INEXACT REFLECTIVE NEWTON METHODS FOR LARGE-SCALE OPTIMIZATION SUBJECT TO BOUND CONSTRAINTS

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Mary Ann Branch
January 1996

INEXACT REFLECTIVE NEWTON METHODS FOR LARGE-SCALE OPTIMIZATION SUBJECT TO BOUND CONSTRAINTS

Mary Ann Branch, Ph.D.

Cornell University 1996

This thesis addresses the problem of minimizing a large-scale nonlinear function subject to simple bound constraints. The most popular methods to handle bound constrained problems, active-set methods, introduce a combinatorial aspect to the problem. For these methods, the number of steps to converge may be related to the number of constraints. For large problems, this behavior is particularly detrimental.

Reflective Newton methods avoid this problem by staying strictly within the constrained region. As a result, these methods have strong theoretical properties. Moreover, they behave experimentally like an unconstrained method: the number of steps to a solution is not strongly correlated with problem size.

In this thesis, we discuss the reflective Newton approach and how it can be combined with inexact Newton techniques, within a subspace trust-region method, to efficiently solve large problems. Two algorithms are presented. The first uses a line search as its globalizing strategy. The second uses a strictly trust-region approach to globally converge to a local minimizer. Global convergence and rate of convergence results are established for both methods.

We present computational evidence that using inexact Newton steps preserves the properties of the reflective Newton methods: the iteration counts are as low as when "exact" Newton steps are used. Also, both the inexact and exact methods are robust when the starting point is varied. Furthermore, the inexact reflective Newton methods have fast convergence when negative curvature is encountered, a trait not always shared by similar active-set type methods.

The role of negative curvature is further explored by comparing the subspace trust-region approach to other common approximations to the full-space trust-region problem. On problems where only positive curvature is found, these trust-region methods differ little in the number of iterations to converge. However,

for problems with negative curvature, the subspace method is more effective in capturing the negative curvature information, resulting in faster convergence.

Finally a parallel implementation on the IBM SP2 is described and evaluated; the scalability and efficiency of this implementation are shown to be as good as the matrix-vector multiply routine it depends on.

Biographical Sketch

Mary Ann Branch was born on the western coast of the United States in the city of San Bernardino, California. For her first four years, she followed her parents around to various midwestern states and also Japan. Her family then headed south and she spent the next fourteen years in Austin, Texas.

Her college education began at Austin College in Sherman, Texas. She then went on to Texas A&M University. After completing five years of undergraduate education, she had in hand a B.A. in mathematics from Austin College and an B.S. (summa cum laude) in computer science from Texas A&M University. She moved to Ithaca, New York and entered Cornell University where she earned a M.S. in computer science in 1993. Her Ph.D. followed in 1996.

Mary Ann continued her eastward trek across the United States, next stopping in the city of Boston, Massachusetts. Here on the east coast she joined The MathWorks in Natick, Massachusetts.

To my parents Patricia and Leon.

Acknowledgements

I would like to first thank the Chair of my thesis committee, Tom Coleman, for making this work possible. Tom was generous with his guidance and advice, while also leaving me free to explore this work on my own and in my own fashion. His insights were invaluable. I am very grateful for his unwavering encouragement, support, and humor.

My thanks also go to my other committee members, Steve Vavasis and Mike Todd, for carefully reading this thesis and giving me valuable comments. In particular, the extensive comments offered by Mike improved the presentation of the theoretical work.

I would like to thank Yuying Li: our many discussions about the reflective Newton methods helped me clarify much of the material in this thesis. My appreciation also goes to Anne Trefethen, whose knowledge and advice made my trips into the world of parallel computing smoother, more productive, and more fun. I am grateful to Nick Trefethen for his encouragement and for the opportunities I gained through the various seminars and colloquia he organized.

I thank my officemate Jianguo Liu and my other ACRI colleagues Wei Yuan, Chunguang Sun, and Shirish Chinchalkar, for all their help. And I also am grateful to Helene Croft and Cindy Robinson for their administrative support. I also wish to express my gratitude to Richard Johnson, who gave me help and support in too many ways to enumerate.

This research was partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-90ER25013.A000, and by NSF, AFOSR, and ONR through grant DMS-8920550, and by the Advanced Computing Research Institute, a unit of the Cornell Theory Center which receives major funding from the National Science Foundation and IBM Corporation, with additional support from New York State and members of its Corporate Research Institute.

The work reported in Chapter 9 was conducted using the resources of the Cornell Theory Center, which receives major funding from the National Science Foundation (NSF) and New York State, with additional support from the Advanced Research Projects Agency (ARPA), the National Center for Research Resources at the National Institutes of Health (NIH), IBM Corporation, and other members of the center's Corporate Research Institute.

Finally I would like to thank my parents Patricia and Leon, and my brothers James, David, and John, for their constant love and support, and for always believing in me and my work.

Table of Contents

1 Introduction **1**
- 1.1 Issues . . . 2
  - 1.1.1 Bound constraints . . . 2
  - 1.1.2 Large-scale problems . . . 3
- 1.2 Contributions and outline . . . 4

2 Background **6**
- 2.1 Active-set methods . . . 6
- 2.2 Reflective Newton methods . . . 8
- 2.3 Unconstrained minimization techniques . . . 9
  - 2.3.1 Trust-region method . . . 9
  - 2.3.2 Subspace trust-region method . . . 9
  - 2.3.3 Inexact Newton method . . . 10
- 2.4 Related work . . . 11
  - 2.4.1 Bound-constrained methods . . . 11
  - 2.4.2 Trust-region strategies . . . 12
  - 2.4.3 Inexact methods . . . 12
  - 2.4.4 Parallel methods . . . 13

3 Reflective Newton methods **14**
- 3.1 Subproblem derivation . . . 14
- 3.2 Scaling and large-step-size . . . 19
- 3.3 SLIR and STIR: Two reflective Newton methods . . . 21
  - 3.3.1 SLIR: Subspace line-search interior reflective method . . . 22
  - 3.3.2 STIR: Subspace trust-region interior reflective method . . . 24

4 Inexact SLIR: Convergence results **29**
- 4.1 MPCG: Computing an inexact Newton step . . . 29
- 4.2 Large-step-size and consistency . . . 32
- 4.3 Subspace determination . . . 37
- 4.4 First-order convergence . . . 40
- 4.5 Rate of convergence . . . 40
- 4.6 Second-order convergence . . . 52

5 Inexact STIR: Convergence results **53**
- 5.1 First-order convergence . . . 53
- 5.2 Rate of convergence . . . 56

6 SLIR: Computational results for quadratic problems **59**
6.1 Implementation 59
6.1.1 Inexact Newton step 60
6.1.2 Preconditioning 60
6.2 Computational results 60
6.2.1 QPGEN 61
6.2.2 QP3D 66
6.2.3 QP2D 66
6.2.4 OBS 71
6.3 Discussion 71

7 STIR: Computational results for nonlinear problems **75**
7.1 Implementation 75
7.2 Computational experience 76

8 Solving the trust-region subproblem **84**
8.1 Introduction 84
8.2 Computational results 90
8.3 Discussion 93

9 Parallel STIR **108**
9.1 Computational issues 109
9.2 Parallel architecture and software specifications 109
9.3 Data distribution and computation 110
9.3.1 Dot product 110
9.3.2 Matrix-vector multiply 110
9.4 Obstacle problem 111
9.5 Computational results 111
9.6 Discussion 116

10 Conclusion **119**
10.1 Future work 120

A Reflective line search **123**

B Preconditioners for MPCG **125**

C STIR: Trust region update **128**

Bibliography **129**

List of Tables

6.1 QPGEN test problems generated with various parameters. 62
6.2 QPGEN1 major iterations and error values. 62
6.3 QPGEN1 Mflop counts and speedup. 63
6.4 QPGEN2 major iterations and error values. 63
6.5 QPGEN2 Mflop counts and speedup. 67
6.6 QPGEN2 error values with ExactCG-SLIR. 67
6.7 QPGEN2 major iterations with ExactCG-SLIR. 68
6.8 QPGEN2 Mflop counts and speedup with ExactCG-SLIR. 68
6.9 QP3D test problems generated with various parameters. 69
6.10 QP3D major iterations and error values. 70
6.11 QP3D Mflops counts and speedup. 70
6.12 QP2D test problems generated with various parameters. 72
6.13 QP2D major iterations and error values. 72
6.14 QP2D Mflops counts and speedup. 73
6.15 OBS major iterations. 73
6.16 OBS Mflops counts and speedup. 74

7.1 STIR method with exact Newton steps: number of iterations. . . . 77
7.2 STIR method with inexact Newton steps, $\eta = 0.1$: number of iterations. 79
7.3 STIR method with exact Newton steps for $n = 1000$: number of iterations. 80
7.4 STIR method with inexact Newton steps for $n = 1000$: number of iterations. 81
7.5 STIR with inexact Newton steps versus LANCELOT SBMIN on a convex quadratic: number of iterations. 83
7.6 STIR with inexact Newton steps versus LANCELOT SBMIN when negative curvature exists: number of iterations. 83

8.1 Comparison when only positive curvature is encountered: number of iterations. 91
8.2 Comparison when negative curvature is encountered: number of iterations 91
8.3 Comparison of iteration counts for four methods. 94
8.4 Comparison of iteration counts for four methods. 95
8.5 Results for the SUBSP2 method: n=260. 96
8.6 Results for the SUBSP2 method: n=260. 97
8.7 Results for the SUBSP3 method: n=260. 98
8.8 Results for the SUBSP3 method: n=260. 99
8.9 Results for the STEIH method: n=260. 100

8.10 Results for the STEIH method: n=260. 101
8.11 Results for the DOGLEG method: n=260. 102
8.12 Results for the DOGLEG method: n=260. 103
8.13 Results for the DOGLEG method using $\|\cdot\|_C$: n=260. 105
8.14 Results for the DOGLEG method using $\|\cdot\|_C$: n=260. 106

9.1 Dot product: wallclock timings on IBM SP2. 112
9.2 Hessian-vector product: wallclock timings on IBM SP2 for obstacle problem I Hessian. 113
9.3 Parallel MPCG: wallclock timings on IBM SP2 for obstacle problem I. For $n = 160000$, solved in 14 iterations and 254 CG iterations; for $n = 2560000$, solve in 19 iterations and 1007 CG iterations. . . 114
9.4 PSTIR: wallclock timings on IBM SP2 with 16 processors for obstacle problem I. 117
9.5 Solving the obstacle problem I with PSTIR on the IBM SP2 with 16 processors, with the massively parallel implementation of GPCG on the CM-2, with the parallel successive over-relaxation of De Leone on the Intel iPSC/860, and the vector implementation of interior point algorithms of Han et al. on the IBM 3090-600S. Times are in wallclock seconds for the SP2, CM seconds for the CM-2, and CPU seconds for the iPSC and IBM 3090. * was solved on a 32K CM-2. NA: not available. 117

List of Figures

3.1 The dependence of v on g. 17
3.2 Reflecting and truncating the step s. 20
3.3 Large-step-size of $s \Rightarrow$ if $s_i^T g_i < 0$ then $\frac{u_i - x_i}{s_i} > 0$. 21
3.4 A line-search reflective Newton algorithm. 23
3.5 Satisfying line search conditions in SLIR Algorithm 1. 24
3.6 A trust-region reflective Newton algorithm. 25
3.7 Minimizing along the dogleg path. 27
3.8 Minimizing along the reflected dogleg path. 28

4.1 MPCG: modified preconditioned conjugate gradient algorithm. . . 30
4.2 Determining a descent direction with large-step-size and consistency. 38
4.3 A *local* inexact reflective Newton method. 44
4.4 A *global* inexact Newton reflective path algorithm. 48

