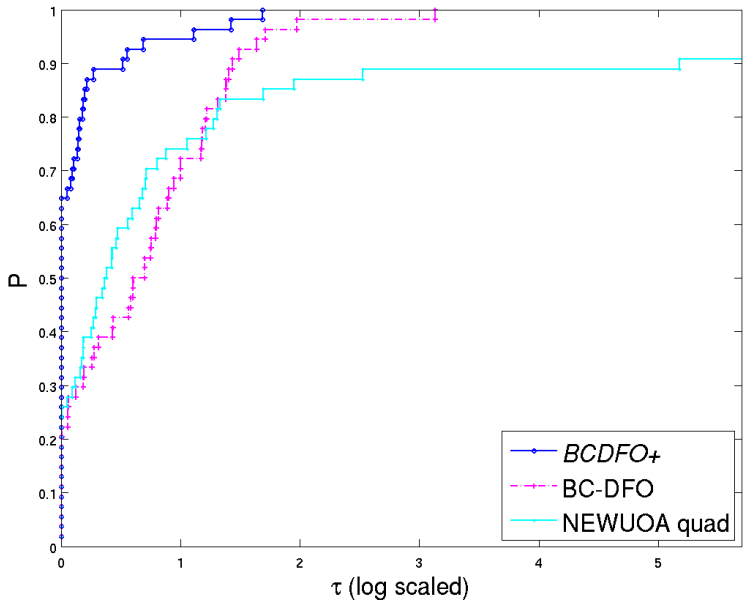


Figure 3.10: Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (4 digits of accuracy required in final function value)

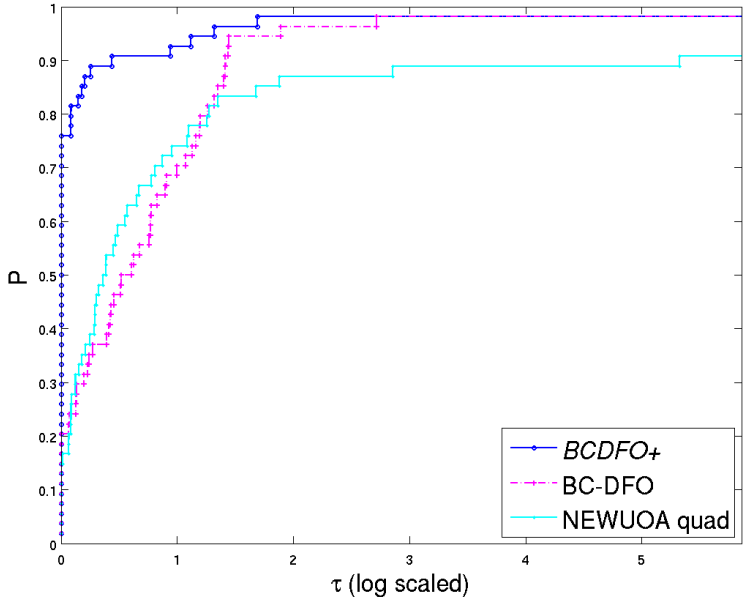


Figure 3.11: Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (6 digits of accuracy required in final function value)

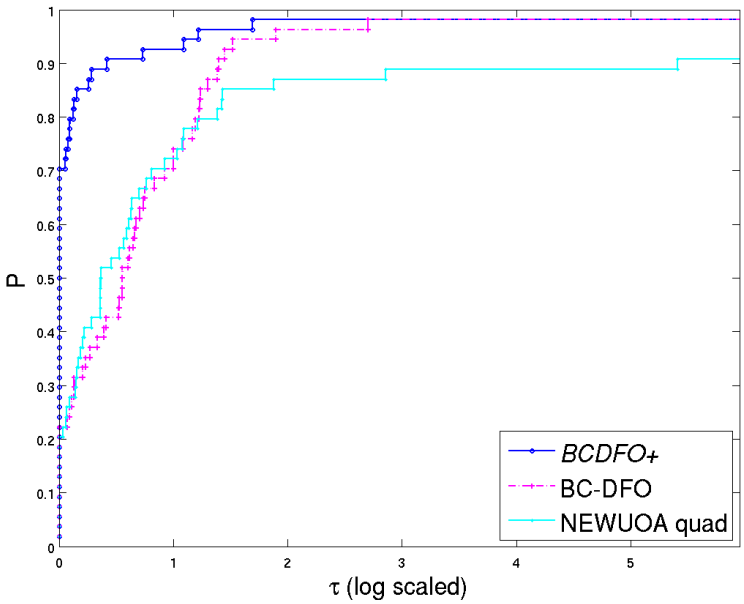


Figure 3.12: Comparison of BCDFO+, BC-DFO and NEWUOA on unconstrained CUTEr problems (8 digits of accuracy required in final function value)

3.5.8 Bound-constrained comparisons with BOBYQA

Again, we first want to report the results obtained with BC-DFO, the bound-constrained extension to Algorithm 2.3 and described in [74], when compared to BOBYQA, the bound-constrained extension to NEWUOA, developed by Powell.

The profiles reported in Figures 3.13-3.16 show that BC-DFO compares well with BOBYQA in the bound-constrained experiments. BOBYQA manages to solve 38 of the 40 test problems where it couldn't solve the problems 3PK and PALMER1A in 15000 function evaluations to a more accurate level. BC-DFO fails to solve the test problem SPECAN in all four cases and the problem HS25 to a highly accurate level.

The overall conclusion is that both solvers are equally robust, but that BC-DFO's dominance in efficiency increases with the desired level of accuracy. For the case where 2 correct significant figures are required (in Figure 3.13), BC-DFO solves 58% of the test cases faster than BOBYQA and BOBYQA solves 45% of the problems faster. For 8 correct significant figures, BC-DFO solves 67.5% of the test cases faster, and BOBYQA solves 35% of the problems faster. The performance of both codes does not vary significantly between requiring 4 or 6 correct significant figures.

In Figures 3.17-3.20, ***BCDFO+*** is considered in the comparison in addition to its predecessor BC-DFO and NEWUOA and it again outperforms the two other solvers. In terms of robustness, the three solvers perform equally good. ***BCDFO+*** is unable to solve the prob-

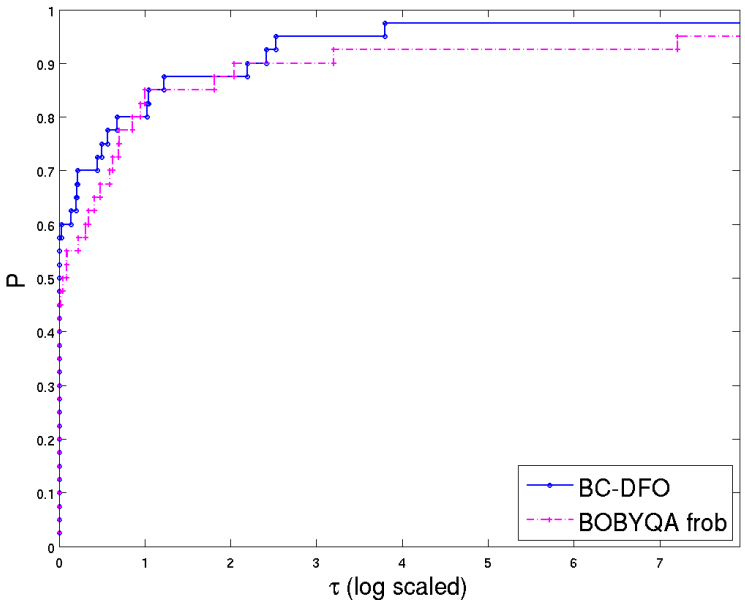


Figure 3.13: Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (2 digits of accuracy required in final function value)

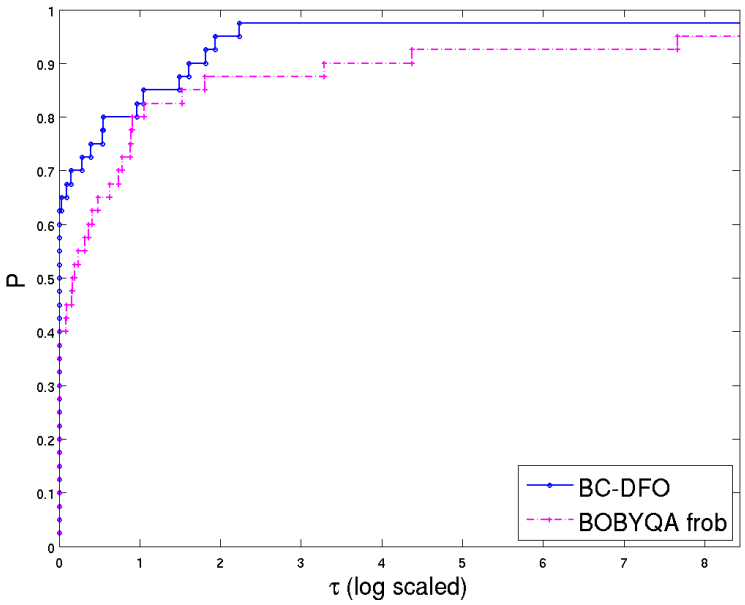


Figure 3.14: Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (4 digits of accuracy required in final function value)

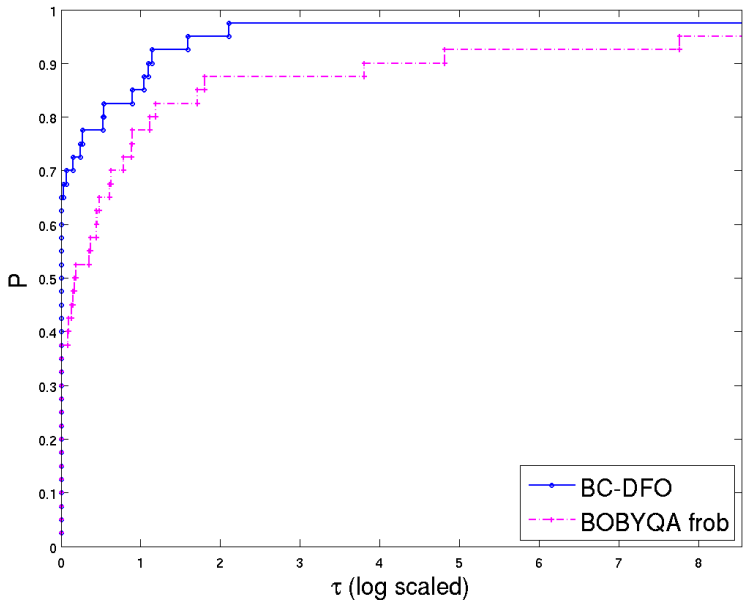


Figure 3.15: Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (6 digits of accuracy required in final function value)

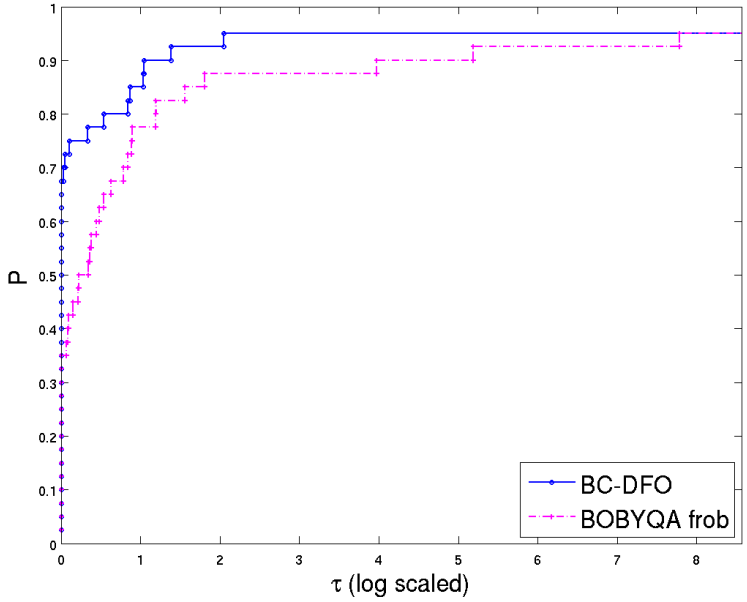


Figure 3.16: Comparison of BC-DFO and BOBYQA on bound-constrained CUTEr problems (8 digits of accuracy required in final function value)

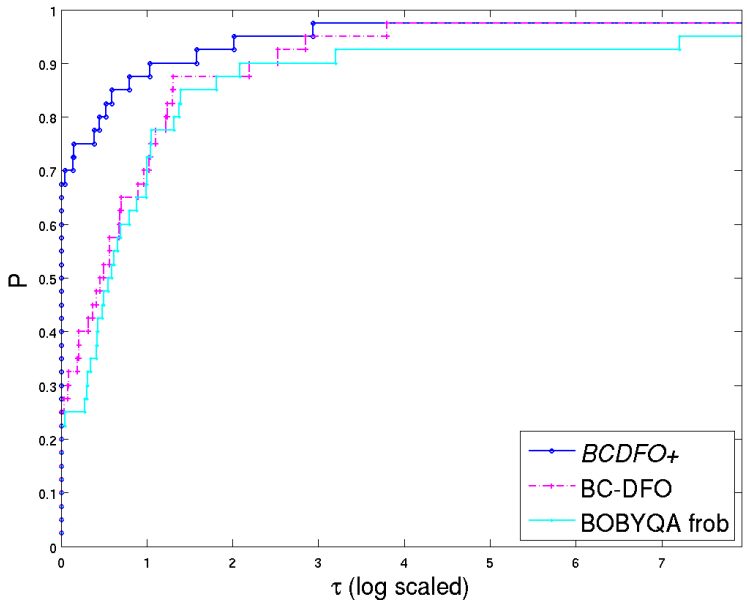


Figure 3.17: Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (2 digits of accuracy required in final function value)

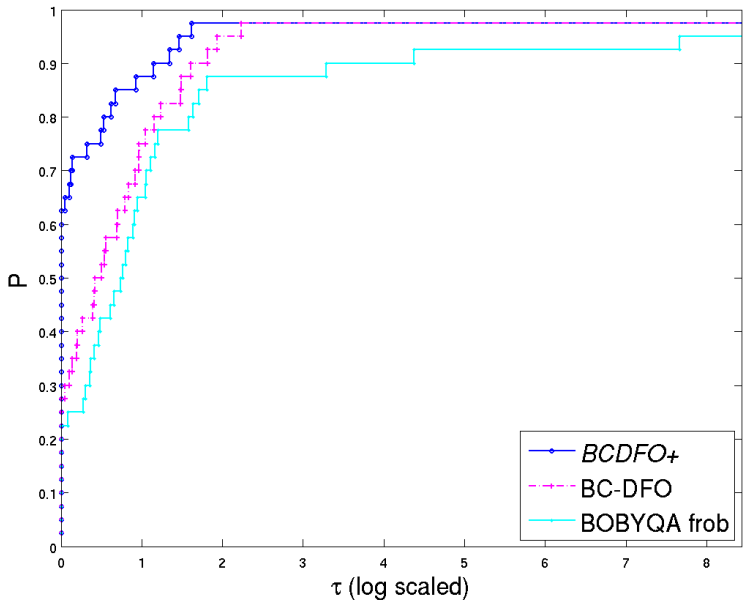


Figure 3.18: Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (4 digits of accuracy required in final function value)

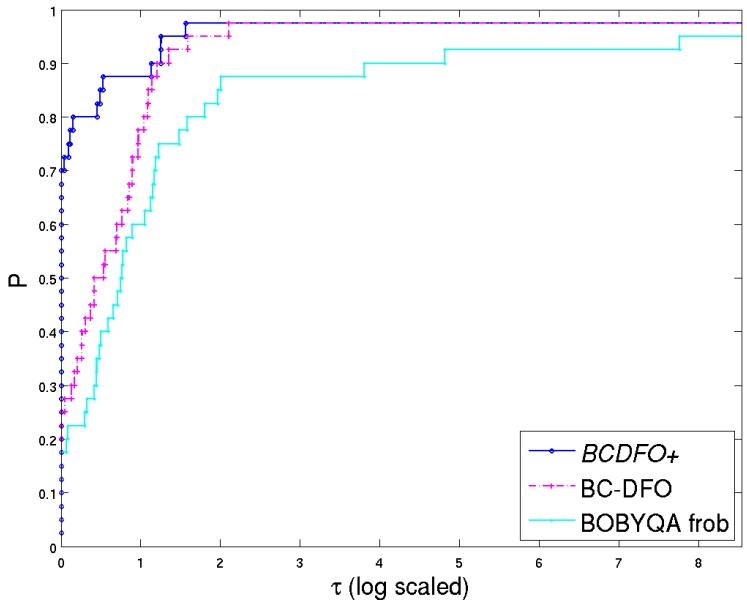


Figure 3.19: Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (6 digits of accuracy required in final function value)

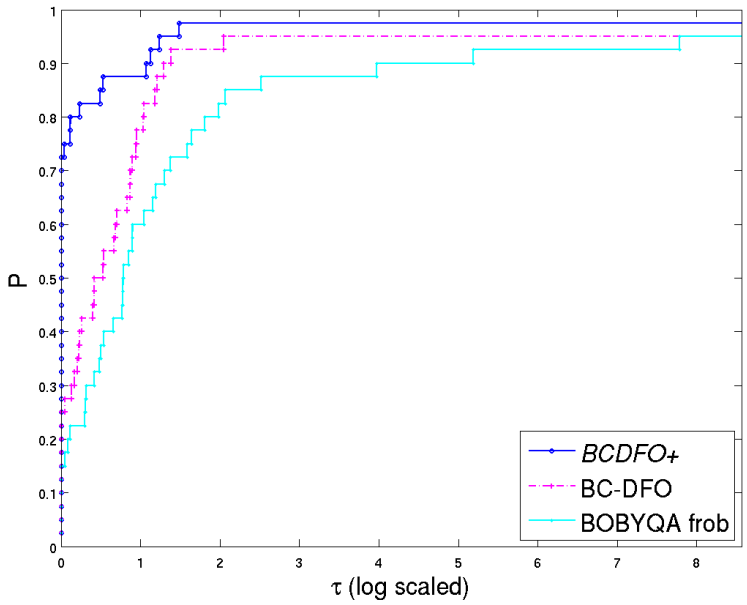


Figure 3.20: Comparison of BCDFO+, BC-DFO and BOBYQA on bound-constrained CUTEr problems (8 digits of accuracy required in final function value)

lem 3PK in the limit of 15000 function evaluations but it solves the problem SPECAN instead.

For high required accuracy, in Figure 3.20, ***BCDFO+*** solves 72.5% of the test problems faster than the other two. BC-DFO solves 25% of the test cases faster and BOBYQA is the fastest in 15% of the cases. For low required accuracy (in Figure 3.17), BOBYQA could gain some percentages and solves 22.5% of the test problems faster than the other two solvers. BC-DFO is still at 25% and ***BCDFO+*** solves the majority of 67.5% of the test cases fastest.

Table B.5 in the appendix contains the detailed results on the bound-constrained test set for the three solvers and the four accuracy levels.

3.5.9 Unconstrained comparisons with direct-search solvers

In this section, we want to compare our solvers to a selection of direct-search solvers. First, we put BC-DFO, NEWUOA and SID-PSM in one picture to see how those three solvers compare for unconstrained problems. As Figure 3.21-3.24 show, NEWUOA performs better than BC-DFO and SID-PSM for low accuracy whereas BC-DFO and NEWUOA are performing equally well for higher required accuracy where SID-PSM loses a bit over the other two solvers. For instance, BC-DFO solves 37% of the problems fastest, NEWUOA solves 33.5% of the test cases fastest and SID-PSM solves 28% of the problems faster than the two others, when 6 digits of accuracy are requested. Further, BC-DFO appears to be more robust, irrespective of the accuracy required. Whereas, as we have seen above, the robustness of BC-DFO and BOBYQA

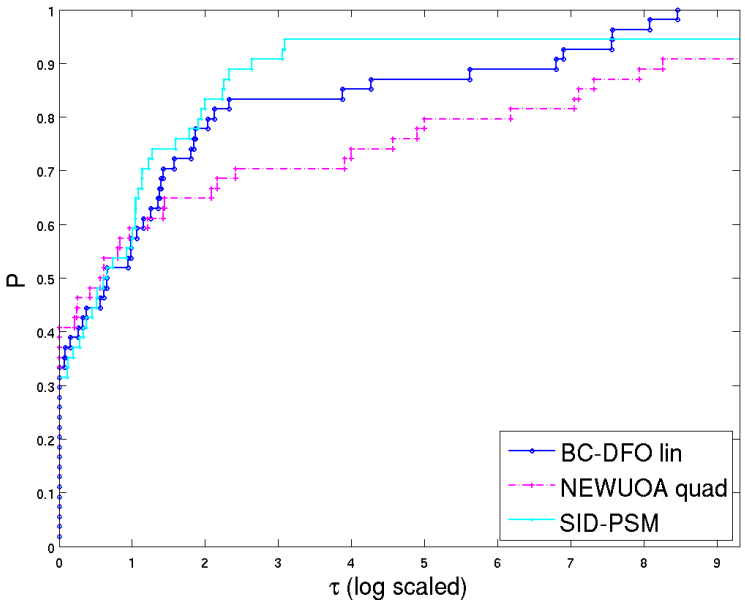


Figure 3.21: Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (2 digits of accuracy required in final function value)

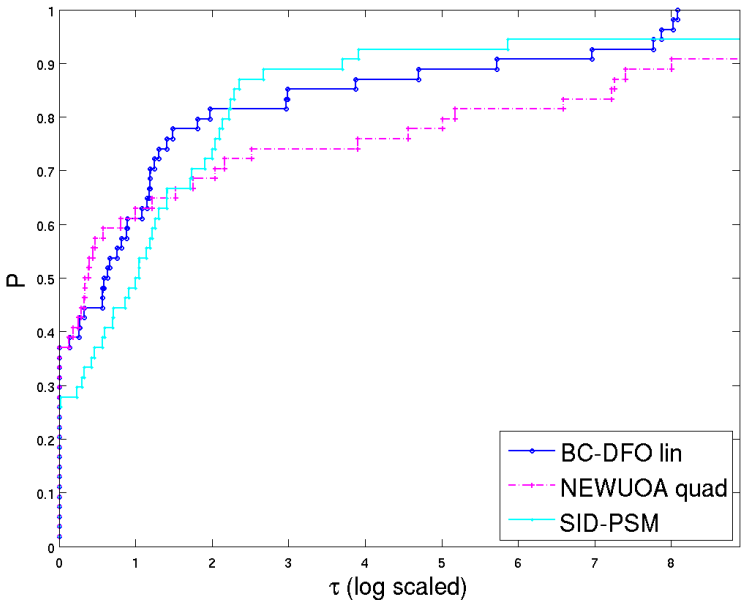


Figure 3.22: Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (4 digits of accuracy required in final function value)

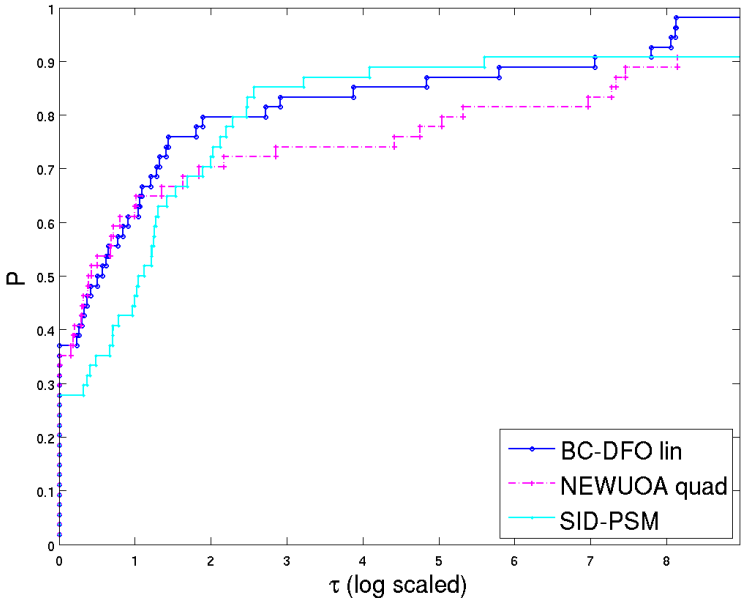


Figure 3.23: Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (6 digits of accuracy required in final function value)

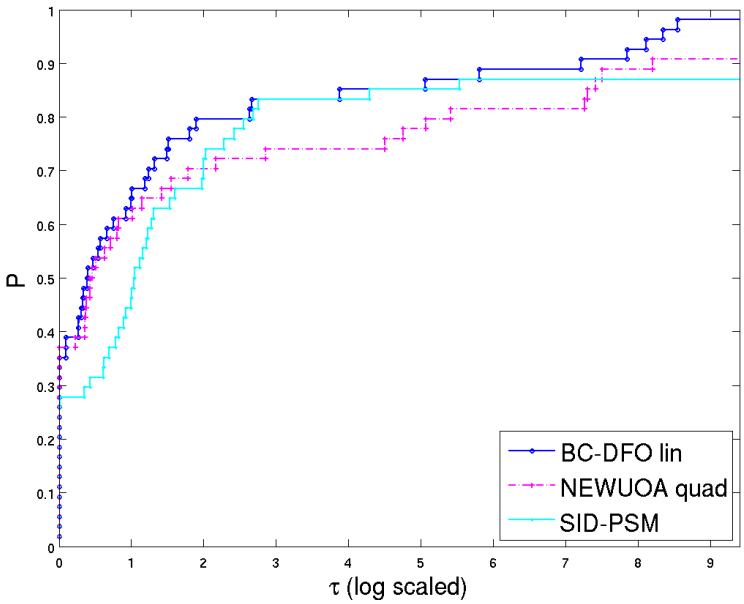


Figure 3.24: Comparison BC-DFO, NEWUOA and SID-PSM on unconstrained CUTEr problems (8 digits of accuracy required in final function value)

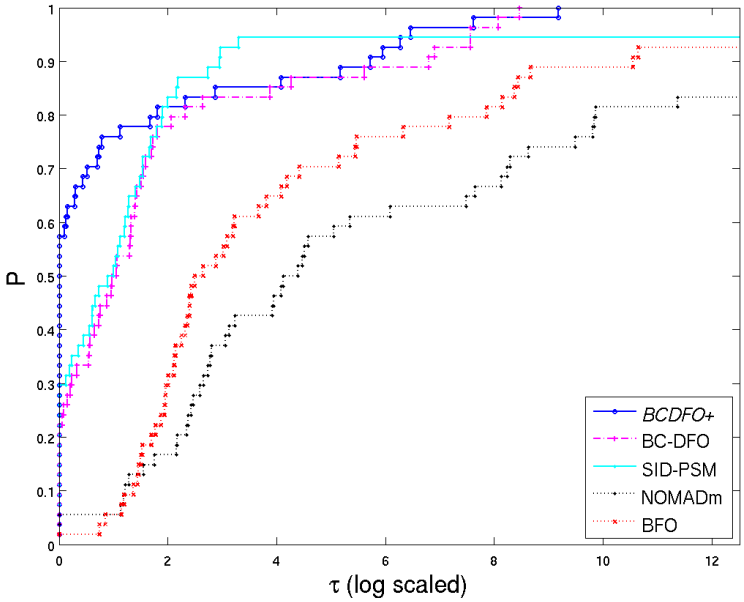


Figure 3.25: Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (2 digits of accuracy required in final function value)

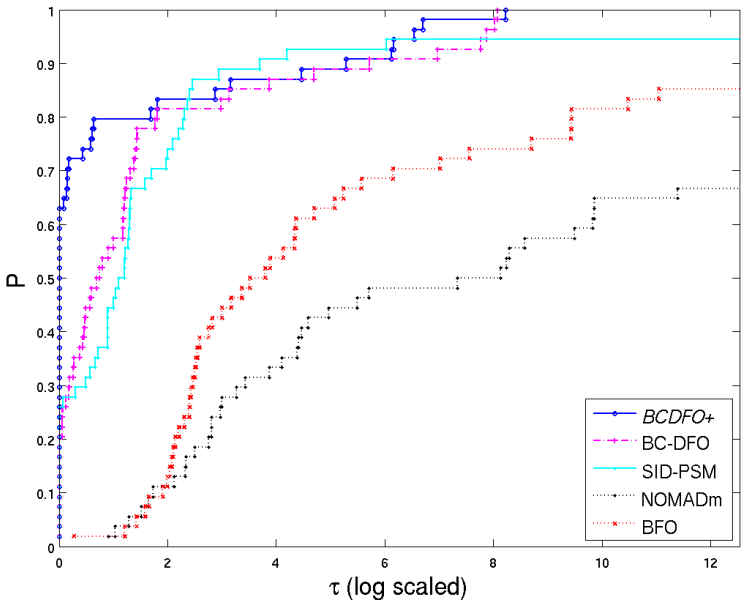


Figure 3.26: Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (4 digits of accuracy required in final function value)

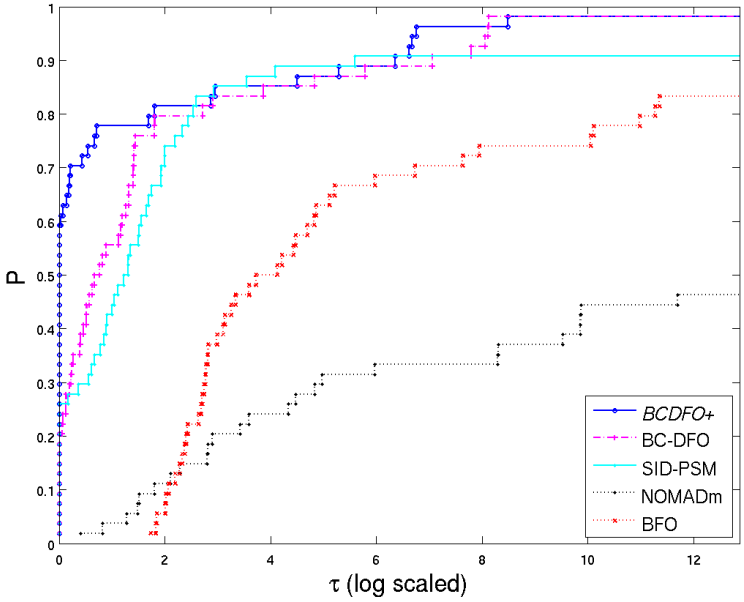


Figure 3.27: Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (6 digits of accuracy required in final function value)

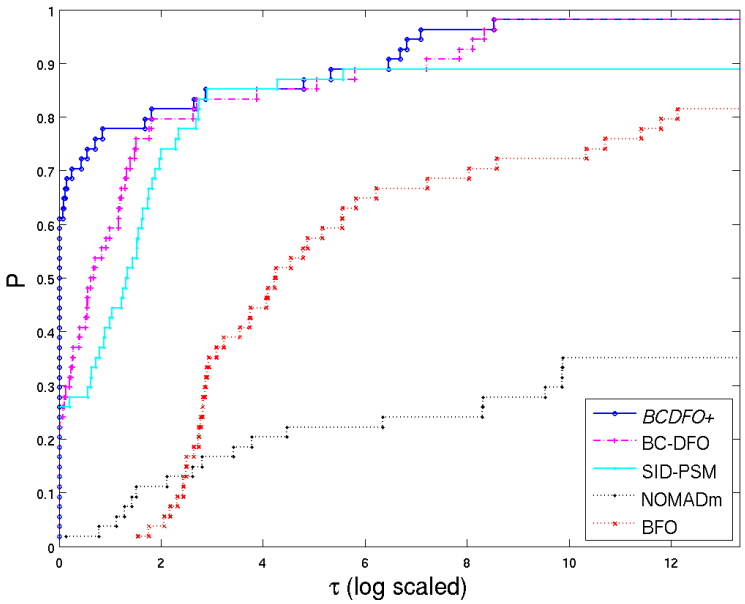


Figure 3.28: Comparison BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on unconstrained CUTEr problems (8 digits of accuracy required in final function value)

are relatively stable, the robustness of SID-PSM is slightly decreasing when higher accuracy is required. However, one could say that the three codes are equally robust as all three algorithms solve more than 90% of the test problems up to 6 required digits in f.

