UNIVERSIDADE D
COIMBRA

Lili Song

Modeling Hessian-Vector Products in Nonlinear Optimization: New Hessian-Free Methods

Tese no âmbito do Programa Inter-Universitário de Doutoramento em Matemática, orientada pelo Professor Doutor Luís Nunes Vicente e pelo Professor Doutor João Eduardo da Silveira Gouveia e apresentada ao Departamento de Matemática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2021

Modeling Hessian-Vector Products in Nonlinear Optimization: New Hessian-Free Methods

Lili Song

UNIVERSIDADE D COIMBRA

U.PORTO

UC|UP Joint PhD Program in Mathematics

Programa Interuniversitário de Doutoramento em Matemática

PhD Thesis | Tese de Doutoramento

January 2022

Acknowledgements

I am especially indebted to my supervisor, Professor Luis Nunes Vicente. As my mentor, he provided me encouragement and was always willing to enthusiastically assist in any way he could during the years we have worked together. His rigor, innovation, and intensity in doing research have benefited me a lot. He has taught me more than I could ever give him credit for.

I would also like to thank Dr. Tommaso Giovannelli who has greatly helped me further improve my writing skills and the overall presentation of this dissertation.

I would also like to extend my gratitude to all my PhD classmates, with whom I have shared moments of excitement but also anxiety. Their presence was important in a process that is often felt as solitaire. Thanks should go to António, Ali, Mbouna, and Mina, who provided me with their assistance and were so helpful in numerous ways. My classmates and great friends Daniela and Sara helped me to adapt to the life in a foreign country, with whom I had the best tea breaks in my life.

Special words of gratitude go to all my friends in Coimbra and Lehigh who gave me the necessary distractions from my research and shared a memorable time together. Especially Xinli and Suyun, who gave me warm encouragement and always managed to make me feel special. I would also like to extend my gratitude to my landlord Teresa and roommate Xinkai, thanking them for giving advice, listening, and supporting me through this entire process.

I am also grateful to *Fundação para a Ciência e Tecnologia* and the Department of Mathematics of the University of Coimbra for the doctoral fellowship, and to the Centre for Mathematics of the University of Coimbra for co-financing my stay at Lehigh University. Finally, I thank the Department of Industrial and Systems Engineering of Lehigh University for the congenial scientific environment provided. Being in one of the best Mathematical Optimization centres on earth was a memorable experience!

Nobody has been more important to me in the pursuit of this project than the members of my family. A special word of thanks goes to my parents, Shaoquan and Jixiang, whose love and support are with me whatever I do. They are my models. Most importantly, I have to thank my loving and supportive husband, Xiaolei, who has been by my side throughout my PhD education, and kept things going on so that I could focus on my research. The very last word goes to my baby boy, Ethan, who has been in my life for the last year and who has given me the extra strength and motivation to get things done.

Table of contents

List of figures vii

List of tables ix

1 Introduction 1

2 Review of algorithms for unconstrained nonlinear optimization 5
2.1 Overview of algorithms for unconstrained nonlinear optimization . . . 5
2.2 Line-search methods . . . 6
2.2.1 Inexact line search techniques . . . 7
2.2.2 Interpolation methods in line searches . . . 9
2.3 Trust-region methods . . . 12
2.4 Inexact Newton methods . . . 15
2.4.1 Line-search Newton-CG method . . . 17
2.4.2 Trust-region Newton-CG method . . . 18
2.5 Quasi-Newton methods . . . 19
2.5.1 The BFGS method . . . 20

3 Review of interpolation models for derivative-free optimization 25
3.1 Generalities on interpolation models . . . 25
3.2 Linear interpolation . . . 26
3.2.1 Error bounds in linear interpolation . . . 28
3.3 Quadratic interpolation . . . 29
3.3.1 Error bounds in quadratic interpolation . . . 31
3.4 Minimum Frobenius norm quadratic models . . . 33
3.4.1 Least Frobenius norm updating of quadratic models . . . 35

4 Existing Hessian approximation attempts in nonlinear optimization 37
4.1 Overview of Hessian approximation attempts . . . 37
4.2 Approximating the Hessian matrix by finite differences . . . 38
4.2.1 Approximating the gradient . . . 38
4.2.2 Approximating the Hessian . . . 39
4.3 Approximating a sparse Hessian matrix with optimal hereditary properties . . . 42
4.4 Approximating a sparse Hessian matrix by solving componentwise secant equations . 44

5 Hessian recovery from Hessian-vector products **49**
5.1 Hessian recovery . . . 49
5.2 Theoretical motivation . . . 50
5.3 Numerical results for the determined case . . . 53
5.4 Numerical results for the determined case when the Hessian sparsity is known . . . 56
5.5 Recovery cost in the general case . . . 58

6 Newton direction recovery from Hessian-vector products **59**
6.1 Newton direction recovery . . . 59
6.2 Theoretical motivation . . . 60
6.3 Numerical results for the determined case using a correction . . . 64

7 Exploring other ideas **67**
7.1 Inverse Hessian recovery from Hessian-vector products . . . 67
7.1.1 Theoretical motivation . . . 68
7.1.2 Re-using previous Hessian-vector products . . . 71
7.1.3 Recovery cost of the Newton direction . . . 71
7.2 On the numerical linear algebra of the Newton direction recovery . . . 73
7.3 Modified Newton direction recovery from Hessian-vector products . . . 75

8 Conclusions **77**

References **79**

Appendix A **83**
A.1 Performance profiles . . . 83

Appendix B **85**
B.1 Very small test problems . . . 85
B.2 Small sparse test problems . . . 85
B.3 Small test problems . . . 86
B.4 Average CPU times . . . 86

List of figures

5.1 Testing the Hessian recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of very small problems of Appendix B.1. The value of p was set to $\frac{n^2+n}{2} - n$. 56

5.2 Testing the Hessian recovery within a line-search algorithm. Performance profiles for number of function evaluations and CPU time, for the set of very small problems of Appendix B.1. The value of p was set to $\frac{n^2+n}{2} - n$. 56

5.3 Testing the Hessian recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of small sparse problems of Appendix B.2. The value of p was set to number of nonzeros minus n. . . 57

5.4 Testing the Hessian recovery within a line-search algorithm. Performance profiles for number of function evaluations and CPU time, for the set of small sparse problems of Appendix B.2. The value of p was set to number of nonzeros minus n. 58

6.1 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of very small problems of Appendix B.1. 64

6.2 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of function evaluations and CPU time, for the set of very small problems of Appendix B.1. 65

6.3 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of small problems of Appendix B.3. 65

6.4 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of function evaluations and CPU time, for the set of small problems of Appendix B.3. 65

List of tables

B.1 List of 48 very small CUTEst test problems 85
B.2 List of 12 sparse small CUTEst test problems 85
B.3 List of 26 small CUTEst test problems 86
B.4 Average CPU times (s). 86

Chapter 1

Introduction

Smooth problems with a twice continuously differentiable objective function frequently arise in the unconstrained nonlinear optimization literature and in its applications. Efficient strategies to address this class of problems are typically based on Newton's method, which requires the solution of a linear system at each iteration. The matrix of this system is the Hessian of f and its right-hand side is the negative of the gradient. In many application instances, the Hessian is not available for factorization or is too large to factorize at a reasonable cost, but Hessian-vector products are available and affordable. In such cases, the linear systems cannot be solved directly, but an iterative method can be applied. When solving certain application problems, it becomes relatively cheap to compute true Hessian-vector products for arbitrary vectors using the problem's structure. Examples are problems governed by differential equations [1, 6, 43] and the training of neural networks for deep learning [45, 48]. When the problem structure cannot be used to calculate Hessian-vector products, one can use techniques from numerical analysis and computer science to accurately compute these products, either by applying finite-differences using gradient calls [33, 52], or by using automatic differentiation techniques (see [4, 7, 38]), in particular those tailored to the calculation of Hessian-vector products [26, 41].

When one is using an iterative method to solve the linear system, it is known that there is a residual error in the application of the iterative solver and that such a residual can be made smaller by asking more from the solver. This reasoning gave rise to the so-called inexact or truncated Newton methods, which have formed an important numerical tool for many decades. (We will compare the proposed approaches against the inexact Newton method.) It is well known since the contribution [21] what conditions one should impose on the norm of the residual of the linear system to obtain linear, superlinear, or quadratic local convergence in the iterates of the underlying method (see [52]). Global convergence of inexact Newton methods is also well studied [27, 40]. One knows well also how to deal with negative curvature while solving the linear system using Krylov-type methods (conjugate gradients or Lanczos), either using a trust-region technique [35, 61] or a line search [50].

When Hessian or Hessian-vector products are not available, estimating the Hessian within an optimization approach can then play an important role, however the existing approaches are not entirely satisfactory. If the Hessian matrix is sparse and its sparsity pattern is known, the approach in [30] enforces multiple secant equations in a least squares sense, solving then a positive semi-definite system of equations in the nonzero Hessian components. Their approach does not show a significant improvement compared to the L-BFGS or Newton trust-region methods. In [58] the

Hessian is estimated by finite differences in the gradient, but by dividing the Hessian columns first into groups. Using symmetry and the known sparsity of the Hessian, it is possible to find approximations to different Hessian columns at once. This method is cheap in computer arithmetic and provided better results when compared to [20]. A more recent approach [34] imposes the secant equations componentwise, leading to fewer equations when taking into account the available sparsity pattern. The numerical results show that the algorithm can find the Hessian approximation fast and accurately when the number of nonzero entries per row is relatively low.

In this dissertation, two techniques are proposed and analyzed for the Hessian-free scenario where only Hessian-vector products are available for use. Our goal is to use as few of these products as possible without losing the ability to converge to a solution or a stationary point of the original problem. Having this in mind we form a quadratic model around a point x, using function and gradient values at x and function values at the interpolating points y^ℓ, $\ell = 1,\ldots,p$. The Hessian matrix H of this model or some kind of Newton step has then to be recovered.

Our first approach enriches these interpolating conditions with the information coming from a single true Hessian-vector product $\nabla^2 f(x)(y-x)$, for a point y different from any of the y^ℓ's of those conditions. In fact, to avoid degeneracy in the enriched interpolating conditions (which are affine conditions on H), one has to choose y differently from those y^ℓ's and one cannot consider more than one of these products. The computation of the model Hessian is carried out by minimizing its norm or its distance to a previous model Hessian (say from a previous iteration of the optimization method)[1] subject to the enriched interpolating conditions. Such a Hessian recovery can then lead to the computation of an approximate Newton step.

Our second approach allows us to consider more than one Hessian-vector product in the model formulation. The interpolating conditions are now enriched by the second-order information coming from the Hessian-vector products $\nabla^2 f(x)(y^\ell - x)$, $\ell = 1,\ldots,p$. Then, avoiding degeneracy and the inverse of the Hessian model, the recovery is done in the space of the Newton direction models, using a modified set of enriched interpolating conditions. Again, the computation of the Newton direction model is carried out by minimizing its norm or its distance to a previous Newton direction model subject to the modified enriched interpolating conditions.

In both cases we will provide some theoretical support for the recoveries by proving that the absolute error (in model Hessian or in model Newton direction) is decreasing in the case where the enriched interpolating conditions are underdetermined. The main results will be established for the case where f is quadratic but theoretical insight is given for the non-quadratic case as well. We will also provide accuracy-type upper bounds for the absolute error coming from the enriched interpolating conditions (in a determined situation). We report numerical results to confirm that both approaches are sound and can lead to a significant reduction in the number of Hessian-vector products. The dimension of the problems tested is rather small. The linear algebra is dense, and the number of function evaluations used can be relatively high. It is left for future research the application to

[1] The case of minimizing the distance to a previous model Hessian resembles the spirit of quasi-Newton methods [22, 23, 28] when they are motivated by least-change secant updating principles [24]. Limited memory quasi-Newton methods [46, 51] have been extensively used for large-scale problems since they avoid the storage and the factorization updates of $n \times n$ secant/quasi-Newton approximation matrices. Anyhow, quasi-Newton or limited memory quasi-Newton techniques are typically applied when no second-order information whatsoever is available.

medium/large-scale problems. The second approach based on a Newton direction model can be also easily parallelized.

The two main approaches proposed in this dissertation pave the way for three ideas that can be explored to extend or improve the methodology underlying the recovery problems. In particular, two additional recovery approaches can be developed based on the enriched interpolating conditions used in the Newton direction recovery problem. A first approach concerns the possibility of determining both the Newton direction and the Hessian inverse by applying an appropriate transformation to the interpolating conditions. The idea is to recover the product of the whole inverse times a vector rather than storing the whole inverse, which is computationally expensive. A second approach aims to recover a modified Newton direction where a multiple of the identity matrix is added to the Hessian to ensure positive definiteness, as required by line-search methods to obtain global convergence. Such a direction can be obtained by a further adjustment of the interpolating conditions. Finally, the recovery of the Newton direction may be enhanced by adopting an iterative solver, which may allow for parallel procedures in order to improve the overall performance.

This thesis is organized as follows. In Chapters 2 and 3, we briefly review algorithms for unconstrained nonlinear optimization and interpolation models used in derivative-free optimization, respectively. In Chapter 4, the most popular Hessian approximation approaches in nonlinear optimization are presented. Chapters 5 and 6 focus on the Hessian-free scenario where only Hessian-vector products are available. In particular, in Chapter 5 we present the approach for recovering the model Hessian, while in Chapter 6 we describe the approach for recovering the model Newton direction. For both cases, we report illustrative numerical results on small problems. Some other ideas to extend or improve the general methodology are explored in Chapter 7. The thesis is finished in Chapter 8 with some final remarks and conclusions.

The notation $O(A)$ will be used to represent the product of a constant times A whenever the multiplicative constant is independent of A. All vector and matrix norms are Euclidean, unless otherwise specified. The notation $A^{\dagger}$ is used to denote the Moore-Penrose generalized inverse of a matrix A.

Chapter 2

Review of algorithms for unconstrained nonlinear optimization

This chapter reviews the main classes of methods for unconstrained nonlinear optimization, focusing on the algorithms that are of interest for the methodology proposed in this thesis. It is mainly based on the books [52] and [62].

2.1 Overview of algorithms for unconstrained nonlinear optimization

Given the following unconstrained minimization problem

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

where $f : \mathbb{R}^n \to \mathbb{R}$, all the algorithms considered in this chapter start from an initial point, denoted by x_0, and produce a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ until a reasonable stopping criterion is satisfied. Such a criterion is satisfied when either no more progress can be made by the algorithm or an optimal solution has been approximated with sufficient accuracy. To move from the current point x_k to the next iterate, information about the function f at x_k may be used together with ∇f computed at the same point, according to the class of the algorithm chosen. The evolution of the algorithm may be determined also by exploiting the information from the previous iterates $x_0, x_1, \cdots, x_{k-1}$.

Two fundamental strategies are adopted to develop algorithms for unconstrained nonlinear optimization: *line-search* and *trust-region*. The former strategy is based on determining a suitable stepsize along given search directions, while the latter deals with optimizing a model of the objective function within a ball where the model can be considered a good approximation. To some extent, the two strategies can be viewed as opposite: while in the line-search algorithms the first decision is the search direction and then a suitable stepsize is computed, in trust-region methods first the radius of the ball is chosen and then a step optimizing the model of the function within the resulting trust region is determined. Line-search methods are described in Section 2.2, while trust-region algorithms are reviewed in Section 2.3. Moreover, examples of these types of algorithms are given in Sections 2.4 and 2.5, where we focus on *inexact Newton methods* and *quasi-Newton methods*, which show remarkable convergence properties despite relaxing some properties of Newton's method. In particular, while the

former methods drop the requirement for an exact solution of the Newton system, the latter do not require the knowledge of the Hessian matrix.

Before going into details, it is important to introduce the terminology used to measure the performance of the algorithms in terms of rate of convergence. To this purpose, let $\{x_k\}$ be a sequence in $\mathbb{R}^n$ that converges to x^*. The convergence is said *linear* (or Q-linear) if a constant $r \in (0,1)$ exists such that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r, \text{ for all } k \text{ sufficiently large.}$$

The convergence is said *superlinear* (or Q-superlinear) if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0, \text{ for all } k \text{ sufficiently large.}$$

The convergence is said *quadratic* (or Q-quadratic) if a positive constant M (not necessarily less than 1) exists such that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M, \text{ for all } k \text{ sufficiently large.}$$

Note that a quadratically convergent sequence converges faster than a superlinearly convergent sequence, which, in turn, converges faster than a linearly convergent sequence.

2.2 Line-search methods

In a generic iteration k, line-search methods choose a direction d_k and, starting from the current iterate x_k, search a new iterate with a lower objective function value along this direction. The step used to move along d_k can be determined by solving the following problem

$$\min_{\alpha>0} f(x_k + \alpha d_k), \tag{2.2}$$

which allows finding a positive scalar α, referred to as the *step length* or the *stepsize*. If at each iteration α_k is required to exactly solve problem (2.2) in the direction d_k (i.e., $\alpha_k = \alpha^*$), the line search is said *exact* and α_k is the *optimal step length*. Therefore, the maximum benefit is obtained from the direction d_k since the optimal stepsize allows achieving the point with the minimum function value along d_k. However, computing an exact solution for problem (2.2) may be very expensive and, in practice, unnecessary since an approximate solution allows achieving satisfactory performance by requiring a lower number of function evaluations. Hence, we usually find an approximate solution α_k of problem (2.2) that leads to a new iterate where the objective function value is lower with respect to the current point, i.e., $f(x_k) - f(x_k + \alpha_k d_k) > 0$. Since the optimality requirement is relaxed, such a line search is called *inexact* or *approximate*.

The iteration of a line-search algorithm can be written as follows

$$x_{k+1} = x_k + \alpha_k d_k. \tag{2.3}$$

Once a new point is obtained, a new search direction and step length are computed based on the information collected, and the process is then repeated. An effective choice of both the search direction

d_k and the step length α_k guarantees the success of the line search. In particular, most line-search algorithms require d_k to be a *descent direction*, i.e., $d_k^T \nabla f(x_k) < 0$, which ensures that the value of the objective function f can be reduced along d_k. Several options are available for the choice of the search direction, and a different algorithm is associated with each of them.

The steepest-descent direction $-\nabla f(x_k)$ ensures the largest amount of decrease in f. The resulting optimization algorithm is referred to as the *steepest-descent method*. Although the anti-gradient appears to be a reasonable choice, the rate of convergence of the steepest-descent method towards the optimal solution is linear, which is a slow rate. Therefore, a better choice as a search direction is given by

$$d_k = -H_k^{-1} \nabla f(x_k), \tag{2.4}$$

where H_k is a symmetric and nonsingular matrix. While in *Newton's method* H_k is the exact Hessian $\nabla^2 f(x_k)$ of the function f at x_k, in *quasi-Newton methods* H_k is an approximation to the Hessian that is updated at each iteration by means of the additional knowledge obtained. Both types of directions guarantee fast rates of convergence towards the optimal solution due to the use of the Hessian or an approximation to it, thus demonstrating the importance of the knowledge of the second-order derivatives. In particular, while Newton's method shows a quadratic local convergence rate, in quasi-Newton method a superlinear rate is achieved. Note that the steepest-descent method can be obtained from (2.4) by simply considering the identity matrix I as H_k. The direction d_k is a descent direction if H_k is a positive definite matrix since $d_k^T \nabla f(x_k) = -\nabla f(x_k)^T H_k^{-1} \nabla f(x_k) < 0$ (assuming that $\nabla f(x_k)$ is not zero).

An important class of algorithms for solving large-scale problems is represented by *nonlinear conjugate gradient methods*, whose search direction can be expressed as

$$d_k = -\nabla f(x_k) + \beta_k d_{k-1},$$

where the scalar β_k ensures that the directions d_k and d_{k-1} are conjugate with respect to H_k, i.e., $d_k^T H_k d_{k-1} = 0$, which is a property that plays a crucial role. Although these methods do not show the fast local convergence rate of Newton's method or quasi-Newton methods, storage of matrices is not required, thus showing a significant advantage.

Once a search direction d_k is fixed, a suitable stepsize must be chosen to identify the next iterate along d_k. Sections 2.2.1 and 2.2.2 review the conditions used to identify good values of the stepsize and the main techniques adopted to determine such values.

2.2.1 Inexact line search techniques

Given a search direction d_k, the goal of an exact line search is to find a step length α such that

$$f(x_k + \alpha_k d_k) = \min_{\alpha > 0} f(x_k + \alpha d_k), \tag{2.5}$$

or,

$$\alpha_k = \min\{\alpha | \nabla f(x_k + \alpha d_k)^\top d_k = 0, \alpha > 0\}.$$

In other words, it aims to find the value of α that minimizes the function f along d_k.

Let us define the function

$$\phi(\alpha) = f(x_k + \alpha d_k),\ \alpha > 0. \tag{2.6}$$

In general, solving the one-dimensional problem in (2.5) to optimality is not very effective, especially when the iterate is far away from the optimal solution of the original unconstrained problem, due to the significant number of function evaluations required. Therefore, in many optimization methods, a step length is accepted as long as it allows achieving a sufficient decrease of the objective function value. Although problem (2.5) is not solved exactly, the overall performance of the optimization algorithm is not affected and a noticeable saving in the computational effort is obtained.

There are various termination conditions for line search algorithms. A very simple one is to choose α_k such that $f(x_k + \alpha_k d_k) < f(x_k)$. However, for some problems this is not sufficient to guarantee that the sequence of iterates produced by the algorithm will converge to the optimal solution x^*. To avoid this behavior, we have to enforce a *sufficient decrease* condition, which is given by

$$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^\top d_k, \tag{2.7}$$

where $c_1 \in (0,1)$. The sufficient decrease condition requires that the reduction of function value f is proportional to both the step length α_k and the directional derivative $\nabla f(x_k)^\top d_k$. Inequality (2.7) is also called the *Armijo condition*. Note that the sufficient decrease condition cannot ensure that the algorithm performs steps that are sufficiently large and makes reasonable progress. Indeed, since in line-search methods d_k is supposed to be a descent direction, even small values of α satisfy the Armijo condition. To avoid that small values of α are accepted, a second condition needs to be introduced by requiring α_k to satisfy

$$\nabla f(x_k + \alpha_k d_k)^\top d_k \geq c_2 \nabla f(x_k)^\top d_k, \tag{2.8}$$

where $c_2 \in (c_1, 1)$. The inequality (2.8) is called the *curvature condition*. It is easy to find that the left-hand side of (2.8) is the derivative $\phi'(\alpha_k)$. That is to say, the curvature condition requires the slope of ϕ at α_k to be greater than or equal to c_2 times the slope $\phi'(0)$. Stopping the line search when this condition is satisfied makes sense because the function value f can be reduced significantly by moving along the direction if the slope $\phi'(\alpha)$ is strongly negative. On the other hand, it means that the function value f cannot be reduced too much if $\phi'(\alpha_k)$ is only slightly negative or even positive. Therefore, it is reasonable to stop the line search procedure. In Newton method or quasi-Newton method, c_2 is typically chosen equal to 0.9, while in nonlinear conjugate gradient methods, c_1 is typically fixed to 0.1.

The sufficient decrease and curvature conditions are also referred to as the *Wolfe conditions*. We rewrite them for the sake of clarity as

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^\top d_k, \tag{2.9a}$$

$$\nabla f(x_k + \alpha_k d_k)^\top d_k \geq c_2 \nabla f(x_k)^\top d_k, \tag{2.9b}$$

where $0 < c_1 < c_2 < 1$.

An issue with (2.9a) and (2.9b) is that the stepsize may satisfy the Wolfe conditions even if it is not close to a minimizer of ϕ. To resolve this issue, we can force α_k to lie in a broad neighborhood of

a local minimizer or stationary point of ϕ by requiring the *strong Wolfe conditions*:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^\top d_k, \tag{2.10a}$$

$$|\nabla f(x_k + \alpha_k d_k)^\top d_k| \leq c_2 |\nabla f(x_k)^\top d_k|, \tag{2.10b}$$

where $0 < c_1 < c_2 < 1$. In these conditions, we can exclude points that are far away from the stationary points of ϕ by ruling out the step length α_k whose derivative $\phi'(\alpha_k)$ is too positive. Therefore, while the Wolfe conditions ensure that the stepsize is not too short and the sufficient decrease of the objective function is achieved, the strong Wolfe conditions ensure that the new points are not very far away from the stationary points of ϕ. Lemma 2.2.1 ensures the existence of step lengths satisfying the Wolfe conditions.

Lemma 2.2.1. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. Let d_k be a descent direction at x_k, and assume that f is bounded below along the ray $\{x_k + \alpha d_k | \alpha > 0\}$. Then, if $0 < c_1 < c_2 < 1$, there exist intervals of step lengths satisfying the Wolfe conditions (2.9) and the strong Wolfe conditions (2.10).*

It is important to point out that considering the sufficient decrease condition (2.7) without the curvature condition is not enough to guarantee that the algorithm can make reasonable progress along the given search direction. However, we can dispense with the extra condition (2.9b) and only use the sufficient decrease condition to terminate the line search procedure by appropriately choosing the step lengths of the line-search algorithm. To this end, we introduce the so-called *backtracking line search* as follows

Algorithm 1: Backtracking Line Search

Choose $\bar{\alpha} > 0, \rho \in (0,1), c \in (0,1)$.
Set $\alpha \leftarrow \bar{\alpha}$.
repeat until $f(x_k + \alpha d_k) \leq f(x_k) + c\alpha \nabla f(x_k)^\top d_k$
    $\alpha \leftarrow \rho \alpha$.
end (repeat)
Terminate with $\alpha_k = \alpha$.

When we use Algorithm 1 in Newton and quasi-Newton methods, we usually choose the step length $\bar{\alpha}$ equal to 1. However, in other algorithms, such as the conjugate gradient methods and the steepest-descent method, different values can be chosen. Note that, if d_k is a descent direction, an acceptable step length is eventually found since at the end of the line search α_k becomes small enough to satisfy the sufficient decrease condition (2.7). Moreover, in practice, a different contraction factor ρ can be chosen in each iteration of the line search. Note that the accepted value α_k is either equal to the initial stepsize $\bar{\alpha}$ or within a factor ρ from the previous trial value, i.e., α_k/ρ, which was rejected for being overly large and, accordingly, for violating the sufficient decrease condition.

2.2.2 Interpolation methods in line searches

The idea of interpolation methods used in line-search algorithms is to approximate the univariate function $\phi(\alpha) = f(x + \alpha d)$ by first fitting a quadratic or cubic polynomial in α to known data ($\phi(\hat{\alpha})$

and $\phi'(\hat{\alpha})$ at some point $\hat{\alpha}$), and then choosing a new value for α by minimizing the fitted polynomial. In particular, every algorithm of this type is composed of two phases: a *bracketing phase*, which aims to find an interval where acceptable values of the stepsize are contained, and a *selection phase*, which has the goal to determine the final value of the stepsize within this interval. In the selection phase, the bracketing interval is reduced by using the information collected across the iterations of the algorithm. Generally speaking, the interpolation methods enhance the procedure described in Algorithm 1 and are superior to the section algorithms (such as the golden section method and the Fibonacci method, which are based only on the comparison of the function values, as described in [62]) when the function has good analytical properties, such as derivatives that are easily available.

Quadratic interpolation methods

Suppose that the interpolation function in α is constructed by using a quadratic polynomial as follows

$$q(\alpha) = a\alpha^2 + b\alpha + c. \tag{2.11}$$

We aim to find a, b, and c by using the data at hand. To this end, three cases are possible according to the number of points used for the interpolation and the known data.