5.1 The STIR method: determining a subspace. 53

6.1 Speedup of CG-SLIR over SLIR on QPGEN1 problems. 64
6.2 Speedup of CG-SLIR over SLIR on QPGEN2 problems. 65
6.3 Speedup of ExactCG-SLIR over SLIR on QPGEN2 problems. . . . 69

7.1 Updating the trust-region size. 76
7.2 STIR performance with exact Newton steps. 78
7.3 STIR method with inexact Newton steps. 80
7.4 STIR method with exact Newton steps at varied starting points. . 81
7.5 STIR method with inexact Newton steps at varied starting points. 82

8.1 The preconditioned conjugate gradient algorithm. 86
8.2 The 2-dimensional subspace algorithm. 87
8.3 The 3-dimensional subspace algorithm. 87
8.4 The Steihaug algorithm. 88
8.5 The modified dogleg algorithm. 89
8.6 Comparison of SUBSP2 and STEIH trust-region methods for unconstrained problems. 92
8.7 Comparison of SUBSP2 and STEIH trust-region methods for unconstrained problems. 92

9.1 Comparison of speedup for n= 160000. 115
9.2 Comparison of efficiency for n= 160000. 115

A.1 Determine the linear reflective path p. 124

C.1 STIR: updating Δ_k. 128

Chapter 1

Introduction

Faster computers with larger memories naturally heighten our interest in solving larger and larger computational problems. For unconstrained optimization, this interest has produced many successful large-scale methods. These methods may take advantage of sparsity in the problem. They often use approximations that reduce the computation but still retain reasonable convergence properties and accuracy. For constrained problems the water is murkier. We would like to have methods where the number of steps to convergence correlates only loosely with the number of variables (constraints) as in unconstrained methods. However typical constrained methods, with or without adaptations for large-scale problems, do not exhibit this desired behavior.

In this thesis, we look at how to minimize a *large-scale nonlinear* function subject to *bound constraints*,

$$\min_{x \in \Re^n} f(x), \quad l \leq x \leq u, \tag{1.1}$$

where $l \in \{\Re \cup \{-\infty\}\}^n$, $u \in \{\Re \cup \{\infty\}\}^n$, $l < u$, and f is a smooth function, $f : \Re^n \to \Re^1$. When every variable has a finite upper and lower bound the feasible region forms a box, earning simple bounds of this type the name box constraints. We emphasize that we are looking for a local, not global, minimizer to the objective function f. That is, while a global minimizer x_*^G satisfies: $l \leq x_*^G \leq u$, and $f(x_*^G) \leq f(x)$ for all x such that $l \leq x \leq u$, a local minimizer x_* satisfies the weaker condition: $l \leq x_* \leq u$, and $f(x_*) \leq f(x)$ for all x in some neighborhood of x_* such that $l \leq x \leq u$.

We assume f is smooth so we can compute the gradient $\nabla f(x)$ and the Hessian $\nabla^2 f(x)$. We prefer to denote the gradient of f as $g(x) = \nabla f(x)$ and the Hessian matrix of f as $H(x) = \nabla^2 f(x)$. We will write f_k for $f(x_k)$, and similarly for g_k and H_k.

We shall also consider in detail the case where the function f is a quadratic, since we can take advantage of this restricted class of problems to further improve performance.

Whether the objective function f is quadratic or nonlinear, we shall assume the problem is large, that is, n is some number greater than 100. Furthermore, we assume the Hessian matrix for the function f is sparse. We can also handle the case where the Hessian is dense but rather than explicitly forming it, we suppose we have available a subroutine to compute the product of the Hessian matrix with a vector.

1.1 Issues

Most optimization algorithms, constrained or unconstrained, begin with some point x_1 and then improve that point over some number of iterations. Each improved point x_{k+1} is found by taking a step s_k in a direction that reduces the function. This computation may be summarized as:

```
For k = 1,2,....
  1. Compute step s_k
  2. x_{k+1} = x_k + s_k
```

Then the total amount of computation required by the algorithm is given by

*total computation = (iteration count) * (computation per iteration)*

where the *iteration count* is the number of iterations at the time when the algorithm's termination criteria are satisfied, and the *computation per iteration* is the amount of work expended at each iteration to compute the step s_k (the work to update x_{k+1} is trivial).

The main difficulty in solving *constrained* problems is to keep the iteration count low. For *large* problems, we would also like to reduce the amount of computation per iteration. Typically, we reduce the computation by using some sort of approximation. However, we do *not* want to make approximations that will increase the number of iterations, which is the usual trade-off.

1.1.1 Bound constraints

We look at large-scale problems with simple bound constraints in this thesis. Even with small problems, the addition of such seemingly innocuous constraints can be detrimental in terms of time to solve the problem. What about these constraints makes the problem hard?

For unconstrained minimization, since there are no restrictions on the variables, we can make progress towards a solution in every dimension at every step. For bound-constrained problems, the typical approach is to restrict the search for a new step to a smaller space. (When we think we have correctly constrained some of the variables, we only search in the space of the remaining variables.) But then progress towards a solution is also restricted to the dimensions in the smaller space. Furthermore, the information in the smaller space may be misleading. On the one hand, if we are in the correct restricted space, then this approach is extremely effective and even saves computation (we only do work in the smaller space). But searching in the wrong space can be viewed as wasted time in some cases (when the information is misleading), and restricted progress in all cases (since we are not making progress in the dimensions we mistakenly left out of the search).

How can we instead make progress towards a solution in every dimension at every step? In this thesis we consider a newer approach that transforms the constrained problem in such a way that this is possible. The transformation is based on the first-order necessary conditions for a constrained minimizer. The result is a new scaled subproblem with a modified Hessian, but this subproblem is unconstrained. Now at each iteration the constraints are handled *implicitly* by the transformed subproblem. We work in the whole space at each step so we have the opportunity to make progress in all dimensions at every iteration. And, since we are looking at the whole space we are not misled by restricted information. We shall refer to this newer class of methods as the reflective Newton methods, first described in [CLb,CL94,CLa]. Evidence of the good theoretical properties and experimental performance of these methods will be given in later chapters.

1.1.2 Large-scale problems

Using a reflective Newton method solves the problem of how to handle the bound constraints. But if we want to handle large problems we still have work left to do. We will adopt ideas shown to be effective for large unconstrained minimization problems. The main ideas we will use are a reduced subspace trust-region strategy and an inexact Newton technique. The inexact Newton idea will be implemented using a preconditioned (linear) conjugate gradient method. Recall that we want these approximations to reduce our work per step, but not to increase the number of iterations. We shall see that for the reflective Newton methods with these approximations, we can ensure good performance by preserving certain properties of our transformed problem. We note that an alternative is to solve for an "exact" Newton step using some sort of sparse Cholesky factorization [GL81,Col84]. Such sparse factorizations are typically more efficient when the matrix is preordered to reduce fill-in of the factors. However, finding a good preordering for some matrices may be difficult. Furthermore, even the optimal preordering may produce factors of substantially higher density than the original matrix.

The reflective Newton approach has a further advantage that comes when choosing how to make the approximations. For other methods, such as active-set strategies, the choice of how to approximate the trust-region subproblem solution is limited since the subproblem to be solved usually includes some box constraints. But with reflective Newton methods, there are multiple ways to reduce the trust-region computation and use inexact Newton ideas at the same time since the subproblem is unconstrained. We shall see that the subspace trust-region strategy with inexact Newton directions appears to be superior to the inexact trust-region method developed by Steihaug [Ste83]. Although the Steihaug method lends itself to the active-set type strategies, the subspace trust-region method with inexact steps is not possible in that framework.

Finally, to solve large problems, it is sometimes advantageous to use some form of parallel computing. This first allows larger problems to be solved as the problem is typically distributed in some manner across the processors. And, given a good implementation, we will also reduce the overall computation time. Since most optimization algorithms have the inherently sequential format of computing a series of approximations to a minimizer, this part is not parallelizable. However, given a method with a low number of iterations, if those iterations contain enough work that is parallelizable, we can speed up the overall execution time of our method. This is most evident when contrasted with a method that takes more iterations that each require less work: if the amount of work per iteration is not great enough, the overhead to set up the parallel computation will be more than the saved execution time. Also, the more straightforward the parallel implementation, the more attractive the method. We shall see that the inexact reflective Newton methods, which are the focus of this thesis, have just this structure: low number of iterations with plenty of work per iteration, but a straightforward implementation.

1.2 Contributions and outline

In this thesis we develop two algorithms to handle large problems with bound constraints: a line search method and a trust-region method. In both algorithms, reflective Newton ideas are used to handle the simple bounds, and an inexact Newton technique within a subspace trust-region strategy is used to solve large problems efficiently. We prove first-order convergence for both the line search and trust-region algorithms. We also give the rate of convergence for both methods.

We demonstrate the computational characteristics of the algorithms. In particular we compare the algorithms to their "exact" counterparts, that is, where a direct factorization is used to compute the exact, as opposed to inexact, Newton step (as in [CLb,CL94,CLa]). First we solve quadratic problems using the line search method and show what sort of speedup the inexact Newton implementation gives over an implementation using a direct factorization. We also explore the trade-off between accuracy of the solution and the amount of computation on ill-conditioned problems. We solve nonlinear problems using the trust-region method and show that even as problem size grows, the iteration counts stay low and steady for a standard set of test problems. Moreover, the inexact trust-region method mimics the behavior of the exact version in terms of the number of iterations. We also give evidence that this method is robust in terms of starting points, again similar to the original exact method. Furthermore, we compare this method to an active-set gradient projection method, which also uses a trust-region strategy, and demonstrate that our method has the advantage when we encounter negative curvature.

The role that negative curvature plays is explored further when we compare various methods that use inexact Newton ideas to approximate a full-space trust-region solution. These methods include a subspace trust-region method, a method by Steihaug [Ste83], and a dogleg strategy. We show that the subspace method is particularly effective when negative curvature is involved.

Finally, we implement the trust-region algorithm on the IBM SP2. This implementation illustrates first that the computation of this algorithm maps well onto an architecture like the SP2. Moreover, it shows that the scalability of the algorithm is mainly dependent on the scalability of the Hessian-vector multiply and dot product subroutines. Efficient, scalable multiply and dot product subroutines result in an efficient, scalable inexact reflective Newton algorithm.

This dissertation is organized as follows. In Chapter 2 we review the methods currently available to solve (1.1). We describe *active-set* methods which are typically used for this type of problem, and we briefly review reflective Newton methods. Then we discuss the ways we shall handle large problems, and how this fits into the reflective Newton method framework. We shall save the theory behind reflective Newton methods for Chapter 3. This chapter also describes the actual algorithms and assumptions. Chapter 3 first describes the line search algorithm, followed by the (strictly) trust-region algorithm. We give the theoretical results in Chapters 4 and 5. As our experimental results exceed what is predicted by the theory, we give a variety of these next in Chapters 6 and 7. Finally Chapter 8 discusses various inexact trust-region strategies and provides supporting evidence that the subspace trust-region method with inexact steps is the right approach. We end with our parallel implementation and experimental results on the IBM SP2 in Chapter 9. This thesis is concluded in Chapter 10 where we summarize our results and comment on future work.

Chapter 2

Background

In this chapter we first discuss various methods for handling bound constraints. In particular, we describe active-set methods and the related method of projected gradients. Then we give a minimal description of the reflective Newton method, deferring a more complete description to Chapter 3. This is followed by a quick review of the concepts from unconstrained minimization that we use in our algorithms. These ideas include trust-region methods, subspace trust-region methods, and inexact Newton ideas.