To extend our comparison, we are also considering the two pure direct-search solvers NOMADm and BFO in our comparison on the set of unconstrained test problems. Furthermore, we add also our new solver ***BCDFO+*** to the picture. The results are depicted in Figures 3.25-3.28. We see that in terms of efficiency, pure direct-search solvers have no big influence on the comparison whereas ***BCDFO+*** is outperforming the other solvers regardless of the accuracy required (as seen in Figures 3.21-3.24). But we need to mention that NOMADm manages to solve 5% of the test cases fastest for low required accuracy and 2% when higher accuracy is required. In terms of robustness, NOMADm can evolve from 35% solved test problems for 8 required correct digits in f to 82% solved problems for 2 required correct digits. BFO is amazingly stable and solves in any case more than 80% of the test problems and even 92% if low accuracy is required.

The detailed results for the three direct-search solvers on the unconstrained CUTEr test set can be found in Table B.6 in the appendix.

3.5.10 Bound-constrained comparisons with direct-search solvers

In this section, we want to compare our solvers to the mentioned direct-search solvers but on a set of bound-constrained problems from the CUTEr library. We start by comparing BC-DFO

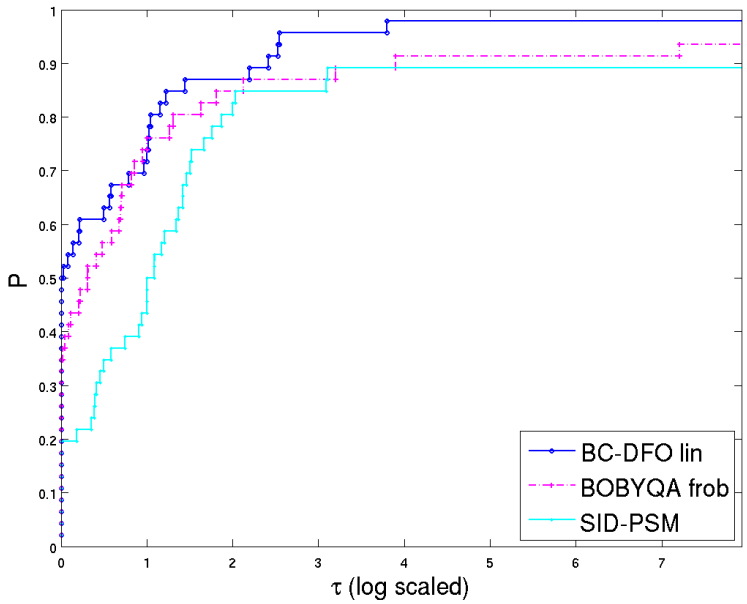


Figure 3.29: Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (2 digits of accuracy required in final function value)

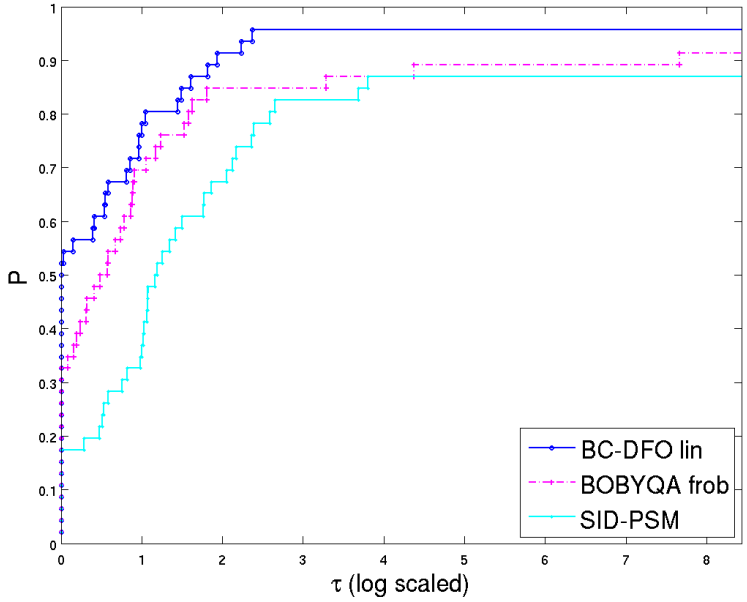


Figure 3.30: Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (4 digits of accuracy required in final function value)

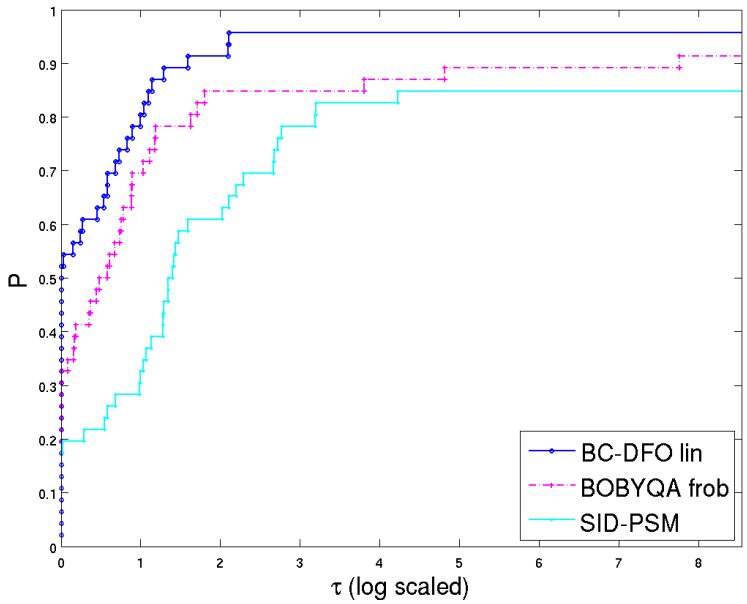


Figure 3.31: Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (6 digits of accuracy required in final function value)

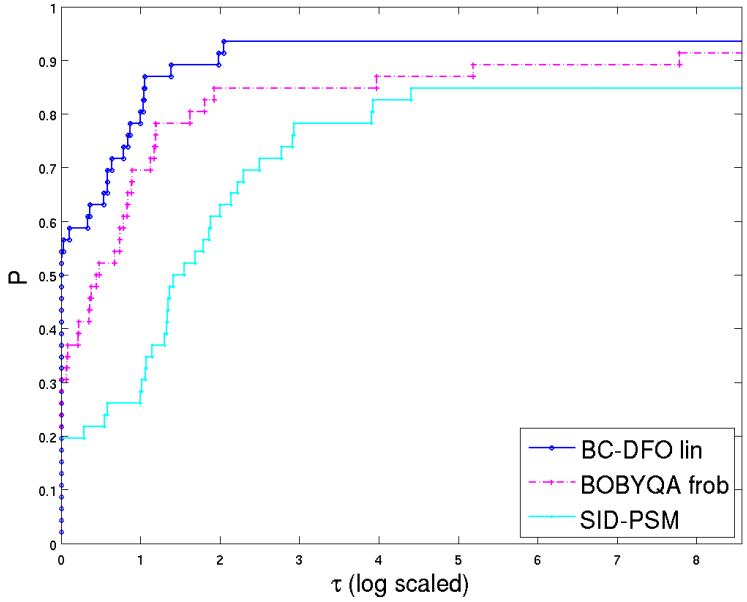


Figure 3.32: Comparison BC-DFO, NEWUOA and SID-PSM on bound-constrained CUTEr problems (8 digits of accuracy required in final function value)

to BOBYQA, the bound-constrained extension to NEWUOA, and to SID-PSM, the direct-search method enhanced by the use of polynomial models. The profiles reported in Figures 3.29-3.32 show that BC-DFO compares well with the two other codes in the bound-constrained experiments. SID-PSM solves 37 problems for low accuracy and 35 problems when high accuracy was required. In fact, our intuition is that all three solvers were able to solve the whole set of test problems because when the solvers were stopped due to a maximum number of function evaluaions, they were still progressing.

The overall conclusion is that BC-DFO is a little more robust and SID-PSM is a little less robust than BOBYQA, and that BC-DFO's dominance increases with the desired level of accuracy. For the case where 2 correct significant figures are required, BC-DFO solves 47.5% of the test cases faster than the other two, BOBYQA solves 35% of the problems fastest and SID-PSM has 22.5% of wins. For 8 correct significant figures, BC-DFO solves 52.5% of the test cases faster, BOBYQA is fastest for 30% of the problems and SID-PSM solves 22.5% of the problems faster than the other two solvers. The performance of the three codes is the same between requiring 4 or 6 correct significant figures.

To do a complete comparison, we are also comparing ***BCDFO+***, BC-DFO, SID-PSM to the two pure direct-search solvers NOMADm and BFO on our set of bound-constrained test problems. The results, depicted in Figures 3.33-3.36, follow our expectations. In particular, for low required accuracy, ***BCDFO+*** solves 67.5% of the test cases faster than the other solvers, BC-DFO solves 25% fastest, SID-PSM solves 17.5% of the problems fastest, NOMADm solves

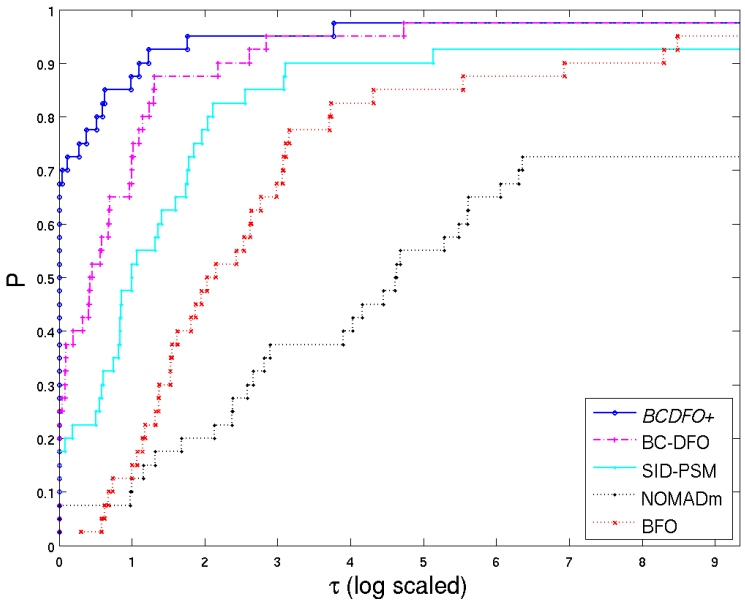


Figure 3.33: Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (2 digits of accuracy required in final function value)

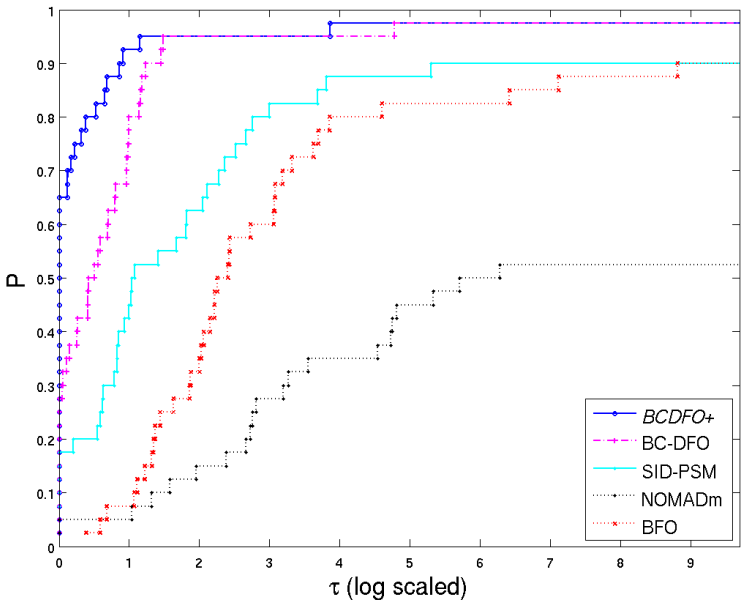


Figure 3.34: Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (4 digits of accuracy required in final function value)

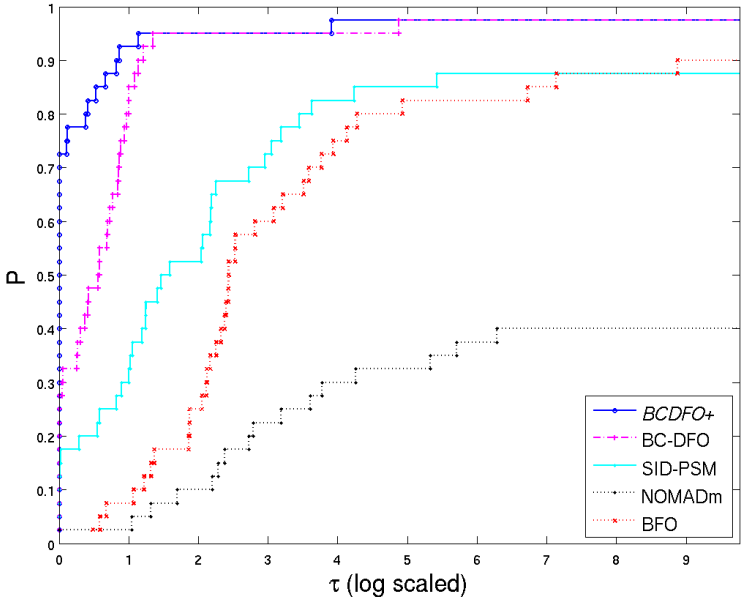


Figure 3.35: Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (6 digits of accuracy required in final function value)

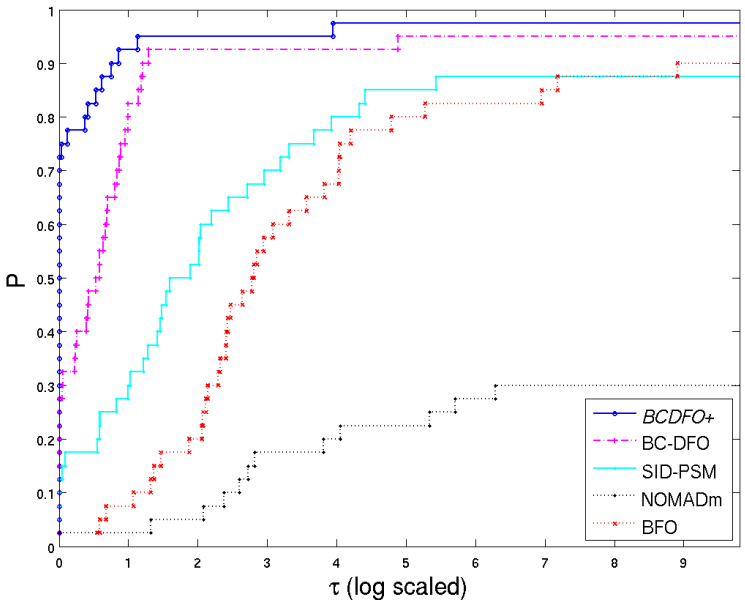


Figure 3.36: Comparison of BCDFO+, BC-DFO, SID-PSM, NOMADm, BFO on bound-constrained CUTEr problems (8 digits of accuracy required in final function value)

7.5% of the test cases faster than the other four solvers. This does not change much when high accuracy is required. ***BCDFO+*** and BC-DFO win a bit more with 72.5% and 27.5% respectively, and SID-PSM looses a bit and is the fastest solver in 12.5% of the test problems. NOMADm is never the fastest here. Regarding robustness, SID-PSM and BFO are equally robust with 88% and 90% of solved test problems for 8 required digits in f and 92.5% and 95% for 2 required correct digits in f, respectively. NOMADm manages to improve robustness the less accuracy is required. Explicitly, from 30% of the problems for high to 72.5% for low required accuracy.

Table B.7 in the appendix contains the detailed results for each of the three direct-search methods and each of the bound-constrained problems for the various accuracy levels.

Chapter 4

Industrial application incorporating noise

Aerospace industry is increasingly relying on advanced numerical flow simulation tools in the early aircraft design phase. Today's flow solvers based on the solution of the Euler and Navier-Stokes equations are able to predict aerodynamic behaviour of aircraft components under different flow conditions quite well. Nevertheless, the simulation of complex phenomena invariably requires the approximation of a function f_∞ by a function f that can be evaluated with a finite number of elementary operations, so that truncation errors $\epsilon_{\text{trunc}} = f_\infty - f$ come into play when attempting to minimize f_∞. Furthermore, we are faced with the computational noise generated by evaluating f in finite precision.

The development of suitable algorithms for dealing with noisy optimization problems has always been a challenging task [7, 19, 25, 47]. Some works (for instance [135, 136]) have in particular studied the impact of computational noise on simulation-based optimization problems where the parameters depend on the solution of a differential equation. Furthermore, several comparisons have been made of derivative-free algorithms on noisy optimization problems that arise in applications (e.g. in [55, 59, 108]).

In this thesis, we want to contribute to different aspects of studies concerning noise in general and a noisy aerodynamic application in particular. In Section 4.2, we demonstrate how the amplitude of the noise in a real-life application can be estimated using a tool which was originally developed to calculate higher order derivatives and to estimate round-off. In Section 4.3, our algorithm ***BCDFO+***, presented in the previous section, is adapted to solve noisy optimization problems from the CUTEr library and an aerodynamic shape-design problem in Section 4.4. In Section 4.5, we present a theoretical study on the allowed noise on a gradient which is used in a gradient-based line search method. In Section 4.6, the derivative-free method SNOBFIT, developed by Huyer and Neumaier [86, 87], is presented in the context of global optimization and we show the performance gain by enhancing this method with inexact gradient information. But first, we want to give a short description of the optimization tool OPTaliA which is used at Airbus to perform aerodynamic shape optimization. It will be used throughout this chapter to test our new algorithm, variations of it and other algorithms on a real-life application.

4.1 Presentation of the optimization suite OPTaliA

To perform aerodynamic shape optimization, Airbus uses the internally developed optimization suite OPTaliA. We can find a detailed description of this optimization suite in [90, 91, 92] which we quote here in parts. This high-fidelity optimization suite can improve aerodynamic performance of an aircraft by changes in the external shape. No reduced model is employed between the aerodynamic solver and the optimizer. The performed shape deformations are usually of small size to limit the impact on the other disciplines.

4.1.1 Optimization framework

A general optimization framework, represented in Figure 4.1, has been set up in OPTaliA. In order to be able to use various types of optimizers (gradient-based, model-based DFO, genetic, response surface) inside OPTaliA, it is organized as a black-box optimization framework. In

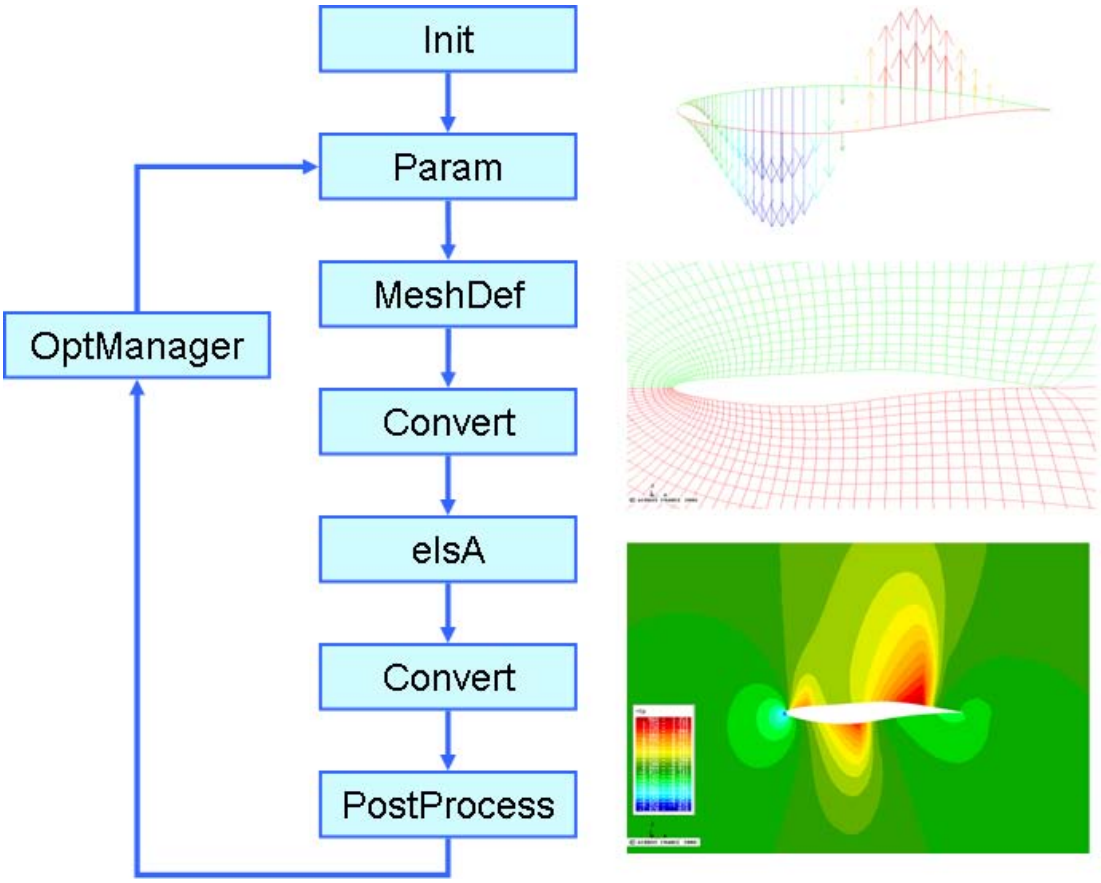


Figure 4.1: Optimization and evaluation framework in OPTaliA

particular, the optimization process can be interpreted as a subsequent alternation of two main tasks: function evaluation and optimization. Within the evaluator, the function value and if

needed the gradient value corresponding to the parametrized shape are computed. Once the shape has been evaluated, the optimizer proposes a new shape by using the new information on functions (and possibly gradients) and the next iteration is performed. In addition to the internal stopping criteria of the optimizer, the termination can be forced at the OPTaliA level if the number of function evaluations exceeds a given threshold *maxeval*. OPTaliA is able to perform function evaluations in parallel by running multiple jobs on high performance computers. One of the challenges in aerodynamic shape optimization is to manage running efficiently the evaluator and the optimizer automatically in batch mode. More particularly, the evaluator itself is a complex process requiring large computational resources.

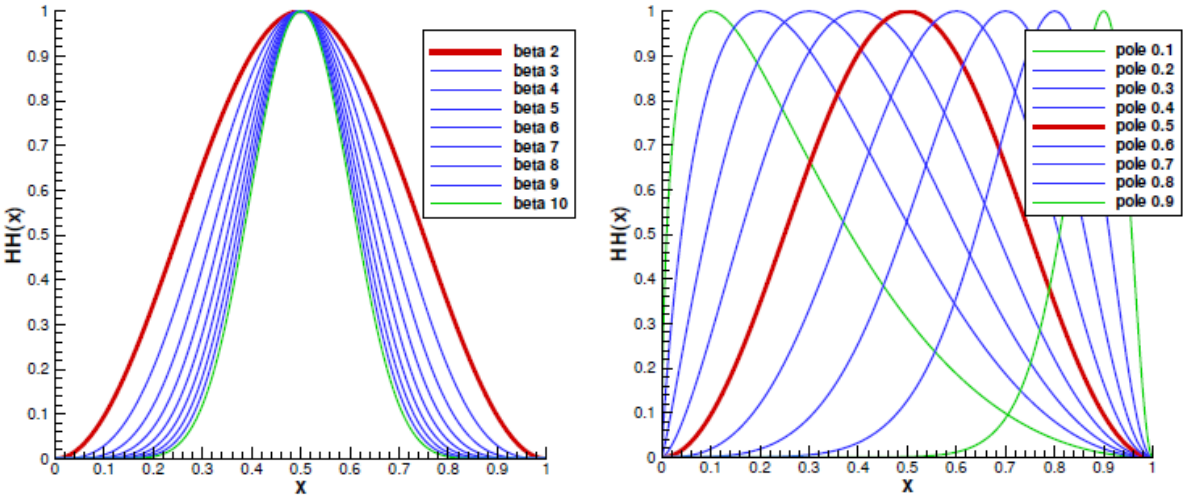
4.1.2 Evaluator for CFD functions

4.1.2.1 Shape parameterization and mesh deformation

The shape parameterization consists of applying Hicks-Henne sinusoidal bumps [83] on a surface skin of an initially block-structured mesh. Each bump is defined by three shape variables (A, p, β) as in the following formulation

$$HH(x) = A * \sin^{\beta}(\pi x^{\frac{\ln 0.5}{\ln p}}),$$
$$\text{with} \quad x \in [0;1], A \in \mathbb{R}, \beta \in [2,10], p \in]0;1[,$$

where A denotes the amplitude, p the position and β the width expansion of a Hicks-Henne bump. In Figure 4.2 from [90], we can see the evolution of the shape of such a bump depending on the parameters β and p when varied inside their parameter bounds, respectively. This type



(a) Variation of parameter β

(b) Variation of parameter p

Figure 4.2: Illustration of the parameters β and p

of deformation was initially developed by Hicks and Henne [83] for numerical optimization

of airfoils. When applied on a bidimensional surface corresponding to a three dimensional shape, a linear propagation of the bump is done in the second direction using fixed propagation distances. In Figure 4.3, we have the illustration of the deformation vectors of a wing profile

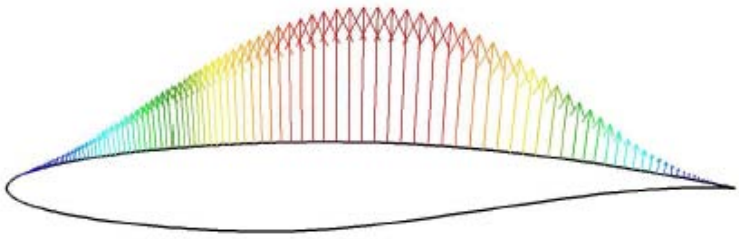


Figure 4.3: Two-dimensional Hicks-Henne sinusoidal bump

by application of a two-dimensional Hicks-Henne sinusoidal bump where the amplitude of the bump is purposedly exaggerated for illustrative reasons.