(1) *Quadratic Interpolation Method with Two Points (I)*
Given two points α_1 and α_2, their function values $\phi(\alpha_1)$, and $\phi(\alpha_2)$, and one of the derivative values $\phi'(\alpha_1)$ and $\phi'(\alpha_2)$ (without loss of generality, we assume to know $\phi'(\alpha_1)$), we can write the following system

$$\begin{aligned} q(\alpha_1) &= a\alpha_1^2 + b\alpha_1 + c = \phi(\alpha_1), \\ q(\alpha_2) &= a\alpha_2^2 + b\alpha_2 + c = \phi(\alpha_2), \\ q'(\alpha_1) &= 2a\alpha_1 + b = \phi'(\alpha_1). \end{aligned} \tag{2.12}$$

Solving (2.12), we determine a, b, and c. We can show that $q(\alpha)$ achieves its minimum at the following point

$$\bar{\alpha} = -\frac{b}{2a} = \alpha_1 - \frac{1}{2}\frac{(\alpha_1 - \alpha_2)\phi'(\alpha_1)}{\phi'(\alpha_1) - \frac{\phi(\alpha_1)-\phi(\alpha_2)}{\alpha_1-\alpha_2}}. \tag{2.13}$$

Hence, the iteration formula can be written as follows

$$\alpha_{k+1} = \alpha_k - \frac{1}{2}\frac{(\alpha_k - \alpha_{k-1})\phi_k'}{\phi_k' - \frac{\phi_k-\phi_{k-1}}{\alpha_k-\alpha_{k-1}}}, \tag{2.14}$$

where $\phi_k = \phi(\alpha_k)$, $\phi_{k-1} = \phi(\alpha_{k-1})$ and $\phi_k' = \phi'(\alpha_k)$.

(2) *Quadratic Interpolation Method with Two Points (II)*
Given two points α_1 and α_2, and one of the function values $\phi(\alpha_1)$ and $\phi(\alpha_2)$ (without loss of generality, we assume to know $\phi(\alpha_1)$), and two derivative values $\phi'(\alpha_1)$ and $\phi'(\alpha_2)$. With the same

discussion as above, we obtain that

$$\bar{\alpha} = -\frac{b}{2a} = \alpha_1 - \frac{\alpha_1 - \alpha_2}{\phi'(\alpha_1) - \phi'(\alpha_2)}\phi'(\alpha_1). \tag{2.15}$$

Thus, the iteration scheme is

$$\alpha_{k+1} = \alpha_k - \frac{\alpha_k - \alpha_{k-1}}{\phi'_k - \phi'_{k-1}}\phi'_k. \tag{2.16}$$

We can prove (see, e.g., [62, Theorem 2.4.1]) that if ϕ is three-times continuously differentiable, the sequence $\{\alpha_k\}$ generated by (2.16) converges to a point α^*, such that $\phi'(\alpha^*) = 0$ and $\phi''(\alpha^*) \neq 0$, with an order of convergence equal to 1.618.

(3) *Quadratic Interpolation Method with Three Points*
Given three distinct points α_1, α_2, and α_3, and their function values, we can construct the quadratic interpolation function by using the following conditions

$$q(\alpha_i) = a\alpha_i^2 + b\alpha_i + c = \phi(\alpha_i), \ i = 1,2,3. \tag{2.17}$$

Once we determine a and b by solving equations (2.17), we can write

$$\bar{\alpha} = -\frac{b}{2a} = \frac{1}{2}(\alpha_1 + \alpha_2) + \frac{1}{2}\frac{(\phi(\alpha_1) - \phi(\alpha_2))(\alpha_2 - \alpha_3)(\alpha_3 - \alpha_1)}{(\alpha_2 - \alpha_3)\phi(\alpha_1) + (\alpha_3 - \alpha_1)\phi(\alpha_2) + (\alpha_1 - \alpha_2)\phi(\alpha_3)}. \tag{2.18}$$

We can prove (see, e.g., [62, Theorem 2.4.3]) that if ϕ has continuous fourth-order derivatives, the sequence $\{\alpha_k\}$ generated by (2.18) converges to a point α^*, such that $\phi'(\alpha^*) = 0$ and $\phi''(\alpha^*) \neq 0$, with an order of convergence equal to 1.32.

We can reduce the bracketing interval by comparing α_{k+1} with α_k after obtaining the new α_{k+1}. The procedure will continue until the length of the interval is less than a prescribed tolerance.

Cubic interpolation method

The cubic interpolation method approximates the objective function $\phi(\alpha)$ by a cubic polynomial. Four interpolation conditions are required to construct the cubic polynomial $p(\alpha)$. We can use either function values at four different points, three function values and one derivate value at one point, or function values and derivate values at two different points. It is worth mentioning that, in general, the cubic interpolation method has better convergence than the quadratic interpolation method. However, more information is required, thus resulting in a more expensive computation. Therefore, this kind of interpolation is often used for smooth functions.

We now focus on the cubic interpolation method that uses the function and derivative values at two points. In particular, suppose that we know two points a and b, their function values $\phi(a)$ and $\phi(b)$, and their derivative values $\phi'(a)$ and $\phi'(b)$. The cubic interpolation polynomial can be constructed as follows

$$p(\alpha) = c_1(\alpha - a)^3 + c_2(\alpha - a)^2 + c_3(\alpha - a) + c_4, \tag{2.19}$$

where $c_1,\ c_2,\ c_3$, and c_4 are coefficients of the polynomial. From the interpolation conditions on the function and derivative values we obtain

$$\begin{aligned} p(a) &= c_4 = \phi(a), \\ p(b) &= c_1(b-a)^3 + c_2(b-a)^2 + c_3(b-a) + c_4 = \phi(b), \\ p'(a) &= c_3 = \phi'(a), \\ p'(b) &= 3c_1(b-a)^2 + 2c_2(b-a) + c_3 = \phi'(b). \end{aligned} \tag{2.20}$$

Moreover, since we want a minimizer of $p(\alpha)$, we write the following sufficient optimality conditions

$$\begin{aligned} p'(\alpha) &= 3c_1(\alpha-a)^2 + 2c_2(\alpha-a) + c_3 = 0, \\ p''(\alpha) &= 6c_1(\alpha-a) + 2c_2 > 0. \end{aligned} \tag{2.21}$$

By combining (2.20) and (2.21), according to the steps described in [62], we obtain the minimizer of $p(\alpha)$, namely,

$$\bar{\alpha} = a + (b-a)\frac{w - \phi'(a) - z}{\phi'(b) - \phi'(a) + 2w}, \tag{2.22}$$

where $z = 3\frac{\phi(b)-\phi(a)}{b-a} - \phi'(a) - \phi'(b)$ and w satisfies $w^2 = z^2 - \phi'(a)\phi'(b)$. It is important to point out that [62] states that the cubic interpolation method with two points has a convergence rate with order 2.

2.3 Trust-region methods

The main idea of trust-region methods is to build an approximate model of the objective function around the current iterate and determine the next iterate by minimizing the model within a suitable region, which is called a *trust region*, where the model is supposed to be a sufficiently accurate approximation of the objective function (see, e.g., [15] and [52] for an extensive analysis of these methods). While line-search methods find first a search direction and then select a suitable step length along this direction, trust-region methods adopt a different perspective by determining the direction and the stepsize at the same time.

To ensure that trust-region methods are able to converge to critical points regardless of the initial point (global convergence property), the model used to approximate the objective function is required to satisfy Taylor-like error bounds on the quality of the approximation. Let $\Delta > 0$ be the radius of the trust region. Global convergence to a first-order critical point requires the model to be fully linear, which means that the norm of the error in the function values at points within the trust region is bounded by Δ^2, while the norm of the error in the gradient values is bounded by Δ. Global convergence to a second-order critical point requires the model to be fully quadratic, which implies stronger conditions. In this case, the norm of the error in the function values is bounded by Δ^3, the norm of the error in the gradient values is bounded by Δ^2, and the norm of the error in the Hessian matrix is bounded by Δ.

A popular choice is to represent the model used at each iteration as a quadratic function. According to the Taylor-series expansion of f around the current iterate x_k, we have

$$f(x_k+d) = f_k + g_k^\top d + \frac{1}{2} d^\top \nabla^2 f(x_k+td)d, \tag{2.23}$$

where $f_k = f(x_k)$, $g_k = \nabla f(x_k)$, and t is a scalar in the interval $(0,1)$. Therefore, at the k-th iteration, the following quadratic model $m : \mathbb{R}^n \to \mathbb{R}$ can be used to approximate the objective function

$$m_k(d) = f_k + g_k^\top d + \frac{1}{2} d^\top H_k d, \tag{2.24}$$

where H_k is a symmetric matrix. Note that if m_k is a first-order Taylor model, then $m_k(0) = f(x_k)$ and $\nabla m_k(0) = g_k = \nabla f(x_k)$; if m_k is a second-order Taylor model, we also have $\nabla^2 m_k(0) = H_k = \nabla^2 f(x_k)$. If H_k is chosen to be the true Hessian $\nabla^2 f(x_k)$, we call the method *trust-region Newton method*. Moreover, in this case the resulting error in the model function m_k is $O(\|d\|^3)$. When we use an approximation H_k to the Hessian in the second-order term, we can prove that the difference between $m_k(d)$ and $f(x_k+d)$ is $O(\|d\|^2)$, which is negligible when d is very small.

To obtain a minimizer of the objective function f, at each iteration the model $m_k(d)$ is used to approximate the true objective function f within a suitable neighborhood to seek a solution of the trust-region subproblem

$$\min_{d\in\mathbb{R}^n} m_k(d) = f_k + g_k^\top d + \frac{1}{2} d^\top H_k d \quad \text{s.t.} \quad \|d\| \le \Delta_k, \tag{2.25}$$

where $\Delta_k > 0$ is the trust-region radius, and $\|\cdot\|$ could be an iteration-dependent norm, but in the majority of our discussion, we define $\|\cdot\|$ to be the Euclidean norm. When H_k is positive definite and $\|H^{-1}g_k\| \le \Delta_k$, the solution of (2.25) is the unconstrained local minimum of the quadratic function $m_k(d)$, which is given by the *full step* $d_k^H = -H_k^{-1} g_k$. The solution of the subproblem (2.25) is not so obvious in other cases, but usually it does not require a high computational expense. Whatever case we consider, we only need to find an approximate solution to obtain convergence and good practical behavior.

The radius of the trust region is very important for the effectiveness of the method. If the trust-region radius is too small, we may lose the chance to move to the minimizer of the objective function in fewer iterations. If the trust-region radius is too large, the minimizer of the model could be far away from the minimizer of the function in this region. Therefore, we have to reduce the size of the region and try again. In practice, the size of the region is chosen according to the performance of the algorithm during the previous iterations. If the model is reliable in the previous iterations and the behavior of the objective function along these steps can be accurately approximated, then we enlarge the trust-region radius. On the contrary, if our model cannot accurately approximate the objective function in the current trust region, we need to reduce the trust-region radius and try again.

To choose the trust-region radius Δ_k, at each iteration we define the ratio between the actual and predicted reduction, namely,

$$\rho_k = \frac{f(x_k) - f(x_k+d_k)}{m_k(0) - m_k(d_k)}, \tag{2.26}$$

which describes the agreement between the objective function f and the model function m_k. Since the predicted reduction is always positive, if ρ_k is close to 1, it means that there is good agreement between the model m_k and the function f over this step. Therefore, we can enlarge the trust-region radius for the next iteration. If ρ_k is close to zero or negative, we need to shrink the trust region by reducing Δ_k. Note that if $\rho_k < 0$, it means that the new objective function value $f(x_k + d_k)$ is larger than the current value $f(x_k)$ and, accordingly, the step needs to be rejected. The scheme of the procedure is reported in Algorithm 2.

Algorithm 2: Trust Region

Let $\hat{\Delta} > 0$, $\Delta_0 \in (0, \hat{\Delta})$, and $\eta \in \left[0, \frac{1}{4}\right)$.
for $k = 0, 1, 2, \cdots$
  Obtain d_k by (approximately) solving (2.25).
  Evaluate ρ_k from (2.26).
  if $\rho_k < \frac{1}{4}$ **then** $\Delta_{k+1} = \frac{1}{4}\Delta_k$.
  else
    if $\rho_k > \frac{3}{4}$ and $\|d_k\| = \Delta_k$ **then** $\Delta_{k+1} = \min(2\Delta_k, \hat{\Delta})$.
    else $\Delta_{k+1} = \Delta_k$.
  if $\rho_k > \eta$ **then** $x_{k+1} = x_k + d_k$.
  else $x_{k+1} = x_k$.
end (for)

In this scheme, $\hat{\Delta}$ represents an overall bound on the step lengths. If $\|d_k\| < \Delta_k$, since the trust-region radius Δ_k does not interfere with the progress of the algorithm, we keep the same radius also in the next iteration when $\rho_k > \frac{3}{4}$. In particular, the radius is increased only when $\|d_k\|$ reaches the boundary of the trust region.

Now we are going to focus on solving the trust-region subproblem (2.25). For the convenience of discussion, we drop the iteration subscript k and restate the problem (2.25) as follows

$$\min_{d \in \mathbb{R}^n} m(d) \stackrel{\text{def}}{=} f + g^\top d + \frac{1}{2} d^\top H d \quad \text{s.t.} \quad \|d\| \leq \Delta. \tag{2.27}$$

Moré and Sorensen in [49] show that the solution d^* of (2.27) satisfies

$$\lambda\, d^* = -\nabla m(d^*) \tag{2.28}$$

for some $\lambda \geq 0$, which means that d^* and $-\nabla m(d^*)$ point in the same direction. In general, we can characterize the solution of the trust-region subproblem by using Theorem 2.3.1.

Theorem 2.3.1. *The vector d^* is a global solution of the trust-region subproblem (2.27) if and only if d^* is feasible and there is a scalar $\lambda \geq 0$ such that the following conditions are satisfied:*

$$(H + \lambda I) d^* = -g, \tag{2.29}$$

$$\lambda(\Delta - \|d^*\|) = 0, \tag{2.30}$$

$$(H + \lambda I) \quad \textit{is positive semidefinite.} \tag{2.31}$$

Note that (2.29) is equivalent to (2.28) and, together with (2.30), it implies that when d^* lies on the boundary of the trust-region and H is assumed to be the true Hessian, d^* is the Newton direction. Moreover, if the condition in (2.31) is replaced by the requirement of positive definiteness, then d^* is proved to be unique.

Similarly to line-search methods, we guarantee the global convergence of a trust-region method by requiring the approximate solution d_k to ensure a sufficient reduction in the model. To quantify the sufficient reduction, we introduce the Cauchy point d_k^C, which is defined as the point that achieves the same amount of reduction in m as the steepest-descent direction. The analytical form of the Cauchy point is

$$d_k^C = -\tau_k \frac{\Delta_k}{\|g_k\|} g_k. \tag{2.32}$$

where

$$\tau_k = \begin{cases} 1 & \text{if } g_k^T H_k g_k \leq 0; \\ \min(\|g_k\|^3/(\Delta_k g_k^\top H_k g_k), 1) & \text{otherwise.} \end{cases} \tag{2.33}$$

One should note that no matrix factorization is required to calculate the Cauchy step d_k^C, thus resulting in a cheap computation. The global convergence of a trust-region method relies on the requirement that approximate solutions of the trust-region subproblem must achieve a reduction in the model m_k that is at least some positive multiple of the decrease attained by the Cauchy step. However, if we always take the Cauchy point as our step, we are simply doing the steepest-descent method with a special step length, which does not perform well even if taking the optimal step at every iteration.

Methods for finding the approximate solution of (2.27) are out of the scope of this review and we refer the reader to [52, Theorem 4.9] for further details.

2.4 Inexact Newton methods

Given the optimization problem (2.1) and assuming that f is twice continuously differentiable, the Newton iteration is given by

$$x_{k+1} = x_k + \alpha_k d_k^N,$$

where α_k is the stepsize and $d_k^N = -\nabla^2 f(x)^{-1} \nabla f(x)$ is the search direction. Assuming that x^* is the limit point of the sequence $\{x_k\}$ generated by Newton's method, by considering a step length equal to 1 and other essential assumptions (such as Lipschitz continuity of the Hessian, $\nabla f(x^*) = 0$, $\nabla^2 f(x^*)$ positive definite, and an initial point x_0 sufficiently close to x^*), we can prove that Newton's method shows a quadratic local convergence rate. Note that this noticeable convergence result can be proved only locally, namely, the initial point must be sufficiently close to the limit point.

One should note that d_k^N is not always a descent direction since the Hessian matrix $\nabla^2 f(x)$ may not be positive definite. To obtain a descent direction to be used in a line-search method, one can make the Hessian matrix positive definite by performing proper modifications. The strategy is to add a positive diagonal matrix E_k to the true Hessian, namely,

$$H_k = \nabla^2 f(x_k) + E_k,$$

so that the modified Hessian H_k is positive definite. To preserve most of the second-order information given by the Hessian, we want the modification to be as small as possible. One simple approach consists in finding a scalar $\tau > 0$ such that H_k is sufficiently positive definite when $E_k = \tau I$. We refer to $-H_k^{-1}\nabla f(x)$ as a *modified Newton direction*, which will be briefly discussed at the end of Chapter 7.

Despite the remarkable fast local convergence, Newton's method presents the significant drawback of requiring the knowledge of the Hessian, which is not affordable when either the problem is large-scale or computing the derivatives is not an available option. Another disadvantage is that the Newton step d_k^N is obtained by solving the symmetric $n \times n$ linear system

$$\nabla^2 f(x_k) d_k^N = -\nabla f(x_k), \tag{2.34}$$

which needs to be solved at each iteration. Since determining the exact solution by using direct methods, such as Gaussian elimination, is expensive, alternative ways to keep both the storage and computational cost of the optimization algorithm at an acceptable level are desirable. The next sections describe algorithms that compute an approximate solution to the Newton system by using an iterative scheme.

The idea of inexact Newton methods is to approximately solve (2.34) with efficient algorithms for linear systems, such as the conjugate gradient (CG) method or the Lanczos method. First, we discuss how relaxing the requirement for an exact solution of the system (2.34) affects the local convergence of inexact Newton methods. Then, we show how the CG method (possibly using matrix preconditioning) and both line-search and trust-region methods are used in the literature to obtain an approximate solution of (2.34). This family of methods is referred to as the *inexact Newton methods*.

One natural stopping criterion used in solving (2.34) is based on the relative residual $\|r_k\|/\|\nabla f(x_k)\|$, where

$$r_k = \nabla^2 f(x_k)\, d_k + \nabla f(x_k), \tag{2.35}$$

and d_k is the inexact Newton step. This kind of inexact Newton method provides a trade-off between the amount of work per iteration and the accuracy adopted to solve the Newton system. In general, we terminate the inexact Newton method when

$$\|r_k\| \leq \eta_k \|\nabla f(x_k)\|, \tag{2.36}$$

where $\{\eta_k\}$ (with $0 < \eta_k < 1$ for all k) is called the *forcing sequence*, which is used to control the level of accuracy. One can prove that if the forcing sequence $\{\eta_k\}$ is bounded away from 1, then inexact Newton methods are locally convergent. That is to say, the sequence of $\{x_k\}$ converges to x^* if the initial guess x_0 is good enough, as stated in Theorem 2.4.1 (see [21]).

Theorem 2.4.1. *Suppose that $\nabla^2 f(x)$ exists and is continuous in a neighborhood of a minimizer x^*, with $\nabla^2 f(x^*)$ positive definite. Consider the iteration $x_{k+1} = x_k + d_k$ where d_k is such that (2.36) is satisfied, and assume that $\eta_k \leq \eta$ for some constant $\eta \in [0,1)$. Then, if the starting point x_0 is sufficiently near x^*, the sequence $\{x_k\}$ converges to x^* linearly and satisfies*

$$\|\nabla^2 f(x^*)(x_{k+1} - x^*)\| \leq \hat{\eta} \|\nabla^2 f(x^*)(x_k - x^*)\|. \tag{2.37}$$

for some constant $\hat{\eta}$ *with* $\eta < \hat{\eta} < 1$.

Moreover, in [21] the authors prove that the sequence of iterates $\{x_k\}$ converge to x^* at the same rate as the sequence of gradients $\{\nabla f(x_k)\}$ converges to zero. From this fact, one can prove that if $\lim_{k\to\infty} \eta_k = 0$, the sequences $\{\|\nabla f(x_k)\|\}$ and $\{x_k\}$ are superlinearly convergent. If the Hessian matrix $\nabla^2 f(x)$ is Lipschitz continuous near x^*, then the two sequences converge quadratically. These results are summarized in the following theorem.

Theorem 2.4.2. *Suppose that the conditions of Theorem 2.4.1 hold, and assume that the iterates* $\{x_k\}$ *generated by the inexact Newton method converge to* x^*. *Then the rate of convergence is superlinear if* $\eta_k \to 0$. *If, in addition,* $\nabla^2 f(x)$ *is Lipschitz continuous at points x close to* x^* *and if* $\eta_k = O(\|\nabla f(x_k)\|)$, *then the convergence is quadratic.*

In practice, if we set $\eta_k = \min(0.5, \sqrt{\|\nabla f(x_k)\|})$, then superlinear convergence is obtained. If we set $\eta_k = \min(0.5, \|\nabla f(x_k)\|)$, then quadratic convergence is achieved.

We now introduce two implementations of Newton's method that approximately solve the Newton system by using the CG method, namely, the line-search Newton-CG method and the trust-region Newton-CG method. It is important to point out that incorporating inexact Newton strategies within line-search and trust-region frameworks allows the resulting algorithms to show good global convergence results, not only local, thus overcoming one of the drawbacks of Newton's method.

2.4.1 Line-search Newton-CG method

The Line-search Newton-CG method is also known as the *truncated Newton method*. In this approach, the search direction is computed by applying the CG method to the Newton system (2.34). Note that the CG method is designed to solve systems with a positive definite coefficient matrix and terminates once a negative curvature direction is detected. This strategy guarantees that d_k is always a descent direction and the fast convergence rate of Newton's method is preserved.

In order to describe the line-search Newton-CG method, we denote $\nabla^2 f(x_k)$ as H_k and we write the linear system (2.34) in the following form

$$H_k d = -\nabla f(x_k). \tag{2.38}$$

Determining an approximate solution to the Newton system is the goal of the inner iteration of the method, where we apply the CG algorithm. Once the inner iteration ends, the approximate solution to the system is used as a direction in the major iteration, where the next iterate is determined according to a suitable stepsize. In the inner iterations, the sequence of search directions generated by the CG method is denoted by $\{p_j\}$ and the sequence of iterates is referred to as $\{z_j\}$.

The sequence z_j converges to the Newton direction d_k^N, which is the exact solution of the system (2.38), when H_k is positive definite. To obtain superlinear convergence rate, we define the forcing sequence to be $\eta_k = \min(0.5, \sqrt{\|\nabla f(x_k)\|})$ at each major iteration (of course, other choices are also acceptable). A tolerance ε_k is used to allow the CG method to terminate at an inexact solution. The

resulting scheme is reported in Algorithm 3.

Algorithm 3: Line-Search Newton-CG

Let x_0 be an initial point.
for $k = 0, 1, 2, \ldots$
  Define a tolerance $\varepsilon_k = \min(0.5, \sqrt{\|\nabla f(x_k)\|})\|\nabla f(x_k)\|$.
  Set $z_0 = 0$, $r_0 = \nabla f(x_k)$, and $p_0 = -r_0 = -\nabla f(x_k)$.
  for $j = 0, 1, 2, \ldots$
    if $p_j^\top H_k p_j \leq 0$
      if $j = 0$
        return $d_k = -\nabla f(x_k)$.
      else
        return $d_k = z_j$.
    Set $\alpha_j = r_j^\top r_j / p_j^\top H_k p_j$.
    Set $z_{j+1} = z_j + \alpha_j p_j$.
    Set $r_{j+1} = r_j + \alpha_j H_k p_j$.
    if $\|r_{j+1}\| < \varepsilon_k$
      return $d_k = z_{j+1}$.
    Set $\beta_{j+1} = r_{j+1}^\top r_{j+1} / r_j^\top r_j$.
    Set $p_{j+1} = -r_{j+1} + \beta_{j+1} p_j$.
  end (for)
  Set $x_{k+1} = x_k + \alpha_k d_k$, where α_k satisfies the Wolfe, Goldstein, or Armijo backtracking conditions (using $\alpha_k = 1$ if possible).
end (for)

Note that if the negative curvature is detected at the first inner iteration $j = 0$, the steepest-descent direction $d_k = -\nabla f(x_k)$ is returned. Moreover, it is important to point out that the explicit form of the Hessian matrix $H_k = \nabla^2 f(x_k)$ is not necessary since it is sufficient to have Hessian-vector products $\nabla^2 f(x_k)p$ for any vector p, which allow significant savings in terms of computational resources. Thus, Algorithm 3 is suited for large-scale problems. However, when the Hessian matrix is nearly singular, the quality of the search direction obtained may be poor. This may result in many function evaluations required by the line-search procedure and in a small decrease of the function value. One way to deal with this problem is to use matrix preconditioning in the CG iterations. An alternative way is to normalize the Newton step. Since it is difficult to define good rules for performing this normalization, which may also affect the fast convergence of Newton's method when the pure Newton step is well-scaled, the trust-region Newton-CG is a preferable option when dealing with this problematic situation.

2.4.2 Trust-region Newton-CG method

The idea underlying the trust-region Newton-CG method is to use a modified CG algorithm to find an approximate solution of the trust-region subproblem (2.25) that achieves a larger reduction in m_k

than the Cauchy point. In particular, when $H_k = \nabla^2 f(x_k)$ for every k, this procedure is equivalent to solving the system (2.38) by applying the CG method.

Let ε_k be a tolerance used to stop the method at an inexact solution. Given $H_k = \nabla^2 f(x_k)$, let p_j and z_j be the sequences of search directions and iterates of the inner CG iteration, respectively. The scheme of the resulting method, which is proposed in [61], is reported in Algorithm 4.

Algorithm 4: CG-Steihaug

Let $\varepsilon_k > 0$ be a given tolerance.
Set $z_0 = 0$, $r_0 = \nabla f(x_k)$, $p_0 = -r_0 = -\nabla f(x_k)$.
if $\|r_0\| < \varepsilon_k$
  return $d_k = z_0 = 0$.
for $j = 0, 1, 2, \ldots$
  if $p_j^\top H_k p_j \leq 0$
    Find τ such that $d_k = z_j + \tau p_j$ minimizes $m_k(d_k)$ in (2.25) and satisfies $\|d_k\| = \Delta_k$.
    return d_k.
  Set $\alpha_j = r_j^\top r_j / p_j^\top H_k p_j$.
  Set $z_{j+1} = z_j + \alpha_j p_j$.
  if $\|z_{j+1}\| \geq \Delta_k$
    Find $\tau \geq 0$ such that $d_k = z_j + \tau p_j$ satisfies $\|d_k\| = \Delta_k$.
    return d_k.
  Set $r_{j+1} = r_j + \alpha_j H_k p_j$.
  if $\|r_{j+1}\| < \varepsilon_k$
    return $d_k = z_{j+1}$.
  Set $\beta_{j+1} = r_{j+1}^\top r_{j+1} / r_j^\top r_j$.
  Set $p_{j+1} = -r_{j+1} + \beta_{j+1} p_j$.
end (for)

Note that the step d_k is obtained by the intersection of the trust-region boundary with the current search direction. The method stops when one of the three following stopping criteria is satisfied: the search direction p_j is a direction of nonpositive curvature along H_k; the size of z_j is out of the trust-region bound; the required accuracy in the solution of the system (2.38) has been achieved.

As discussed in Theorems 2.4.1 and 2.4.2, when the trust-region bound constraint becomes inactive near the solution x^*, the trust-region Newton-CG method reduces to the inexact Newton method. To keep the cost of the trust-region Newton-CG method low, it is very important to choose the tolerance ε_k wisely at every iteration. If the tolerance ε_k is chosen similarly to Algorithm 3, we can obtain rapid convergence as well. One can also prove that the decrease obtained by Algorithm 4 is at least half as good as the optimal decrease [66].

2.5 Quasi-Newton methods

Quasi-Newton methods are approaches that use only the knowledge of the gradient of the objective function at each iteration, without requiring the Hessian. The idea underlying these methods is to construct a model of the objective function by using differences of the gradient at two successive

iterates. If the search direction is an accurate approximation of the Newton direction, superlinear convergence is attained. Although both quasi-Newton methods and steepest-descent method use only the gradient of the objective function without relying on the Hessian, quasi-Newton methods achieve a significant improvement compared to the steepest-descent method, especially on difficult nonconvex problems. Moreover, since the knowledge of the second-order derivatives is not required, quasi-Newton methods are sometimes more efficient than Newton's method.