2.1 Active-set methods

Most methods to solve the problem (1.1) fall into the category of *active-set* methods. An inequality constraint is *active* at a point if it is satisfied as an equality at that point. For bound constraints, we would say the constraint $x_i \leq u_i$ is active at x if $x_i = u_i$. The idea behind these methods is that if we knew which constraints were active at the solution, then we could simply fix those variables at the appropriate bounds. As these fixed variables are now essentially constants, we have a new problem with fewer variables (the variables associated with the inactive constraints). Furthermore, this new, smaller problem is unconstrained. And how to solve unconstrained problems is well understood.

Of course, we do not typically start the problem knowing which constraints will be active at the solution. Then the algorithm becomes a two-step process. The first step is to predict the active constraints. We call this prediction the "working set" of constraints (it may actually be a subset of the constraints that are "active" at the current point). The second is to search for a minimizer in the space of the *free* variables (the variables that do not have constraints in the working set). We may summarize these steps as

```
For  k = 1, 2, ....

  1. Determine the current active constraints (working
     set).

  2. Minimize the objective function in the subspace of
     the free variables.
```

We repeat these two steps, adjusting the active set at each iteration until the active set settles down *and* we converge to a local minimizer in the free space.

One way to determine the active set is to look at the Lagrange multipliers (see [GMW81]). For simple bound constraints, this amounts to finding i such that $x_i = l_i$ and $g_i(x) < 0$, or $x_i = u_i$ and $g_i(x) > 0$; then we delete the ith constraint from the working set. To add a constraint, after determining a search direction in Step 2 above, we follow the direction until a bound is hit. This constraint is added to the working set.

But this typically only allows the active set to change slightly at each iteration (a few variables may move in or out of the active set each time). A more effective approach is to use the steepest descent direction to determine the active set. Such *gradient projection* type methods vary in details, but basically determine the working set by moving in the steepest descent direction. If the steepest descent direction takes us away from a bound, then the bound is removed from the working set; if it takes us to a bound, the bound is added to the working set. For example, we can define the *projection* P onto the feasible region as

$$P(x) = \mathrm{mid}(l, u, x)$$

where $\mathrm{mid}(l, u, x)$ is the vector whose ith component is the median of the set $\{l_i, x_i, u_i\}$. Then the gradient projection step is $x_{k+1} = P[x_k - \alpha_k g_k]$ for some $\alpha_k > 0$. We may summarize this algorithm as

```
For  k = 1, 2, ....

  1. Repeatedly search in the projected gradient
     direction until some set of conditions is
     satisfied (usually until the working set does not
     change for several iterations, or some sufficient
     decrease condition is met).

  2. Perform a subspace minimization on the restricted
     space.
```

If the gradient direction moves away from previously active constraints, those constraints are removed from the working set. As bounds are encountered by moving in the projected gradient direction, those bounds are added to the working set. At some point, we switch to a search in the free-space that may also use second-order information. Then we repeat the gradient search.

The rationale for this approach is that the steepest descent direction gives us a good direction to move in and detect active constraints. It is cheap to compute the projected gradient as the active set changes (simply zero out the components associated with any new active constraints to get the new direction). Note that for a direction like the Newton direction, that direction would need to be recomputed when the active set changes. Just zeroing out the appropriate component will not give us the projected Newton direction.

The convergence proofs for these active-set and gradient projection types of methods then have two parts. First we must show that the correct active set is found in a finite number of steps. Then we show the usual convergence proof as in unconstrained minimization, but here it is restricted to the free space.

2.2 Reflective Newton methods

We shall defer a more complete description of these methods until Chapter 3. Briefly, these methods differ from active-set methods in that they

- stay strictly in the interior of the bound constraints
- use a scaling matrix determined from the gradient and the distance to the constraints
- solve a non-standard *unconstrained* quadratic subproblem at each iteration; some constraint information is packed into the quadratic (but it is not a penalty or barrier term)
- use a trust-region method to ensure directions with desirable properties (specific to this method)
- use a "reflection" technique to improve the steps at each iteration.

A summary description of these algorithms is

For $k = 1, 2, \ldots.$

1. Solve $\min_{s\in\Re^n}\{\psi_k(s): \ \|D_k s\|_2 \le \Delta_k\}$, where $\psi_k(s) \stackrel{\text{def}}{=} s^T g_k + \frac{1}{2} s^T (H_k + C_k) s.$
2. Move some length in the direction s, possibly reflecting off the bound constraints.

The effect of the scaling matrix D and the C matrix, which we define and motivate in the next chapter, is to keep our iterates centered within the feasible region. In other words, we move towards the "correct" bounds at the same rate in each component. If we move too close to an "incorrect" bound, we reflect off this bound back into the center region.

2.3 Unconstrained minimization techniques

2.3.1 Trust-region method

For both the line search and trust-region algorithms, our approach will solve a trust-region subproblem

$$\min_{s\in\Re^n}\{\psi_k(s): \ \|D_k s\|_2 \le \Delta_k\} \tag{2.1}$$

where $\psi_k(s)$ is our quadratic model at the point x_k. The solution to this subproblem can be completely characterized and a global solution can be found [MS83] even when the Hessian H_k is indefinite. However, such a computation involves at least solving a linear system and may include finding a minimum eigenvalue of the Hessian matrix of our subproblem. For large problems, the cost of such computations is prohibitive and we seek alternative approximate solutions of this subproblem. Recently, Sorensen [Sor94] proposed a new method for solving the subproblem (2.1) using matrix-vector multiplications. Nonetheless, the effectiveness of this approach for large-scale minimization, particularly in the context of the reflective Newton algorithms, is yet to be investigated. The next two subsections highlight other methods that reduce this trust-region subproblem computation.

2.3.2 Subspace trust-region method

We shall approach solving (2.1) by solving the smaller problem

$$\min_{s\in\Re^n}\{\psi_k(s): \ \|D_k s\|_2 \le \Delta_k, \ s \in \mathcal{S}_k\}, \tag{2.2}$$

an idea introduced in [SSB85,BSS88b] in the unconstrained minimization context. We shall use a subspace $\mathcal{S}_k$ of two dimensions. Assuming that this subspace $\mathcal{S}_k$ is well chosen, the solution to (2.2) will be a good approximation to the solution of (2.1). The computation to solve problem (2.2) is small as it only involves a two-dimensional problem. Now, in each iteration the bulk of the work is determining the two-dimensional subspace $\mathcal{S}_k$.

What should be included in this subspace? We would like to include both first and second-order directions. In the unconstrained setting, when the Hessian of the quadratic model is positive definite, reasonable directions to include are the steepest descent direction $-g_k$ and the Newton direction s_k^N. Otherwise, we can include the steepest descent direction and a direction of negative curvature p_k (with respect to the Hessian), that is, p_k such that $p_k^T H_k p_k < 0$. Then our subspace is defined by

$$\mathcal{S}_k = \text{span}\{s_k^N, -g_k\} \qquad or \qquad \mathcal{S}_k = \text{span}\{p_k, -g_k\}$$

where span$\{s_k^N, -g_k\}$ denotes the subspace spanned by the vectors $\{s_k^N, -g_k\}$. We shall discuss the merits of this method in much more detail in Chapter 8.

2.3.3 Inexact Newton method

As was explored in the unconstrained setting in [DES82], it is possible to use an approximation to the Newton direction and retain certain convergence properties of the minimization algorithm. Typically, to find the unconstrained Newton direction, we directly solve the Newton equations for s_k^N satisfying

$$H_k s_k^N = -g_k.$$

In an inexact Newton method, we iteratively solve the Newton equations for an *inexact* Newton step s_k^{IN}, that is, s_k^{IN} solves

$$H_k s_k^{IN} = -g_k + r_k, \qquad \|r_k\|/\|g_k\| \leq \gamma_k.$$

Rather than continue computation until the residual r_k of our iterative equation is zero, we stop when a weaker bound is satisfied as shown.

What is shown in [DES82] is that the order of convergence of the minimization routine using inexact steps is governed by the choice of the residual bound. For example, for a positive constant $\gamma_k = \gamma < 1$, we can guarantee linear convergence to a point that satisfies the first-order necessary conditions for a local minimizer. For $\gamma_k = \alpha\|g_k\|$ for some constant α, quadratic convergence can be guaranteed. This topic is discussed further in the context of reflective Newton methods in Chapter 4.

2.4 Related work

2.4.1 Bound-constrained methods

As discussed previously, an active-set strategy where the Lagrange multipliers are used to adjust the active set can be found in [GM76]. These methods are not attractive for large-scale problems because the method is allowed to add or drop only one constraint per iteration. Thus, if the original active set differs from the one at the solution by k constraints, then it will take at least k iterations to converge.

Projected gradient methods overcome this drawback. These methods attracted attention because the active set can change by many constraints at each iteration. These methods were first introduced separately by Goldstein and also Levitin and Polyak [Gol64,LP66]. Around this time McCormick [McC69] also published this idea for the simpler case of bounds of the form $x \geq 0$. This paper was later followed by Bertsekas in 1976 [Ber76]. These two papers incorporate "bending" which avoids "zig-zagging". Zig-zagging is when the working set oscillates between different subsets of the constraints in the problem (see [Fle87]) so that the active set does not settle down and convergence is impeded. The strategy of bending says not to always recompute a new gradient each time a bound is hit, but instead "bend" the old gradient direction along the encountered constraint and keep going. Dunn gives some more general convergence results [Dun81].

Bertsekas in 1982 [Ber82] also has a *projected Newton method*, but it is not clear what to do for nonconvex problems for this method.

A strict active-set method (without the projected gradient search) exploring inexact Newton ideas, using the conjugate gradient method to minimize the subproblem in the free variables, is given by O'Leary in [O'L80]. Of course, this method then has the same drawbacks as the method on which it is based [GM76]. Dembo and Tulowitzki [DT83] give a combination algorithm where both conjugate gradient ideas and also projected gradient ideas are used in alternate stages on quadratic problems. Calamai and Moré [CM87] have some results for more general (linear) constraints and nonlinear problems. A version with more complete convergence results can be found in Moré and Toraldo [MT89a,MT89b] for bound-constrained quadratic problems. Most of these methods use line-search ideas to globalize the minimization. Conn, Gould and Toint instead use trust-region ideas for the globalizing strategy [CGT88a,CGT88b]. These last two papers also differ from earlier methods in how the step length for the projected gradient is chosen. Burke and Moré [BM88] characterize algorithms that identify the optimal constraints in a finite number of iterations.

Fletcher and Jackson [FJ74] give an active-set method that uses direct factorizations. Lötstedt [Löt84] uses an active-set method for quadratic (minimal least squares) problems. Sparse least squares is explored in Björck [Bjö84]. Lötstedt uses conjugate gradients but Björck uses a direct factorization. Both are active-set methods of the same type as [GM76]. Coleman and Hulbert [CH89] discuss an active-set method that uses a Newton-like projection. This method handles indefiniteness and uses exact subspace minimization, not inexact ideas. Byrd, Lu and Nocedal give a limited memory BFGS method that utilizes gradient projection with active-set ideas [BLN93].

Some methods that are strictly interior methods include [HPY90] and [CH93]. The first by Han, Pardalos and Ye is an interior point method which they show converges in a finite number of steps, assuming the Hessian of a quadratic problem is positive semidefinite. Coleman and Hulbert [CH93] describe a superlinearly convergent method for convex problems. This method generates infeasible points (it is a dual method) and uses direct factorizations.