4.1.2.2 Flow simulation

Flow analyses were performed with the elsA [24] software developed by ONERA - the French Aerospace Lab - and CERFACS. The flow is simulated by solving the Reynolds Averaged Navier-Stokes (RANS) equations associated with the one-equation Spalart-Allmaras turbulence model on block structured meshes using a cell-centered finite volume approach. The second order Roe's upwind scheme with the Van Albada limiter is used as spatial scheme coupled with an implicit time resolution. Diffusive terms are discretized with a second order centered scheme. Multigrid and local time stepping techniques are used to increase the convergence rate. An overview of various results obtained with elsA is given by Cambier and Veuillot [24].

One of the main requirement from designers is to obtain the same results when using the CFD solver inside or outside the automatic optimization tool. As hysterisis phenomena are common when dealing with transonic flows, the same initial flow condition (uniform flow) was used for all the simulations performed during the optimization. Therefore, the computational cost of CFD simulations cannot be reduced by using a restart strategy using the flow solution corresponding to the previous shape. Hence, the computational cost of the optimization grows linearly with the number of function evaluations.

The sensitivity of the objective function with respect to the design variables is computed using the discrete adjoint method [88]. For an explicit presentation of the adjoint system solved within elsA and an evaluation of the accuracy of the sensitivities, the reader is referred to Peter et al. [112] and Meaux et al. [98]. This method enables to compute the sensitivity of a single function with respect to n design variables at the cost of one linear system resolution (same size as the linearized RANS system). The gradient vector is computed using approximatively the same computational time ($factor \approx 1.5$) as one mean-flow simulation. For typical aerodynamical problems considering tens or hundreds of design variables and a few functions (lift,

drag), this is a considerable improvement over the classical method of forward finite differences requiring as many flow solutions as design variables.

4.1.2.3 Aerodynamic function computation

The objective function chosen is the far-field pressure drag,

$$f = CDP = CDVP + CDW + CDI \tag{4.1}$$

where $CDVP$, CDW and CDI denote the viscous pressure drag, the wave drag and the induced drag, respectively. The friction drag, CDF, is excluded from the objective function as it does not change significantly with the amplitude of deformation considered. The wetted surface is almost unaffected by the shape deformation. The post-processing code used is *ffd41* [50] and is also developed by ONERA. It implements a far-field drag break-down method. The two main advantages of this approach are its ability to decompose pressure drag into physical components (wave drag, induced drag, viscous pressure drag) and its accuracy through a filtering of non-physical drag (spurious drag).

4.1.3 Interface Matlab – OPTaliA

To be able to test our algorithm in a black-box-type manner on real CFD test problems provided by Airbus, we had to interface our optimization software packages as they are currently developed in Matlab. The optimization suite OPTaliA at Airbus provides the possibility to simply call for a function or gradient evaluation of a specific problem. To use this possibility we had to adapt the Matlab routine `[f,g]=get_fg(x)` which is called by an optimizer to obtain function and gradient values. An evaluation in OPTaliA either comprises a function evaluation or a function and gradient evaluation. So, depending on the number of output-arguments of `get_fg(x)`, one or two is possible, the required calculations are done in OPTaliA. Each call to OPTaliA in the Matlab routine `get_fg(x)` is done in five steps

1. Write the x-vector to a file
2. Open connection to frontal node via ssh and calling a shell-script:
   (a) Source the bash-shell
   (b) Change directory to specific problem
   (c) Call OPTaliA to evaluate either f or f and g at point x
3. Exit connection to frontal node
4. Wait
5. Read either f-value or f- and g-values from a file

As the optimization algorithm in Matlab is running on a calculation node of the server and OPTaliA is to call from the frontal node of the server, in Step 2 of the calling routine, a shell-script is executed which then calls OPTaliA.

4.2 Estimating noise by Hamming's difference table

Coming back to the target of optimizing a noisy function f, a very important fact is to have an idea on the amplitude of the noise of the function to minimize. Dennis and Schnabel in [49] use a parameter $fdigits$ which specifies the number of reliable digits returned by the objective function. It is used in the form $10^{-fdigits}$ to specify the relative noise in f. They suggest, if the objective function f is suspected to be noisy but the approximate value of $fdigits$ is unknown, that it should be estimated by the routine of Hamming [81], which is also given in the book by Gill, Murray and Wright [65]. We are going to explain this technique now.

4.2.1 The idea

The difference table was originally developed to calculate higher order derivatives by finite differences. Finite difference formula for derivatives of arbitrary order are defined from high-order differences of f. In [81], the simplest formula for a derivative of order k is considered. Here, the forward-difference operator Δ is defined by the relation

$$\Delta f(x) = f(x+h) - f(x), \tag{4.2}$$

where the division by h, as for traditional finite difference formula, is suppressed. Higher-order forward-difference operators Δ^k can be defined applying the recurrence formula

$$\Delta^k f(x) = \Delta(\Delta^{k-1} f(x)) = \Delta^{k-1} f(x+h) - \Delta^{k-1} f(x). \tag{4.3}$$

If we defined $f_j = f(x+jh)$, the numbers $\Delta^k f_j$ can be arranged in a difference table as is depicted in Table 4.1.

f_j	Δf_j	$\Delta^2 f_j$	$\Delta^3 f_j$	..
f_0				
	Δf_0			
f_1		$\Delta^2 f_0$		
	Δf_1		$\Delta^3 f_0$	
f_2		$\Delta^2 f_1$		..
	Δf_2		$\Delta^3 f_1$	
f_3		$\Delta^2 f_2$	:	
	Δf_3	:		
f_4	:			
:				

Table 4.1: Difference table by Hamming

Each of the differences is computed by substraction of two entries in the previous column. It can be shown that $\Delta^k f = h^k f^{(k)}(x) + O(h^{k+1})$ [65]. Given a difference table, constructed

from the function values $f(x+jh), j=0,1,...,k$, we obtain

$$f^{(k)}(x) \approx \frac{1}{h^k}\Delta^k f_0. \tag{4.4}$$

To get an accuracy estimation of a function, we have to evaluate it at a set of values $\{x_i\}$ (about twenty values to be able to construct the 10-th finite differences), where the point x_i is defined by $x_i = x + ih$, and $|h|$ is small. We assume that each computed value $\tilde{f}_i$ is of the form

$$\tilde{f}_i = f(x_i) + \delta_i \equiv f(x_i) + \theta_i \epsilon_A, \tag{4.5}$$

where $|\theta_i| \leq 1$ and ϵ_A is called the absolute precision or absolute noise level. We can obtain a difference table as described above, by considering the set of values $\tilde{f}_i$ as the first column of a table, and defining each successive column as the difference of the values in the previous column. By the linearity of the difference operator Δ, after k differences we will have

$$\Delta^k \tilde{f}_i = \Delta^k f_i + \Delta^k \delta_i, \tag{4.6}$$

where we know from above that $\Delta^k f = h^k f^{(k)} + O(h^{k+1})$. Under mild conditions on f, the value $|h^k f^{(k)}|$ will become very small for moderate values of k if h is small enough. Therefore, the higher differences of the computed values should reflect almost entirely the differences of the errors δ_i.

Under the assumptions on the distribution of $\{\theta_i\}$ as random variables that the errors are uncorrelated and have the same variance, the later differences $\Delta^k \tilde{f}_i$ tend to be similar in magnitude and to alternate in sign. In practice, this desired pattern of behaviour typically begins when k is 4 or 5, and the largest value of k that would usually be required is 10 [65]. The formula that has been suggested in [81] for the estimate of ϵ_A from the k-th column is

$$\epsilon_A^{(k)} \sim \frac{\max_i |\Delta^k \tilde{f}_i|}{\beta_k} \tag{4.7}$$

where the division by

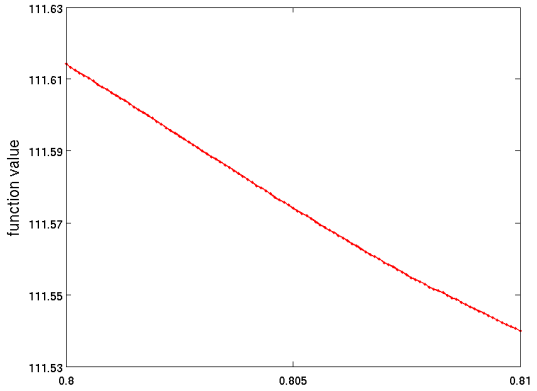
$$\beta_k = \sqrt{\frac{(2k)!}{(k!)^2}}.$$

is performed to remove the round-off error coming from calculating the k-th differences.

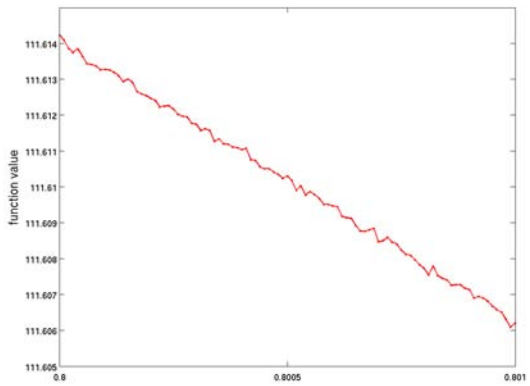
4.2.2 Case study of an aerodynamic function and gradient

We shall consider applying the technique described in this section to a specific numerical example and, of course, we are most interested in the noise level of the objective function of our aerodynamic application provided by Airbus. In this thesis, all our experiments on this application are carried out on OPTaliA version 3.5. The noise may have already been significantly reduced in a more recent version of the code.

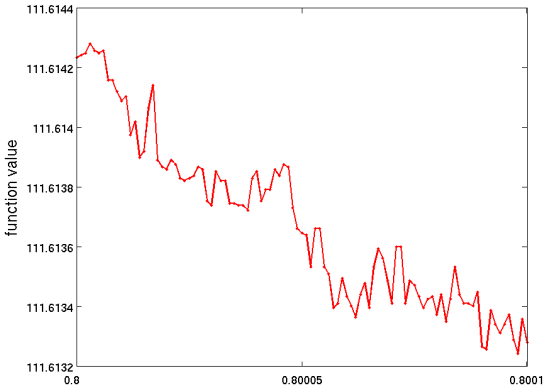
We consider a one-dimensional objective function, called CDP, describing the evolution of the pressure drag when changing only the position of one bump close to the value $x = 0.8$. The



(a) Stepsize $h = 10^{-4}$



(b) Stepsize $h = 10^{-5}$



(c) Stepsize $h = 10^{-6}$

Figure 4.4: Navier Stokes pressure drag objective function

function was sampled three times at 100 values beginning from $x = 0.8$ at sampling distances $h = 10^{-4}, h = 10^{-5}$ and $h = 10^{-6}$, respectively. Whereas in Figure 4.4(a), no noise is visible to the naked eye, Figure 4.4(b) and 4.4(c) let get visible the real nature of the function. To estimate the noise of the example function and compare the results of the three samplings, the technique applying a difference table as described above is used. We compare the estimated noise level to the average regression error value obtained by computing different linear least-squares approximations. For this, we fit a polynomial p of degree 2, 3 or 4 to the given data set and evaluate the polynomial at each sample point. The average error between p and the sampled data is then obtained by applying the formula

$$\epsilon_A = \frac{\|f(x) - p(x)\|}{\sqrt{98}}.$$

The comparison is depicted in Table 4.2 and it shows a good consistency of the computed noise

method	ϵ_A for $h = 10^{-4}$	ϵ_A for $h = 10^{-5}$	ϵ_A for $h = 10^{-6}$
Diff. table	1.28e-4	1.77e-4	7.85e-5
LLS 2nd order	3.33e-4	8.51e-5	8.25e-5
LLS 3rd order	1.12e-4	8.46e-5	8.15e-5
LLS 4th order	1.07e-4	8.44e-5	8.08e-5

Table 4.2: Noise levels in CDP obtained by different strategies and sampling distances h

levels ϵ_A by the different strategies used. The noise level of the aerodynamic function CDP turns out to be of the order of 10^{-4} for all techniques and sampling distances applied.

This noise estimating technique should be directly applicable to estimate the noise level of the gradient of a function of one single variable. We consider the adjoint state gradient of an

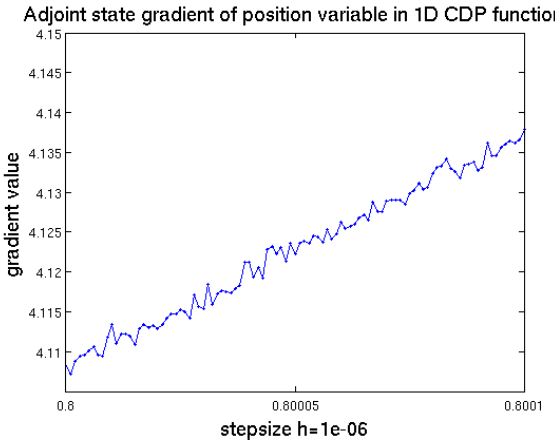


Figure 4.5: Navier Stokes pressure drag adjoint state gradient

Airbus Navier-Stokes test case consisting only of the position variable (depicted on Figure 4.5) and evaluated the gradient at 100 points beginning from $x = 0.8$ with a sampling distance of $h = 10^{-6}$. Table 4.3 gives the columns of differences when the noise-estimation method described above is applied to this example. The data set used in the difference table are the

g_i	Δg_i	$\Delta^2 g_i$	$\Delta^3 g_i$	$\Delta^4 g_i$	...	$\Delta^{10} g_i$
4.10838	-1.146e-3	2.659e-3	-3.514e-3	3.834e-3	...	5.455e-2
4.10723	1.514e-3	-8.539e-4	3.201e-4	6.66e-4	...	-6.771e-2
4.10875	6.597e-4	-5.338e-4	9.861e-4	-1.510e-3	...	2.377e-2
4.10941	1.259e-4	4.523e-4	-5.242e-4	-9.259e-4	...	-2.865e-2
4.10953	5.782e-4	-7.19e-5	-1.450e-3	3.871e-3	...	1.925e-1
4.11011	5.063e-4	-1.522e-3	2.421e-3	-9.011e-4	...	-4.226e-1
4.11062	-1.016e-3	8.985e-4	1.519e-3	-4.611e-3	...	5.143e-1

Table 4.3: Difference table for the gradient of CDP with $h = 10^{-6}$

first 20 sample points. We observe that the fourth differences (and in addition, five through nine, which are not shown), display already some kind of alternation in sign. The desired pattern of behaviour is finally reached for the differences at $k = 10$. Using formula (4.7), the estimates of ϵ_A corresponding to $k = 4, \ldots, 10$ are $1.49 \cdot 10^{-3}, 1.48 \cdot 10^{-3}, 1.37 \cdot 10^{-3}, 1.31 \cdot 10^{-3}, 1.27 \cdot 10^{-3}, 1.19 \cdot 10^{-3}$, and $1.20 \cdot 10^{-3}$. The approximate error using the linear least-squares method (using a cubic polynomial) was $8.78 \cdot 10^{-4}$, and hence the estimate of ϵ_A is reasonably good.

Testing the data of all 100 sampled gradient values from Figure 4.5, the estimate of ϵ_A gives $1.74 \cdot 10^{-3}$ and when approximating by linear least-squares, we get an error estimation of $8.98 \cdot 10^{-4}$ which is very close to the results using only 20 samples. We can therefore conclude that it is not necessary that the number of sample points is exceedingly big to get a reliable error estimation.

4.3 Numerical experiments on noisy CUTEr test problems

In this section, we present numerical experiments of our new algorithm on noisy test problems. As we want to show in the next section the ability of our algorithm to solve the presented real-life application, we decided to adapt the test setting to this situation already in this section as much as possibe. This means, the construction of the noisy test set has been done as to resemble simulation-based problem functions. Furthermore, as in industry the computational budget is most of the time the most restricting parameter, we present our results using data profiles as introduced by Moré and Wild [102] for exactly this reason.

4.3.1 Noisy test problems

For our testing on noisy functions, we use the same test problems from the CUTEr testing environment as described in Section 3.5.2 for the numerical experiments on smooth problems. We want to mimic simulations that are defined by an iterative process and these simulations are not stochastic but tend to produce results that are generally considered noisy (as we have observed in the previous sections). The noise in this type of simulation is better modeled by a function with both high-frequency and low-frequency oscillations. We thus define a noisy problem by

$$f_n(x) = f(x) + \epsilon_f \phi(x) \tag{4.8}$$

with ϵ_f being the absolute noise level and the noise function $\phi : \mathbb{R}^n \mapsto [-1, 1]$ is defined in terms of the cubic Chebyshev polynomial P_3 by

$$\phi(x) = P_3(\phi_0(x)), \quad P_3(\alpha) = \alpha(4\alpha^2 - 3), \tag{4.9}$$

where

$$\phi_0(x) = 0.9 \sin(100\|x\|_1) \cos(100\|x\|_\infty) + 0.1 \cos(\|x\|_2) \tag{4.10}$$

as suggested in [102]. Contrary to [102], where a relative noise level is applied to the smooth functions, we consider an absolute noise level as we believe from our experience that the noise level is not decreasing with the function value in our considered application.

The function $\phi_0(x)$ defined by (4.10) is continuous and piecewise continuously differentiable. The composition of $\phi_0(x)$ with P_3 eliminates the periodicity properties of $\phi_0(x)$ and adds stationary points to $\phi(x)$ at any point where $\phi_0(x)$ coincides with the stationary points $(\pm\frac{1}{2})$ of P_3.

4.3.2 Stopping criterion for the comparison

In [102], a stopping criterion is suggested which is based on the achieved reduction in the function value relative to the best possible reduction achieved by one of the solvers. They propose to use the convergence test

$$f(x) \leq f_L + \tau(f(x_0) - f_L), \tag{4.11}$$

where τ is a tolerance, x_0 is the starting point for the problem, and f_L is computed for each problem $p \in \mathcal{P}$ as the smallest value of f obtained by any solver within a given number of function evaluations. Writing (4.11) as

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_L), \tag{4.12}$$

one can see that (4.11) requires that the reduction $f(x_0) - f(x)$ achieved by x be at least $1 - \tau$ the best possible reduction $f(x_0) - f_L$.

The stopping criterion was also used before in [56] and [97] but with f_L set to an accurate estimate of f at a global minimizer or at a local minimizer obtained by a derivative-based solver, respectively. Setting f_L to an accurate estimate of f at a minimzer is not an appropriate approach in the situation where function evaluations are expensive because no solver may be able to satisfy the convergence test within the given computational budget. Applying (4.11) assures that at least one solver will satisfy the convergence test for any $\tau \geq 0$. The tolerance τ represents the percentage of decrease from the starting value $f(x_0)$, where $\tau = 10^{-1}$ represents a modest decrease but smaller values of τ require larger decreases in the function value.

4.3.3 Data profiles

When benchmarking optimization solvers for problems with expensive function evaluations, Moré and Wild [102] suggest to use so-called *data profiles* instead of performance profiles. Performance profiles provide an accurate view of the relative performance of solvers within a given number of function evaluations but we agree with the statement in [102] that performance profiles do not provide sufficient information for a user with an expensive optimization problem. These users have usually a very limited computational budget and are therefore interested in the percentage of problems that can be solved with α function evaluations. Thus, data profiles were developed to show the performance of solvers as a function of a computational budget.

This information can be obtained by letting $t_{p,s}$ be the number of function evaluations required to satisfy the stopping criterion (4.11) for a given tolerance τ, since then

$$d_S(\alpha) = \frac{1}{|\mathcal{P}|}\text{size}\{p \in \mathcal{P} : t_{p,s} \leq \alpha\} \tag{4.13}$$

is the percentage of problems that can be solved with α function evaluations. If the convergence test is not satisfied after *maxeval* evaluations, $t_{p,s} = \infty$ is set.

Contrary to [102], we do not divide the number $t_{p,s}$ by $n+1$ to take into account that the function evaluations needed to satisfy a given convergence test is likely to grow as the number of variables increase. We use the measure d_S independent of the number of variables in the problem $p \in \mathcal{P}$ as we believe that practicioners have a limited computational budget which does not depend on the dimension of the problem.

4.3.4 Comparison of different model types on unconstrained problems

In this testing, we compare the four model options of our algorithm ***BCDFO+*** as we did in the previous section on smooth problems, hoping to see which one is the most appropriate to use in the case of noise in the objective function. We present the data profiles for $\tau = 10^{-k}$ with $k = \{1, 5\}$ and for maximum evaluation numbers of 200 and 15000 because we are interested in the short- and long-term behaviour of the different model-options as the required accuracy level changes. Figures 4.6 and 4.7 show the comparison of the different model types for a maximum number of 200 function evaluations. We can observed a slight advantage of the sub-basis model

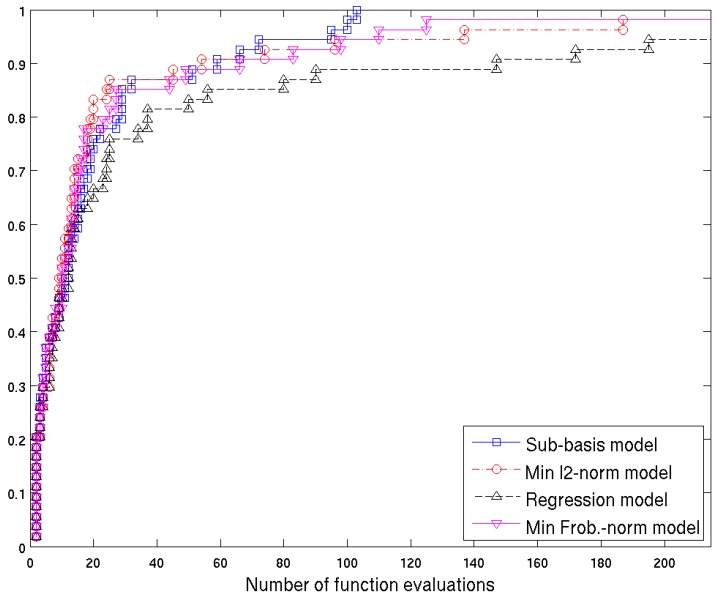


Figure 4.6: Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 200$)

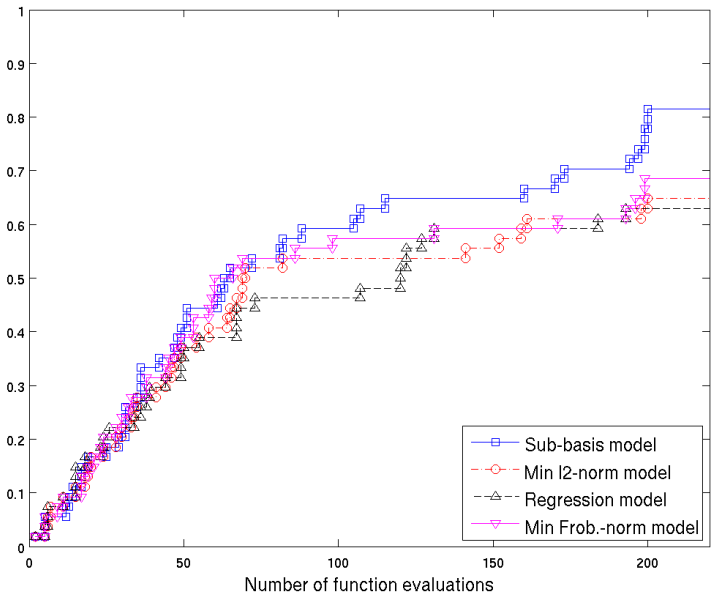


Figure 4.7: Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 200$)

for low and high required accuracy where it even manages to solve all test problems in 100 function evaluations for low required accuracy (within 90% of the best possible reduction achieved by any solver after 200 function evaluations). Regarding only the area of a very low computational budget of less than 30 function evaluations in the case of low required accuracy (Figure 4.6), using the two Minimum norm models seems to be interesting. Using the regression model seems to be not advantageous as only 95% of the test problems can be solved after 200 function evaluations in 90% of the best possible reduction. In the case of a higher required accuracy, the robustness of the solver is much less but the sub-basis model option shows again the best performance on the set of problems. We have a different picture when allowing for a very large computational budget of 15000 function evaluations (see Figures 4.8 and 4.9). For low required accuracy, the 90% reduction in the function value, compared to the best possible reduction, could be achieved after a maximum of 540 function evaluations by each model option, except for the regression model, which could finally not solve the problem KOWOSB in 15000 function evaluations. But regarding high required accuracy in Figure 4.9, show a clear advantage of the regression model. It solves 87% of the problems in the given limit and it shows the best performance if a computational budget of more than 500 function evaluations is available.

The detailed results are contained in Table B.8 and Table B.9 for the testing with a computational budget of 200 function evaluations and for low and high required accuracy, respectively. Table B.10 and Table B.11 contain the test results for the computational budget of 15000 function evaluations.

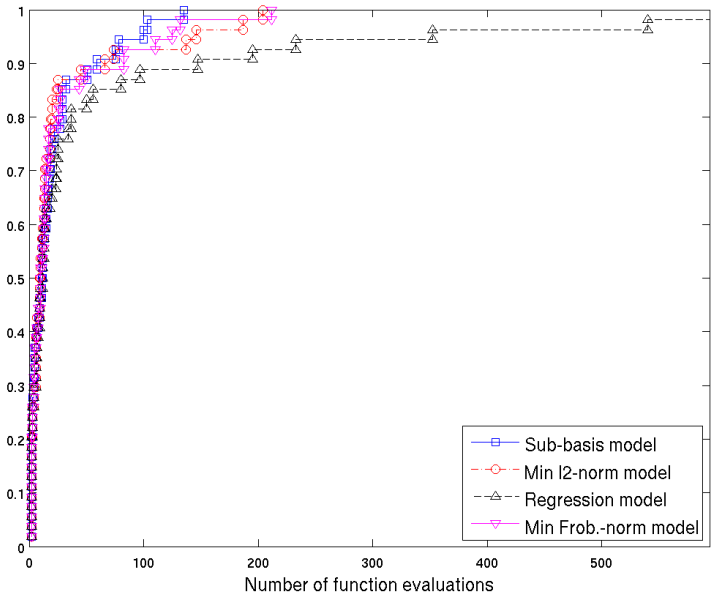


Figure 4.8: Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 15000$)

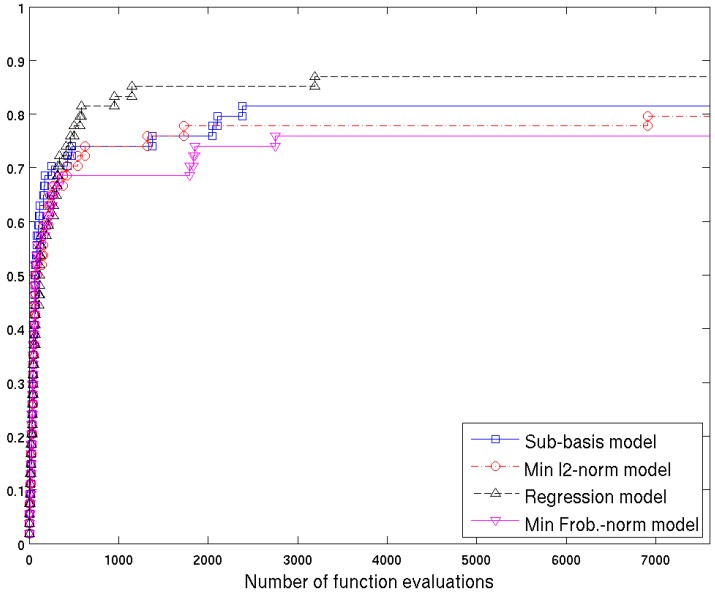


Figure 4.9: Comparison different models on unconstrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 15000$)

4.3.5 Comparison of different model types on bound-constrained problems

Also for the bound-constrained problems, we present the data profiles for $\tau = 10^{-k}$ with $k = \{1, 5\}$ and for maximum evaluation numbers of 200 and 15000 because we are interested in the short- and long-term behaviour of the different model-options as the required accuracy level changes. We can not draw any conclusions when only 200 function evaluations are allowed and low accuracy is required (Figure 4.10). None of the model options can outperform the other options whereas we have the impression that the two minimum-norm models have a slight advantage over the other two when regarding the computational budget between 15 and 75 evaluations. This is different when requiring high accuracy (Figure 4.11). Here, the minimum ℓ_2-norm model performs significantly better than the others when allowing for a computational budget of more than 70 function evaluations. The minimum Frobenius-norm approach performs worst and solves only 58% of the test problems in 99.99% of the best possible function value reduction (what corresponds to $\tau = 10^{-5}$) achieved by one of the other model options after 200 function evaluations.