In every quasi-Newton method, the search direction is of the form

$$d_k = -H_k^{-1}\nabla f(x_k), \tag{2.39}$$

where H_k is a symmetric and positive definite matrix which is updated at every iteration. The new iterate is given by

$$x_{k+1} = x_k + \alpha_k d_k. \tag{2.40}$$

The following theorem states that to attain a superlinear convergence rate, H_k must be a sufficiently accurate approximation of the true Hessian.

Theorem 2.5.1. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. Consider the iteration $x_{k+1} = x_k + d_k$ (i.e., the step length is uniformly 1), where the search direction d_k is given by (2.39). Let us assume also that $\{x_k\}$ converges to a point x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then $\{x_k\}$ converges superlinearly if and only if*

$$\lim_{k\to\infty} \frac{\left\| \left(H_k - \nabla^2 f\left(x^*\right)\right) d_k \right\|}{\|d_k\|} = 0. \tag{2.41}$$

2.5.1 The BFGS method

One of the most popular quasi-Newton methods is *BFGS*, whose name is due to its discoverers Broyden, Fletcher, Goldfarb, and Shanno. Consider the quadratic model of the objective function at the current iteration x_k

$$m_k(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T H_k d, \tag{2.42}$$

where H_k is an $n \times n$ symmetric and positive definite matrix which is updated at every iteration. We can see that the quadratic model m_k and its gradient $\nabla m_k(d)$ match $f(x_k)$ and $\nabla f(x_k)$ at $d = 0$. Moreover, the convex quadratic model m_k achieves its minimum at d_k, which can be expressed as follows

$$d_k = -H_k^{-1}\nabla f(x_k). \tag{2.43}$$

The new iteration is $x_{k+1} = x_k + \alpha_k d_k$, where the stepsize α_k satisfies the Wolfe conditions (2.9). Note that the key difference between the BFGS method and the line-search Newton's method is that the true Hessian $\nabla^2 f(x_k)$ is replaced by the approximate Hessian H_k.

To obtain the approximate Hessian without computing a completely new H_k at each iteration, consider the new model at x_{k+1}

$$m_{k+1}(d) = f(x_{k+1}) + \nabla f(x_{k+1})^\top d + \frac{1}{2} d^\top H_{k+1} d.$$

We want to impose some conditions on the update. One reasonable requirement is that the gradient of m_{k+1} at the iterates x_k and x_{k+1} matches the gradient of the objective function, namely,

$$\begin{aligned} &\nabla m_{k+1}(-\alpha_k d_k) = \nabla f(x_{k+1}) - \alpha_k H_{k+1} d_k = \nabla f(x_k), \\ &\nabla m_{k+1}(0) = \nabla f(x_{k+1}). \end{aligned} \tag{2.44}$$

The second equation in (2.44) automatically satisfies the requirement that the gradient of m_{k+1} matches the gradient of the objective function at x_{k+1}. By rearranging the first equality, we obtain

$$H_{k+1} s_k = y_k, \quad \text{where } s_k = x_{k+1} - x_k = \alpha_k d_k \text{ and } y_k = \nabla f(x_{k+1}) - \nabla f(x_k). \tag{2.45}$$

The formula (2.45) is referred to as the *secant equation*. Note that H_{k+1} is symmetric and positive definite only if the following *curvature condition* holds

$$s_k^\top y_k > 0. \tag{2.46}$$

This can be seen by multiplying the secant equation by s_k^T. To enforce this condition, the stepsize α_k is required to satisfy the Wolfe or strong Wolfe conditions.

To guarantee the uniqueness of the symmetric matrix H_{k+1} satisfying the secant equation, additional conditions are needed to fix the remaining degrees of freedom. To this end, we select the matrix that is closest to the current Hessian approximation H_k in some sense. That is to say, we obtain H_{k+1} by solving the following problem

$$\begin{aligned} \min_H \ & \|H - H_k\| \\ \text{s.t } & H = H^T, \\ & H s_k = y_k, \end{aligned} \tag{2.47}$$

where s_k and y_k satisfy (2.45) and H_k is symmetric and positive definite.

Since in (2.43) we need to compute the inverse of H_k, a significant improvement from the computational point of view can be obtained by directly approximating the inverse of the Hessian. In particular, let G_k be the inverse of H_k. The search direction d_k can now be easily computed by using only one matrix-vector product

$$d_k = -G_k \nabla f(x_k).$$

By following the steps described in [52], one arrives at the following updating formula

$$G_{k+1} = G_k - \frac{G_k y_k y_k^T G_k}{y_k^T G_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}. \tag{2.48}$$

We can see that the last two terms in (2.48) are rank-one matrices, meaning that a rank-two modification is performed in every iteration. We require the approximation G_{k+1} to be symmetric, positive definite, and satisfying the secant equation, which can be written as

$$G_{k+1} y_k = s_k. \tag{2.49}$$

By writing the problem in (2.47) in terms of G and G_k in place of H and H_k and choosing an appropriate matrix norm, we can derive the BFGS formula

$$G_{k+1} = \left(I - \rho_k s_k y_k^T\right) G_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T, \text{ and } \rho_k = \frac{1}{y_k^T s_k}. \tag{2.50}$$

The BFGS method is reported in Algorithm 5.

Algorithm 5: BFGS Method

Let x_0 be a starting point, $\varepsilon > 0$ a convergence tolerance, and G_0 an initial approximation of the Hessian inverse.
Set $k = 0$.
while $(\|\nabla f(x_k)\| > \varepsilon)$
Compute the search direction $d_k = -G_k \nabla f(x_k)$.
Set $x_{k+1} = x_k + \alpha_k d_k$, where α_k is a stepsize computed from a line search satisfying Wolfe conditions.
Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.
Compute $\rho_k = \frac{1}{y_k^T s_k}$ and $G_{k+1} = \left(I - \rho_k s_k y_k^T\right) G_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T$.
$k = k + 1$.
end (while)

Note that from the updating formula (2.50), if $\rho_k = 1/y_k^T s_k$ becomes a very large number, G_{k+1} becomes large as well. Therefore, in this case G_k cannot be considered an accurate approximation to the Hessian inverse. However, one can show that the BFGS method has a self correcting property (see [22, 23]) which implies that the Hessian approximation will tend to correct itself within a few steps if G_k cannot estimate the curvature of the function correctly or the estimation slows down the iteration.

The cost of BFGS method at each iteration is of $O(n^2)$ arithmetic operations plus the cost of function evaluations and gradient evaluations. Therefore, the algorithm is robust and has superlinear convergence rate. Although Newton's method converges faster, its cost per iteration is higher since a solution of a linear system is required. Moreover, BFGS does not require second-order derivatives.

The global convergence of the BFGS method, which was established by Powell in [54], requires the following assumption on the objective function.

Assumption 2.5.1. (i) *The objective function f is twice continuously differentiable.*
(ii) *The level set $L = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$ is convex, and there exist positive constants m and M such that*

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

for all $z \in \mathbb{R}^n$ and $x \in L$.

Now we are going to present the global convergence result for the BFGS method under the Assumption 2.5.1.

Theorem 2.5.2. *Let H_0 be any symmetric positive definite initial matrix, and let x_0 be a starting point such that Assumption 2.5.1 is satisfied. Then the sequence $\{x_k\}$ generated by Algorithm 5 (with $\varepsilon = 0$) converges to the minimizer x^* of f.*

The following result shows that the BFGS method attains superlinear convergence rate under an additional assumption.

Assumption 2.5.2. *The Hessian matrix $\nabla^2 f(x)$ is Lipschitz continuous at x^*, that is,*

$$\|\nabla^2 f(x) - \nabla^2 f(x^*)\| \leq L\|x - x^*\|,$$

for all x near x^*, where L is a positive constant.

Theorem 2.5.3. *Suppose that f is twice continuously differentiable and that the iterates generated by the BFGS algorithm converge to a minimizer x^* such that Assumption 2.5.2 holds. Suppose also that $\sum_{k=1}^{\infty} \|x_k - x^*\| < \infty$. Then $\{x_k\}$ converges to x^* at a superlinear rate.*

Chapter 3

Review of interpolation models for derivative-free optimization

In this chapter, interpolation-based DFO models are reviewed in order to provide useful insight into the methodology proposed in this thesis. The definitions and methodologies in this chapter are based on the book of Conn, Scheinberg and Vicente [17].

3.1 Generalities on interpolation models

The idea behind the interpolation models considered in this thesis is to fit a local polynomial model on a set of sample points where the objective function has been previously evaluated. Therefore, the value of the polynomial model at these points must be the same as the corresponding objective function value. Sampling rules and type of functional models are among the properties that define the accuracy of the interpolation process.

Surrogate modeling is a general term that is commonly adopted to refer to the analytical models used to approximate the objective function by providing a surrogate to be used in place of the true function. Surrogate models are particularly useful when the objective function evaluations are computationally expensive, as is the case in DFO and black-box optimization. Although this term is usually referred to special classes of models, such as radial basis functions, Kriging models, and response surface methodologies, interpolation and regression models can be viewed as surrogates of the true function and many papers in the DFO literature deal with these approaches [16, 17, 47, 53, 57, 65].

Throughout this chapter, two interpolation approaches are considered, namely, linear and quadratic interpolation. Linear functions represent the simplest class of models used for interpolation. However, since they are not able to capture the curvature of the objective function, quadratic functions are the most adopted in practice. Although fully determined quadratic interpolation models require a number of sample points that is approximately equal to the square of the dimension, when dealing with computationally expensive functions it is often more convenient to consider underdetermined quadratic models, which allow reducing the cardinality of the sample set (at the cost of the accuracy of the approximation).

Given a set of sample points $Y = \{y^0, y^1, \cdots, y^p\} \subset \mathbb{R}^n$, referred to as the sample set, a function $m : \mathbb{R}^n \to \mathbb{R}$ is said to be an *interpolation model* if the following interpolating conditions are satisfied:

$$m(y) = f(y), \quad \text{for all } y \in Y, \tag{3.1}$$

where f is the real-valued function of interest. Although m can be any nonlinear function (e.g., radial basis functions have been successfully applied in DFO), the focus of this chapter is on polynomial models, which are commonly adopted in practice. In particular, m is assumed to be a polynomial of degree less than or equal to d. Note that the sample set Y contains $p_1 = |Y| = p+1$ points. Typically, p_1 is selected to be the dimension of the space of polynomials of degree less than or equal to d in $\mathbb{R}^n$, denoted by $\mathscr{P}_n^d$. Specifically, we have that

$$p_1 = p+1 = \binom{n+d}{n},$$

which translates to $p_1 = n+1$ in the case of linear interpolation, where $d = 1$, and $p_1 = (1/2)(n+1)(n+2)$ when quadratic interpolation is considered, where $d = 2$.

To ensure global convergence properties when using interpolation models within DFO algorithms, an accurate approximation of the true objective function must be guaranteed. When the polynomial interpolation model is a truncated Taylor series expansion of first or second order, bounds on the approximation error can be established by exploiting the results derived for the Taylor expansion error bounds. In the DFO literature, one of the most popular results is obtained from [8, Theorem 1], which states that the error bound on the gradient approximation is

$$\|\nabla f(x) - \nabla m(x)\| \leq \frac{1}{(d+1)!} G \Lambda_Y \sum_{i=0}^{p} \left\|y^i - x\right\|^{d+1},$$

where x is a point in the convex hull of Y, while G and Λ_Y are constants depending on the function f and on the interpolation set Y, respectively. The previous error bound provides useful insight into the convergence properties of a DFO method that uses interpolation models. In particular, we can see that the approximation error decreases as the sample points become closer to the point where the gradient is computed. Note that to guarantee the previous result and ensure that the approximation error decreases at the same rate as the sample points approaching x, Λ_Y must be uniformly bounded.

3.2 Linear interpolation

The small number of sample points required for linear interpolation models is the most appealing feature of this kind of models. Despite the low ability in capturing the curvature of the function to interpolate, linear interpolation models have been successfully used in DFO algorithms (see, e.g., [55]).

Given the sample set $Y = \{y^0, y^1, \cdots, y^p\} \subset \mathbb{R}^n$, with $p = n$, linear functions are the simplest models that can be considered for interpolating the sample points in Y. In particular, let $m(x)$ denote

the polynomial of degree $d = 1$ interpolating f at the sample points, i.e.,

$$m(y^i) = f(y^i), \quad i = 0, \cdots, n. \tag{3.2}$$

By using the *natural basis* $\bar{\phi} = \{1, x_1, x_2, \cdots, x_n\}$ (the basis of polynomials as they appear in the Taylor expansion) as a basis for the space $\mathscr{P}_n^1$ of linear polynomials of degree at most 1, the linear interpolating model m of f can be written in the form

$$m(x) = \alpha_0 + \alpha_1 x_1 + \cdots + \alpha_n x_n. \tag{3.3}$$

Note that the natural basis $\bar{\phi}$ is adopted for the sake of simplicity, but other bases ϕ may be also used, e.g., $\{1, 1+x_1, 1+x_1+x_2, \cdots, 1+x_1+x_2+\cdots+x_n\}$. By plugging (3.3) into the interpolating conditions in (3.1) and rewriting in matrix form, we obtain

$$\begin{bmatrix} 1 & y_1^0 & y_2^0 & \cdots & y_n^0 \\ 1 & y_1^1 & y_2^1 & \cdots & y_n^1 \\ \vdots & & \vdots & & \vdots \\ 1 & y_1^n & y_2^n & \cdots & y_n^n \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^n) \end{bmatrix}. \tag{3.4}$$

Therefore, the coefficients $\alpha_0, \cdots, \alpha_n$ of the linear interpolating m can be determined by solving the linear system associated with (3.4). If we define

$$M = M(\bar{\phi}, Y) = \begin{bmatrix} 1 & y_1^0 & y_2^0 & \cdots & y_n^0 \\ 1 & y_1^1 & y_2^1 & \cdots & y_n^1 \\ \vdots & & \vdots & & \vdots \\ 1 & y_1^n & y_2^n & \cdots & y_n^n \end{bmatrix}, \ \alpha_{\bar{\phi}} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}, \text{ and } F(Y) = \begin{bmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^n) \end{bmatrix},$$

the linear system (3.4) can be written as

$$M(\bar{\phi}, Y)\alpha_{\bar{\phi}} = F(Y). \tag{3.5}$$

To ensure that the system (3.5) can be solved, we introduce the definition of poised set for linear interpolation (it is worth pointing out that some authors refer to a poised set for linear interpolation as a *d-unisolvent* set [9]).

Definition 3.2.1. *$Y = \{y^0, y^1, \cdots, y^n\}$ is a poised set for linear interpolation in $\mathbb{R}^n$ if the corresponding matrix $M(\bar{\phi}, Y)$ is nonsingular.*

Note that the definition of poisedness does not depend on the basis chosen. This means that if Y is a poised set for the basis $\bar{\phi}$, it is also a poised set for any other basis ϕ in $\mathscr{P}_n^1$. Furthermore, $m(x)$ is independent of the basis chosen as well. Hence, we conclude that Y is a poised set for linear interpolation if and only if the linear interpolating polynomial $m(x)$ can be uniquely defined. We point out that in the linear case, the notion of poisedness is the same as the notion of affine independence.

After establishing that the existence of solutions to the system (3.5) is related to the poisedness of the interpolation set Y, it is important to investigate the conditions that characterize a *well-poised*

interpolation set. Given a basis ϕ, since the poisedness of Y is defined by the nonsingularity of $M(\phi,Y)$, a natural question is whether the condition number of $M(\phi,Y)$ can be considered an appropriate indicator of a well-poised set. To give an answer to this question, two facts are considered. On the one hand, since the condition number of $M(\phi,Y)$ depends on the choice of ϕ, a different basis $\tilde{\phi}$ can be chosen so that the condition number of $M(\tilde{\phi},Y)$ varies from 1 to $+\infty$ (Y is assumed to be a poised interpolation set). On the other hand, the condition number of $M(\phi,Y)$ also depends on the scaling of Y. Therefore, in general, the condition number of $M(\phi,Y)$ is not a good indicator of poisedness of a sample set Y. However, we can show that the condition number of $M(\phi,Y)$ can be considered a measure of poisedness when the basis is the natural one, i.e., $\phi = \bar{\phi}$, and the interpolation set Y is scaled.

3.2.1 Error bounds in linear interpolation

Suppose that the points in the sample set $Y = \{y^0, y^1, \cdots, y^n\} \subset \mathbb{R}^n$ are in a ball centered at y^0 with radius Δ, where the radius is defined as

$$\Delta = \Delta(Y) = \max_{1\le i\le n} \|y^i - y^0\|.$$

We aim to assess the quality of the approximation of f in the ball of radius Δ centered at y^0 when $m(x)$ is the interpolating function considered. In particular, we assess the quality of the gradient $\nabla m(y)$ of the model when used to approximate $\nabla f(y)$. Let us first write the linear interpolating polynomial (3.3) in the form

$$m(y) = c + g^\top y, \quad \text{where } c = \alpha_0 \text{ and } g = [\alpha_1, \alpha_2, \cdots, \alpha_n]^\top.$$

After performing one step of Gaussian elimination to the matrix $M = M(\bar{\phi},Y)$ in (3.4), we have

$$\begin{bmatrix} 1 & y_1^0 & y_2^0 & \cdots & y_n^0 \\ 0 & y_1^1 - y_1^0 & y_2^1 - y_2^0 & \cdots & y_n^1 - y_n^0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & y_1^n - y_1^0 & y_2^n - y_2^0 & \cdots & y_n^n - y_n^0 \end{bmatrix}.$$

We can rewrite this matrix by using 4 blocks, i.e.,

$$\begin{bmatrix} 1 & (y^0)^\top \\ 0 & L \end{bmatrix},$$

where

$$L = [y^1 - y^0 \; y^2 - y^0 \cdots y^n - y^0]^\top = \begin{bmatrix} (y^1 - y^0)^\top \\ (y^2 - y^0)^\top \\ \vdots \\ (y^n - y^0)^\top \end{bmatrix} = \begin{bmatrix} y_1^1 - y_1^0 & y_2^1 - y_2^0 & \cdots & y_n^1 - y_n^0 \\ \vdots & \vdots & \vdots & \vdots \\ y_1^n - y_1^0 & y_2^n - y_2^0 & \cdots & y_n^n - y_n^0 \end{bmatrix}.$$

Note that the matrix M is nonsingular if and only if L is nonsingular since $\det(M) = \det(L)$. To prove the results on the quality of the linear interpolation, let us now consider the scaled matrix of L, i.e.,

$$\hat{L} = \frac{1}{\Delta}L = \frac{1}{\Delta}[y^1 - y^0 \; y^2 - y^0 \cdots y^n - y^0]^\top = \begin{bmatrix} \frac{y_1^1 - y_1^0}{\Delta} & \frac{y_2^1 - y_2^0}{\Delta} & \cdots & \frac{y_n^1 - y_n^0}{\Delta} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{y_1^n - y_1^0}{\Delta} & \frac{y_2^n - y_2^0}{\Delta} & \cdots & \frac{y_n^n - y_n^0}{\Delta} \end{bmatrix}. \tag{3.6}$$

The scaled matrix $\hat{L}$ is associated with a scaled sample set $\hat{Y}$ whose points are inside a ball centered at y^0/Δ with radius 1, i.e.,

$$\hat{Y} = \{y^0/\Delta, y^1/\Delta, \cdots, y^n/\Delta\} \subset B(y^0/\Delta; 1).$$

The next theorems state the error bounds for linear interpolation. In particular, once we assume a uniform bound on $\|\hat{L}^{-1}\|$ independent of Δ, Theorem 3.2.1 shows that the error in the approximation of the gradient of function f is linear in Δ, while Theorem 3.2.2 proves that the error in the approximation of function f is quadratic in Δ.

Theorem 3.2.1. *Assume that $Y = \{y^0, y^1, \cdots, y^n\} \subset \mathbb{R}^n$ is a poised set for linear interpolation contained in the ball $B(y^0; \Delta)$, the function f is continuously differentiable in an open domain Ω containing $B(y^0; \Delta)$, and ∇f is Lipschitz continuous in Ω with constant $\nu_L > 0$. Then, the gradient of the linear interpolation model satisfies, for all points y in $B(y^0; \Delta)$, an error bound of the form*

$$\|\nabla f(y) - \nabla m(y)\| \leq \kappa_{eg}\Delta, \tag{3.7}$$

where $\kappa_{eg} = \nu_L(1 + n^{\frac{1}{2}}\|\hat{L}^{-1}\|/2)$ and $\hat{L}$ is given by (3.6).

Theorem 3.2.2. *Consider the assumptions of Theorem 3.2.1 hold, then, the interpolation model satisfies, for all points y in $B(y^0; \Delta)$, an error bound of the form*

$$|f(y) - m(y)| \leq \kappa_{ef}\Delta^2, \tag{3.8}$$

where $\kappa_{ef} = \kappa_{eg} + \nu_L/2$ and κ_{eg} is given in Theorem 3.2.1.

3.3 Quadratic interpolation

Quadratic polynomial models are the simplest nonlinear models that can be considered for the interpolation of a set of points. The theory developed in this section could be extended to the general polynomial interpolation, but it is here presented with respect to quadratic interpolation since this is the type of interpolation used in the next chapters of the thesis.

Let us consider the space of polynomials $\mathscr{P}_n^2$ with degree less than or equal to 2 in $\mathbb{R}^n$. A basis $\phi = \{\phi_0(x), \phi_1(x), \ldots, \phi_p(x)\}$ of $\mathscr{P}_n^2$ is a set of polynomials in $\mathscr{P}_n^2$ with dimension equal to $p_1 = (1/2)(n+1)(n+2)$. Hence, any polynomial $m(x) \in \mathscr{P}_n^2$ can be represented as $m(x) = \sum_{j=0}^{p} \alpha_j \phi_j(x)$, where $\alpha_j's$ are real coefficients. In particular, the natural basis of $\mathscr{P}_n^2$ can be written as follows

$$\bar{\phi} = \{1, x_1, x_2, \cdots, x_n, \frac{1}{2}x_1^2, x_1x_2, \cdots, x_1x_n, \frac{1}{2}x_2^2, x_2x_3, \cdots, x_{n-1}x_n, \frac{1}{2}x_n^2\}. \tag{3.9}$$

Assume that we are given a set of interpolation points $Y = \{y^0, y^1, \cdots, y^p\} \subset \mathbb{R}^n$, and let $m(x)$ denote the quadratic interpolating model that interpolates the given function f at the points in Y. By plugging $m(x) = \sum_{j=0}^{p} \alpha_j \phi_j(x)$ into the interpolating conditions in (3.1), we obtain

$$\sum_{j=0}^{p} \alpha_j \phi_j(y^i) = f(y^i), \quad i = 0, 1, ..., p. \tag{3.10}$$

Rewriting (3.10) in matrix form, we have

$$\begin{bmatrix} \phi_0(y^0) & \phi_1(y^0) & \cdots & \phi_p(y^0) \\ \phi_0(y^1) & \phi_1(y^1) & \cdots & \phi_p(y^1) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(y^p) & \phi_1(y^p) & \cdots & \phi_p(y^p) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^p) \end{bmatrix}. \tag{3.11}$$

Therefore, the coefficients $\alpha_0, \cdots, \alpha_p$ of the quadratic interpolating model m can be determined by solving the linear system associated with (3.11). If we define

$$M = M(\phi, Y) = \begin{bmatrix} \phi_0(y^0) & \phi_1(y^0) & \cdots & \phi_p(y^0) \\ \phi_0(y^1) & \phi_1(y^1) & \cdots & \phi_p(y^1) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(y^p) & \phi_1(y^p) & \cdots & \phi_p(y^p) \end{bmatrix}, \ \alpha_\phi = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_p \end{bmatrix}, \text{ and } F(Y) = \begin{bmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^p) \end{bmatrix},$$

the linear system (3.11) can be written as

$$M(\phi, Y)\alpha_\phi = F(Y). \tag{3.12}$$

If we use the natural basis $\bar{\phi}$ of $\mathscr{P}_n^2$ to construct a quadratic model, then

$$M(\bar{\phi}, Y) = \begin{bmatrix} 1 & y_1^0 & y_2^0 & \cdots & y_n^0 & \frac{1}{2}(y_1^0)^2 & y_1^0 y_2^0 & \cdots & y_1^0 y_n^0 & \frac{1}{2}(y_2^0)^2 & y_2^0 y_3^0 & \cdots & y_{n-1}^0 y_n^0 & \frac{1}{2}(y_n^0)^2 \\ 1 & y_1^1 & y_2^1 & \cdots & y_n^1 & \frac{1}{2}(y_1^1)^2 & y_1^1 y_2^1 & \cdots & y_1^1 y_n^1 & \frac{1}{2}(y_2^1)^2 & y_2^1 y_3^1 & \cdots & y_{n-1}^1 y_n^1 & \frac{1}{2}(y_n^1)^2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & y_1^p & y_2^p & \cdots & y_n^p & \frac{1}{2}(y_1^p)^2 & y_1^p y_2^p & \cdots & y_1^p y_n^p & \frac{1}{2}(y_2^p)^2 & y_2^p y_3^p & \cdots & y_{n-1}^p y_n^p & \frac{1}{2}(y_n^p)^2 \end{bmatrix}.$$

For example, suppose that $n = 2$ and the number of interpolation points is $p = 4$. The previous matrix can be written as

$$M(\bar{\phi}, Y) = \begin{bmatrix} 1 & y_1^0 & y_2^0 & \frac{1}{2}(y_1^0)^2 & y_1^0 y_2^0 & \frac{1}{2}(y_2^0)^2 \\ 1 & y_1^1 & y_2^1 & \frac{1}{2}(y_1^1)^2 & y_1^1 y_2^1 & \frac{1}{2}(y_2^1)^2 \\ 1 & y_1^2 & y_2^2 & \frac{1}{2}(y_1^2)^2 & y_1^2 y_2^2 & \frac{1}{2}(y_2^2)^2 \\ 1 & y_1^3 & y_2^3 & \frac{1}{2}(y_1^3)^2 & y_1^3 y_2^3 & \frac{1}{2}(y_2^3)^2 \\ 1 & y_1^4 & y_2^4 & \frac{1}{2}(y_1^4)^2 & y_1^4 y_2^4 & \frac{1}{2}(y_2^4)^2 \end{bmatrix}.$$

We point out that if the linear system in (3.12) is fully determined, then the number of sample points in Y is equal to $p_1 = \frac{1}{2}(n+1)(n+2)$.

We can now introduce for quadratic interpolation models a definition similar to Definition 3.2.1.

Definition 3.3.1. *$Y = \{y^0, y^1, ..., y^p\}$ is a poised set for quadratic interpolation in $\mathbb{R}^n$ if the corresponding matrix $M(\phi, Y)$ is nonsingular for some basis ϕ in $\mathscr{P}_n^2$.*

If the matrix $M(\phi, Y)$ is nonsingular, the following lemma shows that there exists a unique quadratic interpolating model m such that (3.12) holds.

Lemma 3.3.1. *Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and a poised set $Y \in \mathbb{R}^n$, the quadratic interpolating polynomial $m(x)$ exists and is unique.*

Note that the poisedness of Y and the uniqueness of the interpolating model depend neither on f nor on the basis ϕ. Moreover, we point out that the discussion on the condition number of $M(\phi, Y)$ reported in Section 3.2 applies to the case of quadratic interpolation as well.

3.3.1 Error bounds in quadratic interpolation

In order to derive bounds on the approximation error, assume that the interpolation set $Y = \{y^0, y^1, \cdots, y^p\}$ is poised for quadratic interpolation, it is contained in the ball $B(y^0; \Delta)$ of radius Δ, and the number of interpolation points is $p_1 = p + 1 = (n+1)(n+2)/2$. Furthermore, assume that the function f is twice continuously differentiable in an open domain Ω containing $B(y^0; \Delta)$ and $\nabla^2 f$ is Lipschitz continuous in Ω with constant $\nu_Q > 0$.