Coleman and Li [CLb,CL94,CLa] discuss how to solve general quadratic and nonlinear problems using reflective Newton methods. A line search version of the reflective Newton method is described in [CLb,CL94]. Then [CLa] is a strictly trust-region approach. In [BCL95], the trust-region idea is expanded to a subspace trust-region method.

An approach using penalty methods for bound-constrained problems is given in [FL92]; also, a barrier method is discussed in [NS93,NPS94] for solving box-constrained problems.

2.4.2 Trust-region strategies

An excellent survey paper of trust-region strategies for unconstrained optimization is [Mor83]. The idea of approximating a trust-region step confined to a low dimensional subspace, for unconstrained problems, is discussed in [SSB85,BSS88b].

2.4.3 Inexact methods

Convergence results for inexact Newton methods for nonlinear systems of equations and optimization for unconstrained problems may be found in [DES82,DS83]. Combining inexact Newton (also called truncated-Newton) methods with trust-region ideas was discussed in [Ste83]. How to use Lanczos type computation with truncated-Newton methods, and also how to incorporate preconditioning, is discussed in [Nas84,Nas85]. These last two papers use line-search as opposed to trust-region globalizing strategies. Also, Nash and Nocedal discuss quasi-Newton updates in conjunction with truncated-Newton methods for unconstrained problems in [NN91].

2.4.4 Parallel methods

Most parallel methods to date have been for unconstrained problems. These include [NS89,NS91] which discuss parallel implementations of truncated-Newton methods. These methods employ block conjugate gradient (Lanczos) algorithms, which give fairly coarse parallelization and limit the scalability. Along this same vein, a barrier method that handles bound constraints is presented in [Nas93] using basically the same parallel implementation (block conjugate gradient) as the unconstrained methods in [NS89,NS91].

Byrd, Schnabel and Shultz describe a quasi-Newton parallel method for unconstrained problems in [BSS88a]. The focus of this paper is to parallelize the function and gradient evaluations as well as the quasi-Newton update.

A parallel implementation of the GPCG algorithm by Moré and Toraldo is described in [Mor90] which is implemented on the Alliant/FX. In this paper, partitioning techniques are used on partially separable functions to improve the function/gradient evaluation speed and to also improve the speed of the Hessian-vector products. Since the GPCG method employs the idea of a working set, care must be taken to load balance the computation as the working set changes. A second implementation of the GPCG method on the Thinking Machines CM-2 is described in [MMZ95].

Finally, an extensive bibliography is given by Zenios in [Zen89].

Chapter 3

Reflective Newton methods

We begin this chapter with a discussion of the theory behind the reflective Newton methods. Then we describe two reflective Newton algorithms. Both solve the same subproblem as their main computation, but they differ in the strategy used to ensure global convergence to a local minimizer. The first uses a specialized line search along a direction generated by a trust-region subproblem. The second is a strictly trust-region approach. How these methods are modified to handle large-scale problems, and how these modifications fit into the methods, is given in more detail in Chapters 4 and 5.

For discussion purposes, we will describe both algorithms in terms of solving nonlinear problems. However, experimentally we have found the line search method to be most effective for quadratic problems. For more general objective functions, the line search becomes too expensive and the trust-region method is the better alternative.

3.1 Subproblem derivation

What differentiates the reflective Newton methods from previous methods? We have seen that previous methods work by searching a restricted space. Equivalently, this can be seen as moving along the hyperplanes, or edges, formed by the constraints. For the reflective Newton methods, our iterates will be strictly feasible at all times, that is, $x_k \in int(\mathcal{F}) \stackrel{\text{def}}{=} \{x : l < x < u\}$. In fact we shall see that the behavior of the methods will be to keep the iterates centered within the box of constraints.

Past methods have also handled the constraints directly. That is, the bounds must be considered while solving each subproblem. For these reflective Newton methods, the bounds do not appear explicitly in the subproblem. We handle the constraints implicitly by scaling our system as well as using a special quadratic model that contains constraint information as well as the usual quadratic approximation to the objective function. The constraint information is *not* included as a penalty or barrier term.

To describe these methods we begin with the theory that is the basis of the reflective Newton methods. This summary is based on [CL94] where the reader is referred for further details. This theory will define our quadratic model and also determine the scaling matrix. We will start with a formulation of the first order necessary conditions, for the nonlinear problem (1.1), as a system of nonlinear equations. By looking at the Newton step for this system of equations, we will derive a quadratic model for the scaled space. We then transform this model back to the original variables.

We denote the gradient and Hessian as

$$g(x) \stackrel{\text{def}}{=} \nabla f(x) \tag{3.1}$$

$$H(x) \stackrel{\text{def}}{=} \nabla^2 f(x). \tag{3.2}$$

The *feasible set* is denoted as $\mathcal{F} \stackrel{\text{def}}{=} \{x : l \leq x \leq u\}$ and the strict interior of the feasible set as $int(\mathcal{F}) \stackrel{\text{def}}{=} \{x : l < x < u\}$.

We will make use of the following assumptions:

(AS.1) Given an initial point $x_0 \in \mathcal{F}$, it is assumed that $\mathcal{L}$ is compact, where $\mathcal{L}$ is the level set, $\mathcal{L} = \{x : x \in \mathcal{F},\ f(x) \leq f(x_0)\}$.

(AS.2) $f : \mathcal{F} \rightarrow \Re$ is twice continuously differentiable on $\mathcal{L}$.

From (AS.1) and (AS.2) we know that there exists constants χ_g and χ_H such that $\|H_k\| \leq \chi_H$ and $\|g_k\| \leq \chi_g$ for all k.

The *first-order Karush-Kuhn-Tucker conditions* for x_* to be a local minimizer for the problem (1.1), assuming feasibility, may be written as

$$D^{-2}(x_*)g(x_*) = 0, \tag{3.3}$$

where[1]

$$D^{-1}(x) \stackrel{\text{def}}{=} \text{diag}(|v(x)|^{1/2}), \tag{3.4}$$

and $v(x)$ is defined below.

[1] Notation: if z is a vector then $|z|$ denotes a vector with the ith component equal to $|z_i|$ and $|z|^{1/2}$ denotes a vector with the ith component equal to $|z_i|^{1/2}$.

DEFINITION 3.1.1 *The vector $v(x) \in \Re^n$ is defined:*
for each $1 \leq i \leq n$,

(i). *If $g_i < 0$ and $u_i < \infty$ then $v_i \stackrel{\text{def}}{=} x_i - u_i$;*

(ii). *If $g_i \geq 0$ and $l_i > -\infty$ then $v_i \stackrel{\text{def}}{=} x_i - l_i$;*

(iii). *If $g_i < 0$ and $u_i = \infty$ then $v_i \stackrel{\text{def}}{=} -1$;*

(iv). *If $g_i \geq 0$ and $l_i = -\infty$ then $v_i \stackrel{\text{def}}{=} 1$.*

Thus $D^{-1}(x)$ is a diagonal matrix with the ith diagonal component equal to $|v_i(x)|^{1/2}$.

The dependence of v on g is illustrated in Figure 3.1. Pictured in this simple example are the finite box constraints for a two-dimensional problem and our current point x. The steepest descent direction $-g$ points at the upper bound u_1 and the lower bound l_2. Since both these bounds are finite, $v_1 = x_1 - u_1$ and $v_2 = x_2 - l_2$.

What are the second-order conditions for x_* to be a local minimizer for the problem (1.1)? Let $Free_*$ denote the set of indices corresponding to the unconstrained variables at the point x_*, that is,

$$Free_* \;=\; \{i : l_i < (x_*)_i < u_i\}.$$

Let $H_*^{Free_*}$ be the submatrix of $H(x_*)$ consisting of the rows and columns specified by the index set $Free_*$. Then the *second-order necessary conditions* may be written: if a feasible point x_* is a local minimizer of (1.1) then $D_*^{-2} g_* = 0$ and $H_*^{Free_*}$ is positive semi-definite.

To specify sufficiency conditions, we need the following nondegeneracy definition.

DEFINITION 3.1.2 *A point $x \in \mathcal{F}$ is* **nondegenerate** *if, for each index i,*

$$g(x)_i = 0 \Longrightarrow l_i < x_i < u_i. \tag{3.5}$$

A problem (1.1) is nondegenerate if (3.5) holds for every $x \in \mathcal{F}$.

Then the *second-order sufficiency conditions* are: if a feasible point x_* is nondegenerate and satisfies the first-order necessary conditions (i.e., $D_*^{-2} g_* = 0$), and $H_*^{Free_*}$ is positive definite, then x_* is a local minimizer of (1.1).

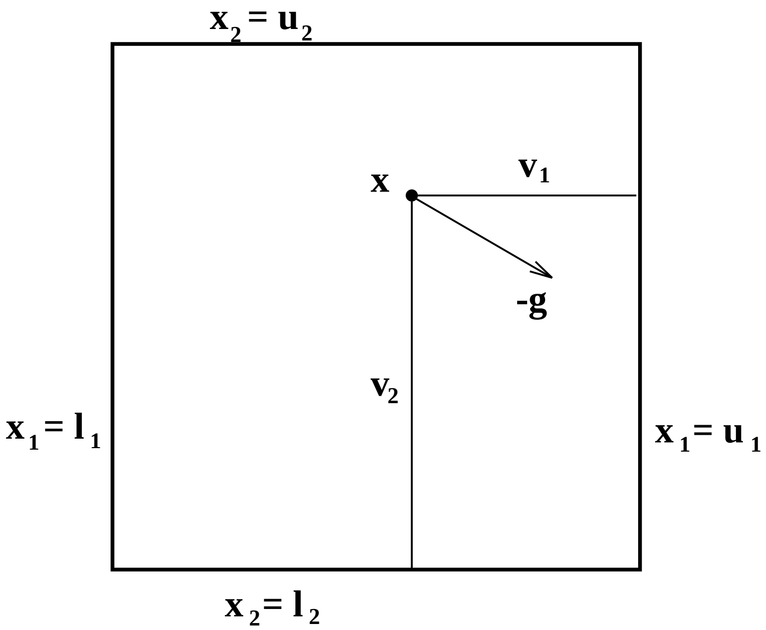

Figure 3.1: The dependence of v on g.

A solution to the nonlinear system of equations (3.3) corresponds to a first-order minimizer of (1.1). This system is continuous, but not everywhere differentiable. First, nondifferentiability occurs when $v_i = 0$. Since only strictly feasible points $x_k \in int(\mathcal{F})$ will be generated, this is not a concern. Therefore assume that $x_k \in int(\mathcal{F})$. Let $J^v(x) \in \Re^{n\times n}$ be the Jacobian matrix of $|v(x)|$ whenever $|v(x)|$ is differentiable. It is easy to see that J^v is a diagonal matrix with each diagonal component equal to zero or ± 1. For example, if all the components of l and u are finite, and each component of g is nonzero, then $J^v = \mathrm{diag}(\mathrm{sgn}(g))$.[2] But $|v_i|$ may not be differentiable at a point where $g_i = 0$. We define $J^v_{ii} = 0$ at such a point. Nondifferentiability of this type is not a cause for concern because, for such a component, it is not significant which value v_i takes. Furthermore, $|v_i|$ will still be discontinuous at this point, but the function $|v_i| \cdot g_i$ is continuous.

It follows that to find a local minimizer to the original problem (1.1), we can solve the system of nonlinear equations (3.3). Consider then the Newton step s_k^N for this system of equations,

$$\hat{B}_k D_k s_k^N = -\hat{g}_k, \tag{3.6}$$

where

$$\begin{aligned} \hat{g}_k &\stackrel{\mathrm{def}}{=} D_k^{-1} g_k, \\ \hat{B}_k &\stackrel{\mathrm{def}}{=} D_k^{-1} H_k D_k^{-1} + \mathrm{diag}(g_k) J_k^v. \end{aligned}$$

[2]If z is a vector then $w = \mathrm{sgn}(z)$ is a vector: $w_i = 1$ if $z_i \geq 0$, $w_i = -1$ if $z_i < 0$.