In Figures 4.12 and 4.13, we can see the comparison when allowing for a large computational budget of 15000 function evaluations. The pictures are close to the ones of the comparison on unconstrained test problems but here the regression model perform a bit better than the other models beginning from a budget of 250 evaluations. It reaches a reduction of 90% of the best possible reduction (achieved by any model option in 15000 function evaluations) for all test

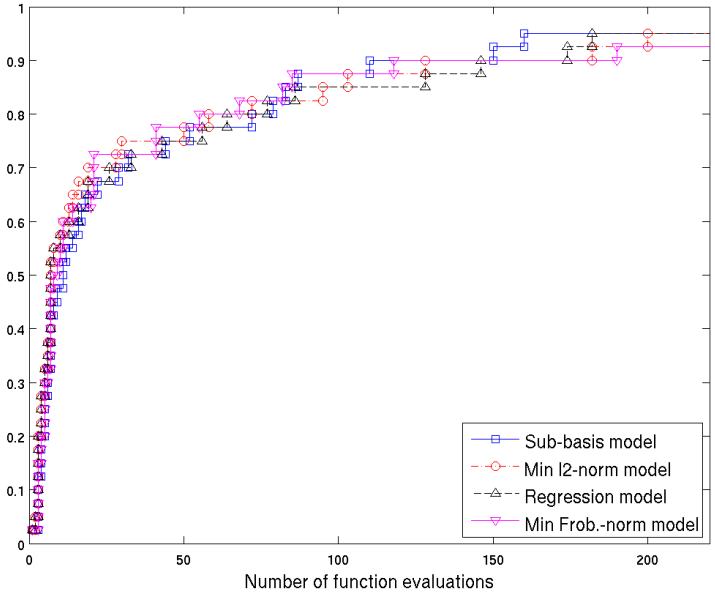


Figure 4.10: Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 200$)

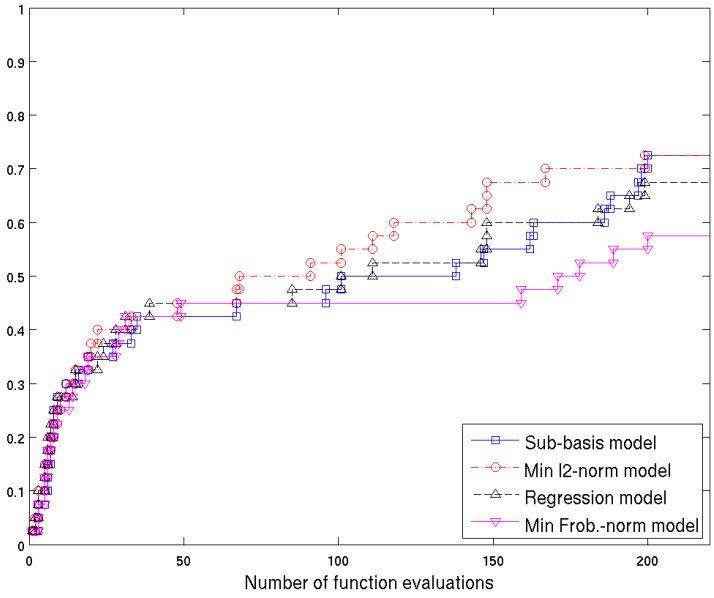


Figure 4.11: Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 200$)

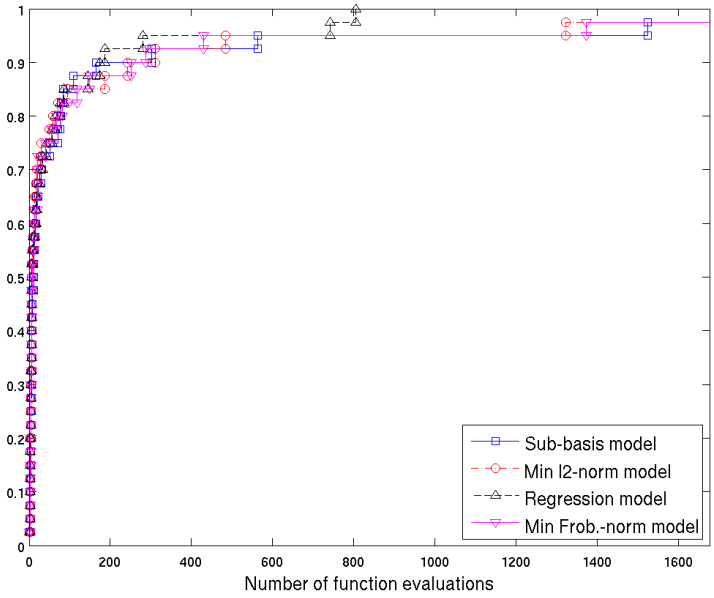


Figure 4.12: Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-1}$ and $maxeval = 15000$)

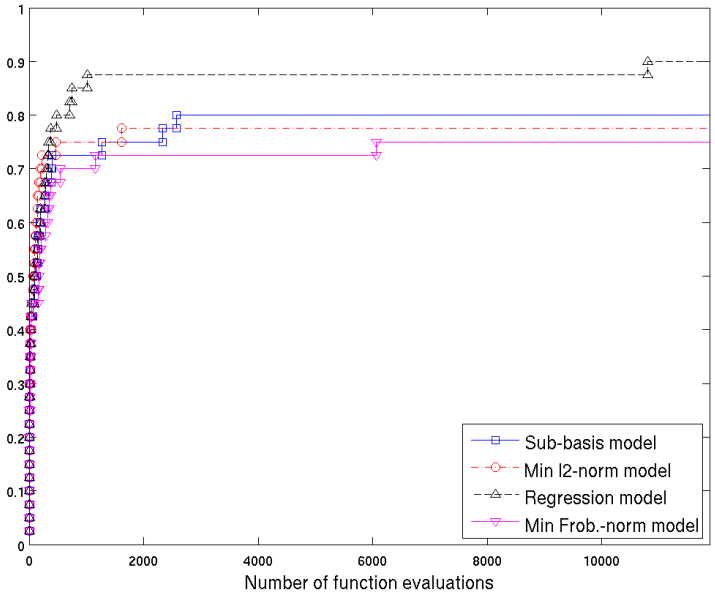


Figure 4.13: Comparison different models on bound-constrained noisy CUTEr problems ($\tau = 10^{-5}$ and $maxeval = 15000$)

problems in 800 function evaluations. After 1600 function evaluations all solver options reach 90% reduction of the best possible reduction. When requiring a reduction of 99.99% of the best reduction achieved (in Figure 4.13), the robustness is clearly worse. But the regression model manages again to outperform the other model options in terms of robustness when a large computational budget is given. And as before in Figure 4.9, we see that the most high-accuracy solutions for a smaller budget are obtained by the minimum ℓ_2-norm model before the regression model takes over at a computational budget of 330 function evaluations.

The detailed results are contained in Table B.12 and Table B.13 for the testing with a computational budget of 200 function evaluations and for low and high required accuracy, respectively. Table B.14 and Table B.15 contain the test results for the computational budget of 15000 function evaluations.

4.4 Numerical experiments on an aerodynamical application

4.4.1 Stopping criterion using noise estimation

When working with real-life applications incorporating expensive and noisy function evaluations, it is of high interest to apply a suitable stopping criterion so that no function evaluations are wasted by searching a solution "in the noise". In fact, if the termination parameter ϵ is prop-

erly set ($\epsilon \approx \sqrt{noisef}$, square root of the noise level in the function), our algorithm ***BCDFO+*** may declare convergence by using its default stopping criterion (a small model gradient in a Λ-poised set). But $noisef$, the exact noise level in the function to minimize, may be unknown beforehand which means that a good choice of ϵ is difficult to make.

To provide the user with an algorithm which is also self-contained when used for noisy optimization problems, we had to adapt the stopping criterion to the noisy case. This is realized by providing two additional parameters to the user: the first one is a binary parameter, where $noisy$="yes" or $noisy$="no" are the possible values while with the second one, the user can provide the function noise level if it is known, otherwise $noisef = 0$ and the algorithm will estimate the noise level during the minimization process in the case the problem was specified by the user as noisy.

In the case when the user defines the problem as noisy and provides the noise level of the function to minimize, the algorithm will use this value in setting the termination thresholds $\epsilon = \sqrt{noisef}$ for the standard stopping criterion and $\Delta_{\min} = \sqrt{noisef}$ for the alternative stopping criterion (see Section 3.3.7). Furthermore, we apply an additional criterion for termination in the noisy case. Here, we compute at each iteration the difference in the function values in the current interpolation set. Thus, we also terminate if $noisef \geq \kappa_{nz}(\max\{f(\mathcal{Y}_k)\} - \min\{f(\mathcal{Y}_k)\})$ with $0 < \kappa_{nz} \leq 1$ holds, which indicates that the solver is already "in the noise".

However, we are also prepared for the case the user is not able to provide an estimation of the function noise level. In this case, the thresholds for termination ϵ and $\Delta_{\min}$ are assumed not to be valid and are approximated by estimating the noise level $noisef$ at some point during the minimization process. The situations which may happen are the following.

The threshold ϵ or $\Delta_{\min}$ is too big. Here, one of the user-defined termination thresholds ϵ or $\Delta_{\min}$ are met and the algorithm would terminate prematurely. But, as we do not trust these thresholds to be appropriate in the noisy case when no noise level is given, we have to verify termination by estimating the noise level. Once $noisef$ is computed we may terminate the run successfully or continue minimization applying the strategy described above when the function noise level is known.

The thresholds ϵ and $\Delta_{\min}$ are too small. In this situation, the algorithm has converged (to a certain accuracy allowed by the noise level) but is not able to terminate. Hence, the solver is proceeding minimization in the noise. What are the possible indicators that the algorithm is in the noise?

1. A second improving step is initiated at the same point.
2. An already explored subspace is tried to enter again (in the bound-constrained case).

In both situations, the strategy for smooth functions is to recompute a poised interpolation set in a reduced trust-region radius. This strategy is not appropriate in the noisy case. Instead, the proper noise level is estimated and we test for successful termination for one of the three stopping criteria mentioned above.

4.4.2 Reference optimizer (DOT-BFGS)

The method currently used to solve aerodynamical design problems at Airbus is a classical quasi-Newton method with BFGS-update. The reference optimizer is from the DOT (Design Optimization Tools) [134] library which is implemented in the optimization suite OPTaliA. This solver uses an adjoint state gradient developed at Airbus. As other gradient-based optimizers, it converges along a descent path until no improvement is achieved during one optimizer iteration or if the norm of the gradient, or projected gradient in the bound-constrained case, is zero. First, the algorithm determines a descent direction, d_k, using the evolution of the gradient vector during the last two iterations. Once the search direction is computed, a linear search aiming at determining the stepsize giving the best improvement is performed. The linear search is driven by a one-dimensional polynomial interpolation and requires successive function evaluations. This type of optimizer is intrinsically sequential as it iteratively follows a single descent path.

In terms of quantity of information, the quasi-Newton gradient algorithm proposes the next set of variables by using only the information about the current internal iteration. The internal iteration contains information about the descent direction (computed using evolution of two gradient values) plus some function evaluations (usually no more than ten). This optimization algorithm proposes a new shape based on $N = 2n + 10$ evaluations of the objective function. Even if the approximated Hessian matrix, H_k, is more and more accurate as the number of internal iterations increases, the algorithm does not retain all the information known about the function but focuses on the information in the vicinity of the current shape.

4.4.3 Test case: Airfoil drag minimization

In our experiments, we attempt to optimize the 2-dimensional RAE2822 airfoil using the Navier-Stokes formulation. We consider the flight situation at Mach number $M = 0.78$ and an angle of attack $AoA = 0.6$. The chord length is one meter and the Reynolds number value is $Re = 6.5 \cdot 10^6$. The C-mesh is formed of 73×458 nodes with its boundary layer refinement. The restitution time for one flow simulation is about 1200 seconds to perform 700 steady iterations. The objective function to minimize is the above mentioned far-field pressure drag Cd_p.

One test case considers one bump ($n = 3$) which is applied to deform the upper surface in the direction of the vertical axis. Only positive deformations are authorized and the maximum amplitude of this bump is 10 millimeters. The position of the bump can vary inside the bounds $p = [0.1, 0.9]$ because the length of a wing is normalized to be 1 meter. In the other test case, three bumps ($n = 9$) are applied on the upper surface where the position-variable has different bounds for each of the bump ($p_1 = [0.1, 0.5], p_2 = [0.25, 0.75], p_3 = [0.5, 0.9]$). The bounds on the expansion-variables are in both cases and for all bumps $\beta = [2.0, 7.0]$.

4.4.4 Numerical results

First, we want to present the comparison of our algorithm with different model options to the reference optimizer DOT on the 3-dimensional test case. The convergence histories are depicted in Figure 4.14. We see that the regression model has the best reduction in the objective function

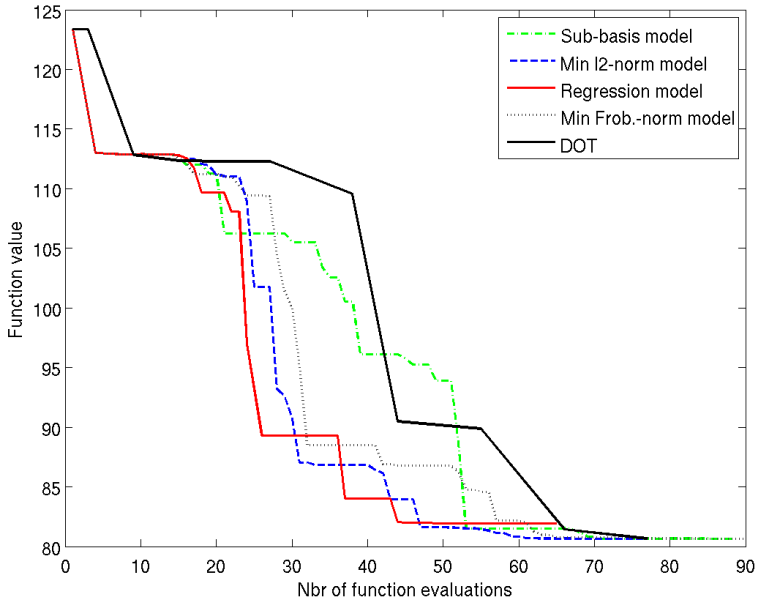


Figure 4.14: Convergence histories on 3-dim. aerodynamic test case

value in the beginning of the minimization process but that it slows down at a certain point and does even converge to another local minimum with a worse function value than all other considered solvers. The Minimum ℓ_2 norm shows the best performance on this test case as it reaches the best reduction in the function value before the other solvers (or model options). Furthermore, Figure 4.14 shows that although the number of function evaluations, needed by DOT and the minimum ℓ_2-norm model to converge, is the same ($neval = 77$), the reduction in the objective function starts much earlier using ***BCDFO+***.

Let's now turn to the test case where nine design variables are considered. In Figure 4.15, we can see a different behaviour of the methods on this test case. The reference solver DOT suggests a fast solution at $f = 92.09$ after only 62 function evaluations but this is obviously far from being the global minimum. Moreover, we do not even know whether the proposed solution is a local minimum as DOT terminates when the function value could not be reduced over two iterations which may also be caused by the inexact gradient or temporary stagnation and not due to criticality at the current iterate. However, most of model-based derivative-free optimization algorithms including ours are only aiming to find a local minimium of the problem but they seem to be able to detect more global optimal points, regardless of the model option

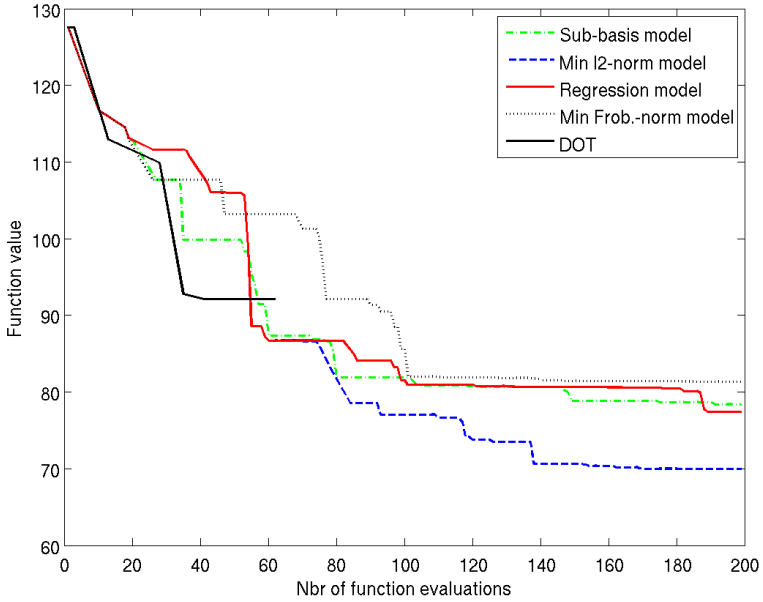


Figure 4.15: Convergence histories on 9-dim. aerodynamic test case

used. This advantage is likely to be due to the fact that such methods always use information from a broader neighbourhood around the current iterate. Nevertheless, such a behaviour is not guaranteed for other cases. Furthermore, we can see that none of the ***BCDFO+*** model-options has converged in the limit of 200 function evaluations whereas again the minimum ℓ_2-norm model performs best amongst them as it terminates with $f = 69.96$. For the others we have that the sub-basis model could reduce the function value to $f = 78.39$, the regression model reaches $f = 77.47$ and using the minimium Frobenius-norm model, the algorithm terminates at $f = 81.43$ after reaching the limit of 200 function evaluations.

This leads to the conclusion that a minimum ℓ_2-norm model embedded in our presented algorithm may be most suitable to solve noisy problems in general and in particular the aerodynamical application considered here in a limited computational budget.

4.5 Modelling the allowed gradient noise in a gradient-based line search method

Surprisingly to us, we observed that gradient-based line search methods, like quasi-Newton methods, are working quite well for many problems even in the presence of relatively high amplitude noise in the gradient.

In this section, we want to shed light on why this is the case and we attempt to model the gradient noise which is allowed by such a method. In Section 4.5.1, we will present some

properties and assumptions on the amplitude of the noise which ensure a descent direction of the algorithm. In Section 4.5.2, we want to establish a bound on the noise in the gradient which still gives rise to a globally convergent algorithm.

For this study, we assume exact function values and noisy gradient values.

4.5.1 Getting a descent direction in the presence of a noisy gradient

4.5.1.1 Deterministic properties

Remember, that if condition

$$g_k^T d_k < 0 \tag{4.14}$$

holds, the search direction d_k is a descent direction.

Steepest descent method

Following (4.14), we have for the steepest descent direction $d_k = -\tilde{g}_k$, where $\tilde{g}_k$ is the inexact gradient, that if condition

$$-g_k^T \tilde{g}_k < 0 \tag{4.15}$$

holds, the negative gradient direction is a descent direction even in the presence of noise. Furthermore, we can write $\tilde{g}_k = g_k + \Delta g_k$, where Δg_k denotes the level of noise in the gradient. From this we get the following property.

Property 4.1. *Assuming that the norm of the noise in the gradient is smaller than the gradient norm*

$$\|\Delta g_k\|_2 < \|g_k\|_2, \tag{4.16}$$

we have that

$$-g_k^T(g_k + \Delta g_k) < 0, \tag{4.17}$$

and the negative noisy gradient will provide a descent direction.

In the interest of readability we will drop the iteration count k in the following proofs.

Proof. We know that

$$\|g\|_2 \|\Delta g\|_2 \geq |g^T \Delta g| \geq -g^T \Delta g \tag{4.18}$$

and that

$$\|g\|_2^2 = g^T g. \tag{4.19}$$

By combining (4.18) and (4.19) with (4.16) multiplied by $\|g\|_2$, we obtain

$$-g^T \Delta g < g^T g$$

and then

$$-g^T(g + \Delta g) < 0$$

which implies that $-g^T \tilde{g} < 0$. □

Quasi-Newton method

Applying (4.14) to a quasi-Newton direction $d_k = -H_k^{-1}\tilde{g}_k$ involving a noisy gradient gives the condition

$$-g_k^T H_k^{-1}\tilde{g}_k < 0. \tag{4.20}$$

In this case, it is possible to derive the following property.

Property 4.2. *Assuming that the matrices H_k are positive definite with bounded condition numbers κ and the noise satisfies*

$$\|\Delta g_k\|_2 < \frac{1}{\sqrt{\kappa}}\|g_k\|_2, \tag{4.21}$$

then we have

$$-g_k^T H_k^{-1}\tilde{g}_k < 0,$$

hence, the direction $-H_k^{-1}\tilde{g}_k$ is a descent direction.

Proof. We have for symmetric positive definite matrices that

$$\sqrt{\kappa} = \kappa(H^{1/2}) = \|H^{1/2}\|_2\|H^{-1/2}\|_2 = \frac{\sigma_{max}(H^{-1/2})}{\sigma_{min}(H^{-1/2})} \geq 1.$$

Combining this with (4.21), we obtain

$$\|\Delta g\|_2 \quad \leq \quad \frac{\sigma_{min}(H^{-1/2})}{\sigma_{max}(H^{-1/2})}\|g\|_2.$$

We know as well that

$$\sigma_{max}(H^{-1/2}) = \|H^{-1/2}\|_2 \;\text{ and }\; \sigma_{min}(H^{-1/2})\|g\|_2 \leq \|H^{-1/2}g\|_2.$$

Therefore, we get

$$\|H^{-1/2}\|_2\|\Delta g\|_2 \quad \leq \quad \|H^{-1/2}g\|_2.$$

Using now that

$$\|H^{-1/2}\Delta g\|_2 \leq \|H^{-1/2}\|_2\|\Delta g\|_2,$$

we obtain

$$\|H^{-1/2}\Delta g\|_2 \leq \|H^{-1/2}g\|_2$$

which gives in turn

$$\|\Delta\hat{g}\|_2 \leq \|\hat{g}\|_2$$

when using the substitutions $\hat{g} = H^{-1/2}g$ and $\Delta\hat{g} = H^{-1/2}\Delta g$. When applying Property 4.1, we have that condition

$$\hat{g}^T(\hat{g} + \Delta\hat{g}) > 0$$

holds. From this we get by re-substitution that

$$g^T H^{-1}(g + \Delta g) > 0$$

what completes the proof. □

After having proved these properties and having observed that an upper bound on the noise is obviously depending on the condition number of the involved Hessian matrix of the optimization problem, we had a closer look at these properties and performed a small test. In fact, we checked the sign of the supposed descent direction (4.20) for different amplitude of Δg and we observed that condition (4.20), of course, always held for Δg smaller than the bound from (4.21) but surprisingly it also held very often for Δg bigger than the bound from (4.21). From this, we got the impression that (4.21) might be too stringent to be an upper bound on the noise what means that Property 4.1 and Property 4.2 are certainly true but they can only be considered as sufficient conditions and should not be regarded as tight upper bounds to ensure a descent direction.

To illustrate this issue on a small test example (Powell 2D problem with $x^* = (1.1 \cdot 10^{-5}, 9.1)$ and a gradient noise level of $noiseg = 10^{-3}$), we assume for the moment that Property 4.2 is a tight upper bound, which means that no noise would be allowed for an ill-conditioned problem and each quasi-Newton method would break down in a noisy situation.

Believing this, it seems natural to regularize the problem whenever the condition number is too big to allow for a higher amplitude noise in the problem. Applying this strategy to an existing line search BFGS method gave interesting results, as it can be seen in Figure 4.16. We implemented an automatic regularization technique which checks at every iteration whether

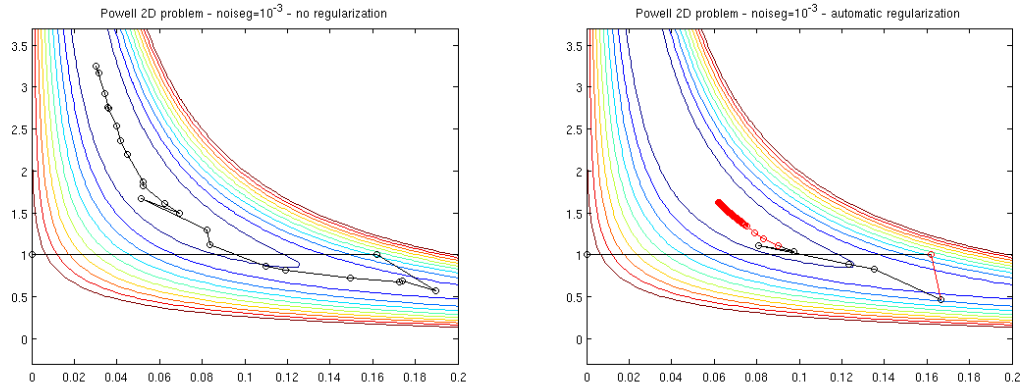


Figure 4.16: Regularization for Powell 2D problem with $noiseg = 10^{-3}$

Property 4.2 holds or not. If it is not satisfied for the given noise level, the condition number is decreased by applying $H_k = H_k + \lambda I$ for an increasing λ until Property 4.2 holds. Such a regularization step is displayed in red on the right-hand side figure. The left-hand side figure shows the minimization of the problem without taking care of the gradient noise.

As it can be seen in Figure 4.16, the results are not as one could expect after having proved Property 4.2. In fact, the regularization had even a negative effect on the convergence path because steps become close to steepest descent steps and get very short due to the conditioning of the matrix.

From this, we get that Property 4.2 is likely to be over stringent. To observe instead how much noise can be afforded in average while condition (4.20) holds, we decided to apply another model for the noise.

4.5.1.2 Statistical approach

We assume a Gaussian nature of the noise with $\Delta g \sim \mathcal{N}(0, \sigma^2 I)$ which is a normal distribution with a mean of zero and a standard deviation of σ. With this statistical approach we want to find out, up to which σ there exists a high probability to get a descent direction. For convenience, we restate here the condition for a quasi-Newton direction with an inexact gradient to be a descent direction

$$- g_k^T H_k^{-1} \tilde{g}_k < 0. \tag{4.22}$$

Now the question is, for which standard deviation σ does (4.22) hold for a given probability? In other words, how big can the noise become in average such that (4.22) is still very likely to hold with a high percentage?

As we know the distribution of Δg, we rewrite (4.22) as

Assumption 1:

$$g^T H^{-1} \Delta g \geq -g^T H^{-1} g, \tag{4.23}$$

where the left hand side is normal distributed with $g^T H^{-1} \Delta g \sim \mathcal{N}(0, \sigma^2 \|H^{-1} g\|_2^2)$. Now we look for the σ up to which Assumption 1 holds with a given probability. From (4.23) we get

$$\mathrm{P}[\text{Assumption 1 holds}] = \frac{2}{\sqrt{\pi}} \int_{-\frac{g^T H^{-1} g}{\sqrt{2}\sigma \|H^{-1} g\|_2}}^{+\infty} e^{-t^2}\, dt = \mathrm{P\_Ass1}. \tag{4.24}$$

This can be expressed in terms of the complementary Gauss error function erfc as

$$\frac{1}{2} \operatorname{erfc}\left[\frac{1}{\sqrt{2}\sigma} \left(- \frac{g^T H^{-1} g}{\|H^{-1} g\|_2} \right) \right] = \mathrm{P\_Ass1}, \tag{4.25}$$

and furthermore, in terms of the inverse complementary Gauss error function erfcinv we get

$$\operatorname{erfcinv}(2\,\mathrm{P\_Ass1}) = \frac{1}{\sqrt{2}\sigma} \left(- \frac{g^T H^{-1} g}{\|H^{-1} g\|_2} \right). \tag{4.26}$$

To see how much noise we can afford and still get a descent direction with a given probability, we extract σ from (4.26) and get

$$\sigma = \frac{1}{\sqrt{2}\, \operatorname{erfcinv}(2\,\mathrm{P\_Ass1})} \left(- \frac{g^T H^{-1} g}{\|H^{-1} g\|_2} \right). \tag{4.27}$$

From this, one could expect that the allowed noise level σ depends on the conditioning of the matrix H, the amplitude of the gradient and of course on the demanded probability. To analyze this, we present a numerical illustration in the following section.

4.5.1.3 Numerical illustration

Here, we want to exercise the results of our analysis applied on a small academic test problem (dimension $n = 10$). In the figures, the allowed noise level is depicted in dependency of the conditioning of the problem. In other words, it is shown which noise can be afforded when the problem becomes more ill-conditioned up to a condition number of $\kappa(H) = 10^{10}$.

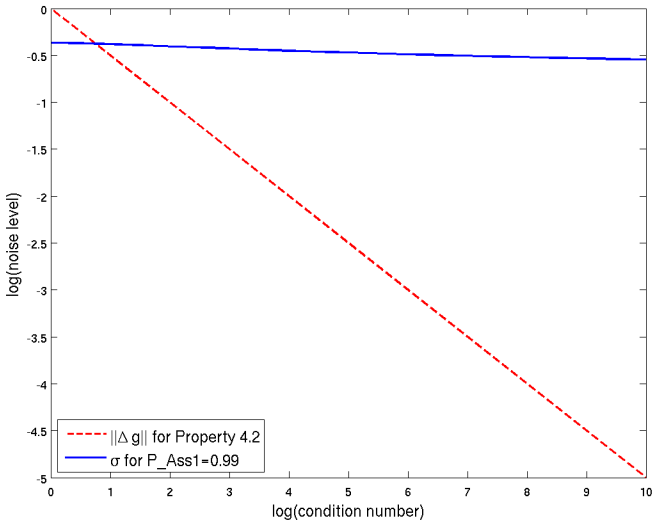


Figure 4.17: Allowed noise level for Property 4.2 and P_Ass1=99%

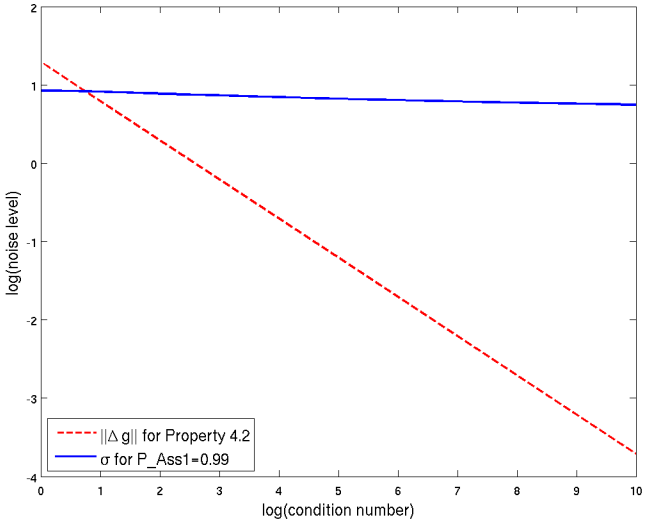


Figure 4.18: Allowed noise level for Property 4.2 and P_Ass1=99% for bigger gradient norm

In each of the Figures 4.17, 4.18 and 4.19, we see the inversely proportional connection between the condition number of H and the allowed noise Δg such that Property 4.2 holds (dashed line). The plain line shows the behaviour of the standard deviation σ of the supposed Gaussian gradient noise Δg such that Assumption 1 holds with 99%.