Suppose that the quadratic interpolation model can be represented in the following form

$$m(x) = c + g^\top x + \frac{1}{2} x^\top H x = c + \sum_{1 \le k \le n} g_k x_k + \frac{1}{2} \sum_{1 \le k, \ell \le n} h_{k\ell} x_k x_\ell, \tag{3.13}$$

where H is a symmetric matrix of order n. We are interested in estimating the error bounds of the quadratic interpolation. Given a point x, we use $e^f(x)$, $e^g(x)$, and $E^H(x)$ to denote the error in the function value, gradient, and Hessian when the quadratic interpolation model is used to approximate the true function. Recalling (3.13), we can write

$$\begin{aligned} e^f(x) &= m(x) - f(x), \\ e^g(x) &= \nabla m(x) - \nabla f(x) = Hx + g - \nabla f(x), \\ E^H(x) &= \nabla^2 m(x) - \nabla^2 f(x) = H - \nabla^2 f(x). \end{aligned}$$

If for all the interpolation points in Y we subtract $m(x) = f(x) + e^f(x)$ from (3.1), we have

$$m(y^i) - m(x) = f(y^i) - f(x) - e^f(x),$$

and, recalling (3.13), we obtain

$$c + g^\top (y^i) + \frac{1}{2}(y^i)^\top H y^i - (c + g^\top x + \frac{1}{2} x^\top H x) = f(y^i) - f(x) - e^f(x), \quad i = 0, \ldots, p.$$

Therefore, rearranging the terms, we have

$$g^\top (y^i - x) + \frac{1}{2}(y^i - x)^\top H (y^i - x) + (y^i - x)^\top H x = f(y^i) - f(x) - e^f(x), \quad i = 0, \ldots, p.$$

Applying the second-order Taylor expansion to f around x, we obtain for all $i = 0, \ldots, p$

$$g^\top (y^i - x) + \frac{1}{2}(y^i - x)^\top H(y^i - x) + (y^i - x)^\top Hx = \nabla f(x)^\top (y^i - x) + \frac{1}{2}(y^i - x)^\top \nabla^2 f(x)(y^i - x) - e^f(x) + O(\Delta^3).$$

Therefore,

$$(Hx + g - \nabla f(x))^\top (y^i - x) + \frac{1}{2}(y^i - x)^\top (H - \nabla^2 f(x))(y^i - x) = O(\Delta^3) - e^f(x), \quad i = 0, \ldots, p,$$

and

$$(e^g(x))^\top (y^i - x) + \frac{1}{2}(y^i - x)^\top (H - \nabla^2 f(x))(y^i - x) = O(\Delta^3) - e^f(x), \quad i = 0, \ldots, p.$$

To cancel $e^f(x)$, we subtract the equation associated with $i = 0$ from all the other equations (note that $y^0 = 0$), obtaining

$$\left(y^i - y^0\right)^\top \left(e^g(x) - E^H(x)x\right) + \frac{1}{2}\left(y^i - y^0\right)^\top \left[H - \nabla^2 f(x)\right]\left(y^i - y^0\right) = O\left(\Delta^3\right), \quad i = 1, \ldots, p.$$

We can write the previous equation in the following form

$$\sum_{1 \le k \le n} (y^i_k - y^0_k) t_k(x) + \frac{1}{2} \sum_{1 \le k \le n} \left(y^i_k - y^0_k\right)^2 E^H_{kk}(x) + \sum_{1 \le \ell < k \le n} \left[(y^i_k - y^0_k)(y^i_\ell - y^0_k)\right] E^H_{k\ell}(x) = O\left(\Delta^3\right), \quad i = 1, \ldots, p,$$

which is a linear system. Equivalently, we can use the matrix form as follows

$$Q_{p \times p} \begin{bmatrix} t(x) \\ e^H(x) \end{bmatrix} = O\left(\Delta^3\right), \tag{3.14}$$

where $t(x) = e^g(x) - E^H(x)x = e^g(x) - \left[H - \nabla^2 f(x)\right]x$ and $e^H(x)$ is a vector of dimension $n + n(n-1)/2$ storing the diagonal elements E^H_{kk}, $k = 1, \ldots, n$, and $E^H_{k\ell}$, with $1 \le \ell < k \le n$.

Remark 3.3.1. *The matrix $Q_{p \times p}$ in linear system (3.14) does not depend on the point x.*

We now want to find an upper bound on the right hand side of (3.14). Note that each element of this vector is the difference of two terms which can be bounded by $\nu_Q \|y^i - x\|^3/6$ and $\nu_Q \|x\|^3/6$, respectively, where ν_Q is the Lipschitz constant of $\nabla^2 f(x)$ in Ω. Since all the interpolation points are in a ball of radius Δ, we have $\|y^i - x\| \le 2\Delta$ and $\|x\| \le \Delta$. As a result, the aforementioned difference can be bounded by $3\Delta^3/2$. Therefore, a bound on the ℓ_2 norm of the right hand side can be expressed as

$$\left\| Q_{p \times p} \begin{bmatrix} t(x) \\ e^H(x) \end{bmatrix} \right\| \le \frac{3}{2} p^{\frac{1}{2}} \nu_Q \Delta^3. \tag{3.15}$$

Now let us consider the scaled matrix of $Q_{p\times p}$,

$$\hat{Q}_{p\times p} = Q_{p\times p}\begin{bmatrix} D_\Delta^{-1} & 0 \\ 0 & D_{\Delta^2}^{-1} \end{bmatrix}, \tag{3.16}$$

where D_Δ is a diagonal matrix of dimension n whose diagonal entries are Δ and D_{Δ^2} is a diagonal matrix of dimension $p-n$ whose diagonal entries are Δ^2. Note that the scaled matrix $\hat{Q}_{p\times p}$ is the same as the the matrix $Q_{p\times p}$ corresponding to the scaled interpolation set $\hat{Y} = Y/\Delta$.

The following theorem states the error bound for the quadratic interpolation case. For the proof of this theorem, we refer the reader to [17].

Theorem 3.3.1. *Let $Y = \{y^0, y^1, \cdots, y^p\} \subset \mathbb{R}^n$, with $p_1 = p+1 = (1/2)(n+1)(n+2)$, be a poised set for quadratic interpolation whose points are contained in the ball $B(y^0;\Delta)$. Let the function f be twice continuously differentiable in an open domain Ω containing $B(y^0;\Delta)$ and assume that $\nabla^2 f$ is Lipschitz continuous in Ω with constant $\nu_Q > 0$.*
Then, for all points y in $B(y^0;\Delta)$, we have that

- *the error between the Hessian of the quadratic interpolation model and the Hessian of the function f satisfies*
$$\|E^H\| \leq \kappa_{eh}\Delta, \tag{3.17}$$
- *the error between the gradient of the quadratic interpolation model and the gradient of the function satisfies*
$$\|e^g\| \leq \kappa_{eg}\Delta^2, \tag{3.18}$$
- *the error between the quadratic interpolation model and the function satisfies*
$$|e^f| \leq \kappa_{ef}\Delta^3, \tag{3.19}$$

where

$$\kappa_{eh} = \left(\alpha_Q^H p^{\frac{1}{2}} \nu_Q \left\|\hat{Q}_{p\times p}^{-1}\right\|\right),$$
$$\kappa_{eg} = \left(\alpha_Q^g p^{\frac{1}{2}} \nu_Q \left\|\hat{Q}_{p\times p}^{-1}\right\|\right),$$
$$\kappa_{ef} = \left(\alpha_Q^f p^{\frac{1}{2}} \nu_Q \left\|\hat{Q}_{p\times p}^{-1}\right\| + \beta_Q^f \nu_Q\right),$$

and

$$\alpha_Q^H = \frac{3\sqrt{2}}{2}, \quad \alpha_Q^g = \frac{3(1+\sqrt{2})}{2}, \quad \alpha_Q^f = \frac{6+9\sqrt{2}}{4}, \quad \beta_Q^f = \frac{1}{6}.$$

3.4 Minimum Frobenius norm quadratic models

In many DFO applications, computing the function value may be time-consuming and using a determined quadratic interpolation model may be impracticable due to the large number of sample points to evaluate. For instance, assume we have to solve a three-dimensional problem. In this case,

building a determined quadratic interpolation model requires 10 function evaluations per iteration. If each function evaluation takes 2 seconds, then it would be reasonable to build an accurate quadratic interpolation model at each iteration. Instead, if each function evaluation takes 1 hour, then it would take 10 hours to build a complete quadratic interpolation model at each iteration. Now, assume we have to solve a problem with 100 variables. In this case, building an accurate quadratic interpolation model requires 5151 function evaluations per iteration. Even if each function evaluation takes only 2 seconds, a complete model would require hours of CPU time.

The previous examples show common issues in DFO and highlight the possible inadequacy of determined models in practice, despite the remarkable quality of the approximation they guarantee. The adoption of underdetermined quadratic interpolation models allows overcoming the aforementioned issue since the number of sample points required is fewer than the number needed by fully determined models. Moreover, ensuring a number of points larger than the number required in linear interpolation enables the adopted optimization algorithm to benefit from capturing some of the curvature of the function f, thus leading to an improved local convergence rate. Additional advantage can be obtained when the sparsity pattern of the Hessian matrix of the function f is known. However, such a topic is out of the scope of this thesis and we refer the reader to [12] and [13] for interpolation-based trust-region algorithms exploiting partial separability of functions.

Now, consider the case where the number of interpolation points p in Y is smaller than the number of elements q in the polynomial basis $\phi = \{\phi_0(x), \phi_1(x), \ldots, \phi_p(x)\}$, i.e., $p < q$. Let $M(\phi, Y)$ be the matrix defined by the interpolation conditions

$$m(y^i) = \sum_{k=0}^{q} \alpha_k \phi_k(y^i) = f(y^i), \quad i = 0, \cdots, p. \tag{3.20}$$

We can see that $M(\phi, Y)$ has now more columns than rows. Thus, the coefficients $\alpha_0, \alpha_1, \cdots, \alpha_q$ are no longer unique.

Let us consider the quadratic interpolation model associated with the natural basis $\bar{\phi}$. To properly introduce the underdetermined models, we split the natural basis into linear and quadratic parts, namely, $\bar{\phi}_L = \{1, x_1, x_2, \cdots, x_n\}$ and $\bar{\phi}_Q = \{\frac{1}{2}x_1^2, x_1x_2, \cdots, x_1x_n, \frac{1}{2}x_2^2, x_2x_3, \cdots, x_{n-1}x_n, \frac{1}{2}x_n^2\}$. Therefore, the interpolation model can be written as

$$m(x) = \alpha_L^\top \bar{\phi}_L(x) + \alpha_Q^\top \bar{\phi}_Q(x),$$

where α_L and α_Q are the corresponding parts of the cofficients α. Recalling that the Frobenius norm of a squared matrix A is $\|A\|_F^2 = \sum_{1 \le i,j \le n} a_{ij}^2$ (equivalently, $\|A\|_F^2 = tr(A^\top A)$, where the trace is the sum of the diagonal entries), the *minimum Frobenius norm* solution α^{mfn} can be defined as a solution to the following optimization problem in α_L and α_Q

$$\begin{aligned} &\min \frac{1}{2}\|\alpha_Q\|^2 \\ &\text{s.t.} \quad M(\bar{\phi}_L, Y)\alpha_L + M(\bar{\phi}_Q, Y)\alpha_Q = F(Y). \end{aligned} \tag{3.21}$$

Due to the choice of the basis $\bar{\phi}(x)$ and to the separation of the coefficients $\alpha = (\alpha_L, \alpha_Q)^\top$, minimizing the norm α_Q is equivalent to minimizing the Frobenius norm of the Hessian of $m(x)$. If $\|Y\| =$

$(1/2)(n+1)(n+2)$ and $M(\bar{\phi},Y)$ is nonsingular, it reduces to determined quadratic interpolation. Problem (3.21) is a constrained quadratic program with a closed form solution. To guarantee that there exists a unique a solution for this problem, the matrix $F(\bar{\phi},Y)$ must be nonsingular.

$$F(\bar{\phi},Y) = \begin{bmatrix} M\left(\bar{\phi}_Q,Y\right)M\left(\bar{\phi}_Q,Y\right)^\top & M\left(\bar{\phi}_L,Y\right) \\ M\left(\bar{\phi}_L,Y\right)^\top & 0 \end{bmatrix}. \tag{3.22}$$

Hence, the poisedness of the sample set Y in the minimum Frobenius norm sense is defined as the nonsingularity of matrix $F(\bar{\phi},Y)$, which in turn implies poisedness in the linear interpolation sense and poisedness for quadratic underdetermined interpolation in the minimum-norm sense.

Remark 3.4.1. *Matrix $F(\bar{\phi},Y)$ is nonsingular if and only if $M(\bar{\phi}_L,Y)$ has full column rank and $M(\bar{\phi}_Q,Y)M(\bar{\phi}_L,Y)^\top$ is positive definite in the null space of $M(\bar{\phi}_L,Y)^\top$.*

3.4.1 Least Frobenius norm updating of quadratic models

Powell [56] suggests that we can choose the solution to the underdetermined interpolation system (3.10) in the Frobenius norm sense by minimizing the change of the second order derivative matrix to the previously calculated one. That is to say, α is the solution of the system

$$\begin{aligned} \min \quad & \frac{1}{2}\left\|\alpha_Q - \alpha_Q^{pre}\right\|^2 \\ \text{s.t.} \quad & M\left(\bar{\phi}_L,Y\right)\alpha_L + M\left(\bar{\phi}_Q,Y\right)\alpha_Q = F(Y). \end{aligned} \tag{3.23}$$

In [56], Powell suggests that one can solve a shifted problem on $\alpha_Q^{dif} = \alpha_Q - \alpha_Q^{pre}$ instead of solving (3.23) directly. Then, we can obtain α_Q as the sum of α_Q^{dif} and α_Q^{pre}. The shifted problem can be stated as

$$\begin{aligned} \min \quad & \frac{1}{2}\left\|\alpha_Q^{dif}\right\|^2 \\ \text{s.t.} \quad & M\left(\bar{\phi}_L,Y\right)\alpha_L^{dif} + M\left(\bar{\phi}_Q,Y\right)\alpha_Q^{dif} = F^{dif}(Y), \end{aligned} \tag{3.24}$$

where $F^{dif}(Y) = F(Y) - m^{pre}(Y)$. Moreover, in [56] Powell shows that when f is quadratic, we have

$$\|H - \nabla^2 f\| \leq \|H^{pre} - \nabla^2 f\|. \tag{3.25}$$

In other words, the absolute error in the optimal solution H^* (relatively to $\nabla^2 f$) is decreasing. This result indicates that Powell's least Frobenius update can achieve good performance.

Chapter 4

Existing Hessian approximation attempts in nonlinear optimization

The knowledge of the Hessian matrix is particularly valuable in nonlinear optimization since algorithms may significantly benefit from the availability of this information. In particular, second-order derivatives may allow the algorithms to achieve faster convergence rates, as is the case of Newton's method, which shows a quadratic local convergence rate. However, two main difficulties prevent using the Hessian in optimization algorithms. On the one hand, when the dimension of the problem is large, calculating the Hessian is impractical due to either limits in computational resources or long computing time required. On the other hand, even if the dimension is small, computing the Hessian for the function of interest may be challenging. Therefore, several methods have been proposed to approximate the Hessian matrix of a function. This chapter reviews the main approaches adopted in nonlinear optimization to achieve this goal.

4.1 Overview of Hessian approximation attempts

Among the many approaches proposed in the literature, techniques from numerical analysis and computer science play an important role. Finite difference methods [33, 52] aim to approximate the derivatives of a function by using finite differences of function values computed at sufficiently close points. The accuracy of the approximation depends on the distance between the points used to evaluate the function. Moreover, when higher-order derivatives are required, as is the case for the second-order partial derivatives of the Hessian matrix, round-off errors may prevent using numerical techniques. In such cases, automatic differentiation methods [4, 7, 38] are preferable. In particular, they are based on techniques allowing the numerical evaluation of the derivatives of a function specified by a computer program. The idea is to repeatedly apply the chain rule to every elementary operation and function executed by the computer program.

Instead of directly approximating the Hessian, a different approach consists in resorting to an iterative method that starts from a first matrix representing a rough estimate of the Hessian and then refines the approximation by updating the matrix at each iteration. For example, the so-called secant equation $\nabla f(x_{k+1}) - \nabla f(x_k) = \nabla^2 f(x_{k+1})(x_{k+1} - x_k)$ plays an important role in the BFGS method, which is the most popular method based on the iterative update of an Hessian approximation (see

Section 2.5 in Chapter 2). By ensuring the positive definiteness of the new Hessian estimation H_{k+1}, BFGS allows the Hessian inverse to be computed at a reasonable computational cost. Nocedal and Wright prove that this method asymptotically converges to a local minimum at a superlinear rate [52]. A step forward is represented by L-BFGS, which is the state of the art method for unconstrained optimization when second-order derivatives are not available [51].

Powell [58] estimates the Hessian matrix using finite differences of the gradient by first dividing the Hessian columns into groups. It is possible to find approximations to different Hessian columns at once by using the symmetry and known sparsity pattern of the Hessian. This method is cheap in computer arithmetic and provides better results when compared to [20], which is devoted to the estimation of sparse Jacobian matrices (we recall that the Hessian matrix of a function can be viewed as the Jacobian matrix of the gradient).

If the Hessian matrix is sparse and its sparsity pattern is known, the approach in [30] enforces multiple secant equations in a least squares sense, solving a positive semi-definite system of equations in the nonzero Hessian components. Their approach does not show a significant improvement compared to the L-BFGS or trust-region Newton-CG method.

A more recent approach [34] imposes the secant equations componentwise, leading to fewer equations when taking into account the available sparsity pattern. The numerical results show that the algorithm can find the Hessian approximation fast and accurately when the number of nonzero entries per row is relatively low.

4.2 Approximating the Hessian matrix by finite differences

In this section, we show how to use Taylor's theorem to approximate first-order and second-order derivatives of a function by using the finite-difference approach. Since derivatives are a measure of the sensitivity of a function to infinitesimal changes in the values of the variables, the idea is to obtain a suitable approximation by taking small and finite perturbations in the variables and then calculating the differences in the function values. In case of gradient approximation, we use finite differences of the function of interest f, while in case of Hessian approximation, we can use either finite differences of f or of the gradient function ∇f.

4.2.1 Approximating the gradient

Given a point $x \in \mathbb{R}^n$ and a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient vector $\nabla f(x)$ can be approximated by calculating proper function values and then performing some elementary arithmetic. One way to approximate the partial derivative $\frac{\partial f}{\partial x_i}$ with respect to the i-th variable at the given point x is to use the *forward-difference* formula, which is given by

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x+\varepsilon e_i) - f(x)}{\varepsilon}, \quad \text{for all } i = 1, \ldots, n, \tag{4.1}$$

where $\varepsilon > 0$ is a small positive scalar and $e_i \in \mathbb{R}^n$ is the i-th canonical vector. An alternative way is to use the *central-difference* formula, which is given by

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x+\varepsilon e_i) - f(x-\varepsilon e_i)}{2\varepsilon}, \quad \text{for all } i = 1, \ldots, n. \tag{4.2}$$

Both finite-difference methods approximate the objective function in a neighborhood of x with a linear function $m : \mathbb{R}^n \to \mathbb{R}$ of the form $m(y) = m(x) + g(x)^\top (y - x)$, where $g(x)$ is the gradient approximation whose components are given by the right-hand side of either (4.1) or (4.2). Note that the forward-difference formula requires the evaluation of f at $(n+1)$ points, namely, the point x as well as the n perturbed points $x + \varepsilon e_i,\ i = 1, \cdots, n$. Instead, the central-difference formula is about twice as expensive as the forward-difference one since f needs to be evaluated at $2n$ points, namely, the points $x \pm \varepsilon e_i,\ i = 1, 2, \cdots, n$. The following theorems show that the central-difference formula allows achieving a more accurate and stable approximation than the forward-difference formula [5].

Theorem 4.2.1. *Suppose that the gradient of the function $f(x)$ is L-Lipschitz continuous. Let $g(x)$ denote the forward-difference approximation to the gradient $\nabla f(x)$. Then, for all $x \in \mathbb{R}^n$,*

$$\|g(x) - \nabla f(x)\| \leq \frac{\sqrt{n} L \varepsilon}{2}. \tag{4.3}$$

Theorem 4.2.2. *Suppose that the Hessian of function $f(x)$ is M-Lipschitz continuous. Let $g(x)$ denote the central-difference approximation to the gradient $\nabla f(x)$. Then, for all $x \in \mathbb{R}^n$,*

$$\|g(x) - \nabla f(x)\| \leq \frac{\sqrt{n} M \varepsilon^2}{6}. \tag{4.4}$$

4.2.2 Approximating the Hessian

To approximate the Hessian matrix of a twice continuously differentiable function f by using finite-difference approximation, approaches based on either the gradients or the function values computed at specific points can be used. The choice of the approach depends on the availability of the gradient. In particular, when the user is not able or is not willing to provide the gradient, it is necessary to approximate the Hessian by using only function values. We start by showing the approach using only function values at specific points and, afterward, we derive a formula that requires only gradient evaluations. It is important to point out that both techniques are based on the application of the Taylor's theorem and that the latter approach allows estimating Hessian-vector products in place of the full Hessian. We assume that the second-order derivatives of f exist and are Lipschitz continuous.

By using the Taylor's theorem, we can write for some $t \in (0, 1)$

$$\begin{aligned} f(x+p) &= f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x+tp) p \\ &= f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x) p + O\left(\|p\|^3\right), \end{aligned} \tag{4.5}$$

where p is referred to as the perturbation vector. If we plug the scaled unit vectors $p = \varepsilon e_i$, $p = \varepsilon e_j$, and $p = \varepsilon(e_i + e_j)$ into (4.5), we obtain three different equations. Combining the results, we have

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x+\varepsilon e_i + \varepsilon e_j) - f(x+\varepsilon e_i) - f(x+\varepsilon e_j) + f(x)}{\varepsilon^2} + O(\varepsilon). \tag{4.6}$$

Equation (4.6) provides a way to approximate each element of the Hessian matrix by using only function values. However, notice that by using (4.6) to approximate the Hessian, which is a symmetric matrix, we need to calculate the function values at $x + \varepsilon e_i + \varepsilon e_j$ for all $i, j = 1, \ldots, n$ such that $i < j$, resulting in $n(n+1)/2$ points, as well as at the n points $x + \varepsilon e_i$, $i = 1, \cdots, n$. The resulting computational cost can be significantly decreased when the Hessian is sparse and the sparsity pattern is known, as it will be shown in the next paragraphs. In this case, the elements equal to zero can be skipped, thus obtaining remarkable savings in the total computational cost.

In some circumstances, given a point $x \in \mathbb{R}^n$, even if the Hessian $\nabla^2 f(x)$ of the function f is not available, the gradient $\nabla f(x)$ may be known. By applying Taylor's theorem, we have

$$\nabla f(x + \varepsilon p) = \nabla f(x) + \varepsilon \nabla^2 f(x) p + O\left(\varepsilon^2\right).$$

Thus,

$$\nabla^2 f(x) p \approx \frac{\nabla f(x + \varepsilon p) - \nabla f(x)}{\varepsilon}. \tag{4.7}$$

Notice that (4.7) provides the forward-difference approximation of the Hessian-vector product $\nabla^2 f(x)p$, where p is a given vector in $\mathbb{R}^n$. Approximating such a product is required in some algorithms, such as the Newton-CG, which are based on the knowledge of the Hessian-vector products along given directions p. Even though computing Hessian-vector products is affordable and cheap in some applications, (4.7) may be a useful tool in DFO contexts. Note that one gradient evaluation is required at the point $x + \varepsilon p$. The error in the approximation given by (4.7) is proved to be $O(\varepsilon)$. To increase the accuracy of the Hessian-vector product approximation, a central-difference formula can be used. However, in this case also the evaluation of $\nabla f(x - \varepsilon p)$ is required (see, e.g., [24, Section 5.6]).

If in (4.7) p is taken to be e_i, where $e_i \in \mathbb{R}^n$ is the i-th canonical vector, we can resort to the finite-difference approach to approximate the whole Hessian matrix $\nabla^2 f(x)$ by estimating one column at a time. For instance, the i-th column of the Hessian can be approximated by using

$$\nabla^2 f(x) e_i \approx \frac{\nabla f(x + \varepsilon_i e_i) - \nabla f(x)}{\varepsilon_i},$$

where ε_i is an appropriate small positive scalar. Note that to determine the full Hessian approximation, $n+1$ gradient evaluations are required.

Now, let us assume that the sparsity pattern of the Hessian matrix is known. We know that the Hessian matrix can be estimated by computing differences of gradients according to (4.7). In general, when the Hessian is sparse, the number of gradients to compute may be small compared to

the dimension of the problem because of the sparsity and symmetry of the Hessian matrix, namely,

$$\nabla^2 f_{ij}(x) = 0, \quad \text{for some } i,j = 1,\cdots,n,$$
$$\nabla^2 f_{ij}(x) = \frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \nabla^2 f_{ji}(x), \quad \text{for all } i,j = 1,\cdots,n.$$

Since estimating a sparse Hessian by using finite-difference approximation of the gradients is an attractive tool, many works have addressed this topic. For example, Coleman and Moré [11] use graph theory to develop algorithms for estimating sparse Hessian matrices by resorting to graph coloring. We illustrate the idea with a simple example that is provided also in [52]. Although the example can be interpreted also in terms of graph coloring, we omit this interpretation and refer the reader to [11] for further details.

Example 4.2.1. *Let $f(x) = x_1 \sum_{i=1}^{n} i^2 x_i^2$. In this case, we know the sparsity pattern of the Hessian matrix $\nabla^2 f(x)$, whose structure is "arrowhead". For instance, when $n = 6$, the Hessian can be written as follows*

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ \times & & \times & & & \\ \times & & & \times & & \\ \times & & & & \times & \\ \times & & & & & \times \end{bmatrix}, \tag{4.8}$$

where $\times$ denotes a nonzero element. We can approximate the first column of $\nabla^2 f(x)$ by setting the perturbation vector $p = e_1$ and using (4.7). Note that the symmetry of the Hessian implies that the first column of $\nabla^2 f(x)$ is equal to its first row. Therefore, the only elements of the Hessian that we need to approximate are the diagonal entries $\nabla^2 f_{22}(x), \nabla^2 f_{33}(x), \cdots, \nabla^2 f_{66}(x)$. Let us focus on the section of the Hessian that we obtain by leaving out the first column and row. We want to simultaneously estimate as many diagonal entries of the Hessian section as possible by choosing the perturbation vector p in (4.7) in a smart way. In particular, observe that the i-th component of $\nabla f(x)$ depends only on x_1 and x_i, thus implying that

$$\nabla f(x + \varepsilon(e_2 + e_3 + \cdots + e_6))_i = \nabla f(x + \varepsilon e_i)_i, \quad \text{for all } i = 1,\ldots,n.$$

Therefore, the perturbation vector

$$p = e_2 + e_3 + \cdots + e_6 = (0,1,1,1,1,1)^\top$$

allows us to approximate the diagonal entries by evaluating the gradient ∇f at the point $x + \varepsilon(e_2 + e_3 + \cdots + e_6)$. Indeed, by applying the forward-difference formula to each component, we obtain

$$\frac{\partial^2 f}{\partial x_i^2}(x) \approx \frac{\nabla f(x + \varepsilon e_i)_i - \nabla f(x)_i}{\varepsilon} = \frac{\nabla f(x + \varepsilon p)_i - \nabla f(x)_i}{\varepsilon}, \quad i = 2,3,\ldots,6. \tag{4.9}$$

As a result, the full Hessian matrix can be approximated by evaluating the gradient at just three points: x, $x+\varepsilon e_1$, and $x+\varepsilon\,(e_2+e_3+\cdots+e_6)$.

In general, suppose that the sparsity pattern of the Hessian is known. We can partition the index set of the components, i.e., $N=\{1,\cdots,n\}$, into m subsets I_j such that $N=\bigcup_{j=1}^{m} I_j$ and $I_h \bigcap I_s = \varnothing$ for all $h,s=1,\ldots,m$. The rule to choose the subsets is to include in each I_j all the indexes that are associated with orthogonal rows of the Hessian. For any $j=1,\ldots,m$, we can write

$$\nabla^2 f(x)\sum_{i\in I_j} e_i \approx \frac{\left[\nabla f\left(x+\varepsilon\sum_{i\in I_j} e_i\right)-\nabla f(x)\right]}{\varepsilon}.$$

Note that only $m+1$ gradients are required. Since $m\leq n$, a significant saving compared to a non-sparse Hessian may be achieved. Moreover, this result can be improved by taking into account the symmetry of the Hessian [11, 30].