Based on the Newton step (3.6) for (3.3), we shall solve the following trust-region subproblem at every iteration,

$$\min_{\hat{s}\in\Re^n}\{\hat{s}_k^T\hat{g}+\tfrac{1}{2}\hat{s}^T\hat{B}_k\hat{s}:\ \|\hat{s}\|_2\le\Delta_k\}. \tag{3.7}$$

Let $s=D_k^{-1}\hat{s}$.

Subproblem (3.7) is equivalent to the following problem in the original variable space:

$$\min_{s\in\Re^n}\{s^Tg_k+\frac{1}{2}s^TB_ks:\ \|D_ks\|_2\le\Delta_k\}, \tag{3.8}$$

where

$$\begin{aligned} C_k &\stackrel{\text{def}}{=} D_k\text{diag}(g_k)J_k^vD_k,\\ B_k &\stackrel{\text{def}}{=} H_k+C_k. \end{aligned} \tag{3.9}$$

In order to use the trust-region subproblem when a problem is degenerate, we replace the second-order matrix B_k in (3.9) by M_k which is a slight modification of B_k. More precisely, we consider a solution of the following trust-region subproblem

$$\min_s\{\psi_k(s):\ \|D_ks\|_2\le\Delta_k\}, \tag{3.10}$$

where

$$\begin{aligned} \psi_k(s) &\stackrel{\text{def}}{=} s^Tg_k+\frac{1}{2}s^TM_ks,\\ M_k &\stackrel{\text{def}}{=} H_k+C_k^+,\\ C_k^+ &\stackrel{\text{def}}{=} D_k\text{diag}(g_k^+)J_k^vD_k, \end{aligned} \tag{3.11}$$

and the vector $g^+(x)$ is an "extended gradient", extended to deal with possible degeneracy. In particular,

$$g_i^+ \stackrel{\text{def}}{=} \begin{cases} g_i+\text{sgn}(g_i)\cdot\tau_\epsilon & \text{if } |g_i|+|v_i|^{\frac{1}{2}}\le\tau_\epsilon\\ g_i & \text{otherwise}\end{cases} \tag{3.12}$$

where τ_ϵ is a very small positive number, e.g., $\sqrt{\epsilon}$ where ϵ is the machine precision. Throughout the remaining presentation, we will refer to a point satisfying $|g_i(x)|+|v_i(x)|^{1/2}>\tau_\epsilon$ for all i as a *sufficiently nondegenerate* point. Clearly if x is a sufficiently nondegenerate point, then $g^+=|g|$, $C^+=C$ and $B(x)=M(x)$.

Then we may write problem (3.10) as

$$\min_{\hat{s}}\{\hat{\psi}_k(\hat{s}):\ \|\hat{s}\|_2\le\Delta_k\} \tag{3.13}$$

where

$$
\begin{aligned}
&\hat{\psi}_k(\hat{s}) \stackrel{\text{def}}{=} \hat{s}^T\hat{g}_k + \frac{1}{2}\hat{s}^T\hat{M}_k\hat{s}, \\
&\hat{M}_k = D_k^{-1}M_kD_k^{-1} = D_k^{-1}H_kD_k^{-1} + J_k^vD_k^{g^+}, \\
&\hat{g}_k = D_k^{-1}g_k, \\
&\hat{s} = D_ks,
\end{aligned}
\tag{3.14}
$$

and D^{g^+} is a diagonal matrix, $D^{g^+} \stackrel{\text{def}}{=} \text{diag}(g^+)$. Similarly, if x is a sufficiently nondegenerate point, then $\hat{M} = \hat{B}$.

The matrices M_k and $\hat{M}_k$ are positive definite in the neighborhood of a minimizer of (1.1) that satisfies the second-order sufficiency conditions (assuming the problem is nondegenerate). We note that this is true for $\hat{M}_k$ despite its possible lack of continuity; the possible discontinuity is a result of the discontinuity of $|v_i|$ when g_i changes sign, and the discontinuity introduced by g_i^+. Whatever values v_i and g_i^+ take on (and there are a finite number of such values), the resulting matrix $\hat{M}_k$ is still positive definite. The matrix M_k only has the discontinuities resulting from g_i^+ since $|v_i| \cdot g_i$ is continuous. However, the matrix M_k may be ill-conditioned as the norm of the matrix C_k^+ is unbounded above (consider the effect of D_k in C_k^+ as a component of v_k approaches zero). The matrix $\hat{M}_k$ does not share this ill-conditioning: due to the scaling by D_k^{-1} the norm of the matrix $\hat{M}_k$ is bounded, assuming smoothness and compactness as in (AS.1) and (AS.2). Thus, we shall choose to solve (3.13) at each iteration to compute a step, which is then transformed back to the original variable space (and this step is a solution to (3.10)).

From (AS.1) and (AS.2), it follows that there exist constants $\chi_{\hat{M}}$ and χ_D such that $\|\hat{M}_k\| \leq \chi_{\hat{M}}$ and $\|D_k^{-1}\| \leq \chi_D$ for all k.

3.2 Scaling and large-step-size

What is the effect of this scaling matrix? Since we handle the constraints implicitly, we may actually compute steps that would take us outside the feasible region. In this case, our algorithms have two choices. First, if the steepest descent direction points away from the constraint we cross, we can "reflect" off this constraint and stay feasible as shown in Figure 3.2. In this figure, x is our current point, s is our computed step, and s^R is the reflection of s off the constraint. The steps s and s^R are the same except in the component corresponding to the crossed constraint where $s_i^R = -s_i$ if that is the ith component. We know that this step s^R permits further descent because $-g_i \cdot s_i^R > 0$. In the second case, the the step s intersects a constraint pointed at by the steepest descent direction. In this case, we do not reflect since the reflection will not give us continued descent. Instead we must truncate s so that it takes us to a feasible point. This case is also shown in Figure 3.2.

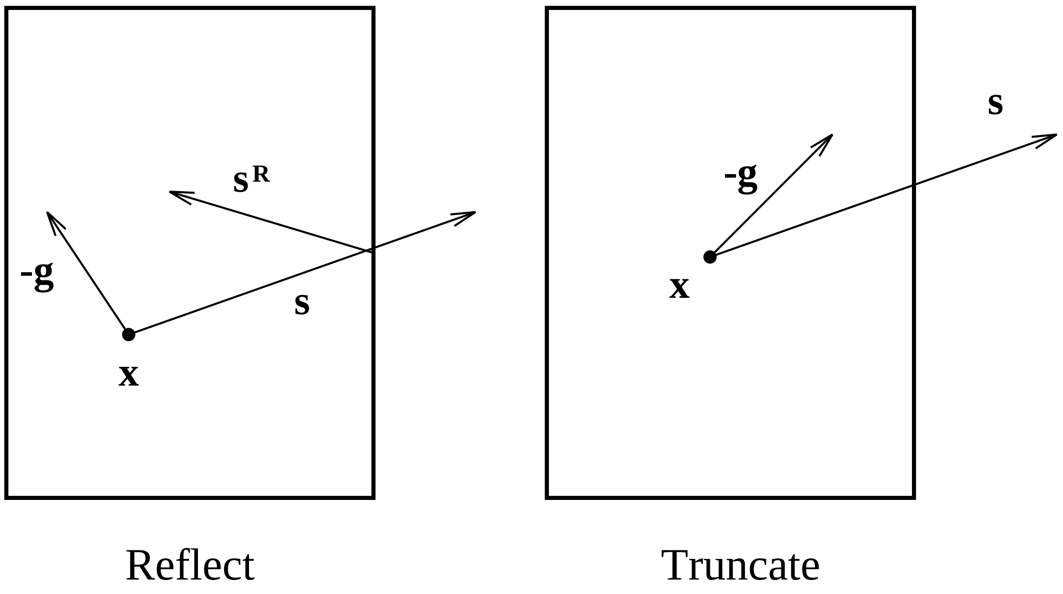

Figure 3.2: Reflecting and truncating the step s.

If we can reflect, we can still take a large step and stay feasible by following s^R. But if we truncate, how can we guarantee that our step will still be large enough to give sufficient decrease of $\psi_k(s)$? The answer is the *large-step-size* property.

DEFINITION 3.2.1 *A sequence of vectors* $\{w_k\}$ *has the* **large-step-size** *property if* $\liminf_{k\to\infty} \|D_k^2 w_k\| < \infty$.

This property, illustrated in Figure 3.3, ensures that when we cannot reflect off the ith constraint, that is $s_i^T g_i < 0$, then, for example, the ratio $\frac{u_i - x_i}{s_i}$ is bounded away from zero. This ratio is a factor in the ith component of the truncated step: $\tilde{s}_i = s_i \cdot \frac{u_i - s_i}{s_i}$. In other words, we are truncating our step but still taking a large enough step to give sufficient decrease in $\psi_k(\tilde{s})$, assuming the full step s gives sufficient decrease. This property is central to the convergence results in Chapters 4 and 5.

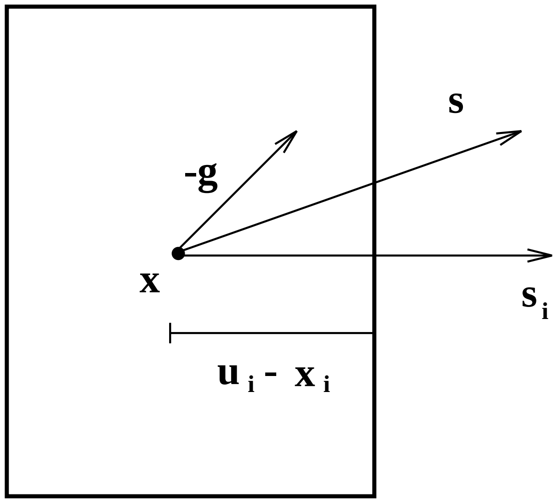

Figure 3.3: Large-step-size of $s \Rightarrow$ if $s_i^T g_i < 0$ then $\frac{u_i - x_i}{s_i} > 0$.

3.3 SLIR and STIR: Two reflective Newton methods

In the next two Subsections 3.3.1 and 3.3.2, we shall discuss the SLIR and STIR algorithms. One is particularly effective for quadratic problems, while the second is more appropriate when the objective function is nonlinear. Both algorithms will solve the trust region subproblem (3.13) to determine the step. For the nonlinear problem method, we use this trust-region subproblem solution as the step. For the quadratic method however, we will also use a special *reflective* line search that can take advantage of the structure of the problem and efficiently improve the step. For nonlinear problems, this line search becomes too expensive computationally and so we prefer a strictly trust-region globalizing strategy.

As mentioned in the previous chapter, the problem (3.13) can be quite expensive for large problems. Instead, we choose to solve

$$\min_{\hat{s} \in \Re^n} \{\hat{\psi}_k(\hat{s}) : \ \|\hat{s}\|_2 \leq \Delta_k, \ D_k^{-1}\hat{s} \in \mathcal{S}_k\}. \tag{3.15}$$

The details of how this problem is solved, in particular how the subspace is chosen, are given in Chapters 4 and 5.

At this point, we emphasize that the main computation of each method is the subproblem (3.15); the two methods differ in their globalizing strategy.

3.3.1 SLIR: Subspace line-search interior reflective method

In this subsection we describe a subspace, line-search, interior and reflective (SLIR) method to solve the problem (1.1). The SLIR algorithm is shown in Figure 3.4. We discuss the trust-region size determination, the line search conditions, and maintaining feasibility.