In Figure 4.18, we see the difference of a gradient with a norm bigger than one ($\|g\|_\infty = 10$) where in Figure 4.17, the gradient is normalized. It is easy to see that a larger gradient norm allows for a larger noise. Hence, the tolerated noise depends on the amplitude of the gradient norm and this expectation from Section 4.5.1.1 turned out to be right.

In Figure 4.19, Assumption 1 is tightened to hold with a probability of 99.99%. As expected, the curves show that the noise has to be smaller than that one from Figure 4.17 with 99% although we also notice that the impact of the gradient norm in Figure 4.18 is more important. Finally, it can be seen in all figures that the dependency on the conditioning of the problem is very marginal in the statistical approach.

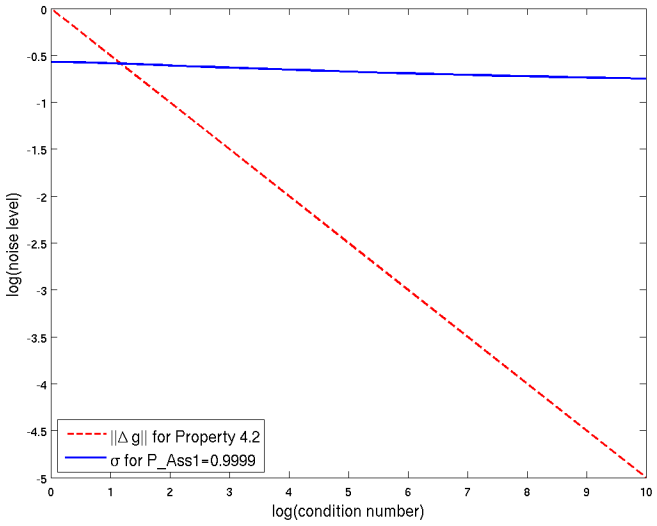


Figure 4.19: Allowed noise level for Property 4.2 and P_Ass1=99.99%

4.5.2 Global convergence in the presence of a noisy gradient

In this section, we want to extend our results from the previous section. We want to establish an upper bound on the gradient noise level to satisfy the condition for a globally convergent line search algorithm in the presence of an inexact gradient and an exact function. We again want to exercise and compare deterministic and statistical approaches.

4.5.2.1 Deterministic properties

The condition for a globally convergent line search algorithm is known from Zoutendijk [143] and writes

$$\cos\varphi_k \geq \delta > 0, \quad \text{for all} k, \tag{4.28}$$

where δ is a positive constant and $\cos\varphi_k$ is the cosine of the angle between d_k and the steepest descent direction $-g_k$, defined by

$$\cos\varphi_k = \frac{-g_k^T d_k}{\|g_k\|\|d_k\|}. \tag{4.29}$$

Steepest descent method

For a steepest descent method using an inexact gradient, condition (4.28) reads

$$\frac{g_k^T \tilde{g}_k}{\|g_k\|_2\|\tilde{g}_k\|_2} \geq \delta, \tag{4.30}$$

where $\tilde{g_k}$ denotes the noisy gradient $\tilde{g_k} = g_k + \Delta g_k$.

Property 4.3. *Assuming that the norm of the gradient noise satisfies*

$$\|\Delta g_k\|_2 \leq \frac{1-\delta}{1+\delta}\|g_k\|_2, \tag{4.31}$$

the condition (4.30) will hold. Hence, the negative noisy gradient will provide a globally convergent algorithm.

Proof. We know that

$$\|g\|_2\|\Delta g\|_2 \geq |g^T \Delta g| \geq -g^T \Delta g.$$

By combining these with (4.31) multiplied by $\|g\|_2$, we obtain

$$-g^T\Delta g + \delta\|g\|_2\|\Delta g\|_2 \quad \leq \quad \|g\|_2^2 - \delta\|g\|_2^2$$

and then

$$\delta\|g\|_2^2 + \delta\|g\|_2\|\Delta g\|_2 \quad \leq \quad \|g\|_2^2 + g^T\Delta g.$$

Using now that $\|g + \Delta g\|_2 \leq \|g\|_2 + \|\Delta g\|_2$ and that $\|g\|_2^2 = g^T g$, we get

$$\delta\|g\|_2\|g + \Delta g\|_2 \leq g^T g + g^T \Delta g,$$

which gives that $\cos\varphi \geq \delta > 0$. □

Quasi-Newton method

For a quasi-Newton method using a noisy gradient, condition (4.28) writes

$$\frac{g_k^T H_k^{-1} \tilde{g}_k}{\|g_k\|_2\|H_k^{-1}\tilde{g}_k\|_2} \geq \delta, \tag{4.32}$$

where $\tilde{g}_k$ denotes again the noisy gradient $\tilde{g}_k = g_k + \Delta g_k$. In this case, it is possible to derive the following property.

Property 4.4. *Assuming that the matrices H_k are positive definite with a uniformly bounded condition number κ, that $\delta \leq \frac{1}{\sqrt{\kappa}}$, and that the noise satisfies*

$$\|\Delta g_k\|_2 \leq \frac{1-\delta\sqrt{\kappa}}{1+\delta\sqrt{\kappa}} \frac{\|g_k\|_2}{\sqrt{\kappa}}, \tag{4.33}$$

the condition (4.32) holds and thus, the algorithm using the search direction $-H_k^{-1}\tilde{g}_k$ is globally convergent.

Proof. We know from Property 4.3 that

$$\text{if } \|\Delta\hat{g}\|_2 \leq \frac{1-\delta}{1+\delta}\|\hat{g}\|_2, \quad \text{then} \quad \frac{\hat{g}^T(\hat{g}+\Delta\hat{g})}{\|\hat{g}\|_2\|(\hat{g}+\Delta\hat{g})\|_2} \geq \delta.$$

If we substitute now $H^{-\frac{1}{2}}g = \hat{g}$ and $H^{-\frac{1}{2}}\Delta g = \Delta\hat{g}$, we get that

$$\text{if } \|H^{-\frac{1}{2}}\Delta g\|_2 \leq \frac{1-\delta}{1+\delta}\|H^{-\frac{1}{2}}g\|_2, \quad \text{then} \quad \frac{g^T H^{-1}(g+\Delta g)}{\|H^{-\frac{1}{2}}g\|_2\|H^{-\frac{1}{2}}(g+\Delta g)\|_2} \geq \delta.$$

The right hand side of this statement can be reformulated using the singular values of $H^{\frac{1}{2}}$ and $H^{-\frac{1}{2}}$, thus we use that $\sigma_{min}(H^{-\frac{1}{2}})\|g\| \leq \|H^{-\frac{1}{2}}g\|_2$, that $\sigma_{min}(H^{\frac{1}{2}})\|H^{-1}(g+\Delta g)\|_2$ and that $\sigma_{min}(H^{-\frac{1}{2}})\sigma_{min}(H^{\frac{1}{2}}) = 1/\kappa(H^{\frac{1}{2}})$. Furthermore, after dividing the complete if-statement by $\|H^{-\frac{1}{2}}\|_2$ and using that $\|H^{-\frac{1}{2}}\Delta g\|_2 \leq \|H^{-\frac{1}{2}}\|_2\|\Delta g\|_2$, we obtain that

$$\text{if } \|\Delta g\|_2 \leq \frac{1-\delta}{1+\delta}\frac{\|H^{-\frac{1}{2}}g\|_2}{\|H^{-\frac{1}{2}}\|_2}, \quad \text{then} \quad \frac{1}{1/\kappa(H^{\frac{1}{2}})}\frac{g^T H^{-1}(g+\Delta g)}{\|g\|_2\|H^{-1}(g+\Delta g)\|_2} \geq \delta,$$

which gives, using the definition of $\cos\varphi$ from (4.29), that

$$\text{if } \|\Delta g\|_2 \leq \frac{1-\delta}{1+\delta}\frac{\|H^{-\frac{1}{2}}g\|_2}{\|H^{-\frac{1}{2}}\|_2}, \quad \text{then} \quad \cos\varphi \geq \frac{\delta}{\kappa(H^{\frac{1}{2}})}.$$

We get by substituting $\delta' = \delta/\kappa(H^{\frac{1}{2}})$ that

$$\text{if } \|\Delta g\|_2 \leq \frac{1-\delta'\kappa(H^{\frac{1}{2}})}{1+\delta'\kappa(H^{\frac{1}{2}})}\frac{\|H^{-\frac{1}{2}}g\|_2}{\|H^{-\frac{1}{2}}\|_2}, \quad \text{then} \quad \cos\varphi \geq \delta'.$$

Expanding the right hand-term of the if-statement by $\|H^{\frac{1}{2}}\|_2$ and using then that $\|H^{\frac{1}{2}}\|_2\|H^{-\frac{1}{2}}g\|_2 \geq \|H^{\frac{1}{2}}H^{-\frac{1}{2}}g\|_2 = \|g\|_2$ and that $\|H^{\frac{1}{2}}\|_2\|H^{-\frac{1}{2}}\|_2 = \kappa(H^{\frac{1}{2}})$, we obtain that

$$\text{if } \|\Delta g\|_2 \leq \frac{1-\delta'\kappa(H^{\frac{1}{2}})}{1+\delta'\kappa(H^{\frac{1}{2}})}\frac{\|g\|_2}{\kappa(H^{\frac{1}{2}})}, \quad \text{then} \quad \cos\varphi \geq \delta'.$$

Writing now $\kappa(H^{\frac{1}{2}})$ as $\sqrt{\kappa}$ completes the proof. □

This property shows again a strong dependence between the tolerated noise and the condition number of the problem. But we did again a small testing which has shown that the dependence on the condition number is only marginal. Hence, we are suspecious that things similar to the previous section may happen and that the condition for a globally convergent algorithm may "in average" allow for higher amplitude noise even if the problem is not well-conditioned. We want again apply here the statistical model.

4.5.2.2 Statistical approach

We assume a Gaussian nature of the noise with $\Delta g \sim \mathcal{N}(0, \sigma^2 I)$ as in the previous section. This time we want to find out, up to which σ does (4.32) hold for a given probability? In other words, how big can the noise become in average such that the condition for global convergence of a line search method still holds with a high percentage?

As we know the distribution of Δg, we rewrite (4.32) as

$$g^T H^{-1} \Delta g \geq \delta \|g\|_2 \|H^{-1}(g + \Delta g)\|_2 - g^T H^{-1} g. \tag{4.34}$$

Furthermore, if we assume that $\|H^{-1}(g + \Delta g)\|_2$ is not far from $\|H^{-1} g\|_2$ (we will look at this later in Assumption 3), we can write (4.34) as

Assumption 2:

$$g^T H^{-1} \Delta g \geq \delta \|H^{-1} g\|_2 \|g\|_2 - g^T H^{-1} g \tag{4.35}$$

and the right hand side of the inequality is independent from the level of noise. From the left hand side we know that it is normal distributed with $g^T H^{-1} \Delta g \sim \mathcal{N}(0, \sigma^2 \|H^{-1} g\|_2^2)$. Now we look for σ up to which Assumption 2 holds with a given probability. From (4.23) we get

$$\mathrm{P}[\text{Assumption 2 holds}] = \frac{2}{\sqrt{\pi}} \int_{\frac{\delta \|g\|_2}{\sqrt{2}\sigma} - \frac{g^T H^{-1} g}{\sqrt{2}\sigma \|H^{-1} g\|_2}}^{+\infty} e^{-t^2} \, dt = \mathrm{P\_Ass2}. \tag{4.36}$$

This can be expressed in terms of the complementary Gauss error function erfc as

$$\frac{1}{2} \operatorname{erfc}\left[\frac{1}{\sqrt{2}\sigma} \left(\delta \|g\|_2 - \frac{g^T H^{-1} g}{\|H^{-1} g\|_2} \right) \right] = \mathrm{P\_Ass2}, \tag{4.37}$$

and furthermore, in terms of the inverse complementary Gauss error function erfcinv, we get

$$\operatorname{erfcinv}(2\,\mathrm{P\_Ass2}) = \frac{1}{\sqrt{2}\sigma} \left(\delta \|g\|_2 - \frac{g^T H^{-1} g}{\|H^{-1} g\|_2} \right). \tag{4.38}$$

To see now how much noise we can afford and still be globally convergent with a given probability, we have to extract σ from (4.38) what results in

$$\sigma = \frac{1}{\sqrt{2}} \left(\delta \|g\|_2 - \frac{g^T H^{-1} g}{\|H^{-1} g\|_2} \right) \frac{1}{\operatorname{erfcinv}(2\,\mathrm{P\_Ass2})}. \tag{4.39}$$

From this one can see that the Gaussian noise level σ depends on the conditioning of the problem, the amplitude of the gradient and of course on the demanded probability.

Now we have to examine the supplementary assumption which we imposed on (4.34) to get Assumption 2 in (4.35).

Assumption 3:

$$2\|H^{-1} g\|_2^2 \geq \|H^{-1}(g + \Delta g)\|_2^2 \geq \frac{1}{2} \|H^{-1} g\|_2^2 \tag{4.40}$$

In fact, we have to divide the two-sided condition (4.40) into two parts what gives

$$\mathrm{P}[\|H^{-1}(g+\Delta g)\|_2^2 \leq c_2] - \mathrm{P}[\|H^{-1}(g+\Delta g)\|_2^2 \leq c_1] = \mathrm{P\_Ass3} \tag{4.41}$$

where $c_2 = 2\|H^{-1}g\|_2^2$ and $c_1 = \frac{1}{2}\|H^{-1}g\|_2^2$. The next question is, up to which σ does Assumption 3 hold for a given probability?

Unfortunately, the distribution of $\|H^{-1}(g+\Delta g)\|_2^2$ cannot be easily computed. Therefore, it is worthwhile to get the right σ by simulation. This is done by imposing a first σ_0, e.g., the one we get from Assumption 2, on the noise Δg, generating 10^5 examples of the random variable $\|H^{-1}(g+\Delta g)\|_2^2$, and computing a CDF (cumulative distribution function) from which we get the probability that Assumption 3 holds. If this is lower than the desired probability, σ_0 is decreased and a new simulation is started.

4.5.2.3 Numerical illustration

The results of our analyses can be seen in the figures below. The allowed noise level is depicted in dependency of the condition of the problem. In other words, it is shown which noise can be afforded when the problem becomes more ill-conditioned up to a condition number of $\kappa(H) = 10^{10}$. In Figure 4.20, we see as a dashed line the inversely proportional relation between the

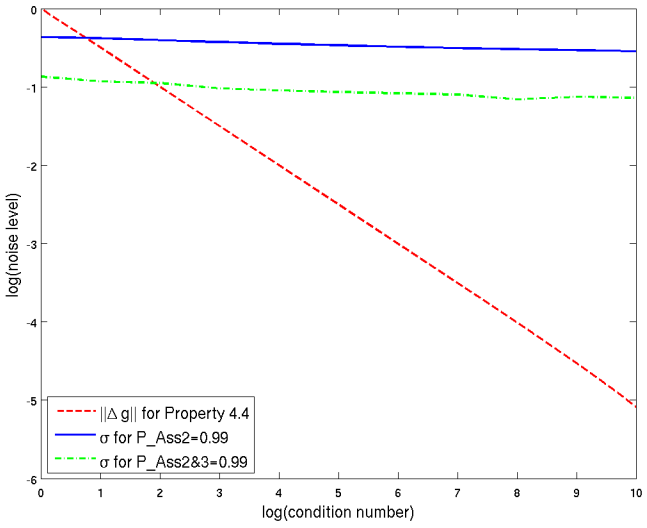


Figure 4.20: Allowed noise level for Property 4.4 and P_Ass2&3=99%

condition number of H and the allowed noise Δg such that Property 4.4 holds. The other curves show the behaviour of the standard deviation σ of the supposed Gaussian noise Δg. The plain curve shows σ up to which Assumption 2 holds with 99% and as dash-dot line, we have the σ up to which also Assumption 3 holds with 99%.

Furthermore, we can observe from this figure that Assumption 2 allows for similar noise levels as Assumption 1 and that Assumption 3 is stronger than both of them. But still, the

statistical approach allows for a higher amplitude of noise than the deterministic one and the dependency on the condition of the problem is not very big in average.

4.5.2.4 Numerical example of an aerodynamical test case

We finally illustrate our theory on some real data from a Navier-Stokes test case provided by Airbus. To do so, we sampled 100 function and gradient values of a one-dimensional function of the CDP position variable in the range of $(0.7, 0.9)$. The adjoint state gradient is displayed as a dotted line on the left hand side of Figure 4.21 and the gradient approximated by finite differences is displayed as a plain line. By visualizing a line at zero, the finite differences gradient

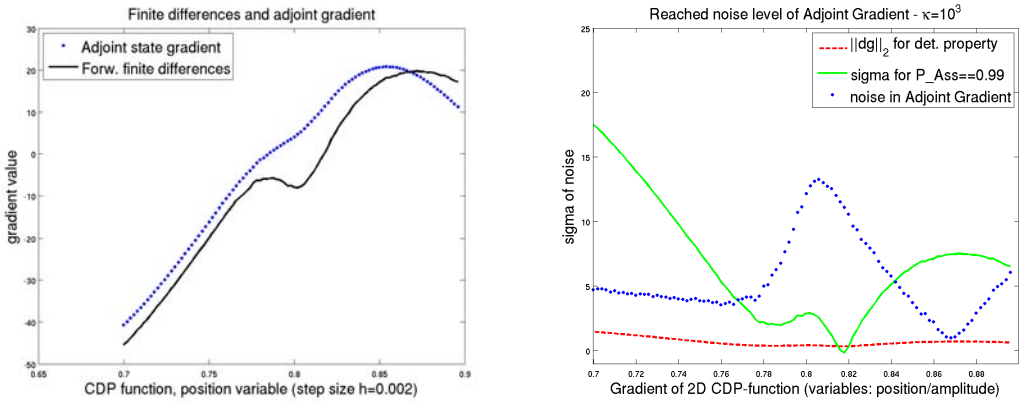


Figure 4.21: Example of CDP adjoint state gradient

which represents the function has a minimum close to 0.82 and the adjoint gradient points out a minimum close to 0.78. This picture shows a big error/noise in the gradient. Our theory should answer the question where to expect any difficulties while running a quasi-Newton method using such an inexact gradient.

On the right hand side of Figure 4.21, the reached noise level of the adjoint state gradient is depicted as a dotted line. To compute the noise, we took the difference between the adjoint gradient and the finite differences gradient from the left figure above. The second dimension for the problem is a constant which is added to be able to construct a problem matrix with a certain condition number. We applied $\kappa = 10^3$ in this example. The dashed line shows the allowed gradient noise for the constructed problem in terms of the deterministic Property 4.4 and the plain line displays the tolerated sigma such that the statistical Assumption 3 holds with 99 percent for this problem. So, if the values of the measured gradient noise are below one of these curves, we do not expect any difficulties for a gradient-based algorithm to converge. This gives a hypothetically safe region up to a value of 0.77 and beginning again from 0.84. The intervall $[0.77, 0.84]$ indicates the region where we expect the iterates to get into trouble.

These results answer our questions which arose in a past work where we compared different gradient-based optimization softwares on this Airbus CDP problem. The solutions returned by

the solvers for this one-dimensional problem ranged from $x^* = 0.7862$ to $x^* = 0.8180$ and the obtained function values differed from $f^* = 111.515$ to $f^* = 111.705$ what was somehow not expected at that time but can now be explained by our theory.

4.5.3 Conclusions

We addressed the question, how much noise in the gradient can a steepest descent method and a quasi-Newton method tolerate. We established two properties on the noise which ensure a descent direction using an inexact gradient in such methods. These properties show a strong dependence between the condition of the problem and the allowed noise. But, a quick check of the established condition with some random noise of different amplitude has shown that this property covers the worst case and seems to be over stringent in the average case.

So, we decided to assume a Gaussian nature of the noise with mean zero and variance σ^2 to see what noise can be afforded in average. We established an assumption which ensures a descent direction in a quasi-Newton method if it holds with a high probability.

Furthermore, we proved another deterministic property which guarantees global convergence of a quasi-Newton algorithm in the presence of a noisy gradient. Here also, we could establish two statistical assumptions which together ensure with a high probability global convergence of the algorithm using an inexact gradient. We simulated the distribution of the random number of Assumption 3 to get an allowed standard deviation σ of the noise up to which the two assumptions hold with a given probability (e.g. 99%).

In fact, we established two sufficient conditions on the global convergence of a quasi-Newton algorithm in presence of a noisy gradient. If the noise satisfies the condition stated in Property 4.4, the algorithm is certainly globally convergent. If the noise satisfies Assumption 2 and Assumption 3, which is very likely to happen up to a certain level of noise, the algorithm is globally convergent.

From our experiments we got confirmed that there is a relation between the amplitude of the gradient norm and the tolerated noise in the gradient. Whereas the dependency between the conditioning of the problem and the tolerated noise in the gradient, which was predicted by the deterministic property, turned out to be not very big in average. Thus, we can conclude that a quasi-Newton method provided with a noisy gradient is globally convergent up to a certain level of noise which is only marginally dependent on the condition of the approximated Hessian.

We could observe that applying Assumption 3 on an Airbus test problem gives us exactly the answer to our question, why different optimizers converged to different minima or even get into trouble while looking for a good solution.

For this study, we assumed exact function values and noisy gradient values. Extension to the more general situation where both function and gradient information are inexact will be considered in a future work.

4.6 Enhancing a DFO method by inexact gradient information in the context of global optimization

An important point in industrial optimization is to obtain a global optimum for approximate function and gradients in the minimal computational time. It is well known that this objective is not very easy to attain for problems of arbitrary size using generic algorithms, even when solving non-noisy problems. Indeed, algorithms which aim at finding a global optimum oftentimes take advantage of building meta-models and using DOE-techniques (Design Of Experiment) which are very costly in terms of function evaluation and therefore in computing time.

Our goal is therefore, viewed more realistically, to envisage this type of algorithm for degrees of freedom of modest dimension (less than 20 variables). The software SNOBFIT (Stable Noisy Optimization by Branch and FIT), as stated by its authors in [86], seems to answer the above specifications in an acceptable manner.

4.6.1 The algorithm SNOBFIT

SNOBFIT, developed by A. Neumaier and W. Huyer, is a software package for robust and fast solution of noisy optimization problems with continuous variables varying within bounds, possibly subject to additional soft constraints.

Objective function values must be provided by a file-based interface. Care is taken that the optimization proceeds reasonably even when the interface produces noisy or even occasionally undefined results (hidden constraints). The interface makes it possible to use SNOBFIT with new data entered by hand, or by any automatic or semiautomatic experimental system.

This makes SNOBFIT very suitable for applications to the selection of continuous parameter settings for simulations or experiments, performed with the goal of optimizing some user-specified criterion. Furthermore, the possibility of evaluating the objective at several points simultaneously by parallel function evaluations should be pointed out.

The method combines a branching strategy to enhance the chance of finding a global minimum with a sequential quadratic programming method based on fitted quadratic models to have good local properties. Various safeguards address many possible pitfalls that may arise in practical applications, for which most other optimization routines are ill-prepared [87].

The package solves the optimization problem

$$\min f(x) \quad \text{s.t. } x \in [u, v] \tag{4.42}$$

where an interval notation for boxes is used

$$[u, v] := \{x \in \mathbf{R}^n | u_i \leq x_i \leq v_i, i = 1, ..., n\}, \tag{4.43}$$

with $u, v \in \mathbf{R}^n$ and $u_i < v_i$ for $i = 1, ..., n$, i.e., $[u, v]$ is bounded with nonempty interior. A box $[u', v']$ with $[u', v'] \subseteq [u, v]$ is called a subbox of $[u, v]$.

Based on the already available function values, the algorithm builds internally, around each point, local models of the function to minimize, and returns at each step a number of points whose evaluation is likely to improve these models or is expected to give better function values. In case the total number of function evaluations is kept low, no guarantees can be given that a global minimum is located.

In each call to SNOBFIT, a possibly empty set of points and corresponding function values are put into the program, together with some other parameters. Then the algorithm generates the requested number of new evaluation points. These points and their function values should preferably be used as input for the following call to SNOBFIT. The suggested evaluation points belong to five classes indicating how such a point has been generated, in particular, whether it has been generated by a local or a global aspect of the algorithm. The points of class 1 to 3 represent the more local aspect of the algorithm and are generated with the aid of local linear or quadratic surrogate models created by linear least squares fits at positions where good function values are expected. The points of classes 4 and 5 represent the more global aspect of the algorithm, and are generated in large unexplored regions. These points are reported together with a model function value computed from the appropriate local model.

SNOBFIT generates at each iteration a user-defined number of new trial points of different character which recommend themselves for evaluation. The points are chosen from five different classes in proportions that are set by the user. This gives the user some control over the desired balance between local and global search. The fact that these selected points, given the opportunity of using parallel computing facilities, can be evaluated at the same time, should again be emphasized here.

All points for which the function has been evaluated are stored in the set X which is used to build the local models at each iteration. These local surrogate models are created around each point using $n + m$ points from X, where n is the dimension of the problem and m is a userdefined number. In the presence of noise, fitting reliable linear models near some point requires the use of a few more data points than parameters. To be able to build such a model, $n+m$ appropriate points have to be selected in a safeguarded manner from X. This is done by using the nearest neighbours of each point.

In step 4 of Algorithm 4.1, the safeguarded nearest neighbours of each point $x = (x_1, ..., x_n)^T \in X$ are determined as follows. Starting with an initially empty list of nearest neighbours, we pick, for each direction $i = 1, ..., n$, a point in X closest to but different from x among the points not yet in the list. This procedure ensures that, for each coordinate i, there is at least one point in the list of nearest neighbours whose i-th coordinate differs from that of x. This gives n points in the list and it is filled up by adding the points from X closest to x but not yet in the list, until it contains $n + m$ points. The list of nearest neighbours is updated for every point at every iteration if necessary.

In step 5 of the algorithm, a local quadratic fit is computed around the best point x^{best},

i.e., the point with the lowest objective function value. Here, the vector $g \in \mathbb{R}^n$ and the symmetric matrix $H \in R^{n\times n}$, gradient and Hessian respectively, are estimated using $K := \min(n(n+3), N-1)$ points which are closest to but distinct from x^{best}, where N denotes the total number of points at which the function values are already evaluated. If $N < \frac{1}{2}(n+1)(n+2)$, the minimum ℓ_2-norm solution of the interpolation conditions is computed. Given these estimations of the first and second derivatives at x^{best}, the potential new evaluation point w is obtained by approximately minimizing the local quadratic model around x^{best}. The minimization is performed with the bound constrained quadratic programming package MINQ [105], designed to find a stationary point, usually a local minimizer. If w is a point already in X, a new point is randomly generated inside a box determined by the farthest of the nearest neighbours of x^{best}.

Algorithm 4.1 SNOBFIT

1: Divide $[u, v]$ into subboxes, each containing one point if function values of some points already available. Choose m, the number of additional points/neighbours considered to build the models.
2: **repeat**
3: Split subboxes which contain more than one point.
4: Compute a vector with $n + m$ nearest neighbours for each point.
5: Determine trial point w by minimizing a quadratic model around the current best point x^{best}.
6: Determine trial points y_j by minimizing local models around $x_j, \forall x_j \in X$.
7: Determine trial points by dividing large subboxes (unexplored regions).
8: Determine trial points by space filling if necessary.
9: Evaluate new function values at suggested trial points.
10: **until** Best f value not improved in a certain number of iterations

In step 6, points of class 2 and 3 are determined by minimizing a local model around each point x_j. Points of class 2 and 3 are alternative good points. They represent another aspect of the local search part of the algorithm. Here, for each point the information on the function at the point itself and its $n + m$ nearest neighbours is used. But to get a more accurate model, a quadratic error term, which accounts for second (and higher) order deviation from linearity and for measurement errors, is considered when building the model. It is chosen such that for points with large expected uncertainties in the function value and for neighbours far away from x_j, a larger error in the model is permitted. A complete quadratic model would be too expensive to construct around each point. The gradient value at x_j is estimated by a linear least squares fit, using the function values of the $n + m$ nearest neighbours. The points y_j obtained by solving the N quadratic programs are potential new evaluation points of class 2 if the point x_j is called local, i.e., if its function value is significantly smaller than that of its nearest neighbours. Otherwise, the points y_j are taken to be in class 3. If y_j is a point already in X, a new point is randomly generated inside a box determined by the nearest of the nearest

neighbours of x_j.