It is important to remark that any differentiable function with a sparse Hessian is partially separable [39]. If f is a partially separable function, then the function f can be expressed as $f(x)=\sum_{i=1}^{\ell} f_i(x)$. In this case, the approximation of the Hessian $\nabla^2 f(x)$ at a given point x is given by

$$\nabla^2 f(x)=\sum_{i=1}^{\ell}\nabla^2 f_i(x),$$

where each $\nabla^2 f_i(x)$ can be approximated by its own secant equation

$$\nabla^2 f_i(x)p\approx \nabla f_i(x+p)-\nabla f_i(x).$$

Since each secant equation involves only a few variables, a computationally cheap sparse approximation can be obtained. This approach has been generalized to group-partial separability and forms the basis of the approximations used in LANCELOT [14].

Effective schemes to approximate a sparse Hessian without resorting to graph coloring are addressed in [58]. From the algorithmic point of view, software for approximating a sparse Hessian has been developed in [10] and [31]. Finally, the application of graph coloring to finite differencing has been also discussed in [32].

4.3 Approximating a sparse Hessian matrix with optimal hereditary properties

This section briefly reviews the method proposed in [30], which addresses nonlinear optimization problems with objective function characterized by a large sparse Hessian matrix and known sparsity pattern. The goal of the proposed approach is to approximate the sparse Hessian matrix of the objective function by taking advantage of the Hessian sparsity. To this end, a least squares problem with quasi-Newton constraints is solved.

Given a function $f:\mathbb{R}^n\to\mathbb{R}$, let

$$\Omega(\nabla^2 f)\stackrel{\text{def}}{=}\{(i,j):\nabla^2 f_{ij}(x)=0 \quad \text{for all } x\in\mathbb{R}^n\}$$

be the sparsity pattern of the Hessian matrix $\nabla^2 f(x)$. We introduce the matrices containing the m most recent differences of points and gradients yielded by the optimization algorithm, namely,

$$\Delta = [s_{k+m-1}, \cdots, s_k] \text{ and } \Gamma = [y_{k+m-1}, \cdots, y_k],$$

where $s_\ell = x_\ell - x_{\ell-1}$ and $y_\ell = \nabla f(x_\ell) - \nabla f(x_{\ell-1})$ for all $\ell = k, \ldots, k+m-1$. Suppose that H is an approximation to the Hessian matrix $\nabla^2 f(x)$. Our goal is to determine H by ensuring that it is symmetric and with the same sparsity pattern as $\nabla^2 f(x)$. That is to say,

$$\begin{aligned} &H = H^\top, \\ &H_{i,j} = 0, \quad \text{if } (i,j) \in \Omega(\nabla^2 f). \end{aligned}$$

Moreover, H is required to satisfy m quasi-Newton conditions simultaneously, i.e.,

$$H\Delta = \Gamma. \tag{4.10}$$

When f is a quadratic function, there always exists a matrix H such that (4.10) is satisfied and the exact Hessian can be recovered from (4.10) by requiring proper linear independence assumptions. When f is non-quadratic, one cannot expect to find a matrix H such that (4.10) is solved exactly with the same sparsity pattern as $\nabla^2 f(x)$. Thus, we attempt to find a least squares solution to the matrix equation (4.10) by solving the following problem

$$\text{(CPP)} \begin{cases} \min\limits_{H} & \|H\Delta - \Gamma\|_F^2 \\ \text{s.t} & H^\top = H \\ & H_{ij} = 0, \text{ for all } (i,j) \in \Omega(\nabla^2 f), \end{cases}$$

where $\|A\|_F^2 = \text{trace}(AA^\top) = \text{trace}(A^\top A)$ is the squared Frobenius norm of A. This problem is often referred to as the *Constrained Procrustes Problem* (CPP) (see, e.g., [42] and [2]). Note that the CPP is a convex quadratic programming problem, the objective function is convex and bounded below by zero, and the constraints of the CPP are linear and consistent with the symmetry of $\nabla^2 f(x)$ (i.e., $(i,j) \in \Omega(\nabla^2 f)$ if and only if $(j,i) \in \Omega(\nabla^2 f)$). Hence, there always exists a solution for this problem.

In [30], the authors prove that the approximate Hessian H can always be computed by solving a positive semi-definite system of equations in the nonzero elements of H. Moreover, the simple structure of the CPP allows a straightforward generalization to the case where some elements of the Hessian are known. It can be also shown that the solution of the CPP can be found by solving a sparse system of equations. By means of the constraints of the CPP, the Hessian approximation is symmetric and the sparsity is preserved. Instead, the positive-definiteness is not guaranteed. Hence, the approach proposed in this paper may be applied inside a trust region method, where positive-definiteness of the Hessian is not required. Necessary and sufficient conditions of the problem have been investigated for the case with positive definite Hessian.

4.4 Approximating a sparse Hessian matrix by solving componentwise secant equations

Drawing inspiration from the work by Fletcher, Grothey, and Leyffer [30], reviewed in Section 4.3, Gould has developed a more appealing approach [34], which is the focus of this section. The idea is to directly approximate the Hessian matrix by using both its sparsity pattern and the data pairs $\{s_\ell\}_{\ell=k}^{k+m-1}$ and $\{y_\ell\}_{\ell=k}^{k+m-1}$ accumulated in the last m iterations. For the sake of simplicity and without loss of generality, assume that ℓ starts from the first iteration so that we can redefine the previous sequences as $\{s_\ell\}_{\ell=1}^{m}$ and $\{y_\ell\}_{\ell=1}^{m}$. Instead of solving all the m conditions (4.10) used in [30], we want to consider as many conditions as possible. Let us focus on the i-th equation of the ℓ-th condition in (4.10), namely,

$$e_i^\top H s_\ell = e_i^\top y_\ell, \tag{4.11}$$

where e_i is the i-th unit vector in $\mathbb{R}^m$. Let s_{ql} and y_{ql} denote the q-th component of the vectors s_ℓ and y_ℓ. Equation (4.11) can be written in terms of the nonzero elements of H as follows

$$\sum_{j\in I_i} H_{ij} s_{j\ell} = y_{i\ell}, \quad \text{where } I_i = \{j : H_{ij} \neq 0\}. \tag{4.12}$$

To approximate the *i-th* row of the Hessian by solving a determined linear system, we need as many equations (4.12) as the number of unknown elements in the row i. Therefore, the number of conditions selected from the m conditions in (4.10) depends on $|I_i|$. If we do not take into account the symmetry of H, the unknown elements can be calculated in any order according to Algorithm 6.

Algorithm 6: Approximating a Sparse Hessian without Symmetry

for $i = 1, \cdots, n$

Calculate the unknown elements in the i-th row of H by solving the system composed of the equations given by (4.12) for all $\ell = m, m-1, \cdots, m-|I_i|+1$.

end (for)

Note that the non-diagonal elements are calculated twice in Algorithm 6. When taking the symmetry of H into account, (4.12) can be rewritten as

$$\sum_{j\in I_i^-} H_{ij} s_{j\ell} = y_{i\ell} - \sum_{j\in I_i^+} H_{ij} s_{j\ell}, \tag{4.13}$$

where

$$I_i^+ = \left\{ j : j \in I_i \text{ and } H_{ij} \text{ is already known } \right\} \text{ and } I_i^- = I_i \backslash I_i^+.$$

Hence, it is important to decide the order of the rows we are going to consider, as it will be shown in Example 4.4.1. In practice, it is more convenient to access the rows in decreasing order starting from the row having the smallest number of non-zero elements.

Example 4.4.1. *Assume $m = n$. Consider the two following Hessian matrices characterized by an "arrowhead" structure*

$$H_1 = \begin{bmatrix} \times & & & & \times \\ & \times & & & \\ & & . & & \vdots \\ & & & \times & \times \\ \times & \times & \cdots & \times & \times \end{bmatrix} \quad \text{and} \quad H_2 = \begin{bmatrix} \times & \times & \cdots & \times & \times \\ \times & \times & & & \\ \vdots & & . & & \\ \times & & & \times & \\ \times & & & & \times \end{bmatrix}. \tag{4.14}$$

Note that H_1 and H_2 are structurally symmetric permutations of one another. As regards H_1, each of the first $n-1$ rows requires two entries to be calculated, namely, the diagonal element and the element in the n-th column, while the last row requires n elements. If we take the symmetry into account, the elements in the last row do not need to be computed since they are equal to the elements in the n-th column. Therefore, the two nonzero elements in each row $i = 1,\ldots,n-1$ of the Hessian matrix can be recovered by using only two conditions (the ones associated with the latest pairs (s_n, y_n) and (s_{n-1}, y_{n-1})). On the contrary, in H_2 there are n unknown elements in the first row. Hence, n pairs (s_ℓ, y_ℓ), $\ell = 1,\cdots,n$, are required to compute such elements. In total, recovering H_1 starting from the first row requires solving $n-1$ systems with two equations each and one equation with only one variable, while the same strategy applied to H_2 requires solving one system with n equations and also $n-1$ equations with one variable (since one of the two elements is known from the solution of the system).

Hessian sparsity can be also analyzed in terms of the adjacency graph $\mathscr{G}$ of the Hessian matrix [11], which is a graph with n nodes, each associated with one of the n variables, such that two nodes i and j are joined by an arc if and only if $\nabla^2 f_{ij}(x) = \partial^2 f(x)/\partial x_i \partial x_j \neq 0$ for some x in $\mathbb{R}^n$. The degree of node i is the number of unknown nonzero entries in the i-th row minus one, namely, $|I_{i^*}^-| - 1$. Let $\mathscr{L}$ be a list of nodes that is updated across the iterations. Algorithm 7 provides a way to approximate the sparse Hessian matrix by using ordering and symmetry.

Algorithm 7: Approximating a Sparse Hessian with Symmetry

Calculate the adjacency graph $\mathscr{G}$ of $\nabla^2 f(x)$, compute the degree of each node, and initialize $\mathscr{L}$ with the set of nodes in $\mathscr{G}$.
while $\mathscr{L} \neq \varnothing$
  Find the node i^* with minimum degree among the nodes in $\mathscr{L}$.
  Calculate the unknown elements in the row i^* by applying (4.13) with $i = i^*$ for all $\ell = m, m-1, \cdots, m-|I_{i^*}^-|+1$.
  Update $\mathscr{L} = \mathscr{L} \backslash \{i^*\}$ and update the degree of each node.
end (while)

Algorithm 7 requires finding the node with the smallest degree at each iteration so that we can keep the number of required pairs (s_ℓ, y_ℓ) small at each iteration. The nodes could be initially ordered using a bucket sort algorithm [19], which has a linear cost $O(n)$ in terms of operations and memory locations.

It is important to note that at each iteration Algorithm 7 requires fewer floating-point operations than its previous iterations. However, there are two disadvantages. On the one hand, while the steps in Algorithm 6 could be performed in parallel, Algorithm 7 is to a large extent sequential; only when the nodes i and j of minimum degree satisfy $I_i^+ \cap I_j^+ = \varnothing$ (i.e., the i-th and j-th do not share the same unknown elements), the steps can be processed in parallel. The second drawback is that inaccurate estimates of components of H obtained from previous iterations lead to increasing errors when solving (4.13), even if the Hessian is a constant matrix ($\nabla^2 f(x) = \nabla^2 f(\bar{x})$ for all $(x,\bar{x}) \in \mathbb{R}^n \times \mathbb{R}^n$). Especially for large Hessian matrices, if errors occur in early iterations, in the last iterations the error may be significant. Since Algorithm 6 is immune to this second drawback because the entries in each row are calculated independently, one way to solve the aforementioned issue is to combine Algorithm 6 with Algorithm 7.

Assume that most of the rows of the Hessian are very sparse and the remaining rows are relatively dense. In particular, after performing a symmetric permutation, we can write the Hessian matrix in the following form

$$\begin{bmatrix} H_{11} & H_{12} \\ H_{12}^\top & H_{22} \end{bmatrix}, \tag{4.15}$$

where H_{11} and H_{12} are relatively sparse and H_{22} is relatively dense. The idea is to find the entries in $[H_{11}\ H_{12}]$ row-by-row by applying Algorithm 6, and then seek the elements in H_{22} by using Algorithm 7 with the symmetric property. Assume that H_{11} is an $n_1 \times n_1$ matrix, while H_{22} is $n_2 \times n_2$. The resulting method is reported in Algorithm 8.

Algorithm 8: Approximating a Sparse Hessian (combined version)

for $i = 1, \cdots, n_1$
  Calculate the unknown elements in the i-th row of $[H_{11}\ H_{12}]$ by solving the system composed of the equations given by (4.12) for all $\ell = m, m-1, \cdots, m-|I_i|+1$.
end (for)
Calculate the adjacency graph $\mathscr{G}$ of H_{22}, compute the degree of each node i, for all $i = n_1+1, \ldots, n_1+n_2$, and initialize $\mathscr{L}$ with the set of nodes in $\mathscr{G}$.
while $\mathscr{L} \neq \varnothing$
  Find the node i^* with minimum degree.
  Calculate the unknown elements in the row i^* by applying (4.13) with $i = i^*$ for all $\ell = m, m-1, \cdots, m-|I_{i^*}^-|+1$.
  Update $\mathscr{L} = \mathscr{L} \backslash \{i^*\}$ and update the degree of each node.
end (while)

Note that we could introduce a threshold η such that the i-th row is treated as dense if $|I_i| > \eta$, sparse otherwise. If in Algorithm 8 we apply Algorithm 7 recursively to the block $[H_{12}^\top\ H_{22}]$, inaccurate estimates of components of H_{22} obtained from previous iterations may lead to significant errors in the last iterations. However, recursion is unnecessary and can be avoided. In order to reduce the instability of Algorithm 8 and, at the same time, keep its benefits, a systematic algorithm is proposed

in Algorithm 9, where $\mathscr{N}$ is a list of nodes updated across the iterations.

Algorithm 9: Approximating Sparse Hessian (reducing instability)

Calculate the adjacency graph $\mathscr{G}$ of $\nabla^2 f(x)$, initialize $\mathscr{N}$ with the set of nodes of $\mathscr{G}$, and set $\mathscr{L} = \varnothing$.
while ($\mathscr{N} \neq \varnothing$)
  Find the node i^* with minimum degree.
  if ($m \geq |I_{i^*}|$)
    Calculate the unknown elements in the i^*-th row of H by solving the system composed of the equations given by (4.12) for all $\ell = m, m-1, \cdots, m-|I_{i^*}|+1$.
  else if ($|I_{i^*}^-| \leq m < |I_{i^*}|$)
    Calculate the unknown elements in the row i^* by applying (4.13) for all $\ell = m, m-1, \cdots, m-|I_{i^*}^-|+1$.
  else
    Skip the i^*-th row and update $\mathscr{L} = \mathscr{L} \cup \{i^*\}$.
  end (if)
  Update $\mathscr{N} = \mathscr{N} \backslash \{i^*\}$ and the degree of the remaining nodes.
end (for)
while ($\mathscr{L} \neq \varnothing$ and the size of $\mathscr{L}$ is decreased with respect to the previous iteration)
  Extract node q from $\mathscr{L}$.
  Attempt to calculate the unknown elements in the q-th row of H by solving the system composed of the equations given by (4.12) for all $\ell = m, m-1, \cdots, m-|I_q|+1$.
  If successful, update $\mathscr{L} = \mathscr{L} \backslash \{q\}$.
end (while)
return $H = (H + H^\top)/2$.

In order to reduce the instability, Algorithm 9 exploits the symmetry of H only when it is necessary. In particular, in the *for loop* the algorithm determines the i^*-th row of H by solving (4.12) if $m \geq |I_{i^*}|$. When $m < |I_{i^*}|$ but $m \geq |I_{i^*}^-|$, the algorithm calculates the row by solving (4.13), which exploits the symmetry. If $m < |I_{i^*}^-|$, the row cannot be uniquely determined since there are not enough conditions. However, since more elements in the considered row of H may become known at some later iteration because of the symmetry, at the end the algorithm attempts to determine the unknown elements in all the rows that have been skipped. Note that, thanks to the symmetry, this framework uses the smallest number of pairs (s_ℓ, y_ℓ) required to uniquely determine each row of H. Finally, the symmetry is enforced also by setting $H = (H + H^\top)/2$, which is effective for reducing the Frobenius norm error $\|H - \nabla^2 f\|_F$.

The last algorithm described in this section uses all the available information to determine the rows of H and it is not limited to the amount of information required for the unique determination. The idea is to create an over-determined system with a unique solution by using all m pairs (s_ℓ, y_ℓ) rather than determining the elements in the *i-th* row of H by using only $|I_i|$ pairs or less. In particular,

the i-th row of H is recovered by determining the least squares solution of the linear system

$$(H^\top)_i = \arg\min_b \|\Delta^\top b - (\Gamma^\top)_i\|_2^2, \tag{4.16}$$

where $H\Delta = \Gamma,\ \Delta = [s_{k+m-1}, \cdots, s_k]$, and $\Gamma = [y_{k+m-1}, \cdots, y_k]$. A situation suitable for applying this technique is when we want to minimize a non-quadratic function with non-constant Hessian within a trust-region method. Indeed, since there is a strong possibility that the vector pairs $\Delta,\ \Gamma$ are not consistent, the over-determined system may not have a unique solution.

It is important to point out that the least-squares approach is beneficial especially if the data in the matrices Δ and Γ are noisy, which may occur when the objective function or its gradient cannot be evaluated exactly. The scheme of the resulting method is reported in Algorithm 10.

Algorithm 10: Approximating Sparse Hessian (using all available information)

Apply Algorithm 9 replacing the statement *by solving (4.12)* with *by solving the least squares problem (4.16)*.

Numerical experiments show that Algorithm 6 can recover the elements of a sparse Hessian fast and accurately provided that the maximum number of entries per row is relatively small. However, if the number of dense rows is one or more, the performance of Algorithm 6 deteriorates. In contrast, Algorithm 7 runs faster since it takes advantage of the symmetry of the Hessian. However, since the accuracy tends to decrease when the entries recovered in the first iterations are affected by non-negligible errors, this approach is not recommended. The combined version, which is shown in Algorithm 8, resolves the main issues of these first two algorithms and, therefore, performs well. However, it is outperformed by both Algorithms 9 and 10 in terms of accuracy. Finally, Algorithm 10 guarantees a smaller approximation error than Algorithm 9 but requires solving a least-squares problem, which is more computationally intensive than finding unique solutions to the linear systems considered in Algorithm 9. Moreover, Algorithms 9 and 10 require the same number of pairs (s_ℓ, y_ℓ), which is the smallest one among all the considered algorithms.

Chapter 5

Hessian recovery from Hessian-vector products

In this chapter, we introduce a new approach to recover the Hessian of a function by using Hessian-vector products.

5.1 Hessian recovery

Let x be a given point. Suppose also that we have calculated f and ∇f at x as well as f at a number of points $y^1, \ldots, y^p$. We can then use quadratic interpolation to fit the data by determining a symmetric matrix H such that

$$f(x) + \nabla f(x)^\top (y^\ell - x) + \frac{1}{2}(y^\ell - x)^\top H (y^\ell - x) \;=\; f(y^\ell), \quad \ell = 1, \ldots, p. \tag{5.1}$$

Furthermore, given a set of vectors $v^1, \ldots, v^m$, with m possibly much smaller than n, suppose that we have calculated $w^j = \nabla^2 f(x) v^j$, $j = 1, \ldots, q$. Hence we could then ask our symmetric Hessian model H to satisfy $Hv^j = w^j$, $j = 1, \ldots, q$. However it is important to notice two immediate facts, reported in Remarks 5.1.1 and 5.1.2.

Remark 5.1.1. *First we cannot have $q > 1$. Any use of a pair v^1, v^2 would make the conditions $Hv^1 = w^1$ and $Hv^2 = w^2$ degenerate in H, in the sense that the matrix multiplying the component variables of H would be rank deficient. This fact can be easily confirmed from multiplying each by the other vector, i.e., by looking at $(v^2)^\top H v^1 = (v^2)^\top w^1$ and $(v^1)^\top H v^2 = (v^1)^\top w^2$. For illustration, suppose that $n = 2$. These two equations would be*

$$\begin{aligned}
(v^2)_1 (v^1)_1 h_{11} + [(v^2)_2 (v^1)_1 + (v^2)_1 (v^1)_2] h_{12} + (v^2)_2 (v^1)_2 h_{22} &= (v^2)^\top w^1, \\
(v^1)_1 (v^2)_1 h_{11} + [(v^1)_2 (v^2)_1 + (v^1)_1 (v^2)_2] h_{12} + (v^1)_2 (v^2)_2 h_{22} &= (v^1)^\top w^2,
\end{aligned}$$

and one can see that the two rows multiplying the H components are the same.

Remark 5.1.2. *Secondly, even when taking $q = 1$, one cannot consider $v^1 = y^\ell - x$, for any ℓ, for the exact same reason. In fact, multiplying $H(y^\ell - x) = w^1$ on the left by $(1/2)(y^\ell - x)^\top$ would lead us to*

$$\frac{1}{2}(y^\ell - x)^\top H(y^\ell - x) = \frac{1}{2}(y^\ell - x)^\top w^1,$$

which has the same term in H as of the corresponding interpolating condition in (5.1),

$$\frac{1}{2}(y^\ell - x)^\top H(y^\ell - x) = f(y^\ell) - f(x) - \nabla f(x)^\top (y^\ell - x).$$

Therefore, we would have two linearly dependent equations in the H components.

From Remark 5.1.1, we know that we can only consider one vector v for the Hessian multiplication $w = \nabla^2 f(x)\, v$, and from Remark 5.1.2, we know that this vector cannot be any of the interpolation vectors $y^\ell - x$. Then, in the same vein as it was done in [18, Section 5.3] for derivative-free optimization, a model Hessian H could then be calculated from the solution of the recovery problem

$$\min_H \quad \text{norm}(H) \quad \text{s.t.} \quad (5.1) \text{ and } Hv = w. \tag{5.2}$$

The norm(H) could be taken in a certain ℓ_1 sense, leading to a linear program (see [3]). It could also be set as the Frobenius norm, namely, norm$(H) = \|H\|_F$, leading to a quadratic program. Alternatively, one can recover a model Hessian in a least secant fashion (as done in [56] for derivative-free optimization using the Frobenius norm)

$$\min_H \quad \text{norm}(H - H^{prev}) \quad \text{s.t.} \quad (5.1) \text{ and } Hv = w, \tag{5.3}$$

where H^{prev} is a previously computed model Hessian (say, from a previous iteration of an optimization scheme).

5.2 Theoretical motivation

Error decrease

We will now see that when f is quadratic the error in the difference between the optimal solution H^* of (5.3) and the true Hessian decreases relatively to the previous estimate H^{prev}. To prove such a result it is convenient to use the Frobenius norm in (5.3) and consider

$$\min_H \quad \frac{1}{2}\|H - H^{prev}\|_F^2 \quad \text{s.t.} \quad (5.1) \text{ and } Hv = w. \tag{5.4}$$

Let us first write the quadratic f centered at x

$$f(y) \;=\; a + b^\top (y - x) + \frac{1}{2}(y - x)^\top C (y - x), \tag{5.5}$$

where $a = f(x)$, $b = \nabla f(x)$, and C is a symmetric matrix. The non-quadratic case will be analyzed after the theorem.

Theorem 5.2.1. *Let f be given by (5.5) and assume that the system of linear equations defined by (5.1) and $Hv = w$ is feasible and underdetermined in H. Let H^* be the optimal solution of problem (5.4). Then*

$$\|H^* - C\|_F^2 \;\leq\; \|H^{prev} - C\|_F^2.$$

Proof. The proof follows the arguments in [56] that lead to [56, Equation (1.8)]. From (5.1), we have $(y^\ell - x)^\top (C - H^*)(y^\ell - x) = 0, \;\; \ell = 1, \ldots, p$. We also have $(C - H^*)v = 0$. Hence, $C - H^*$ is a feasible direction for the affine space in H defined by (5.1) and $Hv = w$. It then turns out that the function

$$m(\theta) \;=\; \frac{1}{2}\|(H^* - H^{prev}) + \theta(C - H^*)\|_F^2$$

has a minimum at $\theta = 0$. From the trace definition of the Frobenius norm

$$m'(\theta) \;=\; [(H^* - H^{prev}) + \theta(C - H^*)]^\top (C - H^*).$$

Hence,

$$(H^* - H^{prev})^\top (C - H^*) \;=\; 0,$$

which then implies (given the symmetry of the matrices and considering only the diagonal entries of the above matrix product)

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(H_{ij}^* - H_{ij}^{prev})(C_{ij} - H_{ij}^*) \;=\; 0.$$

The rest of the proof requires the following calculations:

$$\begin{aligned}
&\|H^{prev} - C\|_F^2 - \|H^* - H^{prev}\|_F^2 - \|H^* - C\|_F^2 \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}[(H_{ij}^{prev} - C_{ij})^2 - (H_{ij}^* - H_{ij}^{prev})^2 - (H_{ij}^* - C_{ij})^2] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}[(H_{ij}^{prev} - C_{ij} + H_{ij}^* - H_{ij}^{prev})(H_{ij}^{prev} - C_{ij} - H_{ij}^* + H_{ij}^{prev}) - (H_{ij}^* - C_{ij})^2] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}[(H_{ij}^* - C_{ij})(2H_{ij}^{prev} - C_{ij} - H_{ij}^* - H_{ij}^* + C_{ij})] \\
&= 2\sum_{i=1}^{n}\sum_{j=1}^{n}[(H_{ij}^* - C_{ij})(H_{ij}^{prev} - H_{ij}^*)] \;=\; 0.
\end{aligned}$$

Hence we have established that

$$\begin{aligned}
\|H^* - C\|_F^2 &= \|H^{prev} - C\|_F^2 - \|H^* - H^{prev}\|_F^2 \\
&\leq \|H^{prev} - C\|_F^2.
\end{aligned}$$

□

When f is not quadratic, a similar result can be obtained under the price of more Hessian-vector products. We will obtain the result by considering the quadratic function that results from a

second-order Taylor expansion of f centered at x:

$$\tilde{f}(y) \;=\; f(x)+\nabla f(x)^\top(y-x)+\frac{1}{2}(y-x)^\top\nabla^2 f(x)(y-x). \tag{5.6}$$

The values of f and $\tilde{f}$ coincide at x up to second-order derivatives: $\tilde{f}(x)=f(x)$, $\nabla\tilde{f}(x)=\nabla f(x)$, and $\nabla^2\tilde{f}(x)=\nabla^2 f(x)$. That is not the case for the function values at y^ℓ, but if we are willing to pay the price of computing p more Hessian-vector products $\nabla^2 f(x)(y^\ell-x)$, $\ell=1,\ldots,p$, then one can indeed calculate $\tilde{f}(y^\ell)$ using (5.6). The new interpolating conditions for H are then given by

$$\tilde{f}(x)+\nabla\tilde{f}(x)^\top(y^\ell-x)+\frac{1}{2}(y^\ell-x)^\top H(y^\ell-x) \;=\; \tilde{f}(y^\ell), \quad \ell=1,\ldots,p. \tag{5.7}$$

Then, H can be calculated like in (5.4) but with (5.1) replaced by (5.7):

$$\min_H \quad \frac{1}{2}\|H-H^{prev}\|_F^2 \quad \text{s.t.} \quad (5.7) \text{ and } Hv=w. \tag{5.8}$$

Corollary 5.2.1. *Assume that the system of linear equations defined by (5.7) and $Hv=w$ is feasible and underdetermined in H. Let H^* be the optimal solution of problem (5.8). Then*

$$\|H^*-\nabla^2 f(x)\|_F^2 \;\le\; \|H^{prev}-\nabla^2 f(x)\|_F^2 \;=\; \|H^{prev}-\nabla^2\tilde{f}(x^{prev})\|_F^2. \tag{5.9}$$

Proof. By applying Theorem 5.2.1 when $\tilde{f}$ is the quadratic function considered, we obtain

$$\|H^*-\nabla^2\tilde{f}(x)\|_F^2 \;\le\; \|H^{prev}-\nabla^2\tilde{f}(x)\|_F^2.$$

The result of the proof follows by considering $\nabla^2\tilde{f}(x)=\nabla^2\tilde{f}(x^{prev})=\nabla^2 f(x)$. □

Further improving the bound of Corollary 5.2.1, in the sense of having $H^{prev}-\nabla^2 f(x^{prev})$ in the right-hand side of (5.9), seems out of reach because it would require incorporating $\nabla^2 f(x^{prev})$ in the objective function of the recovery subproblem (5.8).