The basic steps of the SLIR algorithm are to compute a subspace $\mathcal{S}_k$ and solve the subspace trust-region subproblem (3.15). The solution to this subproblem is then used to define a piecewise linear reflective path (described in Appendix A). An acceptable step size is then computed by minimizing f along this piecewise linear reflective path. We defer the discussion of computing the subspace until Chapter 4, at which time we will also detail how we compute our inexact Newton step.

Trust-region size

In this method, the role of the trust-region is to ensure steps with large-step-size; the line-search gives us global convergence to a local minimizer. Since the trust region does not need to be adjusted to give global convergence, we update Δ in a non-standard way. We use a lower and upper bound for Δ and then the current Δ_k is chosen between these values based on the value of $\|v_k\|$,

$$\Delta_k = \mathrm{mid}(\Delta_L, \|v_k\|,\ \Delta_U).$$

Line search conditions

To ensure convergence a line search is used. In the unconstrained minimization setting, the following conditions may be used to guarantee sufficient descent: Given $0 < \sigma_L < \sigma_U < 1$ and a descent direction s_k with $x_{k+1} = x_k + \alpha_k s_k$, α_k satisfies the modified Armijo-Goldstein conditions [Gol80,Gol65,Arm66] if

$$f(x_{k+1}) < f(x_k) + \sigma_L(\alpha_k g_k^T s_k + \frac{1}{2}\alpha_k^2 \min(s_k^T H_k s_k, 0)) \tag{3.16}$$

and

$$f(x_{k+1}) > f(x_k) + \sigma_U(\alpha_k g_k^T s_k + \frac{1}{2}\alpha_k^2 \min(s_k^T H_k s_k, 0)). \tag{3.17}$$

Both conditions can be combined to form a single expression: If we define

$$\phi_k(\alpha) = \frac{f(x_{k+1}) - f(x_k)}{\alpha_k g_k^T s_k + \frac{1}{2}\alpha_k^2 \min(s_k^T H_k s_k, 0)} \tag{3.18}$$

conditions (3.16) and (3.17) can be expressed as

$$\sigma_L < \phi_k(\alpha_k) < \sigma_U. \tag{3.19}$$

SLIR Algorithm 1

Choose $x_1 \in int(\mathcal{F})$.

For $k = 1, 2, ...$

1. Compute an inexact Newton step s_k^{IN}. If $\|D_k s_k^{IN}\| \leq \Delta_k$ and $\hat{M}_k$ is positive definite then $s_k = s_k^{IN}$.
2. If $\hat{M}_k$ is not positive definite or $\|D_k s_k^{IN}\| > \Delta_k$ then choose a subspace $\mathcal{S}_k$ and Δ_k and solve (3.15) for a descent direction s_k.
3. Determine the piecewise linear reflective path $p_k(\alpha)$ (see Figure A.1 in Appendix A).
4. Compute an acceptable step size α_k by an an approximate piecewise line minimization of $f(x_k + p_k(\alpha))$.
5. $x_{k+1} = x_k + p_k(\alpha_k)$.

Figure 3.4: A line-search reflective Newton algorithm.

The next theorem, taken from Coleman and Li [CL94], establishes when the above conditions (3.16) and (3.17) are satisfiable for the reflective path $p_k(\alpha_k)$ and $x_{k+1} = x_k + p_k(\alpha_k)$, where p_k is defined by (A.2).

Theorem 3.3.1 *Assume that $f(x)$ has two continuous derivatives and that either $g_k^T s_k < 0$, or $g_k^T s_k = 0$ and $s_k^T H_k s_k < 0$ where $x_k \in int(\mathcal{F})$. Then either f is unbounded below along the piecewise linear path $p_k(\alpha)$ or, for $0 < \sigma_L < \sigma_U < 1$, there exists an interval (α_L, α_U), depending on k, such that condition (3.19) is satisfied for $\alpha_k \in (\alpha_L, \alpha_U)$.*

Step 4 in SLIR Algorithm 1 (Figure 3.4) is described in more detail in Figure 3.5. How to compute the piecewise linear reflective path is described in Appendix A, Figure A.1. Note that we have relaxed the conditions of the line search. If the stepsize is bounded away from zero then condition (3.17) need not be satisfied. This variant will not affect the global properties of our convergence results.

Feasibility

Since we require strict feasibility, we must consider the case where α_k is acceptable but corresponds to a breakpoint (see (A.1) in Appendix A). Then we compute $\tilde{\alpha}_k$: $\tilde{\alpha}_k \stackrel{\text{def}}{=} \alpha_k - \epsilon_k$ where $\tilde{\alpha}_k$ is not a breakpoint, $\tilde{\alpha}_k$ satisfies (3.16), and $\epsilon_k < \chi_\alpha \|D_k^{-1} g_k\|$

SLIR: Computing an acceptable step size α_k [ρ is a positive scalar.]

Let s_k be a descent direction for f at x_k, and let p_k be the piecewise linear path defined by x_k, s_k.

- Perform an approximate piecewise line minimization of $f(x_k + p_k(\alpha))$, with respect to α, to determine α_k such that:
  1. α_k does not correspond to a breakpoint
  2. condition (3.16) is satisfied
  3. Either
     (a) α_k satisfies condition (3.17), *or*
     (b) $\alpha_k > \rho > 0$

Figure 3.5: Satisfying line search conditions in SLIR Algorithm 1.

for some $\chi_\alpha > 0$. From this point on, the step size α_k will denote a *feasible* step, that is, if α_k corresponds to a breakpoint, then we use $\alpha_k = \tilde{\alpha}_k$.

3.3.2 STIR: Subspace trust-region interior reflective method

Our second algorithm, a subspace, trust-region, interior and reflective (STIR) method is displayed in Figure 3.6. In the subsection we discuss the STIR algorithm, updating the trust-region size, feasibility, and the trust-region conditions needed for convergence. Again we defer the discussion of the subspace computation until Chapter 5.

For the STIR algorithm, the main computation is computing a subspace $\mathcal{S}_k$ and solving the subspace trust-region subproblem (3.15) for a solution d_k. From d_k we construct the reflected-dogleg path and minimize $\psi_k(s)$ along this path to find the step s_k. (The reflected-dogleg path is described at the end of this section.) If the ratio ρ_k is large enough, the step s_k is *successful* and we update x_{k+1} accordingly. The trust-region size is then updated.

Trust-region size

The STIR algorithm uses a standard trust-region size updating strategy, given in Appendix C.

STIR: Algorithm 1

Let $0 < \mu < 1$, and choose $x_1 \in int(\mathcal{F})$ and $\Delta_1 > 0$.

For $k = 1, 2, ...$

1. Choose a subspace $\mathcal{S}_k$ and solve (3.15) for a descent direction d_k.
2. Compute the step s_k to the minimizer of $\psi_k(s)$ along the reflected-dogleg path.
3. Compute the ratio
$$\rho_k = \frac{f(x_k + s_k) - f(x_k) + \frac{1}{2} s_k^T C_k^+ s_k}{\psi_k(s_k)}.$$
4. If $\rho_k > \mu$ then $x_{k+1} = x_k + s_k$, else $x_{k+1} = x_k$.
5. Update Δ_k.

Figure 3.6: A trust-region reflective Newton algorithm.

Feasibility

The STIR algorithm also requires strict feasibility, i.e., $x_k + s_k \in int(\mathcal{F})$. Assume that $w_k \in \Re^n$. The scalar α_k denotes the stepsize along w_k to the boundary:

$$\alpha_k = \min\{\max\{\frac{l_i - x_{ki}}{w_{ki}}, \frac{u_i - x_{ki}}{w_{ki}}\} : 1 \leq i \leq n\}. \tag{3.20}$$

If problem (1.1) is unconstrained, i.e., $l = -\infty$ and $u = \infty$, we define $\alpha_k = +\infty$. We use $\psi_k^*[w_k]$ to denote the minimum value of $\psi_k(s)$ along the direction w_k within the feasible trust-region, i.e.,

$$\psi_k^*[w_k] \stackrel{\text{def}}{=} \psi_k(\tau_k^* w_k) \stackrel{\text{def}}{=} \min\{\psi_k(\tau w_k) : \|\tau D_k w_k\| \leq \Delta_k, x_k + \tau w_k \in \mathcal{F}\}. \tag{3.21}$$

Note that τ_k^* plays two roles in this definition. As in the unconstrained setting, $\|\tau_k^* w_k\|$ is the length of the step to the minimizer of $\psi_k(s)$ along w_k. However, this minimizer may not be feasible, in which case τ_k^* also includes a factor to scale the step back to be feasible.

Furthermore, since we always require $x_k \in int(\mathcal{F})$, that is x_k must be *strictly* feasible, a possible step-back may be necessary if $x_k + \tau_k^* w_k$ is at a breakpoint. First, the notation $\eta_k = O(\rho_k)$ denotes that there exists a constant $\chi > 0$ such that $|\eta_k| \leq \chi |\rho_k|$. Assume $\theta_L \in (0, 1)$. Define

$$\theta_k \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x_k + \tau_k^* w_k \in int(\mathcal{F}); \\ \theta_k \in [\theta_L, 1) \text{ and } \theta_k - 1 = O(\|w_k\|) & \text{otherwise.} \end{cases} \tag{3.22}$$

We use $\alpha_k^*[w_k]$ to denote the step obtained from w_k with a possible step-back. The exact definition of $\alpha_k^*[w_k]$ is

$$\alpha_k^*[w_k] \stackrel{\text{def}}{=} \theta_k \tau_k^* w_k. \tag{3.23}$$

Trust-region conditions

We specify the conditions needed to ensure sufficient descent, as proposed in [CLa, BCL95]. Here β, β_0, β^q, β_0^q are positive constants.

(STIR.1) $\psi_k(s_k) < \beta\psi_k^*[-D_k^{-2}g_k]$, $\|D_k s_k\| \le \beta_0\Delta_k$, $x_k + s_k \in int(\mathcal{F})$.

(STIR.2) Assume that d_k is a solution to $\min_{s\in\Re^n}\{\psi_k(s) : \|D_k s\| \le \Delta_k, s \in \mathcal{S}_k\}$. Then s_k satisfies $\psi_k(s_k) < \beta^q\psi_k^*[d_k]$, where $\|D_k s_k\| \le \beta_0^q\Delta_k$ and $x_k + s_k \in int(\mathcal{F})$.

Condition (STIR.1) is necessary for first-order convergence; (STIR.2), together with (STIR.1), is necessary for second-order convergence. Both conditions (STIR.1) and (STIR.2) are extensions of convergence conditions for unconstrained trust-region methods. In particular, when $l = -\infty$ and $u = \infty$, these assumptions are exactly what is required of trust-region methods for unconstrained minimization problems. What is contained in the subspace $\mathcal{S}_k$ determines if (STIR.2) can be used to guarantee second-order convergence. For example, if $\mathcal{S}_k$ contains the Newton step when $\hat{M}_k$ is positive definite, and $\mathcal{S}_k$ contains a direction of *sufficient* negative curvature when $\hat{M}_k$ is indefinite, then condition (STIR.2) will guarantee second-order optimality as shown in [BCL95].

When inexact Newton steps are used, however, we do not guarantee the satisfaction of (STIR.2) and so we cannot claim second-order convergence.