The points of class 4 generated in step 7 of the algorithm are points in so far unexplored regions, i.e., they are generated in large subboxes of the current partition. They represent the most global aspect of the algorithm. To maintain an idea of the size of each box as long as the algorithm proceeds, its smallness is assigned to each subbox. It is 0 for the exploration box (u, v) and large for small boxes. For a more systematic global search, large boxes (= having a low smallness) should be prefered when selecting new points for evaluation.

Points of class 5 are only produced if the algorithm does not manage to reach the desired number of points by generating point of classes 1 to 4, which happens in particular in an initial call with an empty set of input points and function values. The points of class 5 are chosen from a set of random points such that they extend X to a space filling design. The algorithm of SNOBFIT is stated schematically as Algorithm 4.1, a more detailed version can be found in [86].

4.6.2 Modifications

Use of provided gradient information Our aim is to develop an improved version of the method described above and we will call this algorithm gbSNO (gradient-based Stable Noisy Optimization). To accelerate convergence speed of the original version, we will exploit the fact that gradient information is available in the OPTaliA suite at Airbus. As we know that the gradient provided comes from an expensive CFD-calculation and may be therefore noisy and certainly comprise an error, the implementation must be done very carefully. The gradient information will be incorporated in the algorithm to provide more accurate information when building up the two kinds of local models. This will be done in a way we attempt to eliminate the possible gradient error by use of the function values of the nearest neighbours around each point.

To determine roughly the quality of the gradient at x, a gradient ratio ρ is computed using the function value from the closest nearest neighbour of x. To get an idea of the gradient accuracy, we try to compare it with a finite difference type gradient. This results in the formula

$$\rho = \frac{f(x) - f(x - s) - \nabla f(x)^T s}{\|s\|}, \tag{4.44}$$

where s is the distance to the closest nearest neighbour of x. So, ρ will be large if the gradient is supposedly incorrect.

Let $x^k, k = 1, ..., n + m$, be the nearest neighbours of x, $f := f(x), f_k = f(x^k)$, and Q_k the error term which accounts for uncertainties in the function values. Furthermore, we want to introduce one additional equation carrying the information of the noisy gradient $g_{noisy} :=$

$g_{noisy}(x)$. Then we consider the equations

$$f_k - f = g^T(x^k - x) + \epsilon_k Q_k, k = 1, ..., n+m, \tag{4.45}$$

$$g_{noisy} - g = \epsilon\rho, \tag{4.46}$$

where ϵ_k are the model errors to be minimized and g is the gradient to be estimated. The new least squares problem to minimize is now written

$$\min_g \left(\sum_{k=1}^{n+m} \left(\frac{f_k - f}{Q_k} - \frac{g^T(x^k - x)}{Q_k} \right)^2 + \left(\frac{g_{noisy} - g}{\rho} \right)^2 \right), \tag{4.47}$$

where the last term represents the penalization we added onto the system to improve the quality of the estimated gradient g. High uncertainties in function and gradient values are reflected in large values of Q_k and ρ, respectively. In this case, a larger error is permitted in the fit.

The gradient g is approximated for each point and is then used in building the local model around the corresponding point. The approximate gradient at x^{best} is also used to build the quadratic model around the current best point.

Use of a BFGS-formula As we are now quite sure to have estimated a more reliable gradient for each x, we do not want to use a Hessian which is estimated by a quadratic fit as it is done in the original version of SNOBFIT. Instead, we decided to use a Hessian approximation in the quadratic model which is updated every iteration by the means of a BFGS-formula where the estimated gradient from (4.47) is used.

4.6.3 Experiments on academic and aerodynamic test problems

From the numerical experiments we have done, we want to detail two problems, one academic and the other one a real life problem provided by Airbus.

Academic test problem The minimal surface area problem is an unconstrained optimization problem. Finding a minimal surface of a boundary with specified constraints is a problem in the calculus of variations and is sometimes known as Plateau's problem. Minimal surfaces may also be characterized as surfaces of minimal surface area for given boundary conditions. As SNOBFIT and also gbSNO are developed to solve problems involving not too many parameters, the size of the problem is taken as $n = 9$ in this case. To consider a noisy test function, we imposed noise levels $noisef = 10^{-2}$ and $noiseg = 10^{-1}$ on the function and gradient respectively to run the codes. At each iteration, three points were generated by the algorithms, where two of them concentrate on the local search at current promising regions and the other one is given to explore the search domain to find possible better local minima to perform the global aspect of the method. The algorithms were stopped when there was no improvement of the best function value over 25 iterations.

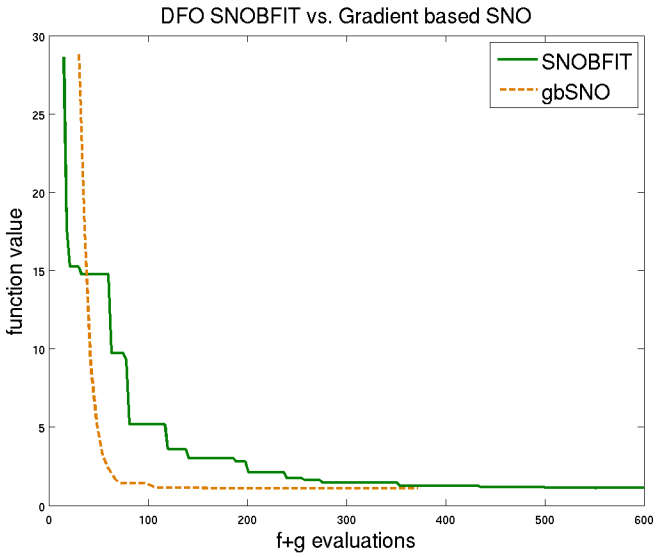


Figure 4.22: Comparison on minimal surface problem

In Figure 4.22, we can observe a quite obvious improvement in terms of function and gradient evaluations of the new gbSNO code over the derivative free SNOBFIT code. The gap at the beginning of the graphs is due to the computation of the initial space filling design which costs 15 function evaluations for SNOBFIT but the double amount for gbSNO as it evaluates additionally the gradient at each point. In the first iteration, $n+6$ points were generated which is the recommanded value from SNOBFIT.

Furthermore, considered that both algorithms allow for parallel evaluations of functions (and gradients if necessary), both algorithms are not forced to wait for sequential computations as other well-known gradient-based algorithms. So, given the hardware facilities, computing three function values at the same time is a great gain in terms of computing time.

Airbus test problem Being able to access a real-life aerodynamic application provided by Airbus, we were testing our algorithm minimizing a CDP function which we were using in the other sections before with the difference that this time an Euler model on a fine mesh, instead of a Navier-Stokes model, is applied in the CFD simulation. The problem comprises three bounded variables (amplitude, position and expansion of a bump). We were running our algorithm gbSNO two times on the given problem. As it starts always with building up a random space filling design over the whole domain, we get two different starting points after this initialization. Nevertheless, as we see on Figure 4.23 (the runs of gbSNO are displayed as dotted lines) the algorithm is able to find the global minimum after a certain time in both runs. From the gradient-based local solvers only Lancelot was able to find the global minimum, probably by chance.

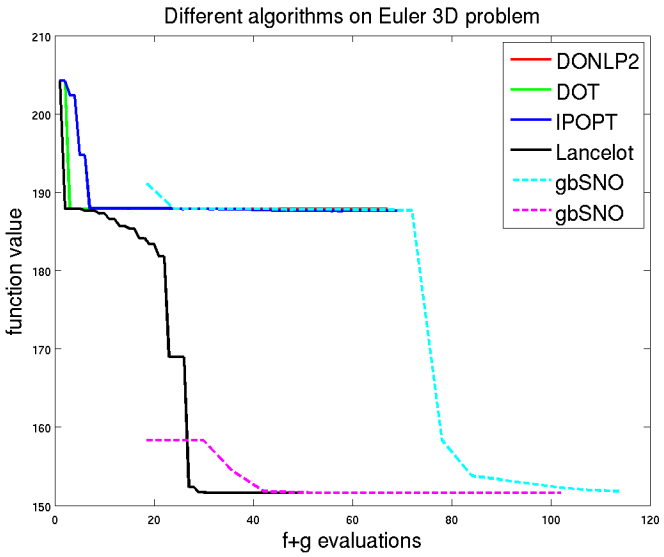


Figure 4.23: Comparison on a Euler CDP problem

4.6.4 Conclusions and perspectives

In this section, we described the new global search algorithm gbSNO which is based on the algorithm SNOBFIT by Huyer and Neumaier which is announced to be especially adapted to cope with expensive noisy functions. We modified this derivative-free-optimization software package to allow for the use of gradient information, even noisy gradient information, with the hope of accelerating convergence speed of the method. We attempted to filter the inexactness of the gradients by using the function values in the close neighbourhood of the points.

We showed on an academic example with added noise on function and gradient values, that although the cost of the initial space filling design is higher than for SNOBFIT, the convergence is much faster by using additional gradient information even in this perturbed case.

Furthermore, we did some experiments on an expensive Airbus problem what provides noisy function values as well as noisy gradient values. The new algorithm gbSNO could achieve an acceptable convergence rate and it shows a strong ability to reach the global minimum.

Chapter 5

Conclusions and future work

In this thesis, we have tried to show that derivative-free optimization (DFO) is currently a very vivid field of research as practitioners more and more want to use sophisticated techniques for solving their optimization problems. But at the same time, derivative-information may be difficult (or only at a high extra cost) to obtain in real-world applications. This suggests the use of derivative-free optimization methods where the class of interpolation-based trust-region methods has shown to be numerically efficient in several recent comparisons [102, 127].

In Chapter 2, we gave a short introduction into trust-region methods which use quadratic polynomials, interpolating or approximating the function on a set of points at which the function has been evaluated. In trust-region methods, the model is trusted to approximate the function within a local neighborhood of the best point found so far. We explained that special care must be taken to control the geometry of the interpolation set which may otherwise deteriorate in certain situations. We presented several methods of this class ranging from basic ones, which involve so-called geometry improving steps, to a recent approach which uses a self-correcting property of the geometry of the interpolation set [128]. Such an algorithm resorts to the geometry improving steps as less as possible (only when the model gradient is small), while still maintaining a mechanism for taking geometry into account.

Moreover, we have presented the new algorithm UDFO+ which is a modified version of the described self-correcting algorithm but where geometry considerations are released in some stages of the algorithm. We proved global convergence to first-order stationary points of this new version. Later, in the numerical section of Chapter 3, we showed that UDFO+ is performing out other solvers from different classes of DFO methods.

As the main contribution of this thesis, we extended this algorithm to handle bound-constrained problems. Chapter 3 introduced the new algorithm *BCDFO+* for solving bound-constrained local optimization problems. The extension to bound-constrained problems is not as straightforward as one could think because points may get aligned at active bounds and hence, the geometry of the interpolation set may deteriorate. An idea to circumvent this problem is to add points to the set which would preserve the geometry, thereby involving several additional function evaluations. But as the idea of using the self-correcting property is to dispense with extra calculations for regularly improving or preserving the geometry, we proposed to handle the bound constraints by an "active-set" approach. Such a strategy creates the opportunity for a model-based DFO method of saving a reasonable amount of function evaluations because

it proceeds minimization in lower-dimensional sub-spaces. A standard active-set method anticipates to update the set of active constraints while adding and/or removing constraints at each iteration. Our approach allows only for adding constraints to the set of active constraints because it then pursues minimization in the subspace of the remaining free variables by calling itself recursively.

Our algorithm comes with several additional interesting features. It is for instance possible to start minimization using only $n+1$ evaluated points and to augment the interpolation set when minimization progresses. In early stages, using $p = n+1$ function values (sufficient to build a linear model) is economical and often sufficient to make progress. In a later stage of the calculation, considering $p = \frac{1}{2}(n+1)(n+2)$ function values (enough to build a complete quadratic model) it is expected to achieve faster progress to a close solution. Different types of models have been proposed to perform under- and overdetermined interpolation. It turned out that the option of a sub-basis model has performed best on a test set of smooth unconstrained and bound-constrained test problems from the CUTEr library.

Another feature has been introduced, targeting to save function evaluations when encountering an active bound and starting to work in the sub-space. In fact, it has turned out to be advantageous to build the first model inside the subspace not only from true function values but also by using the approximate information given by the model of the higher-dimensional space. Furthermore, the algorithm tries to re-use function values from formerly evaluated points whenever a new model in a sub-space or the fullspace has to be computed.

We also presented the new algorithm BC-MS to solve the bound-constrained trust-region subproblem in ℓ_2-norm where we extended the technique of an Moré-Sorensen algorithm. The comparison to a standard truncated conjugate gradient method has shown that using BC-MS as the local solver inside our DFO trust-region algorithm *BCDFO+* is not as efficient as to use the TCG option applied to an infinity-norm trust region.

Numerical experiments have been performed to compare the presented algorithms to the state-of-the-art packages NEWUOA and BOBYQA, which also apply a model-based trust-region method, and three members from the class of direct-search methods. Our new development *BCDFO+* happened to compare very well to the other codes on the test sets of smooth unconstrained and bound-constrained problems from the CUTEr library.

In Chapter 4, we have studied the impact of noise on optimization algorithms in general and adapted our algorithm to handle noisy optimization problems. First, we demonstrated how the level of a low-amplitude noise in a function or a gradient can be estimated using a tool which was originally developed to calculate higher order derivatives and to estimate round-off. This tool is incorporated in our algorithm to estimate the noise level if the user intents to optimize a noisy objective function but is not able to provide the level of noise.

We also presented numerical results on sets of noisy unconstrained and bound-constrained test problems from the CUTEr library and an aerodynamic engineering application. These results support the effectiveness of our algorithm on blackbox functions for which no special

mathematical structure is known or available. We assess different types of interpolation and regression models inside our algorithm *BCDFO+*. The minimum ℓ_2-norm model seems to be the most appropriate option when only a small computational budget is available whereas the regression model is to recommend when the computational budget is large as it turned out to be the most robust to solve problems to a high required accuracy.

Furthermore, we presented a theoretical study on the allowed noise on a gradient which is used in a gradient-based line search method. We could establish deterministic properties on the gradient noise for ensuring globally convergent steepest-descent and quasi-Newton methods in the presence of a noisy gradient. The established properties have shown a strong dependence on the condition of the problem which seemed to be over stringent in the average case. And indeed, after assuming a Gaussian nature of the noise, we established another sufficient condition on the global convergence of a quasi-Newton method, which allows with a high probability for a bigger amplitude noise on the gradient.

The good performance of the algorithm opens many doors for future research. The good robustness of our implementation is of course an invitation to look for a global convergence theory.

Furthermore, we want to show to which extent small perturbations can be applied to control the condition of the system matrix of a quadratic interpolation model without corrupting the error bounds on the gradient. This is planned as an extension to the linear interpolation case which we presented in Section 3.3.4. In addition, we also want to examine the linear and quadratic regression case.

As we are interested in a broad assessment of interpolation and regression models, we want to consider the options of the least-change Frobenius-norm model and the minimum ℓ_1-norm model in our algorithm and compare them to the presented model types.

Another obvious area of research, which is motivated by our shape optimization application, is to extend the algorithm to handle general nonlinear constraints. Standard techniques based on SQP or augmented Lagrangian could be considered in this respect.

Appendix A

Test problems

Table A.1 and Table A.2 depict the unconstrained and bound-constrained test problems taken from the CUTEr testing environment for running our numerical experiments. They show the name and dimension of the problem and, for the bound-constrained problems, give specific details on the number of free variables, the number of variables which are bounded from below, those which are bounded from above and the number of variables which are bounded from below and above.

name	n	f^*
ALLINITU	4	5.74438491032034E+00
ARGLINB	10	4.63414634146338E+00
ARGLINC	10	6.13513513513513E+00
ARWHEAD	16	5.32907051820075E-15
BARD	3	8.21487730657899E-3
BDQRTIC	10	1.82811617535935E+01
BEALE	2	1.03537993810258E-30
BIGGS6	6	5.49981608181981E-16
BOX3	3	1.85236429640516E-20
BRKMCC	2	1.69042679196450E-1
BROWNAL	10	1.49563496755546E-16
BROWNDEN	4	8.58222016263563E+04
CHNROSNB	10	1.21589148855346E-19
CRAGGLVY	10	1.88656589666311E+00
CUBE	2	5.37959996529976E-25
DENSCHND	3	2.15818302178292E-4
DENSCHNE	3	1.29096866601748e-18
DENSCHNF	2	6.51324621983021E-22
DIXMAANC	15	1.00000000000000E+00
DIXMAANG	15	1.00000000000000E+00
DIXMAANI	15	1.00000000000000E+00
DIXMAANK	15	1.00000000000000E+00
DIXON3DQ	10	2.95822839457879E-31
DQDRTIC	10	5.91645678915759E-29
ENGVAL1	2	0.00000000000000E+00
EXPFIT	2	2.40510593999058E-1
FREUROTH	10	1.01406407257452E+03
GENHUMPS	5	9.31205762089110E-33
GULF	3	5.70816776659866E-29
HAIRY	2	2.00000000000000E+01
HELIX	3	1.81767515239766E-28
HILBERTA	2	1.51145573593758E-20
HIMMELBF	4	3.18571748791125E+02
HIMMELBG	2	1.17043537660229E-27
JENSMP	2	1.24362182355615e+02
KOWOSB	4	3.07505603849238E-4
MANCINO	10	1.24143266331958E-19
MARATOSB	2	-1.00000006249999E+00
MEXHAT	2	-4.01000000000000E-2
MOREBV	10	1.85746736253704E-24
NASTY	2	1.53409170790554e-72
OSBORNEB	11	4.01377362935478E-2
PALMER1C	8	9.75979912629838E-2
PALMER3C	8	1.95376385131058E-2
PALMER5C	6	2.12808666605511E+00
PALMER8C	8	1.59768063470262E-1
POWER	10	6.03971630559837E-31
ROSENBR	2	3.74397564313947E-21
SINEVAL	2	7.09027697800298E-20
SINGULAR	4	6.66638187151797e-12
SISSER	2	1.06051492721772E-12
VARDIM	10	1.59507305257139E-26
YFITU	3	6.66972048929030E-13
ZANGWIL2	2	-1.82000000000000E+01

Table A.1: Considered unconstrained CUTEr test problems

name	n	free vars	lbound	ubound	l+ubound	f^*
3PK	30		30			1.72011856739612E+00
BIGGSB1	25	1			24	1.50000000000000E-02
BQP1VAR	1				1	0.00000000000000E+00
CAMEL6	2				2	-1.03162845348988E+00
CHARDIS0	18				18	3.95170009709151E-27
CHEBYQAD	4				4	2.56057805386809E-22
CHENHARK	10		10			-2.00000000000000E+00
CVXBQP1	10				10	2.47500000000000E+00
HARKERP2	10		10			-5.00000000000000E-01
HATFLDA	4		4			1.61711062151584E-25
HATFLDB	4		3		1	5.57280900008425E-03
HATFLDC	25	1			24	3.43494690036517E-27
HIMMELP1	2				2	-6.20539355382574E+01
HS1	2	1	1			7.13660798093435E-24
HS110	10				10	-4.57784755318868E+01
HS25	3				3	1.81845940377455E-16
HS3	2	1	1			1.97215226305253E-36
HS38	4				4	2.02675622883580E-28
HS3MOD	2	1	1			0.00000000000000E+00
HS4	2		2			2.66666666400000E+00
HS45	5				5	1.00000000040000E+00
HS5	2				2	-1.91322295498104E+00
LINVERSE	19	9	10			6.00000000022758E+00
LOGROS	2		2			0.00000000000000E+00
MCCORMCK	10				10	-9.59800619474625E+00
MDHOLE	2	1	1			7.52316384526264E-35
NCVXBQP1	10				10	-2.20500000000000E+04
NCVXBQP2	10				10	-1.43818650000000E+04
NCVXBQP3	10				10	-1.19578050000000E+04
NONSCOMP	25				25	4.42431972353647E-14
OSLBQP	8		5			6.25000000000000E+00
PALMER1A	6	4	2			8.98830583652624E-02
PALMER2B	4	2	2			6.23266904205002E-01
PALMER4	4	1	3			2.28538322742966E+03
PALMER4A	6	4	2			4.06061409159725E-02
PSPDOC	4	3		1		2.41421356237309E+00
QUDLIN	12				12	-7.20000000000000E+03
SIMBQP	2	1			1	0.00000000000000E+00
SPECAN	9				9	1.64565541040970E-13
YFIT	3	2	1			6.66972055747565E-13

Table A.2: Considered bound-constrained CUTEr test problems

Appendix B

Test results

The results of the unconstrained and bound-constrained testing can be seen in the tables below. All tables show the name of the test problems from the CUTEr collection, and the number of function evaluations needed by each tested solver to attain a specified number of significant figures in the objective function value f^*, computed using the package KNITRO or TRON in the un- or bound-constrained case, respectively.

name	nf Sub-basis 6 fig	nf Min ℓ_2-norm 6 fig	nf Regression 6 fig	nf Min Frob.-norm 6 fig
ALLINITU	39	53	81	48
ARGLINB	67	60	60	78
ARGLINC	63	52	52	53
ARWHEAD	16	16	16	16
BARD	32	38	65	44
BDQRTIC	253	242	337	219
BEALE	30	29	45	29
BIGGS6	319	639	630	661
BOX3	34	44	58	34
BRKMCC	11	10	8	10
BROWNAL	408	494	584	406
BROWNDEN	75	61	131	71
CHNROSNB	943	955	1263	947
CRAGGLVY	543	504	674	547
CUBE	94	91	155	81
DENSCHND	68	162	266	126
DENSCHNE	51	97	130	88
DENSCHNF	21	24	33	25
DIXMAANC	296	312	586	311
DIXMAANG	247	408	698	406
DIXMAANI	326	291	691	335
DIXMAANK	1082	456	850	546
DIXON3DQ	100	37	37	130
DQDRTIC	22	67	67	67
ENGVAL1	4	4	4	4
EXPFIT	32	32	45	39
FREUROTH	210	230	343	255
GENHUMPS	1646	2055	1757	3217
GULF	309	342	267	309
HAIRY	31	37	47	52
HELIX	72	70	105	51
HILBERTA	7	8	8	7
HIMMELBF	402	135	136	208
HIMMELBG	23	8	8	8
JENSMP	70	57	108	63
KOWOSB	81	86	160	114
MANCINO	59	79	144	88
MARATOSB	3180	3003	5235	3048
MEXHAT	79	88	254	65
MOREBV	83	82	153	135
NASTY	3	3	3	3
OSBORNEB	1028	1323	1140	994
PALMER1C	failed	failed	failed	failed
PALMER3C	60	88	63	59
PALMER5C	42	41	32	29
PALMER8C	82	63	57	63
POWELLSG	99	103	224	96
POWER	308	414	709	388
ROSENBR	85	73	137	62
SINEVAL	197	198	325	200
SISSER	10	14	17	28
VARDIM	687	802	625	565
YFITU	810	841	526	799
ZANGWIL2	4	4	4	4

Table B.1: Results from comparison of different types of models in ***BCDFO+*** on unconstrained CUTEr problems (see Figure 3.2)

name	nf BC-MS 6 fig	nf TCG 6 fig
ALLINITU	40	39
ARGLINB	67	67
ARGLINC	63	63
ARWHEAD	16	16
BARD	38	32
BDQRTIC	262	253
BEALE	28	30
BIGGS6	396	319
BOX3	29	34
BRKMCC	12	11
BROWNAL	296	408
BROWNDEN	64	75
CHNROSNB	1001	943
CRAGGLVY	failed	543
CUBE	88	94
DENSCHND	143	68
DENSCHNE	143	51
DENSCHNF	22	21
DIXMAANC	370	296
DIXMAANG	435	247
DIXMAANI	332	326
DIXMAANK	525	1082
DIXON3DQ	34	100
DQDRTIC	23	22
ENGVAL1	21	4
EXPFIT	46	32
FREUROTH	246	210
GENHUMPS	575	1646
GULF	failed	309
HAIRY	35	31
HELIX	70	72
HILBERTA	7	7
HIMMELBF	165	402
HIMMELBG	25	23
JENSMP	62	70
KOWOSB	81	81
MANCINO	93	59
MARATOSB	2960	3180
MEXHAT	56	79
MOREBV	77	83
NASTY	3	3
OSBORNEB	947	1028
PALMER1C	failed	failed
PALMER3C	59	60
PALMER5C	29	42
PALMER8C	66	82
POWELLSG	98	99
POWER	477	308
ROSENBR	88	85
SINEVAL	202	197
SISSER	18	10
VARDIM	394	687
YFITU	647	810
ZANGWIL2	7	4

Table B.2: Results from comparison of local solvers BC-MS and TCG in ***BCDFO+*** on unconstrained CUTEr problems (see Figure 3.3)

name	nf BC-MS 6 fig	nf TCG 6 fig
3PK	failed	failed
BIGGSB1	650	38
BQP1VAR	2	2
CAMEL6	20	17
CHARDIS0	190	190
CHEBYQAD	207	143
CHENHARK	93	100
CVXBQP1	33	21
HARKERP2	38	91
HATFLDA	5	5
HATFLDB	34	38
HATFLDC	405	400
HIMMELP1	34	21
HS1	88	75
HS110	97	156
HS25	367	249
HS3	8	8
HS38	237	371
HS3MOD	8	8
HS4	5	5
HS45	12	13
HS5	16	11
LINVERSE	440	478
LOGROS	3	3
MCCORMCK	96	179
MDHOLE	154	159
NCVXBQP1	24	15
NCVXBQP2	61	32
NCVXBQP3	40	31
NONSCOMP	1458	626
OSLBQP	29	31
PALMER1A	3939	4838
PALMER2B	380	670
PALMER4	failed	257
PALMER4A	2025	1892
PSPDOC	35	34
QUDLIN	25	20
SIMBQP	9	9
SPECAN	256	351
YFIT	736	2156

Table B.3: Results from comparison of local solvers BC-MS and TCG in ***BCDFO+*** on bound-constrained CUTEr problems (see Figure 3.4)

	nf ***BCDFO+***				nf BC-DFO				nf NEWUOA			
name	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig
ALLINITU	20	38	39	43	53	66	103	106	50	60	67	73
ARGLINB	52	58	67	67	88	88	88	88	70	70	70	70
ARGLINC	57	62	63	63	84	84	84	84	70	70	70	70
ARWHEAD	16	16	16	16	16	16	16	16	513	579	641	680
BARD	19	28	32	37	62	73	80	83	26	46	51	60
BDQRTIC	140	193	253	254	347	435	578	593	181	236	276	296
BEALE	22	26	30	31	63	69	69	73	24	33	42	46
BIGGS6	176	263	319	559	442	653	715	820	148	265	400	624
BOX3	24	24	34	41	46	56	58	63	18	33	47	59
BRKMCC	10	10	11	16	7	13	13	16	7	7	15	15
BROWNAL	164	268	408	426	320	377	398	432	88	166	212	256
BROWNDEN	56	71	75	77	93	99	107	111	59	66	80	85
CHNROSNB	867	937	943	963	1180	1264	1335	1341	717	776	790	803
CRAGGLVY	422	519	543	580	919	1199	1302	1324	458	538	616	662
CUBE	67	86	94	97	77	103	110	112	105	130	138	151
DENSCHND	51	66	68	83	58	78	86	100	45	45	64	68
DENSCHNE	39	50	51	58	67	76	87	91	87	92	99	110
DENSCHNF	13	20	21	22	15	18	22	22	23	25	25	34
DIXMAANC	232	278	296	309	334	375	400	444	415	447	465	472
DIXMAANG	186	210	247	265	566	654	669	691	479	508	528	543
DIXMAANI	265	312	326	338	810	813	831	971	398	460	483	509
DIXMAANK	587	900	1082	1101	570	779	799	825	736	773	848	881
DIXON3DQ	99	100	100	100	31	31	31	31	72	72	79	83
DQDRTIC	22	22	22	22	44	44	44	44	71	71	81	81
ENGVAL1	4	4	4	4	4	4	4	4	17	23	29	29
EXPFIT	26	30	32	38	65	68	70	72	25	32	34	38
FREUROTH	131	204	210	254	345	553	560	571	174	197	229	243
GENHUMPS	1161	1600	1646	1688	705	1422	1602	1684	613	739	760	793
GULF	186	295	309	314	197	307	338	344	187	336	378	404
HAIRY	25	27	31	32	46	54	55	57	68	68	74	74
HELIX	62	70	72	74	57	66	68	70	66	75	84	90
HILBERTA	7	7	7	7	7	7	7	7	9	9	9	9
HIMMELBF	105	367	402	412	257	400	461	485	168	599	737	1076
HIMMELBG	15	21	23	23	22	32	35	35	14	19	23	23
JENSMP	60	67	70	72	36	44	48	49	10	25	28	31
KOWOSB	5	35	81	90	5	59	125	144	16	38	106	116
MANCINO	43	55	59	71	89	103	110	116	136	136	143	150
MARATOSB	2804	3116	3180	3198	2915	3240	3339	3361	5693	6471	6748	6813
MEXHAT	42	58	79	81	263	508	520	527	64	65	79	83
MOREBV	30	78	83	92	88	108	119	135	68	73	84	112
NASTY	3	3	3	3	3	3	3	3	failed	failed	failed	failed
OSBORNEB	636	946	1028	1047	1917	2564	2743	2764	858	1126	1190	1211
PALMER1C	61	65	failed	failed	68	68	failed	failed	failed	failed	failed	failed
PALMER3C	56	58	60	62	66	66	66	67	failed	failed	failed	failed
PALMER5C	42	42	42	42	46	46	46	46	37	37	44	44
PALMER8C	70	79	82	83	70	71	71	78	failed	failed	failed	failed
POWELLSG	46	61	99	142	67	99	136	173	60	82	105	137
POWER	159	215	308	409	358	704	839	1117	218	289	375	460
ROSENBR	75	82	85	87	46	72	75	78	82	94	98	107
SINEVAL	190	195	197	199	171	177	178	182	203	217	218	234
SISSER	3	7	10	13	3	8	16	26	16	27	32	35
VARDIM	610	665	687	706	2022	2278	2409	2412	464	579	648	648
YFITU	712	795	810	816	897	965	981	985	failed	failed	failed	failed
ZANGWIL2	4	4	4	4	4	4	4	4	7	7	7	7