Error bound

Let α represent the coefficients of H in $(1/2)w^\top Hw$ in terms of the monomial basis. The quadratic components of this basis are of the form $(1/2)w_i^2$, $i=1,\ldots,n$ and w_iw_j, $1\le i<j\le n$. So, we have $(1/2)h_{11}w_1^2=(1/2)\alpha_1w_1^2,\ldots,h_{1n}w_1w_n=\alpha_nw_1w_n$, $(1/2)h_{22}w_2^2=(1/2)\alpha_{n+1}w_2^2$ and so on. The recovery problem (5.4) can then be formulated approximately[1] as

$$\min_\alpha \quad \frac{1}{2}\|\alpha-\alpha^{prev}\|^2 \quad \text{s.t.} \quad M\alpha \;=\; \delta, \tag{5.10}$$

where

$$M=\begin{bmatrix} M^1 \\ M^2 \end{bmatrix}, \quad \delta=\begin{bmatrix} \delta^1 \\ \delta^2 \end{bmatrix},$$

[1] The norm used in (5.10) for α is a minor variation of the Frobenius norm of H.

$$M^1\alpha = \begin{bmatrix} \frac{1}{2}(y^1-x)^\top H(y^1-x) \\ \vdots \\ \frac{1}{2}(y^p-x)^\top H(y^p-x) \end{bmatrix}, \quad M^2\alpha = Hv,$$

$$\delta^1 = \begin{bmatrix} f(y^1)-f(x)-\nabla f(x)^\top(y^1-x) \\ \vdots \\ f(y^p)-f(x)-\nabla f(x)^\top(y^p-x) \end{bmatrix}, \quad \delta^2 = w.$$

Another piece of motivation for this approach comes from the fact that the enriched interpolating conditions defined by (5.1) and $Hv = w$, once determined (i.e., with as many equations as variables), may produce a model Hessian H that used together with $\nabla f(x)$ can give rise to a fully quadratic model. Such a model has the same orders of accuracy as a Taylor-based model [17] (see also [18]).

Theorem 5.2.2. *If p is chosen such that $p+n = \frac{n^2+n}{2}$ and if M is nonsingular, then the model Hessian H resulting from $M\alpha = \delta$ in (5.10) can give rise to a fully quadratic model, in other words, one has*

$$\|H - \nabla^2 f(x)\| \;=\; O(\Delta_y),$$

where $\Delta_y = \max_{1\le\ell\le p} \left\|y^\ell - x\right\|$ and the constant multiplying Δ_y depends on the inverse of an appropriate scaled version of M.

Proof. Consider that x is at the origin, without any lost of generality. One can start by making a Taylor expansion of f around x along all the displacements $y^\ell - x$, $\ell = 1,\ldots,p$, leading to

$$\delta^1 - M^1\alpha^x \;=\; O(\Delta_y^3), \tag{5.11}$$

where α^x stores the components of $\nabla^2 f(x)$ and each component of the right-hand side is bounded by $(1/6)L_{\nabla^2 f}\|y^\ell - x\|^3$, with $L_{\nabla^2 f}$ the Lipschitz constant of $\nabla^2 f$. From (5.11) and $M^1\alpha = \delta^1$, we obtain

$$M^1(\alpha - \alpha^x) \;=\; O(\Delta_y^3). \tag{5.12}$$

On the other hand, one also has

$$M^2(\alpha - \alpha^x) \;=\; 0.$$

Now we divide each row of (5.12) by Δ_y^2. The proof is concluded by considering $[M^1/\Delta_y^2; M^2]$ as the scaled version of M alluded in the statement of the result. □

5.3 Numerical results for the determined case

As we have discussed in Theorem 5.2.2, if p is chosen such that $p+n = \frac{n^2+n}{2}$ and if the matrix M is nonsingular and well conditioned, the model Hessian H resulting from $M\alpha = \delta$ in (5.10) becomes fully quadratic. The error between the Hessian model H and $\nabla^2 f(x)$ is then of the $O(\Delta_y)$, where $\Delta_y = \max_{1\le\ell\le p}\|y^\ell - x\|$.

In this section we will report some illustrative numerical results to confirm that an approach built on such a Hessian model can lead to an economy of Hessian-vector products. Our term of

comparison will be the inexact Newton method (as described in [52, Section 7.1]), where the system $\nabla^2 f(x)d^{IN} = -\nabla f(x)$ is solved by applying a truncated linear conjugate gradient (CG) method (stopping once a direction of negative curvature is found or a relative error criterion is met). In our case, after computing H from solving $M\alpha = \delta$ in (5.10), to compute our search direction d^{MH}, we apply the exact same truncated CG method to $Hd^{MH} = -\nabla f(x)$ as in the inexact Newton method. The computed directions d^{IN} or d^{MH} are necessarily descent in the sense of making an acute angle with $-\nabla f(x)$.

For both the inexact Newton method and our model Hessian approach, a new iterate is of the form $x+\alpha d$, where d is given by d^{IN} or d^{MH} respectively. The same cubic interpolation line search [62, Section 2.4.2] is used to compute the stepsize α^{IN} and α^{MH}. In this line search, the objective function is approximated by a cubic polynomial with function values at three points and a derivative value at one point (see Section 2.2.2 for further details). The line search starts with a unit stepsize and terminates either successfully with a value α satisfying a sufficient decrease condition for the function (of the form $f(x+\alpha d) \leq f(x) + c_1\alpha\nabla f(x)^\top d$, with $c_1 = 10^{-4}$) or unsuccessfully with a stepsize smaller than 10^{-10}.

To form the model described in (5.2) one needs p interpolation points $y^1, \ldots, y^p$ and one vector v for Hessian multiplication. We have used the following scheme: before the initial iteration, we have randomly generated a set of p points, $\{y^1, \ldots, y^p\}$, and a vector v, in the unit ball $B(0;1)$ centered at the origin. Then, at each iteration x_k, the interpolation points $Y_k = \{y_k^1, \ldots, y_k^p\}$ used were of the form $x_k + r_k y^\ell$, $\ell = 1, \ldots, p$, and the vector v_k of the form $r_k v$, where $r_k = \min\{10^{-2}, \max\{10^{-4}, \|x_k - x_{k-1}\|\}\}$, $k = 1, 2, \ldots$.

For the purpose of this numerical illustration, we selected 48 unconstrained (smooth and nonlinear) very small problems from the CUTEst collection (see Appendix B.1), also used in the papers [34, 37]. Both methods were stopped when an iterate x_k was found such that $\|\nabla f(x_k)\| < 10^{-5}$. We built performance profiles (see Appendix A.1) using as performance metric the numbers of Hessian-vector products and iterations (Figure 5.1) and the number of function evaluations and CPU time (Figure 5.2). One can see that our approach can effectively lead to a significant reduction on the number of Hessian-vector products. Both approaches take on average 2CG inner iterations to compute a direction, and the number of main iterations is comparable. Hence, we estimate that this reduction is approximately 50% as we only do one Hessian-vector product per main iteration. Of course, one has to pay a significant cost in the number of function evaluations, which is of the order of n^2 per main iteration.

Note that the interpolation points around the current iterate x are of the form $x + r_k y^\ell$, where y^ℓ, $\ell = 1, \ldots, p$, are randomly generated in the unit ball $B(0;1)$ at the very beginning. The idea is to consider an equivalent scaled version of the system in (5.10). Recalling the definition of δ in (5.10)

and given the matrix M,

$$M = \begin{bmatrix} M^1 \\ M^2 \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1}{2}(y_k^1 - x_k)_1^2 & (y_k^1 - x_k)_1(y_k^1 - x_k)_2 & \cdots & (y_k^1 - x_k)_1(y_k^1 - x_k)_n & \frac{1}{2}(y_k^1 - x_k)_2^2 & \cdots & \frac{1}{2}(y_k^1 - x_k)_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \frac{1}{2}(y_k^p - x_k)_1^2 & (y_k^p - x_k)_1(y_k^p - x_k)_2 & \cdots & (y_k^p - x_k)_1(y_k^p - x_k)_n & \frac{1}{2}(y_k^p - x_k)_2^2 & \cdots & \frac{1}{2}(y_k^p - x_k)_n^2 \\ \hline (v^k)_1 & (v^k)_2 & \cdots & (v^k)_n & 0 & \cdots & 0 \\ 0 & (v^k)_1 & \cdots & 0 & (v^k)_2 & \cdots & (v^k)_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & (v^k)_1 & 0 & \cdots & (v^k)_n \end{bmatrix}, \tag{5.13}$$

consider the scaled version of M and δ, namely,

$$\tilde{M} = \begin{bmatrix} \tilde{M}^1 \\ \tilde{M}^2 \end{bmatrix} = \begin{bmatrix} M^1/(r_k)^2 \\ M^2/r_k \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1}{2}(y^1)_1^2 & (y^1)_1(y^1)_2 & \cdots & (y^1)_1(y^1)_n & \frac{1}{2}(y^1)_2^2 & \cdots & \frac{1}{2}(y^1)_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \frac{1}{2}(y^p)_1^2 & (y^p)_1(y^p)_2 & \cdots & (y^p)_1(y^p)_n & \frac{1}{2}(y^p)_2^2 & \cdots & \frac{1}{2}(y^p)_n^2 \\ \hline (v)_1 & (v)_2 & \cdots & (v)_n & 0 & \cdots & 0 \\ 0 & (v)_1 & \cdots & 0 & (v)_2 & \cdots & (v)_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & (v)_1 & 0 & \cdots & (v)_n \end{bmatrix}, \tag{5.14}$$

$$\tilde{\delta} = \begin{bmatrix} \tilde{\delta}^1 \\ \tilde{\delta}^2 \end{bmatrix} = \begin{bmatrix} \delta^1/(r_k)^2 \\ \delta^2/r_k \end{bmatrix}. \tag{5.15}$$

Note that the scaled matrix $\tilde{M}$ is independent of x_k and r_k. Solving problem (5.10) is then equivalent to solving the following problem

$$\min_{\alpha} \quad \frac{1}{2}\|\alpha - \alpha^{prev}\|^2 \quad \text{s.t.} \quad \tilde{M}\alpha = \tilde{\delta}. \tag{5.16}$$

It is important to point out that once the p interpolation points $y^1, \cdots, y^p$ are generated, the matrix $\tilde{M}$ can be determined and it is not required to be updated at each iteration.

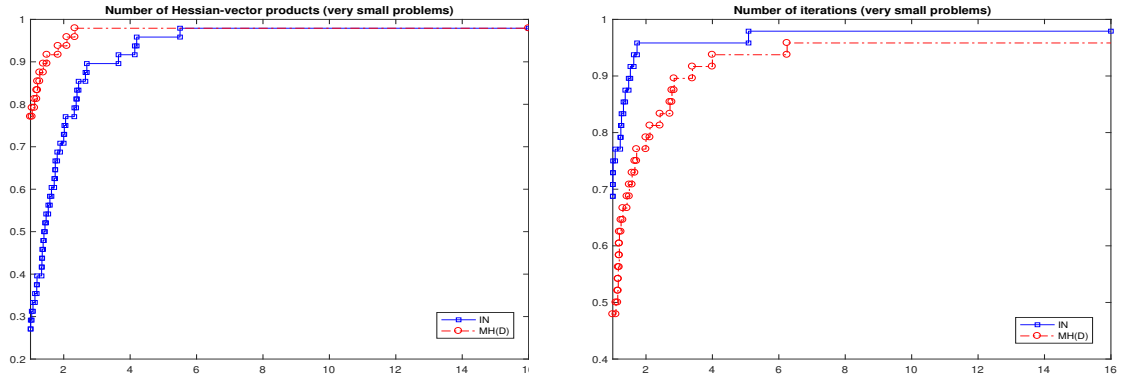

Fig. 5.1 Testing the Hessian recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of very small problems of Appendix B.1. The value of p was set to $\frac{n^2+n}{2} - n$.

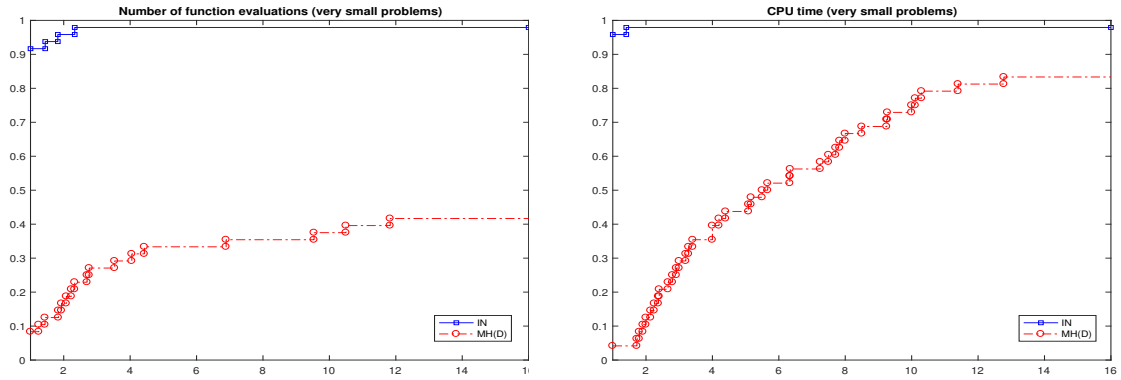

Fig. 5.2 Testing the Hessian recovery within a line-search algorithm. Performance profiles for number of function evaluations and CPU time, for the set of very small problems of Appendix B.1. The value of p was set to $\frac{n^2+n}{2} - n$.

5.4 Numerical results for the determined case when the Hessian sparsity is known

In many optimization problems, the Hessian matrix of the objective function is sparse and the corresponding sparsity pattern is known in advance. This is the case for problems governed by partial differential equations [1, 6, 41, 43], and, in general, for all problems involving partially separable functions of the form $f(x) = \sum_{i=1}^{m} f_i(x)$, where each of the element functions f_i depends on only a few components of x (see [13, 64]). The CUTEst [36] collection lists many constrained and unconstrained sparse problems, and tools for accessing the sparsity pattern of the Hessian of the objective functions are made available. CUTEst problems for which the sparsity pattern of the Hessian of the objective function is accessible arise from interconnected markets, power network, circuit simulation, computational fluid dynamics, chemical process simulation, among many other

applications. Taking advantage of the Hessian sparsity pattern to approximate Hessian values has thus been the subject of research [29, 30, 58, 63].

Let $\Omega(\nabla^2 f) = \{(i,j) : i \leq j, \nabla^2 f_{ij}(x) = 0 \text{ for all } x\}$ be the sparsity pattern of $\nabla^2 f$. When $|\Omega(\nabla^2 f)| \ll n(n+1)/2$, it is then beneficial and often necessary to use specialized algorithms and data structures that take advantage of the known sparsity pattern. One can tailor our model Hessian approach to problems with sparse Hessian matrices when the sparsity patterns are known. We require the Hessian model to share the same sparsity pattern of the true Hessian, recovering only the nonzero elements. In fact, instead of solving problem (5.10) with respect to the whole Hessian matrix, we solve problem

$$\min_{\alpha_\Omega} \quad \frac{1}{2}\|\alpha_\Omega - \alpha_\Omega^{prev}\|^2 \quad \text{s.t.} \quad M_\Omega \alpha_\Omega = \delta, \tag{5.17}$$

where the elements in the rows of M_Ω and in the vector α_Ω correspond now only to nonzero entries.

We have tested our sparse Hessian recovery approach using the same algorithmic environment of Section 5.3, the only difference being in the usage of the model equation $M_\Omega \alpha_\Omega = \delta$ in (5.17) and a smaller value of p (now given by the difference between the number of nonzeros of the Hessian and n, so that the matrix M_Ω is squared). The sparse problems used are listed in Appendix B.2. The experiments are reported in Figures 5.3 and 5.4 in the form of performance profiles. The conclusions are similar to those in Section 5.3.

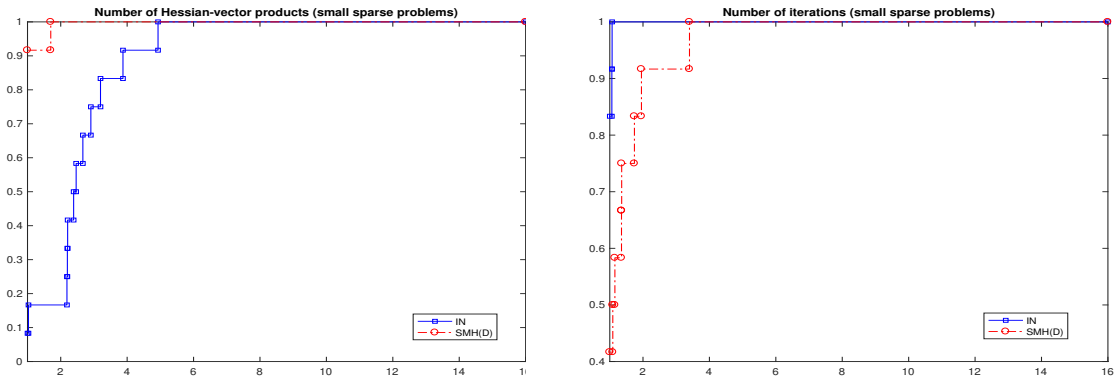

Fig. 5.3 Testing the Hessian recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of small sparse problems of Appendix B.2. The value of p was set to number of nonzeros minus n.

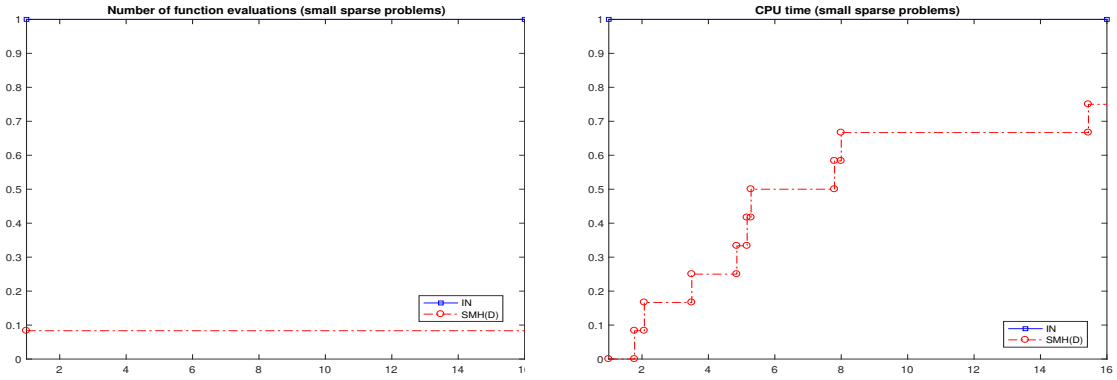

Fig. 5.4 Testing the Hessian recovery within a line-search algorithm. Performance profiles for number of function evaluations and CPU time, for the set of small sparse problems of Appendix B.2. The value of p was set to number of nonzeros minus n.

5.5 Recovery cost in the general case

The necessary and sufficient optimality conditions for the convex QP (5.10) can be stated as

$$\begin{aligned} \alpha - \alpha^{prev} - M^\top \lambda = 0, \\ M\alpha = \delta, \end{aligned} \tag{5.18}$$

where λ denotes the vector of Lagrange multipliers. Such multipliers can then be recovered by solving

$$MM^\top \lambda \;=\; \delta - M\alpha^{prev}. \tag{5.19}$$

The system (5.19) can either be solved directly or iteratively. If solved directly the cost is of the order of $(p+n)^2 n^2$ to form $MM^\top$ and of $(p+n)^3$ to factorize it, and the overall storage of the order of $(p+n)^2$. If the conjugate gradient (CG) method is applied, the overall cost is of the order of $c_g(p+n)n^2$, where c_g is the number of CG iterations. In fact, each matrix vector multiplication with either $M^\top$ or M costs $O((p+n)n^2)$. Solving the KKT system (5.18) using an indefinite factorization is even less viable given that the storage space would be of the order of $(n^2+p)^2$.

Chapter 6

Newton direction recovery from Hessian-vector products

In this chapter, we introduce a new approach to recover the Newton direction from Hessian-vector products without requiring an explicit recovery of the Hessian matrix.

6.1 Newton direction recovery

Let us first consider a quadratic Taylor expansion of the form

$$f(x)+\nabla f(x)^\top(y^\ell-x)+\frac{1}{2}(y^\ell-x)^\top\nabla^2 f(x)(y^\ell-x) \;\simeq\; f(y^\ell), \quad \ell=1,\ldots,p, \tag{6.1}$$

made using a sample set $\{y^1,\ldots,y^p\}$. We will synchronize expansion (6.1) with Hessian-vector products along $y^\ell-x$, $\ell=1,\ldots,p$. In fact, we require the calculation of

$$z^\ell \;=\; \nabla^2 f(x)(y^\ell-x), \quad \ell=1,\ldots,p. \tag{6.2}$$

Since our interest relies specifically on the calculation of the Newton direction, assuming that the model Hessian $\nabla^2 f(x)$ is nonsingular, we obtain from (6.1) and (6.2)

$$f(x)+(\nabla^2 f(x)^{-1}\nabla f(x))^\top\nabla^2 f(x)(y^\ell-x)+\frac{1}{2}(y^\ell-x)^\top z^\ell \;\simeq\; f(y^\ell), \quad \ell=1,\ldots,p. \tag{6.3}$$

Then, introducing the model vector $d\simeq-\nabla^2 f(x)^{-1}\nabla f(x)$, one arrives at a new set of enriched interpolating conditions

$$(z^\ell)^\top d \;=\; -f(y^\ell)+f(x)+\frac{1}{2}(y^\ell-x)^\top z^\ell, \quad \ell=1,\ldots,p. \tag{6.4}$$

Equations (6.4) lead then to a new recovery problem

$$\min_d \quad \text{norm}(d-d^{prev}) \quad \text{s.t.} \quad (6.4). \tag{6.5}$$

When d^{prev} is the previously recovered Newton direction, we are following the spirit of a quasi-Newton least secant approach. One could also consider the case $d^{prev} = 0$ as it was done in some derivative-free approaches for Hessian recovery. Let us now give two arguments to motivate this approach.

6.2 Theoretical motivation

Error decrease

First, as we did in Section 5.2 for the Hessian recovery approach, we can provide motivation for the Newton direction recovery approach when f is assumed quadratic (5.5), this time with a nonsingular Hessian C. Here we need to consider the square of the ℓ_2-norm in (6.5)

$$\min_d \quad \frac{1}{2}\|d - d^{prev}\|^2 \quad \text{s.t.} \quad (6.4). \tag{6.6}$$

We will show that in the quadratic case the error in the approximation of the Newton direction is monotonically non increasing. The non-quadratic case will be discussed at the end of this section.

Let us consider the following quadratic f centered at x

$$f(y) \;=\; a + b^\top (y-x) + \frac{1}{2}(y-x)^\top C(y-x), \tag{6.7}$$

where $a = f(x)$, $b = \nabla f(x)$, and C is a symmetric matrix.

Theorem 6.2.1. *Let f be given by (6.7) with C nonsingular and assume that the system of linear equations (6.4) is feasible and underdetermined in d. Let d^* be the optimal solution of problem (6.6). Then*

$$\|d^* - (-C^{-1}b)\|^2 \;\leq\; \|d^{prev} - (-C^{-1}b)\|^2. \tag{6.8}$$

Proof. From the expression of f, one has

$$f(y^\ell) \;=\; a + (C^{-1}b)^\top C(y^\ell - x) + \frac{1}{2}(y^\ell - x)^\top C(y^\ell - x), \quad \ell = 1, \ldots, p,$$

and hence, using $z^\ell = C(y^\ell - x)$, $\ell = 1, \ldots, p$, and (6.4), one arrives at $(z^\ell)^\top (d^* - (-C^{-1}b)) = 0$. The conclusion is that $d^* - (-C^{-1}b)$ is a feasible direction for the affine space in d defined by (6.4).

The rest of the proof follows the same lines as in the proof of Theorem 5.2.1. The function

$$m(\theta) \;=\; \frac{1}{2}\|(d^* - d^{prev}) + \theta(-C^{-1}b - d^*)\|^2$$

has a minimum at $\theta = 0$, from which we conclude that $(d^* - d^{prev})^\top (-C^{-1}b - d^*) = 0$. From here we obtain

$$\begin{aligned} \|d^* - (-C^{-1}b)\|^2 &= \|d^{prev} - (-C^{-1}b)\|^2 - \|d^* - d^{prev}\|^2 \\ &\leq \|d^{prev} - (-C^{-1}b)\|^2. \end{aligned}$$

□

This result does not measure, however, the decrease in the absolute error occurred in the current approximate Newton direction because the gradient of f at x^{prev} is not b, but rather $b+C(x^{prev}-x)$. Hence, what we would like to have in the right-hand side of the bound (6.8) of Theorem 6.2.1 is

$$d^{prev}-(-C^{-1}\nabla f(x^{prev})) \;=\; d^{prev}-(-C^{-1}(b+C(x^{prev}-x))).$$

To achieve such a result we need to change (6.6) to

$$\min_{d} \quad \frac{1}{2}\|d-(d^{prev}+x^{prev}-x)\|^2 \quad \text{s.t.} \quad (6.4), \tag{6.9}$$

and the next corollary states it rigorously.

Corollary 6.2.1. *Let f be given by (6.7) with C nonsingular and assume that the system of linear equations (6.4) is feasible and underdetermined in d. Let d^* be the optimal solution of problem (6.9). Then*

$$\|d^*-(-C^{-1}\nabla f(x))\|^2 \;\leq\; \|d^{prev}-(-C^{-1}\nabla f(x^{prev}))\|^2. \tag{6.10}$$

Proof. All we need to do is to apply Theorem 6.2.1 to problem (6.9) instead, which leads to

$$\|d^*-(-C^{-1}b)\|^2 \;\leq\; \|(d^{prev}+x^{prev}-x)-(-C^{-1}b)\|^2. \tag{6.11}$$

Finally, notice that $d^{prev}+x^{prev}-x-(-C^{-1}b) = d^{prev}-(-C^{-1}(b+C(x^{prev}-x)))$. □

As we did in Corollary 5.2.1 for the Hessian recovery approach of Chapter 5, we can also shed some light on what happens when f is non-quadratic. Consider the quadratic function in (5.6) that results from the second-order Taylor expansion of f centered at x. Again, the values of f and $\tilde{f}$ coincide at x up to second-order derivatives ($\tilde{f}(x)=f(x)$, $\nabla\tilde{f}(x)=\nabla f(x)$, and $\nabla^2\tilde{f}(x)=\nabla^2 f(x)$), but that is not the case for the function values at y^ℓ. However, we can calculate $\tilde{f}(y^\ell)$ using (5.6) and the Hessian-vector products (6.2). The new set of enriched interpolating conditions is then given by

$$(z^\ell)^\top d \;=\; -\tilde{f}(y^\ell)+\tilde{f}(x)+\frac{1}{2}(y^\ell-x)^\top z^\ell, \quad \ell=1,\ldots,p, \tag{6.12}$$

and a new recovery problem is formulated as

$$\min_{d} \quad \frac{1}{2}\|d-(d^{prev}+x^{prev}-x)\|^2 \quad \text{s.t.} \quad (6.12). \tag{6.13}$$

Corollary 6.2.2. *Assume that $\nabla^2 f(x)$ is non-singular and the system of linear equations (6.12) is feasible and underdetermined in d. Let d^* be the optimal solution of problem (6.13). Then*

$$\|d^*-(-\nabla^2 f(x)^{-1}\nabla f(x))\|^2 \;\leq\; \|d^{prev}-(-\nabla^2\tilde{f}(x^{prev})^{-1}\nabla\tilde{f}(x^{prev}))\|^2. \tag{6.14}$$

Proof. The proof is a combination of the proofs of Corollaries 5.2.1 and 6.2.1. □

Note that $\nabla^2 \tilde{f}(x^{prev})$ is equal to $\nabla^2 f(x)$, not to $\nabla^2 f(x^{prev})$. Note also that $\nabla \tilde{f}(x^{prev})$ is not the same as $\nabla f(x^{prev})$. Having the Hessian and the gradient of f at x^{prev} in the right-hand side of the bound (6.14) would require the knowledge of the true Hessian or true Newton direction at x^{prev}.