Reflected-dogleg path

We can however guarantee "sufficient Cauchy decrease" as in (STIR.1) using inexact Newton steps. This can be done in two ways. We shall use the technique of the dogleg path, but note that the trust-region size can also be used to ensure this condition as in [CLa]. In the dogleg path technique, if d_k is the solution to the subspace trust-region subproblem (3.15), and we define the step $s_k^{TR} \stackrel{\text{def}}{=} \alpha_k^*[d_k]$ and the step $s_k^{CP} \stackrel{\text{def}}{=} \alpha_k^*[-D_k^{-2}g_k]$. Then we minimize $\psi_k(s)$ along the line segment joining $x_k + s_k^{CP}$ and $x_k + s_k^{TR}$ for a step s_k^{DL}, as shown in Figure 3.7. Note that all the points on this line segment are feasible in terms of the bound constraints and the trust-region bound. The step s_k^{DL} satisfies (STIR.1).

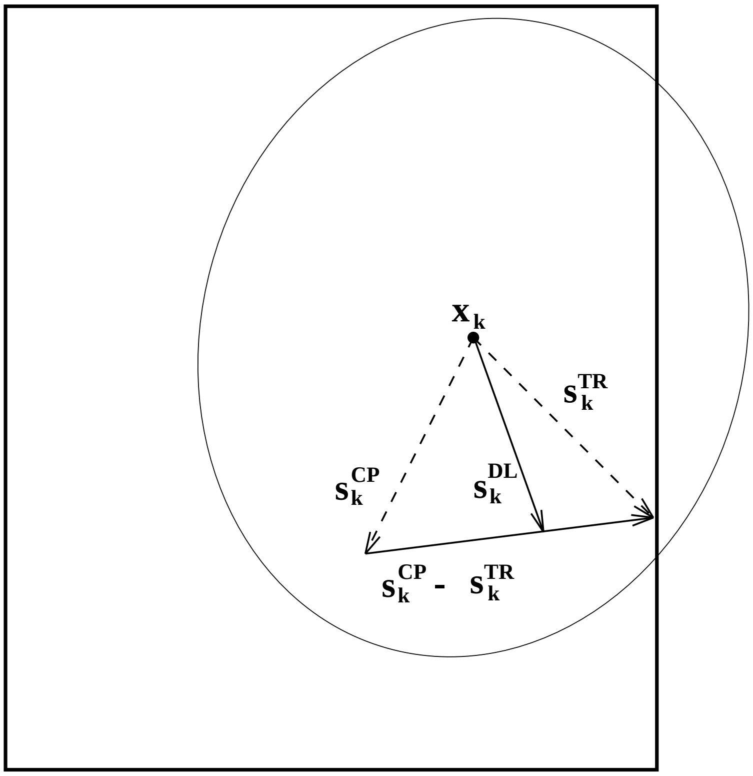

Figure 3.7: Minimizing along the dogleg path.

However, this may not lead to an efficient computation process. A reflection technique may be used as in [CLb], [CL94] and [BCL95], to improve the reduction of the objective function along a reflection path on the boundary. Thus we also compute d_k^R as described in Section 3.2. Let s_k^R be the line segment beginning at $x_k + s_k^{TR}$ and following d_k^R for as long as we remain strictly feasible and within the trust-region bound. Then we find the minimizer of $\psi_k(s)$ along the vector s_k^R; we denote this minimizer as x_*^R. If $\psi_k(x_*^R - x_k) < \psi_k(s_k^{DL})$ then $s_k = (x_*^R - x_k)$. Otherwise we choose $s_k = s_k^{DL}$. Thus computing s_k is equivalent to finding the step to the minimizer of $\psi_k(s)$ along the reflected-dogleg path, which is the path

$$x_k \implies x_k + s_k^{CP} \implies x_k + s_k^{TR} \implies x_k + s_k^{TR} + s_k^R.$$

This path is shown in Figure 3.8 (the three solid arrows).

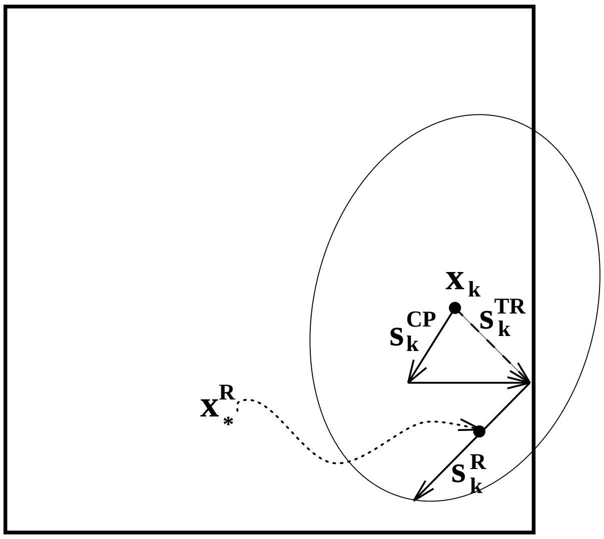

Figure 3.8: Minimizing along the reflected dogleg path.

Chapter 4

Inexact SLIR: Convergence results

We prove that first-order convergence is achieved using *inexact* Newton steps in the SLIR algorithm defined in Coleman and Li [CL94]. This algorithm is briefly described in Figure 3.4. To show first-order convergence for this algorithm, we must establish *large-step-size* and *consistency* (see [CL94] for the motivation behind these properties, referred to as *constraint compatibility* and consistency). We also show global convergence of the algorithm and we establish the rate of convergence in relation to the residual size of our inexact Newton step.

4.1 MPCG: Computing an inexact Newton step

First we describe how we compute our inexact Newton step for the SLIR algorithm. For SLIR (and later for STIR) we implement our inexact Newton method by using the preconditioned conjugate gradient method [GVL89] to find an inexact Newton step $\hat{s}_k^{IN}$ that satisfies

$$\hat{M}_k \hat{s}_k^{IN} = -\hat{g}_k + \hat{r}_k, \qquad \|\hat{r}_k\|/\|\hat{g}_k\| \leq \eta. \tag{4.1}$$

Our version of this method, *MPCG*, is shown in Figure 4.1. We will say more about the criteria that the residual $\hat{r}_k$ must meet, particularly how the stopping condition shown in Step 6 of MPCG implies (4.1), in Section 4.5. For now, we assume the above condition is met for some η where $0 < \eta < 1$.

We make the following assumptions about our MPCG algorithm.

(MPCG.1) The linear system to be solved is $\hat{M}_k \hat{s}_k = -\hat{g}_k$.

(MPCG.2) The preconditioners $P_k = C_k^2$ are symmetric positive definite where $\|C_k\| \leq \sigma_1$ and $\|C_k^{-1}\| \leq \sigma_2$.

Step 0: $i = 0$; $p_0 = 0$; $d_0 = 0$; $r_0 = -\hat{g}$;
% Note: η and ϵ are positive constants.
for $i = 1, 2, \ldots$

Step 1: Solve $Pz_i = r_i$

Step 2: $i = i + 1$

Step 3: **if** $i = 1$

$d_1 = z_0$

else

$$\beta_i = r_{i-1}^T z_{i-1} / r_{i-2}^T z_{i-2}$$
$$d_i = z_{i-1} + \beta_i d_{i-1}$$

end

Step 4: $\gamma_i = d_i^T \hat{M} d_i$

Step 5: **if** $\gamma_i < 0$, **exit**: $d = d_i$, $p = p_{i-1}$ (4.2)

else if $\gamma_i \leq \epsilon(d_i^T P d_i)$, **exit**: $d = d_i$, $p = p_{i-1}$ (4.3)

else

$$\alpha_i = r_{i-1}^T z_{i-1} / \gamma_i$$
$$p_i = p_{i-1} + \alpha_i d_i$$
$$r_i = r_{i-1} - \alpha_i \hat{M} d_i$$

end

Step 6: **if** $|r_i| \cdot \leq \eta \cdot \min(|\hat{g}|, |g|)$, **exit**: $d = d_i$, $p = p_i$ (4.4)

Step 7:

if $i >$ maximum iteration, **exit**: $d = d_i$, $p = p_i$ (4.5)

end

Figure 4.1: MPCG: modified preconditioned conjugate gradient algorithm.

(MPCG.3) If MPCG takes m iterations, then for $j = 1, \ldots, m-1$,

$$\frac{d_j^T \hat{M}_k d_j}{d_j^T P_k d_j} > \epsilon \tag{4.6}$$

for some small $\epsilon > 0$. If MPCG exits in Step 6 or 7 then (4.6) holds for $j = m$ as well.

We further assume that the preconditioners P_k are chosen so that the linear system $Pz = r$ is easily solved. The last assumption (MPCG.3) is enforced by MPCG in Step 5 by (4.3). Note that for nondegenerate problems and for small enough ϵ, this condition will always hold for x_k in the neighborhood of a minimizer x_* ($x_k \in int(\mathcal{F})$) since $\hat{M}_*$ (and each of the finite number of limits of $\hat{M}(x)$ as $x \to x_*$) is positive definite.

Most subscripts shown in the following lemmas and theorems refer to either the outer iteration of the SLIR algorithm, denoted using the subscript k, or the inner iteration (MPCG iteration), denoted by various subscript variables.

To achieve first-order convergence for the line-search method SLIR, we need to satisfy the line search conditions (3.16) and (3.17). However, we also need steps that are *consistent* and have *large-step-size*. The next lemma establishes some relations from the MPCG algorithm needed to show these two properties.

Lemma 4.1.1 *From the MPCG algorithm, the following formulas hold after m iterations when we assume (MPCG.1-3):*

$$\begin{aligned}
r_0 &= -\hat{g}, && && (4.7)\\
p_i &= \sum_{j=1}^{i} \alpha_j d_j, && i = 1, 2, \ldots, m, && (4.8)\\
\alpha_i &= \frac{r_{i-1}^T z_{i-1}}{d_i^T \hat{M} d_i}, && i = 1, 2, \ldots, m, && (4.9)\\
p_i &= \sum_{j=1}^{i} \frac{r_{j-1}^T z_{j-1}}{d_j^T \hat{M} d_j} d_j, && i = 1, 2, \ldots, m, && (4.10)\\
d_i &= r_{i-1}^T z_{i-1} \sum_{j=0}^{i-1} \frac{z_j}{r_j^T z_j}, && i = 0, 1, \ldots, m, && (4.11)\\
r_i^T z_j &= 0, && i \neq j,\ i, j = 0, 1, 2, \ldots, m, && (4.12)\\
r_j^T d_i &= r_{i-1}^T z_{i-1}, && j \leq i-1,\ i, j = 0, 1, 2, \ldots, m, && (4.13)\\
r_0^T d_i &= r_{i-1}^T z_{i-1}, && i = 1, 2, \ldots, m, && (4.14)\\
r_j^T d_i &= 0, && i \leq j, \quad i, j = 0, 1, 2, \ldots, m, && (4.15)\\
r_i^T p_i &= 0, && i = 1, 2, \ldots, m. && (4.16)
\end{aligned}$$

Proof. (4.7), (4.8), and (4.9) follow directly from the algorithm. (4.10) follows from (4.8) and (4.9). (4.11) follows from repeated expansion of the definition of d_i and some straightforward algebraic manipulation. For a proof of (4.12) when no preconditioning is used, see Golub and Van Loan [GVL89] Theorem 10.2.3, p. 521. The extension to the preconditioned case is trivial.

The relation (4.13) follows trivially from (4.11) and (4.12). And (4.14) follows directly from (4.13).