Table B.4: Results from comparison of ***BCDFO+***, BC-DFO and NEWUOA on unconstrained CUTEr problems (see Figures 3.5-3.8 and Figures 3.9-3.12)

	nf ***BCDFO+***				nf BC-DFO				nf BOBYQA			
name	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig
3PK	failed	failed	failed	failed	8834	8834	8927	8959	failed	failed	failed	failed
BIGGSB1	34	38	38	38	35	35	35	35	144	341	489	548
BQP1VAR	2	2	2	2	2	2	2	2	7	7	7	7
CAMEL6	14	15	17	22	16	26	29	35	17	29	37	41
CHARDIS0	189	190	190	190	420	420	420	420	92	119	139	161
CHEBYQAD	115	138	143	151	208	235	259	265	15	50	60	64
CHENHARK	100	100	100	100	133	134	134	134	90	121	151	172
CVXBQP1	21	21	21	21	21	21	21	21	26	39	39	39
HARKERP2	89	91	91	91	62	63	63	63	64	67	67	67
HATFLDA	5	5	5	5	5	5	5	5	46	104	141	182
HATFLDB	26	34	38	40	64	76	81	91	40	52	67	72
HATFLDC	97	381	400	425	699	753	768	790	131	247	360	441
HIMMELP1	17	21	21	24	15	25	26	28	13	19	22	26
HS1	68	73	75	77	133	138	147	148	135	158	167	172
HS110	83	113	156	167	196	316	398	409	144	260	436	521
HS25	43	236	249	252	59	254	298	failed	107	734	978	995
HS3	3	7	8	8	3	8	9	9	6	9	10	10
HS38	354	367	371	375	322	342	347	347	408	440	474	503
HS3MOD	8	8	8	8	13	13	13	13	21	24	24	24
HS4	5	5	5	5	5	5	5	5	7	7	7	7
HS45	13	13	13	13	15	15	15	15	16	16	16	16
HS5	5	10	11	17	8	8	20	20	13	15	18	21
LINVERSE	410	450	478	494	792	905	936	958	137	236	651	2828
LOGROS	3	3	3	3	3	3	3	3	443	609	652	661
MCCORMCK	117	137	179	183	59	152	166	178	29	54	75	87
MDHOLE	156	159	159	159	165	165	165	165	220	220	225	225
NCVXBQP1	15	15	15	15	16	18	18	18	31	31	31	31
NCVXBQP2	32	32	32	32	64	64	64	64	31	31	31	31
NCVXBQP3	31	31	31	31	50	50	50	50	49	49	49	49
NONSCOMP	581	615	626	650	859	907	921	937	779	1072	1406	1684
OSLBQP	30	31	31	31	31	32	32	32	22	22	22	22
PALMER1A	4385	4726	4838	4846	10842	11143	11190	11195	failed	failed	failed	failed
PALMER2B	639	669	670	675	1192	1198	1201	1206	1037	1118	1144	1166
PALMER4	108	251	257	261	145	160	162	170	62	82	87	93
PALMER4A	1283	1764	1892	1934	2753	3445	3511	3536	2366	5767	7606	8105
PSPDOC	28	31	34	38	35	44	44	50	41	55	57	65
QUDLIN	17	20	20	20	22	22	22	22	34	34	34	34
SIMBQP	9	9	9	9	12	12	12	12	12	12	12	12
SPECAN	336	342	351	358	failed	failed	failed	failed	697	765	820	881
YFIT	1355	2139	2156	2159	897	965	981	985	1356	2011	2237	2257

Table B.5: Results from comparison of ***BCDFO+***, BC-DFO and BOBYQA on bound-constrained CUTEr problems (see Figures 3.13-3.16 and Figures 3.17-3.20)

	nf SID-PSM				nf NOMADm				nf BFO			
name	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig
ALLINITU	63	124	156	169	20	119	191	265	125	200	275	365
ARGLINB	145	308	391	450	433	560	failed	failed	134	134	238	334
ARGLINC	211	342	342	471	496	496	failed	failed	229	255	327	481
ARWHEAD	33	33	33	33	355	355	355	355	1283	2069	3177	4195
BARD	55	84	104	114	85	7836	failed	failed	107	320	387	433
BDQRTIC	180	358	476	608	963	1350	1778	failed	545	1130	2179	3418
BEALE	26	26	26	26	3925	4215	failed	failed	116	144	178	191
BIGGS6	192	633	3729	failed	4002	failed	failed	failed	776	1583	3228	failed
BOX3	85	110	114	128	11	45	45	45	103	418	595	767
BRKMCC	8	23	42	59	106	209	313	failed	16	80	97	150
BROWNAL	748	1057	1255	1436	2785	14018	failed	failed	1502	2757	3801	7130
BROWNDEN	122	175	191	257	863	1217	1518	failed	159	383	483	574
CHNROSNB	24	24	24	24	9486	failed	failed	failed	5557	9993	failed	failed
CRAGGLVY	341	546	610	671	failed	failed	failed	failed	570	1393	2486	3861
CUBE	39	121	138	143	2637	failed	failed	failed	496	606	660	686
DENSCHND	3	3	3	3	7933	8050	9986	failed	51	78	85	140
DENSCHNE	32	32	32	32	78	78	78	78	581	645	691	746
DENSCHNF	58	77	96	107	70	91	failed	failed	56	83	102	124
DIXMAANC	3	3	3	3	2790	2790	2790	2790	1216	2061	3192	3870
DIXMAANG	3	3	3	3	2704	2777	2777	2777	991	2079	3324	4999
DIXMAANI	3	3	3	3	2161	2161	2219	2219	1039	4268	7897	13390
DIXMAANK	3	3	3	3	2733	2733	2806	2806	851	2072	6067	10685
DIXON3DQ	124	124	124	124	6182	11854	failed	failed	1374	2200	3286	4619
DQDRTIC	3	3	3	3	941	941	941	941	4487	6334	7440	8184
ENGVAL1	15	52	68	78	28	28	28	28	34	59	89	117
EXPFIT	55	73	91	110	135	202	238	failed	113	129	161	172
FREUROTH	876	1012	1054	1296	827	failed	failed	failed	2789	failed	failed	failed
GENHUMPS	2	179	212	271	561	594	594	594	3204	6764	7361	7441
GULF	646	1435	1870	2017	1031	failed	failed	failed	336	357	2065	2213
HAIRY	66	68	77	87	55	55	55	55	68	119	126	173
HELIX	78	98	108	130	1357	failed	failed	failed	185	326	423	506
HILBERTA	2	2	2	2	42	90	126	162	87	95	126	149
HIMMELBF	failed	failed	failed	failed	534	11470	failed	failed	303	failed	failed	failed
HIMMELBG	14	14	14	14	32	40	40	40	31	44	59	79
JENSMP	83	94	104	124	121	249	failed	failed	265	297	327	337
KOWOSB	1	180	231	259	1	742	failed	failed	1	720	1504	1713
MANCINO	424	424	448	472	1748	failed	failed	failed	1516	1852	2191	3338
MARATOSB	failed	failed	failed	failed	failed	failed	failed	failed	13979	failed	failed	failed
MEXHAT	122	3786	3827	3836	failed	failed	failed	failed	161	230	263	274
MOREBV	97	194	244	271	88	failed	failed	failed	162	1084	2154	5199
NASTY	7	7	7	7	7	13	13	13	failed	failed	failed	failed
OSBORNEB	937	1508	1577	1635	5991	13922	failed	failed	5165	failed	failed	failed
PALMER1C	149	150	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed
PALMER3C	104	108	108	108	failed	failed	failed	failed	failed	failed	failed	failed
PALMER5C	31	31	31	31	542	742	978	failed	435	628	900	1109
PALMER8C	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed
POWELLSG	73	113	171	235	276	479	1193	1945	236	350	785	1885
POWER	3	3	3	3	902	902	950	950	433	563	740	1145
ROSENBR	70	89	97	120	1517	failed	failed	failed	219	270	302	325
SINEVAL	414	437	440	450	failed	failed	failed	failed	899	939	960	985
SISSER	5	13	38	46	20	35	35	35	11	21	36	38
VARDIM	934	1093	1271	1305	failed	failed	failed	failed	2082	2802	3635	4406
YFITU	5574	14598	failed	failed	failed	failed	failed	failed	2762	4495	5296	5463
ZANGWIL2	8	8	8	8	18	43	43	43	11	36	53	68

Table B.6: Results from comparison of ***BCDFO+***, BC-DFO, SID-PSM, NOMADm and BFO on unconstrained CUTEr problems (see Figures 3.21-3.24 and Figures 3.25-3.28)

	nf SID-PSM				nf NOMADm				nf BFO			
name	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig
3PK	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed
BIGGSB1	33	62	231	532	failed	failed	failed	failed	10379	failed	failed	failed
BQP1VAR	3	3	3	3	5	5	5	5	3	3	3	3
CAMEL6	36	53	71	89	28	58	83	93	41	61	74	97
CHARDIS0	337	337	337	337	4130	5343	failed	failed	3756	4606	5793	7349
CHEBYQAD	60	99	146	161	34	86	294	failed	132	219	333	417
CHENHARK	304	632	1235	2006	2484	failed	failed	failed	288	909	1200	1648
CVXBQP1	42	42	42	42	1727	failed	failed	failed	77	77	77	77
HARKERP2	104	133	190	233	4138	failed	failed	failed	124	171	229	265
HATFLDA	43	70	94	106	failed	failed	failed	failed	233	429	528	619
HATFLDB	37	119	181	262	failed	failed	failed	failed	341	492	582	656
HATFLDC	350	591	914	1290	4348	10252	failed	failed	1288	3806	7744	11737
HIMMELP1	25	40	58	67	14	43	68	failed	31	49	77	100
HS1	175	197	214	216	5	5	5	5	611	695	704	724
HS110	88	174	291	407	163	719	719	1181	128	600	839	1289
HS25	167	1596	2715	3214	272	failed	failed	failed	340	986	1190	1340
HS3	6	6	6	6	77	failed	failed	failed	5	13	26	26
HS38	165	195	209	223	failed	failed	failed	failed	1422	1624	1934	2215
HS3MOD	33	33	33	33	391	failed	failed	failed	28	32	36	40
HS4	5	5	5	5	26	26	26	26	8	8	8	8
HS45	10	10	10	10	60	66	66	66	21	21	21	21
HS5	17	41	50	69	22	56	76	103	29	53	58	90
LINVERSE	175	398	583	743	1234	2700	4341	failed	715	2139	3355	5840
LOGROS	8	8	8	8	failed	failed	failed	failed	1075	1353	1412	1444
MCCORMCK	67	236	344	491	1052	1259	2018	2493	367	655	862	1112
MDHOLE	675	687	713	727	12396	12409	12409	12409	192	208	223	232
NCVXBQP1	22	22	22	22	583	605	605	605	81	81	81	81
NCVXBQP2	58	65	65	65	478	748	failed	failed	80	80	80	80
NCVXBQP3	104	178	240	241	506	821	failed	failed	80	80	80	80
NONSCOMP	886	1265	1483	1895	14225	failed	failed	failed	3578	5180	7138	9203
OSLBQP	63	99	140	168	96	363	594	failed	68	96	155	213
PALMER1A	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed	failed
PALMER2B	588	614	677	697	failed	failed	failed	failed	1524	2230	2778	3310
PALMER4	failed	failed	failed	failed	failed	failed	failed	failed	963	1977	2205	2770
PALMER4A	7511	failed	failed	failed	failed	failed	failed	failed	10779	failed	failed	failed
PSPDOC	70	150	281	377	146	299	465	630	81	144	196	275
QUDLIN	11	11	11	11	536	574	574	574	93	93	93	93
SIMBQP	31	72	82	82	67	failed	failed	failed	40	40	52	63
SPECAN	604	617	833	834	749	1022	failed	failed	2287	4429	6136	6602
YFIT	7619	12449	failed	failed	failed	failed	failed	failed	2762	4495	5296	5463

Table B.7: Results from comparison of ***BCDFO+***, BC-DFO, SID-PSM, NOMADm and BFO on bound-constrained CUTEr problems (see Figures 3.29-3.32 and Figures 3.33-3.36)

	$\tau = 10^{-1}, maxeval = 200$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
ALLINITU	9	20	34	16
ARGLINB	2	2	2	2
ARGLINC	2	2	2	2
ARWHEAD	72	187	failed	failed
BARD	5	4	4	14
BDQRTIC	29	14	14	13
BEALE	6	7	12	7
BIGGS6	66	54	90	66
BOX3	5	4	4	5
BRKMCC	4	6	6	4
BROWNAL	3	25	25	83
BROWNDEN	22	18	37	17
CHNROSNB	2	2	2	2
CRAGGLVY	32	6	6	8
CUBE	4	6	7	4
DENSCHND	12	3	3	3
DENSCHNE	3	11	37	4
DENSCHNF	7	9	7	13
DIXMAANC	2	2	2	2
DIXMAANG	2	2	2	2
DIXMAANI	2	2	2	2
DIXMAANK	2	2	2	2
DIXON3DQ	27	13	13	14
DQDRTIC	3	3	3	3
ENGVAL1	2	2	2	2
EXPFIT	15	14	25	15
FREUROTH	5	12	12	15
GENHUMPS	16	10	10	10
GULF	59	74	147	125
HAIRY	9	11	12	27
HELIX	11	13	24	13
HILBERTA	3	3	3	3
HIMMELBF	12	24	18	27
HIMMELBG	12	6	6	6
JENSMP	51	45	80	44
KOWOSB	18	18	failed	17
MANCINO	11	13	13	25
MARATOSB	8	7	8	5
MEXHAT	15	18	24	17
MOREBV	29	20	20	23
NASTY	103	failed	failed	49
OSBORNEB	95	96	172	98
PALMER1C	17	9	9	10
PALMER3C	19	9	9	11
PALMER5C	16	15	15	10
PALMER8C	20	9	9	8
POWELLSG	19	19	50	12
POWER	2	2	2	2
ROSENBR	12	6	23	5
SINEVAL	100	137	195	110
SISSER	14	10	13	10
VARDIM	2	2	2	2
YFITU	29	14	56	14
ZANGWIL2	2	2	2	2

Table B.8: Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.6)

	$\tau = 10^{-5}, maxeval = 200$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
ALLINITU	36	46	73	38
ARGLINB	31	24	24	28
ARGLINC	36	15	15	19
ARWHEAD	199	failed	failed	failed
BARD	failed	41	failed	33
BDQRTIC	170	159	failed	171
BEALE	31	28	39	32
BIGGS6	197	failed	failed	failed
BOX3	48	failed	38	failed
BRKMCC	14	7	6	9
BROWNAL	160	161	failed	failed
BROWNDEN	61	49	107	60
CHNROSNB	200	failed	failed	failed
CRAGGLVY	failed	failed	failed	193
CUBE	81	82	122	66
DENSCHND	35	67	127	49
DENSCHNE	42	failed	failed	failed
DENSCHNF	17	20	23	21
DIXMAANC	failed	200	failed	failed
DIXMAANG	failed	failed	failed	199
DIXMAANI	failed	198	failed	failed
DIXMAANK	200	failed	failed	failed
DIXON3DQ	failed	failed	67	196
DQDRTIC	12	34	34	44
ENGVAL1	2	2	2	2
EXPFIT	31	30	49	30
FREUROTH	194	failed	failed	failed
GENHUMPS	51	152	failed	failed
GULF	failed	failed	193	failed
HAIRY	29	33	36	45
HELIX	62	54	67	38
HILBERTA	5	6	6	5
HIMMELBF	88	failed	120	98
HIMMELBG	19	19	15	17
JENSMP	72	58	120	69
KOWOSB	49	failed	failed	failed
MANCINO	25	44	44	47
MARATOSB	13	18	15	17
MEXHAT	17	47	67	24
MOREBV	failed	failed	failed	199
NASTY	105	failed	failed	53
OSBORNEB	199	failed	failed	failed
PALMER1C	47	65	49	86
PALMER3C	63	69	55	53
PALMER5C	36	35	26	23
PALMER8C	82	69	50	60
POWELLSG	51	64	131	59
POWER	115	141	failed	131
ROSENBR	65	70	122	58
SINEVAL	173	failed	failed	failed
SISSER	failed	failed	18	failed
VARDIM	107	11	11	11
YFITU	failed	failed	184	failed
ZANGWIL2	5	5	5	5

Table B.9: Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.7)

	$\tau = 10^{-1}, maxeval = 15000$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
ALLINITU	9	20	34	16
ARGLINB	2	2	2	2
ARGLINC	2	2	2	2
ARWHEAD	78	187	352	212
BARD	5	4	4	14
BDQRTIC	29	14	14	13
BEALE	6	7	12	7
BIGGS6	76	66	97	83
BOX3	5	4	4	5
BRKMCC	4	6	6	4
BROWNAL	3	25	25	83
BROWNDEN	22	18	37	17
CHNROSNB	2	2	2	2
CRAGGLVY	32	6	6	8
CUBE	4	6	7	4
DENSCHND	12	3	3	3
DENSCHNE	3	11	37	4
DENSCHNF	7	9	7	13
DIXMAANC	2	2	2	2
DIXMAANG	2	2	2	2
DIXMAANI	2	2	2	2
DIXMAANK	2	2	2	2
DIXON3DQ	27	13	13	14
DQDRTIC	3	3	3	3
ENGVAL1	2	2	2	2
EXPFIT	15	14	25	15
FREUROTH	5	12	12	15
GENHUMPS	16	10	10	10
GULF	59	74	147	125
HAIRY	9	11	12	27
HELIX	11	13	24	13
HILBERTA	3	3	3	3
HIMMELBF	12	24	18	27
HIMMELBG	12	6	6	6
JENSMP	51	45	80	44
KOWOSB	18	18	failed	17
MANCINO	11	13	13	25
MARATOSB	8	7	8	5
MEXHAT	15	18	24	17
MOREBV	29	20	20	26
NASTY	103	204	540	49
OSBORNEB	135	146	233	132
PALMER1C	17	9	9	10
PALMER3C	19	9	9	11
PALMER5C	16	15	15	10
PALMER8C	20	9	9	8
POWELLSG	19	19	50	12
POWER	2	2	2	2
ROSENBR	12	6	23	5
SINEVAL	100	137	195	110
SISSER	14	10	13	10
VARDIM	2	2	2	2
YFITU	29	14	56	14
ZANGWIL2	2	2	2	2

Table B.10: Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.8)

	$\tau = 10^{-5}, maxeval = 15000$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
ALLINITU	36	46	73	38
ARGLINB	31	24	24	28
ARGLINC	36	15	15	19
ARWHEAD	failed	544	failed	failed
BARD	failed	41	failed	33
BDQRTIC	171	159	276	172
BEALE	31	28	39	32
BIGGS6	failed	failed	1149	failed
BOX3	48	failed	38	failed
BRKMCC	14	7	6	9
BROWNAL	160	161	277	failed
BROWNDEN	61	49	107	60
CHNROSNB	2050	1320	952	1847
CRAGGLVY	474	379	503	245
CUBE	81	82	122	66
DENSCHND	35	67	127	49
DENSCHNE	42	failed	failed	failed
DENSCHNF	17	20	23	21
DIXMAANC	248	262	462	258
DIXMAANG	426	622	572	327
DIXMAANI	failed	failed	343	failed
DIXMAANK	2379	417	587	failed
DIXON3DQ	failed	failed	67	201
DQDRTIC	12	34	34	44
ENGVAL1	2	2	2	2
EXPFIT	31	30	49	30
FREUROTH	1380	235	323	1838
GENHUMPS	64	159	414	2747
GULF	failed	failed	193	failed
HAIRY	29	33	36	45
HELIX	62	54	67	38
HILBERTA	5	6	6	5
HIMMELBF	88	failed	120	98
HIMMELBG	19	19	15	17
JENSMP	72	58	120	69
KOWOSB	49	failed	failed	failed
MANCINO	25	44	44	47
MARATOSB	2104	1729	3193	1794
MEXHAT	17	47	67	24
MOREBV	failed	failed	failed	277
NASTY	105	215	failed	53
OSBORNEB	failed	6909	failed	failed
PALMER1C	47	65	49	86
PALMER3C	63	69	55	53
PALMER5C	36	35	26	23
PALMER8C	82	69	50	60
POWELLSG	51	64	131	59
POWER	115	141	214	131
ROSENBR	65	70	122	58
SINEVAL	173	214	317	failed
SISSER	failed	failed	18	failed
VARDIM	107	11	11	11
YFITU	failed	failed	306	failed
ZANGWIL2	5	5	5	5

Table B.11: Results from comparison of different model types in ***BCDFO+*** on unconstrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.9)

	$\tau = 10^{-1}, maxeval = 200$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
3PK	failed	182	182	failed
BIGGSB1	5	4	4	9
BQP1VAR	1	1	1	1
CAMEL6	12	11	19	11
CHARDIS0	32	10	10	21
CHEBYQAD	72	58	86	55
CHENHARK	14	30	26	21
CVXBQP1	3	3	3	3
HARKERP2	7	6	6	5
HATFLDA	18	failed	43	20
HATFLDB	17	7	7	7
HATFLDC	150	128	128	failed
HIMMELP1	8	8	8	8
HS1	4	3	3	4
HS110	79	72	146	82
HS25	11	7	7	7
HS3	failed	2	2	failed
HS38	6	3	3	3
HS3MOD	4	4	4	5
HS4	3	3	3	3
HS45	6	6	6	6
HS5	3	3	3	3
LINVERSE	44	16	16	41
LOGROS	22	28	33	21
MCCORMCK	11	7	7	7
MDHOLE	83	95	174	118
NCVXBQP1	5	5	5	5
NCVXBQP2	7	7	7	7
NCVXBQP3	7	7	7	7
NONSCOMP	4	4	4	4
OSLBQP	3	3	3	3
PALMER1A	87	103	failed	85
PALMER2B	160	200	failed	190
PALMER4	110	failed	64	68
PALMER4A	52	50	77	41
PSPDOC	16	19	19	11
QUDLIN	5	5	5	5
SIMBQP	7	7	7	7
SPECAN	9	13	13	10
YFIT	29	14	56	14

Table B.12: Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.10)

	$\tau = 10^{-5}, maxeval = 200$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
3PK	failed	failed	199	failed
BIGGSB1	9	12	14	14
BQP1VAR	1	1	1	1
CAMEL6	16	20	31	18
CHARDIS0	138	111	111	178
CHEBYQAD	101	101	146	171
CHENHARK	96	199	148	failed
CVXBQP1	3	3	3	3
HARKERP2	35	22	22	31
HATFLDA	33	failed	failed	failed
HATFLDB	failed	48	101	failed
HATFLDC	197	failed	failed	failed
HIMMELP1	19	19	28	19
HS1	67	67	85	49
HS110	163	143	failed	failed
HS25	failed	91	failed	failed
HS3	failed	2	2	failed
HS38	failed	68	failed	failed
HS3MOD	6	10	6	6
HS4	3	3	3	3
HS45	6	6	6	6
HS5	failed	failed	24	failed
LINVERSE	162	148	148	failed
LOGROS	27	33	failed	28
MCCORMCK	186	167	failed	159
MDHOLE	147	148	failed	189
NCVXBQP1	5	5	5	5
NCVXBQP2	7	7	7	7
NCVXBQP3	7	15	15	13
NONSCOMP	failed	118	failed	failed
OSLBQP	12	9	9	9
PALMER1A	200	failed	failed	failed
PALMER2B	198	failed	failed	failed
PALMER4	failed	failed	194	failed
PALMER4A	failed	failed	failed	200
PSPDOC	failed	failed	39	29
QUDLIN	8	5	5	5
SIMBQP	8	8	8	8
SPECAN	188	failed	failed	failed
YFIT	failed	failed	184	failed

Table B.13: Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 200$ (see Figure 4.11)

	$\tau = 10^{-1}, maxeval = 15000$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
3PK	564	186	186	252
BIGGSB1	5	4	4	9
BQP1VAR	1	1	1	1
CAMEL6	12	11	19	11
CHARDIS0	32	10	10	21
CHEBYQAD	72	58	86	55
CHENHARK	14	30	26	21
CVXBQP1	3	3	3	3
HARKERP2	7	6	6	5
HATFLDA	18	failed	43	20
HATFLDB	17	7	7	7
HATFLDC	302	485	805	430
HIMMELP1	8	8	8	8
HS1	4	3	3	4
HS110	79	72	146	82
HS25	11	7	7	7
HS3	failed	2	2	failed
HS38	6	3	3	3
HS3MOD	4	4	4	5
HS4	3	3	3	3
HS45	6	6	6	6
HS5	3	3	3	3
LINVERSE	77	18	18	149
LOGROS	22	28	33	21
MCCORMCK	11	7	7	7
MDHOLE	83	95	174	118
NCVXBQP1	5	5	5	5
NCVXBQP2	7	7	7	7
NCVXBQP3	7	7	7	7
NONSCOMP	4	4	4	4
OSLBQP	3	3	3	3
PALMER1A	1525	1323	743	1373
PALMER2B	165	312	281	287
PALMER4	110	242	64	68
PALMER4A	52	50	77	41
PSPDOC	16	19	19	11
QUDLIN	5	5	5	5
SIMBQP	7	7	7	7
SPECAN	9	13	13	10
YFIT	29	14	56	14

Table B.14: Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.12)

	$\tau = 10^{-5}, maxeval = 15000$			
name	nf Sub-basis	nf Min ℓ_2-norm	nf Regression	nf Min Frob.-norm
3PK	failed	failed	10804	failed
BIGGSB1	9	12	14	14
BQP1VAR	1	1	1	1
CAMEL6	16	20	31	18
CHARDIS0	138	111	111	178
CHEBYQAD	101	101	146	171
CHENHARK	96	201	148	failed
CVXBQP1	3	3	3	3
HARKERP2	35	22	22	31
HATFLDA	33	failed	failed	failed
HATFLDB	failed	48	101	failed
HATFLDC	1273	failed	failed	failed
HIMMELP1	19	19	28	19
HS1	67	67	85	49
HS110	163	143	343	209
HS25	failed	91	failed	failed
HS3	failed	2	2	failed
HS38	388	68	372	383
HS3MOD	6	10	6	6
HS4	3	3	3	3
HS45	6	6	6	6
HS5	failed	failed	24	failed
LINVERSE	401	failed	706	6067
LOGROS	27	33	failed	28
MCCORMCK	186	167	275	159
MDHOLE	147	148	272	189
NCVXBQP1	5	5	5	5
NCVXBQP2	7	7	7	7
NCVXBQP3	7	15	15	13
NONSCOMP	287	118	202	349
OSLBQP	12	9	9	9
PALMER1A	2581	failed	1010	failed
PALMER2B	277	464	479	542
PALMER4	failed	failed	202	324
PALMER4A	2326	1614	747	1154
PSPDOC	failed	failed	39	29
QUDLIN	8	5	5	5
SIMBQP	8	8	8	8
SPECAN	194	224	331	293
YFIT	failed	failed	306	failed

Table B.15: Results from comparison of different model types in ***BCDFO+*** on bound-constrained noisy CUTEr problems when $maxeval = 15000$ (see Figure 4.13)

Appendix C

French summary of the thesis

Récapitulation française de la thèse

L'optimisation sans dérivées (OSD) a connu un regain d'intérêt ces dernières années, principalement motivée par le besoin croissant de résoudre des problèmes d'optimisation définis par des fonctions dont les valeurs sont calculées par simulation (par exemple, la conception technique, la restauration d'images médicales ou de nappes phréatiques).

Ces problèmes d'optimisation, coûteux en temps de calcul, surviennent en sciences et en ingénierie parce que l'évaluation des fonctions à optimiser nécessite souvent une simulation complexe déterministe qui est basée sur la résolution des équations qui décrivent les phénomènes physiques sous-jacents (par exemple pour les équations différentielles ordinaires ou partielles). Le bruit de calcul associés à ces simulations complexes implique que l'obtention de dérivées et parfois peu fiable, stimulant un intérêt croissant pour les techniques d'optimisation sans dérivées.