Error bound

The second argument establishes the accuracy of the recovery under the assumption that $p \geq n$ (see the end of this section for a discussion about this assumption and how to circumvent it practice). We will establish a bound on the norm of the absolute error of the recovered Newton direction d^N based on $\Delta_y = \max_{1\leq \ell \leq p} \|y^\ell - x\|$, $\Delta_z = \max_{1\leq \ell \leq p} \|z^\ell\|$, and the conditioning of the matrix M_L^z, whose rows are $(1/\Delta_z)(z^\ell)^\top$, $\ell = 1, \ldots, p$, in other words,

$$M_L^z \;=\; \frac{1}{\Delta_z} \begin{bmatrix} (z^1)^\top \\ \vdots \\ (z^p)^\top \end{bmatrix}.$$

Theorem 6.2.2. *Suppose that $p \geq n$, the matrix M_L^z is full column rank, and $\nabla^2 f(x)$ is invertible. Then, if d^N satisfies (6.4), in a least squares sense when $p > n$, one has*

$$\left\| -\nabla^2 f(x)^{-1} \nabla f(x) - d^N \right\| \;\leq\; \Lambda_z O\left(\frac{\Delta_y^3}{\Delta_z} \right),$$

where Λ_z is a bound on the norm of the left inverse of M_L^z and the multiplicative constant in O depends on the Lipschitz constant of $\nabla^2 f$ at x.

Proof. Expanding f at y^ℓ around x in (6.4) yields

$$\begin{aligned} (z^\ell)^\top d^N &= -\nabla f(x)^\top (y^\ell - x) + O(\Delta_y^3) \\ &= (z^\ell)^\top (-\nabla^2 f(x)^{-1} \nabla f(x)) + O(\Delta_y^3), \quad \ell = 1, \ldots, p, \end{aligned}$$

where the constant in $O(\Delta_y^3)$ depends on the Lipschitz constant of $\nabla^2 f$ at x. Multiplying both terms by the left inverse of M_L^z,

$$\Delta_z \left(-\nabla^2 f(x)^{-1} \nabla f(x) - d^N \right) \;=\; -(M_L^z)^\dagger O(\Delta_y^3).$$

Hence, the result follows by dividing both terms by Δ_z and then taking norms. □

One can derive an estimate solely dependent on Δ_y and on the conditioning of the matrix M_L^y formed by the rows $(1/\Delta_y)(y^\ell - x)^\top$, $\ell = 1, \ldots, p$,

$$M_L^y \;=\; \frac{1}{\Delta_y} \begin{bmatrix} (y^1 - x)^\top \\ \vdots \\ (y^p - x)^\top \end{bmatrix}.$$

In fact, from

$$\Delta_y M_L^y \nabla^2 f(x) \;=\; \Delta_z M_L^z,$$

one has

$$\left\| (M_L^z)^\dagger \right\| \;=\; \frac{\Delta_z}{\Delta_y} \|R_y\|,$$

with

$$R_y \;=\; \left(\nabla^2 f(x) (M_L^y)^\top (M_L^y) \nabla^2 f(x) \right)^{-1} \nabla^2 f(x) (M_L^y)^\top. \tag{6.15}$$

Corollary 6.2.3. *Suppose that $p \geq n$, the matrix M_L^z is full column rank, and $\nabla^2 f(x)$ is invertible. Then, if d^N satisfies (6.4), one has*

$$\left\| -\nabla^2 f(x)^{-1} \nabla f(x) - d^N \right\| \;\leq\; \|R_y\| O(\Delta_y^2),$$

where the multiplicative constant in O depends on the Lipschitz constant of $\nabla^2 f$ at x.

Hence by controlling the geometry of the points y^ℓ, $\ell = 1, \ldots, p$, around x one can provide an accurate bound when the Hessian of f is invertible and $p \geq n$. In general, we can attempt to control the conditioning of M_L^z, replacing some of the points y^ℓ if necessary. Such a conditioning must eventually become adequate if the vectors $y^\ell - x$ are sufficiently linearly independent and lie in eigenspaces of $\nabla^2 f(x)$ corresponding to eigenvalues not too close to zero. (See Chapter 7 for a modified Newton direction recovery approach where the curvature values $(z^\ell)^\top (y^\ell - x)$, $\ell = 1, \ldots, p$, are taken into consideration.)

Using $p = n$ Hessian-vector products at each iteration is certainly not a desirable strategy as that would be equivalent to access the entire Hessian matrix. It is however possible to use $p \ll n$ and still obtain an accurate Newton direction model. The possibility we have in mind is to build upon a previously computed Newton direction model calculated using $p = n$. Let x_{prev} be such an iterate, $y^1_{prev}, \ldots, y^n_{prev}$ be the corresponding sample points and $z^1_{prev}, \ldots, z^n_{prev}$ be the corresponding Hessian-vector products. Suppose we are now at a new iterate x and we would like to reuse $f(y^1_{prev}), \ldots, f(y^n_{prev})$ and $z^1_{prev} = \nabla^2 f(x_{prev})(y^1_{prev} - x_{prev}), \ldots, z^n_{prev} = \nabla^2 f(x_{prev})(y^n_{prev} - x_{prev})$. In such a case what we will have in (6.4) is

$$z^\ell_{prev} \;=\; \nabla^2 f(x_{prev})(y^\ell_{prev} - x_{prev}) \;\simeq\; \nabla f(y^\ell_{prev}) - \nabla f(x_{prev}), \quad \ell = 1, \ldots, p,$$

but what we wish we would have is

$$z^\ell \;=\; \nabla^2 f(x)(y^\ell_{prev} - x) \;\simeq\; \nabla f(y^\ell_{prev}) - \nabla f(x), \quad \ell = 1, \ldots, p,$$

So, one can obtain an approximation to z^ℓ from

$$z^\ell_{prev} + \nabla f(x_{prev}) - \nabla f(x), \quad \ell = 1, \ldots, p. \tag{6.16}$$

The error in such an approximation is of $O(\max\{\|y^\ell_{prev} - x_{prev}\|^2, \|y^\ell_{prev} - x\|^2\})$, which would then has to be divided by Δ_z in the context of Theorem 6.2.2. Of course, if we then keep applying this strategy the error will accumulate over the iterations, but there are certainly remedies such as bringing

a few new, fresh z's at each iteration and applying restarts with $p = n$ whenever the conditioning of M_L^z becomes large.

6.3 Numerical results for the determined case using a correction

To use as few Hessian-vector products as possible, we start by using $p = n$ products at iteration zero, to then replace only one interpolation point at each iteration. We choose to replace the point farthest away from the current iterate x. (A perhaps more sound approach would have been to choose the z^ℓ that has contributed the most to the conditioning of M_L^z.) A new point is then added, generated in the ball $B(x; r)$, where $r = \min\{10^{-2}, \max\{10^{-4}, \|x - x_{prev}\|\}\}$. Therefore, only one more Hessian-vector product and one more function evaluation is required at each iteration. We then replace all other z_{prev}^ℓ's by (6.16). We monitor the condition number of M_L^z, and apply a restart (with $p = n$ as in iteration 0) whenever $\text{cond}(M_L^z) \geq 10^8$.

A Newton direction model d^N is then calculated by solving (6.4) directly. To guarantee that we have a descent direction d, meaning that $-\nabla f(x)^\top d > 0$, we modify the d^N from (6.4) so that $d = d^N - \beta \nabla f(x)$ where β is such that $\cos(d, -\nabla f(x)) = \eta$, and η was set to 0.95.

The modified Newton direction model was then used in a line search algorithm using the same cubic line search procedure of Section 5.3. The comparison is again against the inexact Newton method (as described in [52, Section 7.1]). First we tested the very small problems of Appendix B.1. Again, we plot performance profiles (see Appendix A.1) using as performance metric the numbers of Hessian-vector products and iterations (Figure 6.1) and the number of function evaluations and CPU time (Figure 6.2). The results are quite encouraging. We then selected a benchmark of 26 unconstrained nonlinear small problems from the CUTEst collection [36], listed in Appendix B.3. The experiments are reported in Figures 6.3 and 6.4 in the form of the same performance profiles. The results are similar and again promising.

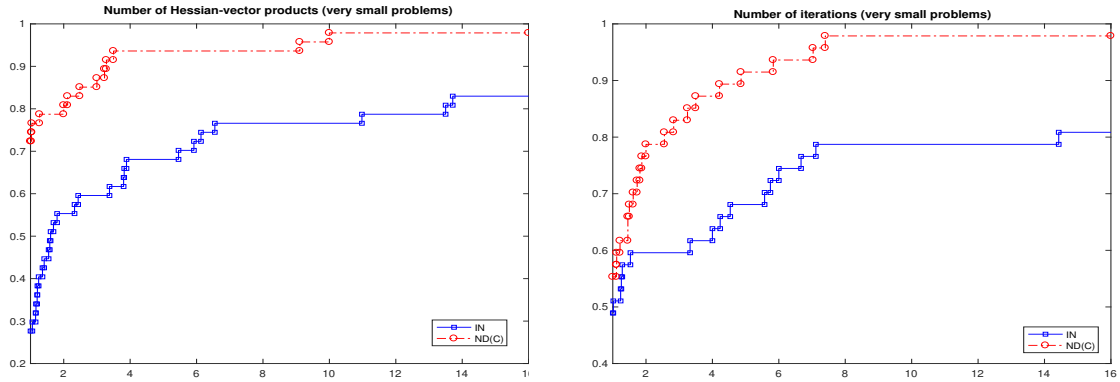

Fig. 6.1 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of very small problems of Appendix B.1.

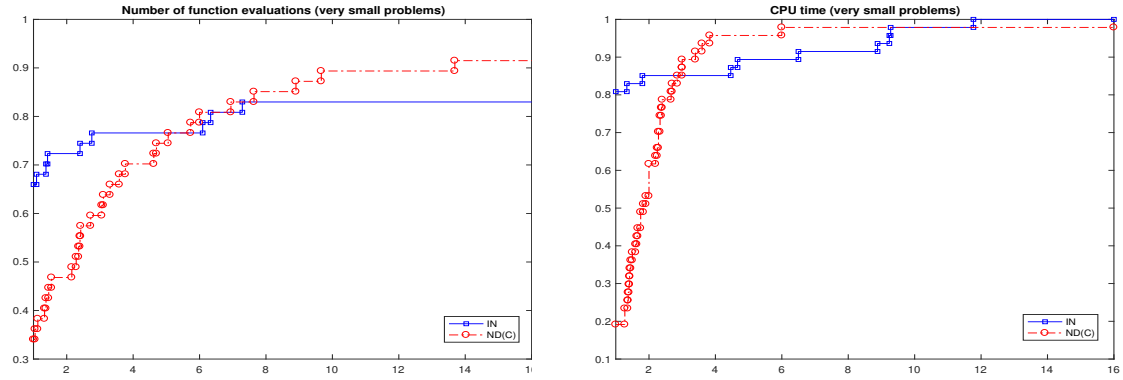

Fig. 6.2 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of function evaluations and CPU time, for the set of very small problems of Appendix B.1.

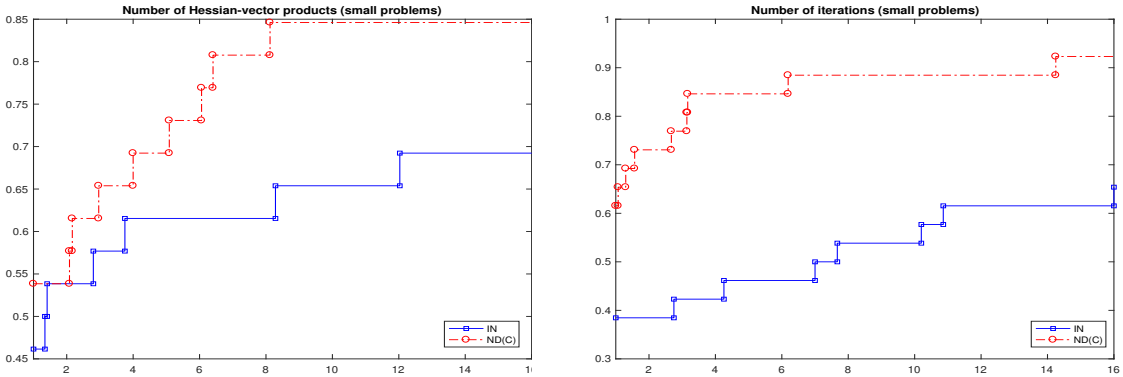

Fig. 6.3 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of Hessian-vector products and iterations, for the set of small problems of Appendix B.3.

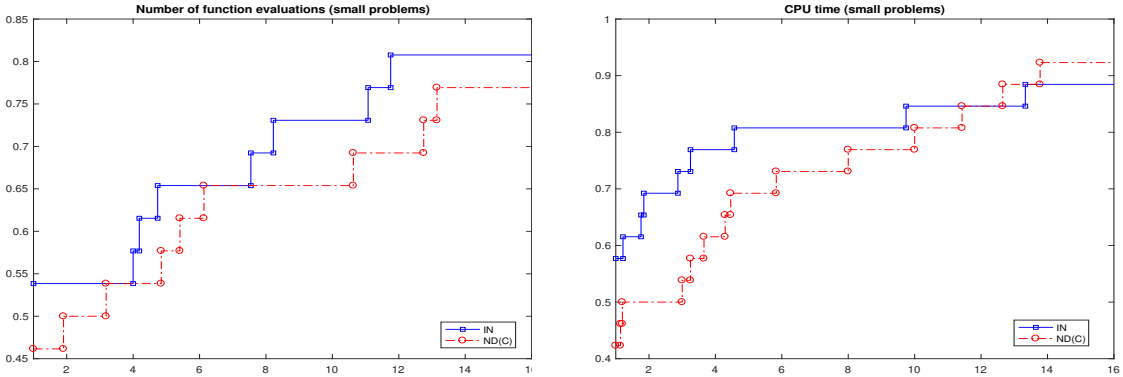

Fig. 6.4 Testing the Newton direction recovery within a line-search algorithm. Performance profiles for the numbers of function evaluations and CPU time, for the set of small problems of Appendix B.3.

Chapter 7

Exploring other ideas

In this chapter, we explore some ideas that improve or extend the methodology described in Chapters 5 and 6, such as concurrent determination of both Newton direction and Hessian inverse, use of an iterative solver to compute the Newton direction model, and recovery of a modified Newton direction.

7.1 Inverse Hessian recovery from Hessian-vector products

In this section, we describe an approach to implicitly recover the Newton direction by determining an approximation to the inverse of the Hessian. Theoretical support for this recovery is developed by proving that the absolute error is decreasing, in case f is quadratic, and by showing that, in general, the error in the enriched interpolating conditions can be expressed as a function of the distance between the Hessian inverse of the model and the previous Hessian inverse approximation. The cost of the Newton direction recovery is analyzed in terms of floating point operations and storage space.

Suppose to be in the same context considered in Chapters 5 and 6. Therefore, let x be a given point and assume that we have calculated f and ∇f at x as well as f at a number of points $y^1, \ldots, y^p$. Consider the quadratic interpolation model at the current point x, which is given by

$$f(x) + \nabla f(x)^\top (y^\ell - x) + \frac{1}{2}(y^\ell - x)^\top H(y^\ell - x) \;=\; f(y^\ell), \quad \ell = 1, \ldots, p. \tag{7.1}$$

Moreover, suppose that we have calculated the Hessian-vector products

$$z^\ell \;=\; \nabla^2 f(x)(y^\ell - x), \quad l = 1, \ldots, p. \tag{7.2}$$

Our goal is to determine the symmetric Hessian model H that satisfies the system of enriched interpolating conditions given by (7.1) and

$$z^\ell \;=\; H(y^\ell - x), \quad \ell = 1, \ldots, p. \tag{7.3}$$

Since our interest relies specifically on the calculation of the Newton direction, one can focus on the action of the inverse of the Hessian on the negative gradient. For the moment, suppose that one can afford to recover the whole inverse of the Hessian. Assuming that the model Hessian H is

invertible and setting $G = H^{-1}$, (7.1) can be written as

$$f(x) + (G\nabla f(x))^\top H(y^\ell - x) + \frac{1}{2}(H(y^\ell - x))^\top G(H(y^\ell - x)) \;=\; f(y^\ell), \quad \ell = 1, \ldots, p. \tag{7.4}$$

Then, using (7.3), one arrives at a new set of interpolating conditions in G

$$f(x) + (G\nabla f(x))^\top z^\ell + \frac{1}{2}(z^\ell)^\top G z^\ell \;=\; f(y^\ell), \quad \ell = 1, \ldots, p. \tag{7.5}$$

Note that the determination of G allows recovering the Newton direction $d^N = -G\nabla f(x)$.

Conditions (7.5) lead then to a new recovery problem

$$\min_G \quad \text{norm}(G - G^{prev}) \quad \text{s.t.} \quad (7.5). \tag{7.6}$$

When G^{prev} is the previously recovered inverse of the Hessian, we are following the spirit of a quasi-Newton least secant approach. One could also consider the case $G^{prev} = 0$ as it was done in some derivative-free approaches for Hessian recovery.

Similarly to Chapters 5 and 6, let us now give two arguments to motivate the minimization of the norm of $G - G^{prev}$.

7.1.1 Theoretical motivation

Error decrease

We will now focus on the case when f is quadratic (with nonsingular Hessian) to show that the error in the difference between the optimal solution G^* of (7.6) and the true inverse of the Hessian decreases relatively to the previous estimate G^{prev}. To prove such a result, it is convenient to use the Frobenius norm in (7.6) and consider

$$\min_G \quad \frac{1}{2}\|G - G^{prev}\|_F^2 \quad \text{s.t.} \quad (7.5). \tag{7.7}$$

Let us first write f centered at x as follows

$$f(y) \;=\; a + b^\top (y - x) + \frac{1}{2}(y - x)^\top C(y - x), \tag{7.8}$$

where $a = f(x)$, $b = \nabla f(x)$, and C is the symmetric nonsingular Hessian matrix $\nabla^2 f(x)$. We want to write $f(y^\ell)$ in (7.5) in terms of z^ℓ. By applying the change of variables $z \;=\; C\,(y - x)$ in (7.8), we can define $g(z)$ as

$$g(z) \;=\; f(C^{-1}z + x) \;=\; f(x) + (C^{-1}\nabla f(x))^\top z + \frac{1}{2} z^\top C^{-1} z.$$

Therefore, to calculate $f(y^\ell)$, we can compute

$$g(z^\ell) \;=\; f(x) + (C^{-1}\nabla f(x))^\top z^\ell + \frac{1}{2}(z^\ell)^\top C^{-1} z^\ell, \quad \ell = 1, \ldots, p. \tag{7.9}$$

Theorem 7.1.1. *Let f be given by (7.8) and assume that the system of linear equations (7.5) is feasible and underdetermined in G. Let G^* be the optimal solution of problem (7.7). Then*

$$\|G^* - C^{-1}\|_F^2 \;\leq\; \|G^{prev} - C^{-1}\|_F^2. \tag{7.10}$$

Proof. Similarly to the proof of Theorem 5.2.1, we follow the arguments in [56]. From (7.5) and (7.9), we obtain $\nabla f(x)^\top (C^{-1} - G^*)\, z^\ell + \frac{1}{2}(z^\ell)^\top (C^{-1} - G^*)\, z^\ell \;=\; 0$. Thus, $C^{-1} - G^*$ is a feasible direction for the affine space in G defined by (7.5).

It then turns out that the function

$$m(\theta) \;=\; \frac{1}{2}\|(G^* - G^{prev}) + \theta(C^{-1} - G^*)\|_F^2$$

has a minimum at $\theta = 0$. From the trace definition of the Frobenius norm

$$m'(\theta) \;=\; \left[(G^* - G^{prev}) + \theta(C^{-1} - G^*)\right]^\top (C^{-1} - G^*).$$

Hence,

$$(G^* - G^{prev})^\top (C^{-1} - G^*) \;=\; 0,$$

which then implies (given the symmetry of the matrices and considering only the diagonal entries of the above matrix product)

$$\sum_{i=1}^{n}\sum_{j=1}^{n} (G^*_{ij} - G^{prev}_{ij})(C^{-1}_{ij} - G^*_{ij}) \;=\; 0.$$

The rest of the proof follows the same steps as the proof of Theorem 5.2.1. □

Error bound

The second argument is similar to the one used in [18] for the motivation of minimum Frobenius norm models. Let us expand f at y^ℓ around x in (7.5)

$$(G\nabla f(x))^\top z^\ell + \frac{1}{2}(z^\ell)^\top G z^\ell \;=\; \nabla f(x)^\top (y^\ell - x) + O(\Delta_y^2), \quad \ell = 1, \ldots, p, \tag{7.11}$$

where $\Delta_y = \max_{1\leq \ell \leq p} \|y^\ell - x\|$ and the constant in $O(\Delta_y^2)$ depends on a bound of $\nabla^2 f$ in the neighborhood of the expansion. Now, rearranging the terms, we have

$$(G\nabla f(x))^\top z^\ell - \nabla f(x)^\top (y^\ell - x) \;=\; -\frac{1}{2}(z^\ell)^\top G z^\ell + O(\Delta_y^2), \quad \ell = 1, \ldots, p. \tag{7.12}$$

Using (7.2), we obtain

$$\left[-\nabla^2 f(x)^{-1}\nabla f(x) - (d^N)\right]^\top (z^\ell) = -\frac{1}{2}(z^\ell)^\top G z^\ell + O(\Delta_y^2), \quad \ell = 1, \ldots, p, \tag{7.13}$$

where $d^N = -G\nabla f(x)$ is the recovered Newton direction. By exploiting the previous equation, we can establish a bound on the norm of the absolute error of the recovered Newton direction. The bound is

based on $\Delta_z = \max_{1\le\ell\le p}\|z^\ell\|$ and on the conditioning of the matrix M_L^z, whose rows are $(1/\Delta_z)(z^\ell)^\top$, $\ell = 1,\dots,p$ (this matrix has been already introduced in Chapter 6 to prove Theorem 6.2.2).

Theorem 7.1.2. *Suppose that $p \ge n$, the matrix M_L^z is full column rank, and $\nabla^2 f(x)$ is invertible. Then, if G satisfies (7.5), one has*

$$\left\|-\nabla^2 f(x)^{-1}\nabla f(x) - d^N\right\| \;\le\; \Lambda_z\left[O(\|G - G^{prev}\|\Delta_z) + O\left(\Delta_z\right) + O\left(\frac{\Delta_y^2}{\Delta_z}\right)\right],$$

where $d^N = -G\nabla f(x)$ and Λ_z is a bound on the norm of the left inverse of M_L^z. Moreover, the multiplicative constant in the second O depends on the norm of G^{prev} and the multiplicative constant in the third O depends on the norm of $\nabla^2 f$ in a neighborhood of x.

Proof. If in (7.13) we add and subtract $(1/2)(z^\ell)^\top G^{prev} z^\ell$, we obtain

$$\left[-\nabla^2 f(x)^{-1}\nabla f(x) - (d^N)\right]^\top (z^\ell) = -\frac{1}{2}(z^\ell)^\top G z^\ell + \frac{1}{2}(z^\ell)^\top G^{prev} z^\ell - \frac{1}{2}(z^\ell)^\top G^{prev} z^\ell + O(\Delta_y^2), \quad \ell = 1,\dots,p.$$
$$(z^\ell)^\top\left[-\nabla^2 f(x)^{-1}\nabla f(x) - (d^N)\right] = \frac{1}{2}(z^\ell)^\top (G^{prev} - G) z^\ell - \frac{1}{2}(z^\ell)^\top G^{prev} z^\ell + O(\Delta_y^2), \qquad \ell = 1,\dots,p.$$

Multiplying both terms by the left inverse of M_L^z,

$$\Delta_z\left[-\nabla^2 f(x)^{-1}\nabla f(x) - (d^N)\right] = \frac{1}{2}(M_L^z)^\dagger (z^\ell)^\top (G^{prev} - G) z^\ell - \frac{1}{2}(M_L^z)^\dagger (z^\ell)^\top G^{prev} z^\ell + O(\Delta_y^2), \qquad \ell = 1,\dots,p.$$

We concludes the proof dividing both terms by Δ_z and taking the norm. □

This result motivates the recovery of the Newton direction based on the minimization of $\|G - G^{prev}\|$ when G^{prev} is itself bounded. It also motivates the recovery of the Newton direction based on the minimization of $\|G\|$ when $G^{prev} = 0$.

One can derive a similar estimate dependent only on Δ_y and on the conditioning of the matrix M_L^y formed by the rows $(1/\Delta_y)(y^\ell - x)^\top$, $\ell = 1,\dots,p$. In fact, from

$$\Delta_y M_L^y \nabla^2 f(x) \;=\; \Delta_z M_L^z$$

one has

$$\left\|(M_L^z)^\dagger\right\| \;=\; \frac{\Delta_z}{\Delta_y}\|R_y\|,$$

with

$$R_y \;=\; \left(\nabla^2 f(x)(M_L^y)^\top (M_L^y)\nabla^2 f(x)\right)^{-1}\nabla^2 f(x)(M_L^y)^\top. \tag{7.14}$$

The desired result uses also

$$\Delta_z \;\le\; \|\nabla^2 f(x)\|\Delta_y.$$

Corollary 7.1.1. *Suppose that $p \ge n$, the matrix M_L^z is full column rank, and $\nabla^2 f(x)$ is invertible. Then, if G satisfies (7.5), one has*

$$\left\|-\nabla^2 f(x)^{-1}\nabla f(x) - d^N\right\| \;\le\; \|R_y\|\left[O(\|G - G^{prev}\|\Delta_y) + O\left(\Delta_y\right)\right],$$

where R_y is given in (7.14). The multiplicative constant in the first O depends on $\|\nabla^2 f(x)\|^2$, while the multiplicative constant in the second O depends on both the norm of G^{prev} and the norm of $\nabla^2 f$ in a neighborhood of x.

Hence, by controlling the geometry of the points y^ℓ, $\ell = 1,\dots,p$, around x, one can obtain an accurate bound when the Hessian of f is invertible and $p \geq n$. This provides us with an algorithmic tool for accurately using Hessian-vectors products. Furthermore, one can still control the quality of the bound by monitoring the conditioning of M_L^z and replacing some of the points y^ℓ if necessary. Such a conditioning must eventually become adequate if the vectors $y^\ell - x$ are sufficiently linearly independent and lie in eigenspaces of $\nabla^2 f(x)$ corresponding to eigenvalues not too close to zero.

7.1.2 Re-using previous Hessian-vector products

Now, let us suppose that we want to take advantage of previous Hessian-vector products. For this purpose, consider

$$\bar{z}^\ell \;=\; \nabla^2 f(\bar{x})(\bar{y}^\ell - \bar{x}),$$

where $\bar{x}$ is a previous iterate and $\bar{y}^\ell$ are corresponding sample points.

If we now use $\bar{z}^\ell$ in place of z^ℓ in the recovery problem (7.6), the analysis based on (7.5) is no longer valid. Instead, we need to use $z^\ell = \nabla^2 f(x)(\bar{y}^\ell - x)$. But

$$\bar{z}^\ell \;=\; z^\ell + \left(\nabla^2 f(\bar{x}) - \nabla^2 f(x)\right)(\bar{y}^\ell - \bar{x}) + \nabla^2 f(x)(x - \bar{x}),$$

and assuming Lipschitz continuity of $\nabla^2 f$, we see that the error in z^ℓ is of the order of $\|\bar{x} - x\|$. The error in the approximate Newton direction will thus contain a term involving $\|\bar{x} - x\|/\Delta_z$.