Ces dernières années, un certain nombre de méthodes d'optimisation sans dérivées ont été développées et en particulier des méthodes de région de confiance, basées sur l'interpolation, qui permettent souvent d'obtenir de bons résultats [102, 127]. Ces méthodes peuvent être classées en deux groupes : les méthodes qui visent de bonnes performances pratiques et qui, jusqu'à présent, n'avaient pas le soutien d'une théorie de convergence et les méthodes pour lesquelles la convergence globale a été démontrée (en ce sens que leur convergence vers un point stationnaire est garantie depuis tout point de départ arbitraire), mais au détriment de l'efficacité.

Dans le Chapitre 2, nous avons donné une brève introduction sur les méthodes de région de confiance, qui utilisent des polynômes quadratiques d'interpolation ou d'approximation de la fonction sur un ensemble de points où la fonction a été évaluée. Dans les méthodes de région de confiance, le modèle est supposé approximer la fonction dans un voisinage local du meilleur point trouvé par le passé. Nous avons présenté les précautions particulières doivant être prises pour contrôler la géométrie de l'ensemble d'interpolation, de manière à contrôler l'erreur entre la fonction à minimiser et son modèle. Nous avons présenté plusieurs méthodes de cette classe allant de celles de base, qui impliquent ce qu'on appelle des pas améliorant la géométrie, à une approche récente, qui utilise une propriété d'auto-correction de la géométrie de l'ensemble d'interpolation [128]. Un tel algorithme qui recourt à des pas améliorant la géométrie le moins possible (uniquement lorsque le gradient modèle est faible), tout en conservant un mécanisme de prise en compte la géométrie a été présenté.

En outre, nous avons présenté le nouvel algorithme UDFO+, qui est une version modifiée de l'algorithme avec auto-correction décrit, mais où les considérations de géométrie sont ignorées dans certaines phases de l'algorithme. Nous avons prouvé la convergence globale jusqu'aux points fixes de premier ordre de cette nouvelle version. Plus tard, dans la section numérique du chapitre 3, nous avons montré que UDFO+ dépasse des autres solveurs de différentes classes de méthodes d'OSD.

Comme contribution principale de cette thèse, nous avons étendu l'algorithme UDFO+ pour traiter les problèmes avec des bornes dans le chapitre 3, donnant ainsi naissance à l'algorithme *BCDFO+*. L'extension à des problèmes avec des bornes n'est pas aussi simple qu'on pourrait le penser, parce qu'au cours d'une optimisaiton, les points peuvent s'aligner le long des contraintes actives et par conséquent, n'appartenir qu'à une face de l'ensemble des contraintes, perdant ainsi la propriété de couverture de l'espace de minimisation : on dit alors que la géométrie de l'ensemble d'interpolation se détériore. Une idée pour contourner ce problème consiste à ajouter des points à l'ensemble qui permettrait de préserver la géométrie, impliquant ainsi plusieurs évaluations de fonctions supplémentaires. Mais comme l'idée d'utiliser la propriété d'auto-correction est de renoncer à des calculs supplémentaires pour améliorer ou préserver régulièrement la géométrie, nous avons proposé de gérer les contraintes de borne par une approche basées sur une activation des contraintes. Une telle stratégie crée la possibilité pour une méthode d'OSD basée sur des modèles d'interpolation d'économiser une bonne quantité d'évaluations de fonctions parce qu'il procède par minimisation dans des sous-espaces. Une telle approche prévoit de mettre à jour l'ensemble des contraintes actives tout en ajoutant et / ou éliminant des contraintes à chaque itération.

Notre algorithme comporte plusieurs caractéristiques intéressantes. Il est, par exemple, possible de lancer la minimisation en utilisant seulement $n + 1$ points évalués et d'augmenter graduellement l'ensemble d'interpolation lorsque la minimisation progresse. Dans les premières étapes, utiliser $p = n + 1$ valeurs de fonction est économique et souvent suffisant pour faire des progrès substantiels, tandis que considérer $p = \frac{1}{2}(n + 1)(n + 2)$ valeurs de fonction dans une étape ultérieure du calcul, peut permettre de s'approcher de la performance de la méthode de Newton. Différents types de modèles ont été proposés pour réaliser une interpolation sous- ou surdéterminée. Il s'est avéré qu'une d'interpolation basée sur un sous-espace des formes quadratiques a obtenu les meilleurs résultats sur un ensemble de problèmes-tests sans contraintes et un ensemble de problèmes-tests avec contraintes de borne issu de la collection CUTEr.

Une autre fonctionnalité a été introduite, ciblée pour économiser des évaluations de fonction lorsque les itérés rencontrent une borne active. En fait, il s'est avéré être avantageux de construire le premier modèle à l'intérieur du sous-espace non seulement à partir des vraies valeurs de la fonction mais aussi en utilisant les informations approximatives donnée par le modèle de l'espace de dimension supérieure. Par ailleurs, l'algorithme essaie de réutiliser les valeurs des fonctions à partir de points précédemment évalués chaque fois qu'un nouveau modèle dans un sous-espace ou dans l'espace-plein doit être calculé.

Nous avons également présenté le nouvel algorithme BC-MS pour résoudre les sous-problèmes dans la région de confiance de la norme ℓ_2 avec contraintes de borne où nous avons étendu la technique d'un algorithme de type Moré-Sorensen. La comparaison avec une méthode de gradient conjugué tronquée standard a montré que l'utilisation de la BC-MS comme solveur local intérieur de notre algorithme de région de confiance d'OSD *BCDFO+* n'est pas aussi efficace que d'utiliser le gradient conjugué tronqué dans une région de confiance en norme infinie.

Des expériences numériques ont été réalisées afin de comparer les algorithmes présentés aux logiciels de l'état de l'art NEWUOA et BOBYQA, qui appliquent eux aussi une méthode de région de confiance basée sur des modèles d'interpolation. Nous avons aussi rajouté à la comparaison trois méthodes de recherche directe. Notre code *BCDFO+* se compare très bien aux autres codes sur un ensemble de problèmes-tests sans contraintes et sur un ensemble de problèmes-tests avec contraintes de borne de la collection CUTEr.

Dans le Chapitre 4, nous avons étudié l'impact du bruit sur les algorithmes d'optimisation, en général, et nous ont adapté notre algorithme pour traiter des problèmes d'optimisation bruitées. Tout d'abord, nous montrons comment le niveau d'un bruit de faible amplitude dans une fonction ou un gradient peut être estimé à l'aide d'un outil qui a été initialement développé pour le calcul des dérivés d'ordre supérieure et pour estimer les erreurs d'arrondi. Cet outil est intégré dans notre algorithme pour estimer le niveau de bruit si l'utilisateur a l'intention d'optimiser une fonction objectif bruitée mais dont il n'est pas en mesure de fournir le niveau de bruit.

Nous avons aussi présenté des résultats numériques sur des ensembles de cas-tests bruités sans contraintes et sur un ensemble de cas-tests avec contraintes de borne. Pour réaliser des expériences numériques qui intègrent le bruit, nous créons un ensemble de cas-tests bruités en ajoutant des perturbations à l'ensemble des problèmes sans bruit issu de la collection CUTEr. Le choix des problèmes bruités a été guidé par le désir d'imiter les problèmes d'optimisation basés sur la simulation. Les résultats numériques confirment l'efficacité de notre algorithme sur des fonctions de type boîte noire pour lesquelles aucune structure mathématique spéciale n'est connue ou disponible. Nous évaluons les différents types de modèles d'interpolation et de régression, comme le modèle qui minimise la norme ℓ_2, le modèle qui minimise la norme Frobenius, le modèle utilisant une sous-base et le modèle de régression à l'intérieur de notre algorithme *BCDFO+*. Le modèle qui minimise la norme ℓ_2 semble d'être l'option la plus appropriée lorsque seulement un petit budget en terme d'effort de calcul n'est disponible. Par contre, le modèle de régression est à recommander fortement lorsque le budget en effort de calcul est grand car il s'est avéré être le plus robuste pour résoudre les problèmes avec une grande précision.

Par ailleurs, nous avons présenté une étude théorique sur le bruit autorisé sur un gradient qui est utilisé dans une méthode de recherche linéaire basée sur gradient. Nous avons pu établir des propriétés déterministes sur le bruit de gradient pour assurer la convergence globale des méthodes de plus grande descente et de quasi-Newton en présence d'un gradient bruité. De plus, après avoir supposé la nature gaussienne du bruit, nous avons établi une autre condition

suffisante sur la convergence globale d'une méthode de quasi-Newton qui autorise un bruit de grande amplitude sur le gradient.

La bonne performance de l'algorithme ouvre de nombreuses portes pour des recherches futures. La robustesse de notre mise en œuvre est bien sûr une invitation à chercher une théorie de la convergence globale de la méthode. Comme nous sommes intéressés à une large évaluation des modèles d'interpolation et de régression, nous voulons à l'avenir considérer l'utilisation d'un modèle qui pénalise l'écart au Hessien précédent en norme de Frobenius, ainsi que celui qui minimise la norme ℓ_1 du Hessien et de comparer ces approches aux types de modèles déjà disponibles dans notre code.

Un autre domaine évident de recherches futures, qui est motivé par notre application d'optimisation de forme, est d'étendre l'algorithme pour lui permettre de gérer des contraintes générale linéaires et non linéaires. Les techniques standard basée sur des approches SQP ou de Lagrangien augmenté pourraient être considérées à cet égard.

Bibliography

[1] MATLAB. The MathWorks Inc. `http://www.mathworks.com`.

[2] M. A. Abramson. *Pattern search algorithms for mixed variable general constrained optimization problems*. PhD thesis, Houston, TX, USA, 2003.

[3] M. A. Abramson. Second-order behavior of pattern search. *SIAM J. on Optimization*, 16:515–530, June 2005.

[4] M. A. Abramson. NOMADm version 4.6 user's guide, 2007.

[5] M. A. Abramson and Ch. Audet. Convergence of mesh adaptive direct search to second-order stationary points. *SIAM J. on Optimization*, 17:606–619, August 2006.

[6] M. A. Abramson, Ch. Audet, Jr. J. E. Dennis, and S. Le Digabel. OrthoMADS: A deterministic MADS instance with orthogonal directions. *SIAM J. on Optimization*, 20:948–966, July 2009.

[7] N. Alexandrov, J. E. Dennis Jr., R. M. Lewis, and V. Torczon. A trust region framework for managing the use of approximation models in optimization. *STRUCTURAL OPTIMIZATION*, 15:16–23, 1998.

[8] M. Andretta, E. G. Birgin, and J. M. Martínez. Practical active-set euclidian trust-region method with spectral projected gradients for bound-constrained minimization. *Optimization*, 54:305–325(21), June 2005.

[9] Ch. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *J. of Global Optimization*, 41:299–318, June 2008.

[10] Ch. Audet and Jr. J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. on Optimization*, 17:188–217, January 2006.

[11] Ch. Audet and Jr. J. E. Dennis. A progressive barrier for derivative-free nonlinear programming. *SIAM J. on Optimization*, 20:445–472, April 2009.

[12] Ch. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. on Optimization*, 17:642–664, September 2006.

[13] A. Bandeira. *Computation of Sparse Low Degree Interpolating Polynomials and their Application to Derivative-Free Optimization*. Master's thesis, University of Coimbra, Coimbra, Portugal, 2010.

[14] A. Bandeira, K. Scheinberg, and L. N. Vicente. Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization, 2011. in preparation.

[15] F. Vanden Berghen and H. Bersini. CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: experimental results and comparison with the DFO algorithm. *J. Comput. Appl. Math.*, 181:157–175, September 2005.

[16] Å. Björck. Algorithms for linear least squares problems. In E. Spedicato, editor, *Computer Algorithms for Solving Linear Algebraic Equations; The State of the Art.*, volume Vol. 77 of *NATO ASI Series F: Computer and Systems Sciences*, pages 57–92, Berlin, 1991. Springer-Verlag.

[17] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, 1996.

[18] J. Borggaard and J. Burns. A pde sensitivity equation method for optimal aerodynamic design. *J. Comput. Phys.*, 136:366–384, September 1997.

[19] D. M. Bortz and C. T. Kelley. The simplex gradient and noisy optimization problems. In *in Computational Methods in Optimal Design and Control*, pages 77–90. Birkhäuser, 1998.

[20] O. Burdakov. An MST-type algorithm for the optimal basis problem. Technical Report TR/PA/95/22, CERFACS, Toulouse, France, 1995.

[21] O. Burdakov. A greedy algorithm for the optimal basis problem. *BIT Numerical Mathematics*, 37(3):591–599, 1997.

[22] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

[23] R. H. Byrd, J. Nocedal, and R. A. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization*, pages 35–59. Springer Verlag, 2006.

[24] L. Cambier and J.-P. Veuillot. Status of the *elsA* CFD software for flow simulation and multidisciplinary applications. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, Nevada, January 2008.

[25] R. G. Carter. On the global convergence of trust region algorithms using inexact gradient information. *SIAM J. Numer. Anal.*, 28:251–265, January 1991.

[26] P. G. Ciarlet and P. A. Raviart. General Lagrange and Hermite interpolation in R^n with applications to finite element methods. *Archive for Rational Mechanics and Analysis*, 46(3):177–199, 1972.

[27] B. Colson. Trust-region algorithms for derivative-free optimization and nonlinear bilevel programming. *4OR: A Quarterly Journal of Operations Research*, 2:85–88, 2004.

[28] B. Colson and Ph. L. Toint. Exploiting band structure in unconstrained optimization without derivatives. *Optimization and Engineering*, 2:399–412, 2001.

[29] B. Colson and Ph. L. Toint. A derivative-free algorithm for sparse unconstrained optimization problems. In A. H. Siddiqi and M. Kocvara, editors, *Trends in Industrial and Applied Mathematics*, Applied Optimization, pages 131–149, Dordrecht, The Netherlands, 2002.

[30] B. Colson and Ph. L. Toint. Optimizing partially separable functions without derivatives. *Optimization Methods and Software*, 20(4-5):493–508, 2005.

[31] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM J. Numer. Anal.*, 25(2):433–460, 1988.

[32] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM J. Numer. Anal.*, 28(2):545–572, 1991.

[33] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Series on Computational Mathematics #17. Springer-Verlag Berlin, Heidelberg, New York, 1992.

[34] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[35] 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, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108, Cambridge, England, 1997.

[36] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79:397–414, 1997.

[37] A. R. Conn, K. Scheinberg, and Ph. L. Toint. A derivative free optimization algorithm in practice, 1998. Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, September 2-4.

[38] A. R. Conn, K. Scheinberg, and L. N. Vicente. Error estimates and poisedness in multivariate polynomial interpolation. Technical report, IBM T. J. Watson Research Center, USA, 2006.

[39] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of interpolation sets in derivative free optimization. *Mathematical Programming, Series A and B*, 111:141–172, 2008.

[40] 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*, 28:721–748(28), 2008.

[41] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. MPS-SIAM Series on Optimization. SIAM, Philadelphia, PA, USA, 2008.

[42] 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.

[43] A. R. Conn and Ph. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In G. Di Pillo and F. Gianessi, editors, *Nonlinear Optimization and Applications*, Plenum Publishing, pages 27–47, New York, 1996.

[44] A. R. Conn, L. N. Vicente, and Ch. Visweswariah. Two-step algorithms for nonlinear optimization with structured applications. *SIAM Journal on Optimization*, 9(4):924–947, 1999.

[45] A. L. Custódio, H. Rocha, and L. N. Vicente. Incorporating minimum Frobenius norm models in direct search. *Computational Optimization and Applications*, 46(2):265–278, 2010.

[46] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM J. on Optimization*, 18:537–555, 2007.

[47] J. E. Dennis and V. J. Torczon. Derivative-free pattern search methods for multidisciplinary design problems. AIAA Paper 94-4349, 1994.

[48] J. E. Dennis and L. N. Vicente. Trust-region interior-point algorithms for minimization problems with simple bounds. In H. Fischer, B. Riedmüller, and S. Schäffler, editors, *Applied Mathematics and Parallel Computing*, pages 97–107, Heidelberg, 1996. Physica.

[49] Jr. J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Siam, Philadelphia, 1996.

[50] D. Destarac and J. van der Vooren. Drag/thrust analysis of jet-propelled transonic transport aircraft; Definition of physical drag components. *Aerospace Science and Technology*, 8(6):545–556, 2004.

[51] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–203, Oct 2002.

[52] E. D. Dolan, J. J. Moré, and T. S. Munson. Optimality measures for performance profiles. *SIAM J. on Optimization*, 16(3):891–909, 2006.

[53] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[54] D. L. Donoho and Y. Tsaig. Fast solution of ℓ_1-norm minimization problems when the solution may be sparse, 2006.

[55] L. Driessen. *Simulation-based Optimization for Product and Process Design*. Phd thesis, Tilburg University, Tilburg, Netherlands, 2006.

[56] C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA Journal of Numerical Analysis*, 15(4):585–608, 1995.

[57] P. Erdős. Problems and results on the theory of interpolation. ii. *Acta Mathematica Hungarica*, 12:235–244, 1961.

[58] G. Fasano, J. L. Morales, and J. Nocedal. On the geometry phase in model-based algorithms for derivative-free optimization. *Optimization Methods Software*, 24(1):145–154, February 2009.

[59] K.R. Fowler, J.P. Reese, C.E. Kees, J. E. Dennis Jr., C. T. Kelley, C.T. Miller, Ch. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray, and T.G. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, 2008.

[60] A. Friedlander, J. M. Martínez, and S. A. Santos. A new trust region algorithm for bound constrained minimization. *Applied Mathematics & Optimization*, 30:235–266, 1994.

[61] J.-J. Fuchs. Recovery of exact sparse representations in the presence of bounded noise. *IEEE Transactions on Information Theory*, 51(10):3601–3608, 2005.

[62] D. M. Gay. Computing optimal locally constrained steps. *SIAM J. Sci. Statist. Comput.*, 2:186–197, 1981.

[63] J. Ch. Gilbert. Automatic differentiation and iterative processes. *Optimization Methods and Software*, 1(1):13–21, 1992.

[64] Ph. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Computing forward-difference intervals for numerical optimization. *SIAM Journal on Scientific and Statistical Computing*, 4(2):310–321, 1983.

[65] Ph. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Elsevier, June 1981.

[66] M. S. Gockenbach, D. R. Reynolds, P. Shen, and W. W. Symes. Efficient and automatic implementation of the adjoint state method. *ACM Trans. Math. Softw.*, 28:22–44, March 2002.

[67] S. M. Goldfeld, R. E. Quandt, and H. F. Trotter. Maximization by quadratic hill-climbing. *Econometrica*, 34(3):541–551, Jul. 1966.

[68] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. The Johns Hopkins University Press, 3rd edition, October 1996.

[69] 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.

[70] N. I. M. Gould, D. P. Robinson, and H. S. Thorne. On solving trust-region and other regularised subproblems in optimization. *Mathematical Programming Computation*, 2:21–57, 2010.

[71] S. Gratton, M. Mouffe, A. Sartenaer, Ph. L. Toint, and D. Tomanos. Numerical experience with a recursive trust-region method for multilevel nonlinear bound-constrained optimization. *Optimization Methods and Software*, 25(3):359–386, 2010.

[72] S. Gratton, M. Mouffe, and Ph. L. Toint. Stopping rules and backward error analysis for bound-constrained optimization. *Numerische Mathematik*, (to appear), 2011.

[73] S. Gratton, M. Mouffe, Ph. L. Toint, and M. Weber Mendonca. A recursive trust-region method for bound-constrained nonlinear optimization. *IMA Journal of Numerical Analysis*, 28(4):827–861, 2008.

[74] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optimization Methods and Software*, to appear, 2011.

[75] G. A. Gray and T. G. Kolda. Algorithm 856: Appspack 4.0: asynchronous parallel pattern search for derivative-free optimization. *ACM Trans. Math. Softw.*, 32:485–507, September 2006.

[76] A. Griewank. Computational differentiation and optimization. In J.R. Birge and K.G. Murty, editors, *Mathematical Programming : State of the Art 1994*, pages 102–131. The University of Michigan, Michigan, 1994.

[77] A. Griewank. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[78] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, Philadelphia, PA, 1991.

[79] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In *Nonlinear Optimization*, Academic Press, New-York, 1981.

[80] W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM Journal on Optimization*, 17(2):526–557, 2006.

[81] R. W. Hamming. *Numerical methods for scientists and engineers (2nd ed.)*. Dover Publications, Inc., New York, NY, USA, 1986.

[82] M. D. Hebden. An algorithm for minimization using exact second derivatives. Technical Report TP515, Atomic Energy Research Establishment, Harwell, England, 1973.

[83] R. M. Hicks and P. A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15:407–412, 1978.

[84] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms*. Springer Verlag, Heidelberg, 1996. Two volumes - 2nd printing.

[85] P. D. Hough, T. G. Kolda, and V. Torczon. Asynchronous parallel pattern search for nonlinear optimization. *SIAM J. Sci. Comput.*, 23:134–156, January 2001.

[86] W. Huyer and A. Neumaier. SNOBFIT - stable noisy optimization by branch and fit. software available at `http://www.mat.univie.ac.at/~neum/software/snobfit`, 2007.

[87] W. Huyer and A. Neumaier. SNOBFIT - stable noisy optimization by branch and fit. *ACM Trans. Math. Softw.*, 35(2):1–25, 2008.

[88] A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3(3):233–260, sep 1988.

[89] C. T. Kelley. Detection and remediation of stagnation in the nelder-mead algorithm using a sufficient decrease condition. *SIAM J. OPTIM*, 10:43–55, 1997.

[90] J. Laurenceau. *Surfaces de réponse par krigeage pour l'optimisation de formes aérodynamiques*. Phd thesis, Institut National Polytechnique de Toulouse, France, Toulouse, France, 2008. TH/CFD/08/62.

[91] J. Laurenceau and M. Meaux. Comparison of gradient and response surface based optimization frameworks using adjoint method. In *4th AIAA Multidisciplinary Design Optimization Specialist Conference*, Schaumburg, Illinois, USA, 2008.

[92] J. Laurenceau, M. Meaux, M. Montagnac, and P. Sagaut. Comparison of gradient-based and gradient-enhanced response-surface-based optimizers. *AIAA Journal*, 48(5):981–994, 2010.

[93] R. M. Lewis and V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM J. on Optimization*, 9:1082–1099, April 1999.

[94] R. M. Lewis and V. Torczon. Pattern search methods for linearly constrained minimization. *SIAM J. on Optimization*, 10:917–941, July 1999.

[95] R. M. Lewis and V. Torczon. A globally convergent augmented lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM J. on Optimization*, 12:1075–1089, April 2002.

[96] C. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[97] M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming, Series A*, 91:289–305, 2002.

[98] M. Meaux, M. Cormery, and G. Voizard. Viscous aerodynamic shape optimization based on the discrete adjoint state for 3D industrial configurations. In *European Congress on Computational Methods in Applied Sciences and Engineering*, July 2004.

[99] M. Weber Mendonca. *Multilevel Optimization: Convergence Theory, Algorithms and Application to Derivative-Free Optimization*. Phd thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 2009.

[100] E. H. Moore. On the reciprocal of the general algebraic matrix. In *THE FOURTEENTH WESTERN MEETING OF THE AMERICAN MATHEMATICAL SOCIETY*, number 26, pages 394–395. Bulletin of the American Mathematical Society, 1920.

[101] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. Statist. Comput.*, 4:553–572, 1983.

[102] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal of Optimization*, (20):172–191, 2009.

[103] M. Mouffe. *Multilevel optimization in infinity norm and associated stopping criteria*. Phd thesis, Institut National Polytechnique de Toulouse and FUNDP University of Namur, Toulouse, France and Namur, Belgium, February 2009. TH/PA/09/49.

[104] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[105] A. Neumaier. MINQ - general definite and bound constrained indefinite quadratic programming, www-document, software available at `http://www.mat.univie.ac.at/~neum/software/minq/`, 1998.

[106] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, Berlin, Heidelberg, 1999.

[107] R. Oeuvray. *Trust-region methods based on radial basis functions with application to biomedical imaging*. Phd thesis, École polytechnique fédérale de Lausanne (EPFL), Lausanne, Switzerland, 2005.

[108] R. Oeuvray and M. Bierlaire. A new derivative-free algorithm for the medical image registration problem. *International Journal of Modelling and Simulation*, 27(2):115–124, 2007.

[109] R. Oeuvray and M. Bierlaire. A doped derivative-free algorithm. *International Journal of Modelling and Simulation*, 28(4):387–393, 2008.

[110] R. Oeuvray and M. Bierlaire. BOOSTERS: a derivative-free algorithm based on radial basis functions. *International Journal of Modelling and Simulation*, 29(1):26–36, 2009.

[111] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.

[112] J. Peter, C.-T. Pham, and F. Drullion. Contribution to discrete implicit gradient and discrete adjoint method for aerodynamic shape optimization. In *European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS*, July 2004.

[113] M. Porcelli and Ph. L. Toint. BFO, a Brute Force Optimizer for mixed integer nonlinear bound-constrained optimization and its self tuning. Technical Report in preparation, 2011.

[114] M. J. D. Powell. A direct search optimization method that models the objective and constraint funcions by linear interpolation. In S. Gomez and J. P. Hennart, editors, *Advances in Optimization and Numerical Analysis*, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico, vol. 275, pages 51–67, Dordrecht, The Netherlands, 1994. Kluwer Academic Publishers.

[115] M. J. D. Powell. A direct search optimization method that models the objective by quadratic interpolation, 1994. Presentation at the 5th Stockholm Optimization Days, Stockholm.

[116] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.

[117] M. J. D. Powell. On trust region methods for unconstrained minimization without derivatives. Technical Report NA2002/02, Departement of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2002.

[118] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92:555–582, 2002.

[119] M. J. D. Powell. On the use of quadratic models in unconstrained minimization without derivatives. Technical Report NA2003/03, Departement of Applied Mathematics and Theoretical Physics, Camridge University, Cambridge, England, 2003.

[120] M. J. D. Powell. Least frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming Series B*, 100:183–215, May 2004.

[121] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In P. Pardalos, G. Pillo, and M. Roma, editors, *Large-Scale Nonlinear Optimization*,

volume 83 of *Nonconvex Optimization and Its Applications*, pages 255–297. Springer US, 2006.

[122] M. J. D. Powell. New developements of NEWUOA for minimization without derivatives. Technical report DAMPT 2007/NA05, University of Cambridge, Cambridge, England, 2007.

[123] M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA Journal of Numerical Analysis*, 28(4):649–664, 2008.

[124] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, University of Cambridge, Cambridge, England, 2009.

[125] C. H. Reinsch. Smoothing by spline functions. *Numerische Mathematik*, 10:177–183, 1967.

[126] C. H. Reinsch. Smoothing by spline functions. ii. *Numerische Mathematik*, 16:451–454, 1971.

[127] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations, 2010.

[128] 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.

[129] S. Singer and S. Singer. Efficient implementation of the Nelder-Mead search algorithm. *Applied Numerical Analysis and Computational Mathematics*, 1:524–534, December 2004.

[130] S. J. Smith. Lebesgue constants in polynomial interpolation. *Annales Mathematicae et Informaticae*, 33:109–123, 2006.

[131] D. C. Sorensen. Newton's method with a model trust region modification. *SIAM J. Numer. Anal.*, 19:409–426, 1982.

[132] G. W. Stewart. *Afternotes Goes to Graduate School*. Society for Industrial and Applied Mathematics, 1998.

[133] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.

[134] G. N. Vanderplaats. *DOT Users Manual, Version 4.20*. Vanderplaats Research & Development, Inc., Colorado Springs, USA, 1995.

[135] K. E. Vugrin. *On the effect of numerical noise in simulation-based optimization*. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2003.

[136] K. E. Vugrin. *On the effects of noise on parameter identification optimization problems.* Phd thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2005.

[137] S. M. Wild. *Derivative-Free Optimization Algorithms for Computationally Expensive Functions.* Phd thesis, Cornell University, Ithaca, NY, USA, 2008.

[138] S. M. Wild. MNH: a derivative-free optimization algorithm using minimal norm hessians. In *Tenth Copper Mountain Conference on Iterative Methods*, April 2008.

[139] S. M. Wild, R. G. Regis, and Chr. A. Shoemaker. ORBIT: optimization by radial basis function interpolation in trust-regions. *SIAM J. on Scientific Computing*, 30(6):3197–3219, 2008.

[140] S. M. Wild and Chr. A. Shoemaker. Global convergence of radial basis function trust-region algorithms. Technical Report Preprint ANL/MCS-P1580-0209, Mathematics and Computer Science Division, February 2009.

[141] D. Winfield. *Function and functional optimization by interpolation in data tables.* Phd thesis, Havard University, Cambridge, USA, 1969.

[142] D. Winfield. Function minimization by interpolation in a data table. *IMA Journal of Applied Mathematics*, 12:339–347, 1973.

[143] G. Zoutendijk. Nonlinear Programming, Computational Methods. In J. Abadie, editor, *Integer and Nonlinear Programming*, pages 37–86, North-Holland, Amsterdam, 1970.