7.1.3 Recovery cost of the Newton direction

Let α_Q represent the coefficients of G in $(1/2)w^\top G w$ in terms of the monomial basis. The quadratic components of this basis are of the form $(1/2)w_i^2$, $i = 1,\dots,n$ and $w_i w_j$, $1 \leq i < j \leq n$. The recovery problem can then be formulated approximately[1] as

$$\min_{\alpha_Q} \quad \frac{1}{2}\|\alpha_Q - \alpha_Q^{prev}\|^2 \quad \text{s.t.} \quad M_Q \alpha_Q \;=\; \delta(Y,x), \tag{7.15}$$

where

$$\begin{aligned}
(M_Q\alpha_Q)_\ell \;&=\; \nabla f(x)^\top G z^\ell + \frac{1}{2}(z^\ell)^\top G z^\ell, \quad \ell = 1,\dots,p,\\
\delta(Y,x)_\ell \;&=\; f(y^\ell) - f(x), \qquad\qquad\qquad\quad \ell = 1,\dots,p,
\end{aligned}$$

[1] The norm used in (7.6) for α_Q is a minor variation of the Frobenius norm of G.

and α_Q^{prev} represents the coefficients of G^{prev}. The necessary and sufficient optimality conditions for the convex QP (7.15) can be stated as

$$\begin{aligned}\alpha_Q - \alpha_Q^{prev} - M_Q^\top \lambda &= 0\\ M_Q \alpha_Q &= \delta(Y,x),\end{aligned}$$

where λ denotes the vector of Lagrange multipliers, which can be recovered by solving

$$M_Q M_Q^\top \lambda \;=\; \delta(Y,z) - M_Q \alpha_Q^{prev}.$$

This system can be solved by applying, e.g., the CG method.

We now aim to assess the cost of each matrix-vector product $M_Q M_Q^\top \lambda$ in terms of both floating point operations and storage space. While the matrix M_Q requires a storage space of the order of pn^2, the matrix $M \equiv M_Q M_Q^\top$ requires an overall cost of the order of $(2p)^2$ and the overall effort is $O(pn^2)$. The calculation of α_Q is only necessary to compute the approximate Newton direction $d^N = -G\nabla f(x)$. One should not store α_Q as it costs $O(n^2)$. Suppose that $\alpha_Q^{prev} = 0$. Then, the calculation of $\alpha_Q = M_Q^\top \lambda$ should be done using vertical blocks of the matrix $M_Q^\top$ with $n-i+1$ lines, $i = 1,\ldots,n$, after which we can immediately calculate the contribution of the respective elements of α_Q in the product $-G\nabla f(x)$. The overall effort here is again $O(pn^2)$, but the storage requires a single auxiliary vector with n components. Therefore, the final effort is $O(c_g pn^2)$, where c_g is the number of CG iterations. The storage required is $O(pn)$, corresponding to the storage of the y's and the z's which dominate all other auxiliary storage.

A problem arises when the α_Q^{prev} from the previous iteration is stored explicitly since it would cost $O(n^2)$ elements. Here the fix requires a limited memory approach where the previous Hessian inverses are stored by keeping track of the previous sets of z's and previous Lagrange multipliers λ (thus allowing the evaluation of the previous $M_Q^\top \lambda$ products). Of course, such a procedure must return an iterate where the corresponding α_Q admits a storage linear in n like, for instance, an identity or a diagonal matrix. Let us explain such a procedure in more detail. For this purpose, we introduce a notation dependent on an iteration counter k. Assume one has

$$\alpha_Q^k - \alpha_Q^{k-1} - (M_Q^k)^\top \lambda^k \;=\; 0.$$

Then, for a $\bar{k} < k$,

$$\alpha_Q^k \;=\; \sum_{i=\bar{k}+1}^{k} (M_Q^i)^\top \lambda^i + \alpha_Q^{\bar{k}}.$$

Storing the previous sets of z points $\{z_i^\ell, i = \bar{k}+1,\ldots,k-1\}$ and previous multipliers $\{\lambda^i, i = \bar{k}+1,\ldots,k-1\}$ allows us to compute the matrix inner products $(M_Q^i)^\top \lambda^i$ in a storage space linear in n. Of course, $\alpha_Q^{\bar{k}}$ must also have a storage space linear in n (as it would be the case of a diagonal Hessian inverse approximation).

7.2 On the numerical linear algebra of the Newton direction recovery

Among the iterative algorithms used to determine a solution of a large-scale linear system, Newton's method shows remarkable convergence properties due to its quadratic local convergence rate [44]. However, at each iteration the exact solution of the Newton system is required, which strongly affects the overall cost. As shown in Chapter 2, inexact Newton methods have been developed to solve this issue. When the problem is large, several options can be considered to determine an approximate solution of the system. In this section, we focus on an iterative method named *Generalized Minimal Residual Method* (GMRES), which was developed by Saad and Schultz in 1986 [60]. The GMRES method approximates the exact solution of a linear system by the vector with minimal residual norm in a Krylov subspace. Our goal is to use the GMRES iterative method to approximately determine the Newton step when Newton's method is applied to solve system (6.4). In this section we focus on the idea behind the GMRES method. Our preliminary results have not been conclusive, and we have decided not to report them here.

Let $\|\cdot\|$ denote the Euclidean norm. We consider the system of linear equation given by $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is assumed to be invertible and not necessarily symmetric. The corresponding *Krylov subspace* of order m generated by a vector $v \in \mathbb{R}^n$ is

$$K_m(A;v) = \text{span}\left\{v, Av, \ldots, A^{m-1}v\right\}.$$

For a fixed m, an orthonormal basis for $K_m(A;v)$ can be computed by the *Arnoldi algorithm*, which is described in Algorithm 11.

Algorithm 11: The Arnoldi Algorithm

Set $v_1 = v/\|v\|$ and generate an orthonormal basis $\{v_1, \cdots, v_m\}$ for $K_m(A;v)$ by using the Gram-Schmidt procedure.
for $k = 1, \cdots, m$
  Compute $h_{ik} = v_i^T A v_k,\ i = 1, 2, \cdots, k,$
    $w_k = Av_k - \sum_{i=1}^{k} h_{ik} v_i,\ h_{k+1,k} = \|w_k\|.$
  if $(w_k = 0)$
    Stop and return $\{v_1, \ldots, v_k\}$.
  else
    Compute $v_{k+1} = w_k/\|w_k\|$ and set $k = k+1$.
  end (if)
end (for)

It is possible to show (see, e.g., [59]) that if the algorithm terminates at the m-th step, then the set of vectors $\{v_1, \cdots, v_m\}$ represents an orthonormal basis of $K_m(A;v)$. If we introduce the matrix V_m whose columns are the vectors v_i, we have

$$V_m^T A V_m = H_m \quad \text{and} \quad V_{m+1}^T A V_m = \widehat{H}_m, \tag{7.16}$$

where $\widehat{H}_m \in \mathbb{R}^{(m+1)\times m}$ is the upper *Hessenberg matrix*

$$\widehat{H}_m = \begin{bmatrix} h_{11} & & \cdots & h_{1m} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ & & h_{m,m-1} & h_{mm} \\ & & & h_{m+1,m} \end{bmatrix},$$

whose entries h_{ij} are given by $h_{ij} = v_i^T A v_j$, with $i = 1,\dots,m$ and $j = 1,\dots,m$, and $h_{m+1,m} = \|w_m\|$, and $H_m \in \mathbb{R}^{m\times m}$ is the restriction of $\widehat{H}_m$ to the first m rows and m columns.

The Arnoldi algorithm can also be used to solve the given system of linear equations $Ax = b$. The resulting algorithm is referred to as the GMRES method, which aims to compute a point $x_k \in W_k = \{v = x_0 + y,\, y \in K_k(A; b - Ax_0)\}$ to minimize the Euclidean norm of the residual $\|b - Ax_k\|$, i.e.,

$$\|b - Ax_k\| = \min_{v\in W_k} \|b - Av\|.$$

Suppose that after k steps the Arnoldi algorithm yields an orthonormal basis for $K_k(A; b - Ax_0)$, which is then stored into the columns of the matrix V_k. The solution at the k-th step can be computed through

$$x_k = x_0 + V_k z_k, \tag{7.17}$$

and the residual is

$$r_k = b - Ax_k = r_0 + AV_k z_k, \quad \text{where } r_0 = v_1/\|b - Ax_0\|. \tag{7.18}$$

Recalling (7.16), notice that (7.18) can be expressed as

$$r_k = V_{k+1}\left(\|r_0\| e_1 - \widehat{H}_k z_k\right), \tag{7.19}$$

where e_1 is the first unit vector in $\mathbb{R}^{k+1}$. Hence, the GMRES method at the k-th iteration can be viewed as the following minimization problem

$$\min_{z_k} \quad \left\| \|r_0\| e_1 - \widehat{H}_k z_k \right\|. \tag{7.20}$$

This is because V_{k+1} is orthogonal and thus $\|V_{k+1}\|_2$ in (7.19) does not change. The GMRES method will terminate at most after n iterations where an exact solution can be obtained. Premature termination is due to a breakdown in the orthonormalization of Arnoldi algorithm (see Algorithm 11). In particular, we can state that a breakdown occurs for the GMRES method at the m-th step (with $m < n$) if and only if the computed solution x_m coincides with the exact solution to the system.

As regards the convergence results, it is possible to prove that the sequence of residuals obtained by the application of the GMRES method is monotonically decreasing and converges after at most n steps, namely, $\|r_{k+1}\| \le \|r_k\|$, and $\|r_n\| = 0$. We point out that $\|r_k\|$ is the smallest residual in the space K_k. GMRES converges monotonically if enlarging K_k to the space K_{k+1}, the norm of the residual can only decrease. Since at the n-th step, the space K_n is $\mathbb{R}^n$, we have $\|r_n\| = 0$.

7.3 Modified Newton direction recovery from Hessian-vector products

Another open question is how to incorporate the concept of a modified Newton method in the approach for recovering the Newton direction, described in Chapter 6. We recall that in Chapter 6 we built a Newton direction model based on $f(x)$, $\nabla f(x)$, $f(y^\ell)$, $\ell = 1,\ldots,p$, and the Hessian-vector products (6.2), namely, $z^\ell = \nabla^2 f(x)(y^\ell - x)$, $\ell = 1,\ldots,p$. We developed the approach by introducing $\nabla^2 f(x)^{-1}$ in the quadratic Taylor expansion (6.1), resulting in (6.3). In the spirit of a modified Newton method, we add to the Hessian of f at x a multiple $\tau \geq 0$ of the identity, and consider instead

$$f(x) + ((\nabla^2 f(x) + \tau I)^{-1}\nabla f(x))^\top (\nabla^2 f(x) + \tau I)(y^\ell - x) + \frac{1}{2}(y^\ell - x)^\top z^\ell \simeq f(y^\ell), \ \ell = 1,\ldots,p.$$

Now $d \simeq -(\nabla^2 f(x) + \tau I)^{-1}\nabla f(x)$ is a vector that models the modified Newton direction. Using the Hessian-vector products (6.2), the new set of enriched interpolating conditions is given by

$$(z^\ell + \tau(y^\ell - x))^\top d = -f(y^\ell) + f(x) + \frac{1}{2}(y^\ell - x)^\top z^\ell, \quad \ell = 1,\ldots,p.$$

The matrix of this linear system is now

$$\begin{bmatrix} (z^1 + \tau(y^1 - x))^\top \\ \vdots \\ (z^p + \tau(y^p - x))^\top \end{bmatrix}.$$

How to initialize and update τ will be certainly crucial for the performance of this modified approach. Setting τ to positive values would depend on the sign and magnitude of the curvature values $(z^\ell)^\top(y^\ell - x)$. Notice also that the geometry of the points y^ℓ, $\ell = 1,\ldots,p$, around x has a more direct impact on the conditioning of the linear system of the recovery.

Chapter 8

Conclusions

In this dissertation, we showed how to use interpolation techniques from derivative-free optimization to model Hessian-vector products. We aimed at presenting new, refreshing ideas, laying down the theoretical groundwork for future more elaborated algorithmic developments. In particular, two main approaches were proposed for recovering the Hessian and the Newton direction by using the knowledge of Hessian-vector products. The idea of the underlying methodology is to interpolate the objective function using a quadratic on a set of points around the current one and concurrently using the curvature information from products of the Hessian times appropriate vectors, possibly defined by the interpolating points. The resulting enriched interpolating conditions form an affine space of model Hessians or model Newton directions, from which a particular one can be computed once an equilibrium or least secant principle is defined. These approaches provide algorithmic tools for accurately using the Hessian-vectors products by controlling the geometry of the interpolating points.

In the first approach, one aims at recovering a model of the Hessian matrix, possibly sparse if the true Hessian sparsity pattern is known. When f is quadratic, the error in the difference between the approximate and the true Hessian is proved to be decreasing relatively to the previous estimate. A similar result can be obtained under the price of more Hessian-vector products when f is non-quadratic. A drawback of this approach is that at most one Hessian-vector product can be used in the recovery.

The second approach aims at directly recovering the Newton direction itself from Hessian-vector products without requiring an explicit recovery of the Hessian matrix. It may incorporate several Hessian-vector products at the same time. However, a dense system of linear equations needs to be solved. Similarly to the approach for Hessian recovery, we prove that the error in the approximation of the Newton direction is monotonously non-increasing when f is quadratic. To achieve a similar result when f is non-quadratic, additional knowledge of the true Hessian or true Newton direction at the previous iterate is required.

Numerical experiments were performed for small smooth nonlinear problems to assess the effectiveness and efficiency achievable with our methodology. To this end, we embedded both approaches in a line-search algorithm that was compared with the inexact Newton method. The results show that both the proposed recovery strategies can effectively lead to a substantial reduction in the number of Hessian-vector products. However, a larger cost in terms of number of function evaluations (of the order of n^2 per main iteration) and CPU time must be paid when using the first approach. Conversely, the promising results for the second approach demonstrate that the Newton direction

recovery is highly competitive in terms of robustness. In general, if Hessian-vector products are expensive and function evaluations are relatively cheap, the approaches proposed in this dissertation are advantageous.

The two approaches lead to three ideas which were explored and analyzed as extensions or improvements of the main methodology. In particular, two additional recovery approaches can be developed by using the same techniques of the first two. One is related to the determination of both Newton direction and Hessian inverse (possibly never storing the whole inverse, rather forming its product times the gradient), while the other aims to recover a modified Newton direction. Furthermore, the development of competitive versions was addressed by proposing the use of an iterative solver for the calculation of a Newton direction model. Being at an early stage of research, these ideas were discussed leaving the numerical experiments for future work.

References

[1] Akçelik, V., Biros, G., Ghattas, O., Hill, J., Keyes, D., and van Bloemen Waanders, B. (2006). Parallel algorithms for PDE-constrained optimization. In *Parallel Processing for Scientific Computing*, pages 291–322. SIAM, Philadelphia.

[2] Andersson, L. E. and Elfving, T. (1997). A constrained Procrustes problem. *SIAM J. Matrix Anal. Appl.*, 18:124–139.

[3] Bandeira, A. S., Scheinberg, K., and Vicente, L. N. (2012). Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. *Math. Program.*, 134:223–257.

[4] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18:5595–5637.

[5] Berahas, A. S., Cao, L., and Scheinberg, K. (2019). A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *arXiv preprint arXiv:1910.04055*.

[6] Biros, G. and Ghattas, O. (2005). Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization. Part I: the Krylov-Schur solver. *SIAM J. Sci. Comput.*, 27:687–713.

[7] Bücker, H. M., Corliss, G. F., Hovland, P. D., Naumann, U., and Norris, B. (2005). *Automatic Differentiation: Applications, Theory, and Implementations*. Lecture Notes in Computational Science and Engineering. Springer, New York.

[8] Ciarlet, P. G. and Raviart, P. A. (1972). General Lagrange and Hermite interpolation in $\mathbb{R}^n$ with applications to finite element methods. *Arch. Rational Mech. Anal.*, 46:177–199.

[9] Clarke, F. H. (1983). *Optimization and Nonsmooth Analysis*. John Wiley & Sons, New York, (Reprinted by SIAM Publications, 1990).

[10] Coleman, T. F., Garbow, B. S., and Moré, J. J. (1984). Software for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 10:329–345.

[11] Coleman, T. F. and Moré, J. J. (1984). Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.*, 28:243–270.

[12] Colson, B. (2004). Trust-region algorithms for derivative-free optimization and nonlinear bilevel programming. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2:85–88.

[13] Colson, B. and Toint, P. L. (2005). Optimizing partially separable functions without derivatives. *Optim. Methods Softw.*, 20:493–508.

[14] Conn, A. R., Gould, N. I. M., and Toint, P. L. (1996). Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Math. Program.*, 14:73–110.

[15] Conn, A. R., Gould, N. I. M., and Toint, P. L. (2000). *Trust-Region Methods*. MPS-SIAM Series on Optimization, SIAM, Philadelphia.

[16] Conn, A. R., Scheinberg, K., and Toint, P. L. (1998). A derivative free optimization algorithm in practice. In *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, page 4718.

[17] Conn, A. R., Scheinberg, K., and Vicente, L. N. (2008). Geometry of interpolation sets in derivative free optimization. *Math. Program.*, 111:141–172.

[18] Conn, A. R., Scheinberg, K., and Vicente, L. N. (2009). *Introduction to Derivative-Free Optimization*. MPS-SIAM Series on Optimization. SIAM, Philadelphia.

[19] Corwin, E. and Antonette, L. (2004). Sorting in linear time-variations on the bucket sort. *J. Comput. Sci. Coll.*, 20:197–202.

[20] Curtis, A. R., Powell, M. J. D., and Reid, J. K. (1974). On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119.

[21] Dembo, R. S., Eisenstat, S. C., and Steihaug, T. (1982). Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408.

[22] Dennis Jr., J. E. and Moré, J. J. (1977). Quasi-Newton methods, motivation and theory. *SIAM Rev.*, 19:46–89.

[23] Dennis Jr., J. E. and Schnabel, R. B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice–Hall, Englewood Cliffs, (republished by SIAM, Philadelphia, in 1996, as Classics in Applied Mathematics,16).

[24] Dennis Jr., J. E. and Schnabel, R. B. (1996). Least change secant updates for quasi-Newton methods. *SIAM Rev.*, 21:443–459.

[25] Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–213.

[26] Dunning, I., Huchette, J., and Lubin, M. (2017). JuMP: a modeling language for mathematical optimization. *SIAM Rev.*, 59:295–320.

[27] Eisenstat, S. C. and Walker, H. F. (1994). Globally convergent inexact Newton methods. *SIAM J. Optim.*, 4:393–422.

[28] Fletcher, R. (1987). *Practical Methods of Optimization*. John Wiley & Sons, New York, 2nd edition.

[29] Fletcher, R. (1995). An optimal positive definite update for sparse Hessian matrices. *SIAM J. Optim.*, 5:192–218.

[30] Fletcher, R., Grothey, A., and Leyffer, S. (1997). Computing sparse Hessian and Jacobian approximations with optimal hereditary properties. In Biegler, L. T., Coleman, T. F., Conn, A. R., and Santosa, F. N., editors, *Large-Scale Optimization with Applications*, The IMA Volumes in Mathematics and its Applications, pages 37–52. Springer, New York.

[31] Garbow, B. S. and Moré, J. J. (1985). Software for estimating sparse Hessian matrices. *ACM Trans. Math. Software*, 11:363–377.

[32] Gebremedhin, A. H., Manne, F., and Pothen, A. (2005). What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Rev.*, 47:629–705.

[33] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. (1983). Computing forward-difference intervals for numerical optimization. *SIAM J. Sci. Statist. Comput.*, 4:310–321.

[34] Gould, N. I. M. (June, 2018). *Personal communication.*

[35] Gould, N. I. M., Lucidi, S., Roma, M., and Toint, P. L. (1999). Solving the trust-region subproblem using the Lanczos method. *SIAM J. Optim.*, 9:504–525.

[36] Gould, N. I. M., Orban, D., and Toint, P. L. (2015). CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.*, 60:545–557.

[37] Gratton, S., Royer, C. W., and Vicente, L. N. (2020). A decoupled first/second-order steps technique for nonconvex nonlinear unconstrained optimization with improved complexity bounds. *Math. Program.*, 179:195–222.

[38] Griewank, A. (2000). *Evaluating Derivatives: Principles and Techniques of Automatic differentiation*. Frontiers in Applied Mathematics. SIAM, Philadelphia.

[39] Griewank, A. and Toint, P. L. (1982). Partitioned variable metric updates for large structured optimization problems. *Numer. Math.*, 39:119–137.

[40] Grippo, L., Lampariello, F., and Lucidi, S. (1986). A nonmonotone line search technique for Newton's method. *SIAM J. Numer. Anal.*, 34:707–716.

[41] Hicken, J. E. (2014). Inexact Hessian-vector products in reduced-space differential-equation constrained optimization. *Optim. Eng.*, 15:575–608.

[42] Higham, N. J. (1988). The symmetric Procrustes problem. *BIT.*, 28:133–143.

[43] Hinze, M., Pinnau, R., Ulbrich, M., and Ulbrich, S. (2008). *Optimization with PDE Constraints*. Springer, New York.

[44] Kelley, C. T. (1995). *Iterative Methods for Linear and Nonlinear Equations*. Frontiers in Applied Mathematics, SIAM, Philadelphia.

[45] Kingsbury, B., Sainath, T. N., and Soltau, H. (2012). Scalable minimum Bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization. In *Interspeech*.

[46] Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45:503–528.

[47] Maggiar, A., Wachter, A., Dollnskaya, I. S., and Staum, J. (2018). A derivative-free trust-region algorithm for the optimization of functions smoothed via gaussian convolution using adaptive multiple importance sampling. *SIAM J. Optim.*, 28:1478–1507.

[48] Martens, J. (2010). Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742.

[49] Moré, J. J. and Sorensen, D. C. (1983). Computing a trust region step. *SIAM J. Sci. Statist. Comput.*, 4:553–572.

[50] Nash, S. G. (1984). Newton-type minimization via the Lanczos method. *SIAM J. Numer. Anal.*, 21:770–788.

[51] Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage. *Math. Comp.*, 35:773–782.

[52] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, Berlin, 2nd edition.

[53] Powell, M. J. D. (1974). Unconstrained minimization algorithms without computation of derivatives. *Boll. Stor. Sci. Mat.*, 9:60–69.

[54] Powell, M. J. D. (1976). Some global convergence properties of a variable metric algorithm for minimization without exact line searches. *SIAM-AMS Proceedings, Vol. IX, R. W. Cottle and C. E. Lemke, eds., SIAM Publications*, 4:53–72.

[55] Powell, M. J. D. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In Gomez, S. and Hennart, J. P., editors, *Advances in Optimization and Numerical Analysis*, volume 275 of *Mathematics and its Applications*, pages 51–67. Springer.

[56] Powell, M. J. D. (2004). Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Math. Program.*, 100:183–215.

[57] Powell, M. J. D. (2006). The NEWUOA software for unconstrained optimization without derivatives. In *Large-scale Nonlinear Optimization*, pages 255–297. Springer.

[58] Powell, M. J. D. and Toint, P. L. (1979). On the estimation of sparse Hessian matrices. *SIAM J. Numer. Anal.*, 16:1060–1074.

[59] Quarteroni, A. (2009). *Numerical Models for Differential Problems*. Springer, Milan, 1st edition.

[60] Saad, Y. and Schultz, M. H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869.

[61] Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, 20:626–637.

[62] Sun, W. and Yuan, Y. (2006). *Optimization Theory and Methods: Nonlinear Programming*. Springer, Berlin.

[63] Toint, P. L. (1977). On sparse and symmetric matrix updating subject to a linear equation. *Math. Comp.*, 31:954–961.

[64] Toint, P. L. (1978). Some numerical results using a sparse matrix updating formula in unconstrained optimization. *Math. Comp.*, 32:839–851.

[65] Wild, S. M., Regis, R. G., and Shoemaker, C. A. (2008). ORBIT: Optimization by radial basis function interpolation in trust-regions. *SIAM J. Sci. Comput.*, 30:3197–3219.

[66] Yuan, Y. (2000). On the truncated conjugate-gradient method. *Math. Program.*, 87:561–573.

Appendix A

A.1 Performance profiles

Performance profiles [25] are used to compare the performance of several solvers on a set of problems. Let S be a set of solvers and P a set of problems. Let $t_{p,s}$ be the performance metric of the solver $s \in S$ on the problem $p \in P$. Then the performance profile of solver $s \in S$ is defined as the fraction of problems where the performance ratio is at most τ,

$$\rho_s(\tau) = \frac{1}{|P|} \left| \left\{ p \in P : \frac{t_{p,s}}{\min\left\{t_{p,s'} : s' \in S\right\}} \leq \tau \right\} \right|,$$

where $|P|$ denotes the cardinality of P. The value of $\rho_s(1)$ expresses the percentage of problems on which solver s performed the best. The values of $\rho_s(\tau)$ for large τ indicate the percentage of problems successfully solved by solver s. Hence, $\rho_s(1)$ and $\rho_s(\tau)$ for large τ are, respectively, measures of the efficiency and robustness of a given solver s. Solvers with profiles above others are naturally preferred.

Appendix B

B.1 Very small test problems

Table B.1 List of 48 very small CUTEst test problems

Name	Dimension	Name	Dimension	Name	Dimension
ALLINITU	4	ARGLINA	10	ARWHEAD	10
BEALE	2	BIGGS6	6	BOX3	3
BROWNAL	10	BRYBND	10	CHNROSNB	10
COSINE	10	CUBE	2	DIXMAANA	15
DIXMAANB	15	DIXMAAND	15	DIXMAANE	15
DIXMAANF	15	DIXMAANG	15	DIXMAANH	15
DIXMAANI	15	DIXMAANJ	15	DIXMAANK	15
DIXMAANL	15	DIXON3DQ	10	DQDRTIC	10
EDENSCH10	10	ENGVAL2	3	EXPFIT	2
FMINSURF	15	GROWTHLS	3	HAIRY	2
HATFLDD	3	HATFLDE	3	HEART8LS	8
HELIX	3	HILBERTA	10	HILBERTB	10
HIMMELBG	2	HUMPS	2	KOWOSB	4
MANCINO	30	MSQRTALS	4	MSQRTBLS	9
POWER	10	SINEVAL	2	SNAIL	2
SPARSINE	10	SPMSRTLS	28	TRIDIA	10

B.2 Small sparse test problems

Table B.2 List of 12 sparse small CUTEst test problems

Name	Dimension	Name	Dimension	Name	Dimension
BDQRTIC	10	BROYDN7D	50	COSINE	200
DQRTIC	10	EDENSCH	200	ENGVAL1	200
LIARWHD	100	NONSCOMP	50	PENTDI	100
SROSENBR	50	TOINTGSS	50	TRIDIA	200

B.3 Small test problems

Table B.3 List of 26 small CUTEst test problems

Name	Dimension	Name	Dimension	Name	Dimension
BOX	200	BOXPOWER	200	BRYBND	100
CHNROSNB	50	DIXON3DQ	200	DQDRTIC	100
EDENSCH	200	ENGVAL1	200	EXTROSNB	100
GENHUMPS	100	HILBERTA	200	HILBERTB	200
INTEQNELS	100	LIARWHD	200	MOREBV	200
PENTDI	100	PENALTY1	100	POWELLSG	36
SPARSINE	100	SROSENBR	50	SROSENBR	100
TESTQUAD	100	TOINTGSS	50	TQUARTIC	100
TRIDIA	200	VAREIGVL	100		

B.4 Average CPU times

Table B.4 Average CPU times (s).

Approaches \ Problems	Very small problems	Small problems	Sparse problems
MH(D)	1.89	–	–
SMH(D)	–	–	1.12
ND(C)	0.16	4.79	–
IN	0.34	7.67	0.10

All methods tested were coded in Matlab R2016b. The experiments were run on a single processor of a system comprising Intel Core i5 CPU clocked at 1.6GHz, with 8 GB of RAM, running the macOS High Sierra operating system.