Relation (4.15) follows from (4.11) and (4.12),

$$\begin{aligned} r_j^T d_i &= r_j^T \left[r_{i-1}^T z_{i-1} \sum_{k=0}^{i-1} \frac{z_k}{r_k^T z_k} \right] \\ &= 0. \end{aligned}$$

Finally (4.16) follows directly from (4.8) and (4.15),

$$\begin{aligned} r_i^T p_i &= \sum_{j=1}^{i} \alpha_j r_i^T d_j \\ &= 0. \end{aligned}$$

□

4.2 Large-step-size and consistency

Now we are ready to show how MPCG produces steps with the large-step-size property, defined in Section 3.2. Recall that large-step-size ensures that we can take a sufficiently long step before hitting a bound when reflecting the step is not possible.

We shall make one further assumption about our MPCG algorithm.

(MPCG.4) If $\{D_k^{-2} w_k\}$ has large-step-size then $\{D_k^{-1} P_k^{-1} D_k^{-1} w_k\}$ has large-step-size. Equivalently, we may say if $\{w_k\}$ is bounded then $\{D_k P_k^{-1} D_k^{-1} w_k\}$ is bounded.

The condition (MPCG.4) is satisfied for diagonal preconditioners. Assume $P_k = C^2$ and $\hat{M} = \hat{M}_k$ where C is the diagonal matrix computed as $C_{ii} = \sqrt{|\hat{M}_{ii}|}$ if $|\hat{M}_{ii}| > \tau$, $C_{ii} = 1$ otherwise, for some small positive constant τ.

Then

$$D_k P_k^{-1} D_k^{-1} w_k = P_k^{-1} w_k \tag{4.17}$$

But each element of P_k is greater than τ, which is some constant greater than zero. This implies $\|P_k^{-1}\|$ is bounded above. Thus the assumption (MPCG.4) holds for this type of diagonal preconditioner. Furthermore, we have also shown that $\|C_k^{-1}\| \le \sigma_2$ (for some σ_2), part of assumption (MPCG.2), holds for this particular preconditioner.

Appendix B discusses how this assumption may be shown for other preconditioners.

In the next theorem, we establish sequences of inexact Newton steps that have large-step-size. In particular, we establish that d_i and p_i have large-step-size for all $i = 1, ..., m$ where m is the number of iterations of MPCG. Because both these steps retain this property at *every* iteration, we know that no matter which exit condition is satisfied in MPCG, the steps d and p returned by MPCG have large-step-size.

Theorem 4.2.1 *Under the compactness and smoothness assumptions (AS.1) and (AS.2), and the four assumptions on our algorithm MPCG, (MPCG.1-4), the following definitions yield large-step-size sequences* $\{s_k\}$*:*

1. $s_k = D_k^{-1}\hat{s}_k$ *where* $\hat{s}_k = d_i$*, for any* $i = 1, \ldots, m$*, and* d_i *is defined by the MPCG algorithm when it executes* m *iterations.*
2. $s_k = D_k^{-1}\hat{s}_k$ *where* $\hat{s}_k = p_i$*, for any* $i = 1, \ldots, m-1$ *when MPCG exits in Step 5 or, for any* $i = 1, \ldots, m$ *when MPCG exits in Step 6 or 7, and* p_i *is defined by the MPCG algorithm when it executes* m *iterations.*

Proof.

1. To show that $\{s_k\} = \{D_k^{-1}d_i\}$ has large-step-size, we prove $D_k^2 s_k = D_k d_i$ is bounded by induction. First, $D_k d_1 = D_k z_0 = -D_k P_k^{-1} D_k^{-1} g_k$, which by assumption is bounded since g_k is bounded by compactness. Also, $D_k r_0 = D_k(-\hat{g}_k) = -g_k$ is bounded by compactness.

   We assume $D_k d_i$ and $D_k r_{i-1}$ are bounded. Then

$$D_k r_i = D_k r_{i-1} - \alpha_i D_k \hat{M}_k D_k (D_k^{-1} d_i). \tag{4.18}$$

   By the induction hypothesis $D_k r_{i-1}$ and $D_k d_i$ are bounded. But $D_k d_i$ bounded implies $D_k \hat{M}_k D_k (D_k^{-1} d_i)$ is bounded. To see this note that

$$\begin{aligned} D_k \hat{M}_k D_k (D_k^{-1} d_i) &= D_k (D_k^{-1} H_k D_k^{-1} + J_k^v D_k^{g+}) d_i \\ &= (H_k D_k^{-1} + D_k J_k^v D_k^{g+}) d_i \\ &= (H_k D_k^{-2} + J_k^v D_k^{g+}) D_k d_i. \end{aligned}$$

   Then by the boundedness of $D_k d_i$ and the compactness assumption (AS.1), it follows that $D_k \hat{M}_k D_k (D_k^{-1} d_i)$ is bounded.

We claim that α_i is bounded. Note that $d_i = z_{i-1} + \beta_i d_{i-1}$ implies

$$\begin{aligned} d_i^T P_k d_i &= z_{i-1}^T P_k z_{i-1} + 2\beta_i d_{i-1}^T P_k z_{i-1} + \beta_i^2 d_{i-1}^T P_k d_{i-1} \\ &= z_{i-1}^T r_{i-1} + \beta_i^2 d_{i-1}^T P_k d_{i-1} \end{aligned}$$

since $d_{i-1}^T P_k z_{i-1} = 0$ by (4.15). Thus $z_{i-1}^T r_{i-1} < d_i^T P_k d_i$ since

$$\beta_i^2 d_{i-1}^T P_k d_{i-1} > 0.$$

Then since $\frac{d_i^T \hat{M}_k d_i}{d_i^T P_k d_i} > \epsilon$, we have that

$$\alpha_i = \frac{z_{i-1}^T r_{i-1}}{d_i^T \hat{M}_k d_i} < \frac{1}{\epsilon}.$$

Thus α_i is bounded, and we have from (4.18) that $D_k r_i$ is bounded.

Next,

$$D_k d_{i+1} = D_k z_i \ + \ \beta_{i+1} D_k d_i \tag{4.19}$$

But $D_k d_i$ is bounded by our induction hypothesis. And

$$D_k z_i \ = \ D_k P_k^{-1} r_i \ = \ D_k P_k^{-1} D_k^{-1}(D_k r_i),$$

which is bounded by assumption since $D_k r_i$ is bounded.

We show that β_{i+1} is bounded. Again we use induction. Note that $\beta_2 = \frac{r_1^T z_1}{r_0^T z_0} = \frac{r_1^T C_k^{-2} r_1}{r_0^T C_k^{-2} r_0}$. Then

$$\sqrt{\beta_2} \ = \ \frac{\|C_k^{-1} r_0 - \alpha_1 C_k^{-1} \hat{M}_k d_1\|}{\|C_k^{-1} r_0\|} \ \leq \ 1 + \alpha_1 \|C_k^{-1} \hat{M}_k C_k^{-1}\|$$

which is bounded by assumptions on C_k^{-1}, the bound on α_1, and the compactness assumption (AS.1); we also note that $\|C_k d_1\| / \|C_k^{-1} r_0\| = 1$. Next we assume that β_i is bounded and $\frac{\|C_k d_{i-1}\|}{\|C_k^{-1} r_{i-2}\|}$ is bounded. Then,

$$\sqrt{\beta_{i+1}} = \frac{\|C_k^{-1} r_{i-1} - \alpha_i C_k^{-1} \hat{M}_k d_i\|}{\|C_k^{-1} r_{i-1}\|} \leq 1 + \alpha_i \|C_k^{-1} \hat{M}_k C_k^{-1}\| \cdot \frac{\|C_k d_i\|}{\|C_k^{-1} r_{i-1}\|}$$

But,

$$\frac{\|C_k d_i\|}{\|C_k^{-1} r_{i-1}\|} = \frac{\|C_k (z_{i-1} + \beta_i d_{i-1})\|}{\|C_k^{-1} r_{i-1}\|} \leq 1 + \beta_i \frac{\|C_k d_{i-1}\|}{\|C_k^{-1} r_{i-2}\|},$$

and this is bounded by our induction hypothesis. Then,

$$\sqrt{\beta_{i+1}} \leq 1 + \alpha_i \|C_k^{-1} \hat{M}_k C_k^{-1}\| \left(1 + \beta_i \frac{\|C_k d_{i-1}\|}{\|C_k^{-1} r_{i-2}\|} \right).$$

And we have that $\sqrt{\beta_{i+1}}$ is bounded.

Now we may conclude from (4.19) that $D_k d_{i+1}$ is bounded. (Note that, while the bounds may increase with i, m is uniformly bounded by *maximum iteration.*)

2. This follows easily from the previous case. From (4.8) we have that

$$D_k^{-1} p_i = \sum_{j=1}^{i} \alpha_j D_k^{-1} d_j$$

but $\{D_k^{-1} d_j\}$ has large-step-size by the previous case. In that case, we also noted that α_j is bounded. We then conclude $\{D_k^{-1} p_i\}$ has large-step-size.

□

Note that for the last two sequences defined above we need the condition $\frac{d_i^T \hat{M}_k d_i}{d_i^T P_k d_i} > \epsilon$ for all $i = 0, 1, \ldots, m$. Thus if we are on the ith iteration and $0 < \frac{d_i^T \hat{M}_k d_i}{d_i^T P_k d_i} < \epsilon$, MPCG is exited with $p = p_{i-1}$. By case 2 above, $s_k = D_k^{-1} p_i$ has large-step-size.

We also need consistency to achieve first-order convergence. Consistency ensures that first-order descent $g_k^T s_k$ is consistent with first-order optimality $D_k^{-2} g_k$. In the next theorem, we establish sequences of inexact Newton steps that satisfy consistency. In this case, we show that for every iteration of MPCG, the direction $D_k^{-1} p_i$ satisfies the consistency condition.

DEFINITION 4.2.1 *A sequence* $\{w_k\}$ *satisfies the* **consistency** *condition if* $\{w_k^T g_k\} \to 0$ *implies* $\{D_k^{-1} g_k\} \to 0$.

Theorem 4.2.2 *Under the compactness and smoothness assumptions (AS.1) and (AS.2), and the assumptions on our algorithm MPCG, (MPCG.1-4), the following definition yields a sequence* $\{s_k\}$ *satisfying the consistency condition:* $s_k = D_k^{-1} \hat{s}_k$ *where* $\hat{s}_k = p_i$, *for any* $i = 1, \ldots, m-1$ *when MPCG exits in Step 5 or, for any* $i = 1, \ldots, m$ *when MPCG exits in Step 6 or 7, and* p_i *is defined by the MPCG algorithm when it executes* m *iterations.*

Proof. The structure of the following proof is similar to Lemma A.2 in Dembo and Steihaug [DS83].

Using relations (4.7), (4.10), and (4.14) in Lemma 4.1.1 and that $d_1 = z_0 = C_k^{-2} r_0$,

$$s_k^T g_k \quad = \quad (D^{-1} \hat{s_k})^T D \hat{g}_k$$

$$
\begin{aligned}
&= \hat{s_k}^T \hat{g}_k \\
&= -p_i^T r_0 \\
&= -\sum_{j=1}^{i} \frac{r_{j-1}^T z_{j-1}}{d_j^T \hat{M}_k d_j} d_j^T r_0 \\
&= -\sum_{j=1}^{i} \frac{(d_j^T r_0)^2}{d_j^T \hat{M}_k d_j} \\
&\leq -\frac{d_1^T r_0}{d_1^T \hat{M}_k d_1} d_1^T r_0 \\
&= -\frac{d_1^T C_k^2 d_1}{d_1^T \hat{M}_k d_1} r_0^T C_k^{-2} r_0 \\
&\leq -\frac{1}{\|C_k^{-1} \hat{M}_k C_k^{-1}\|} r_0^T C_k^{-2} r_0 \\
&\leq -\frac{1}{\|C_k^{-1}\|^2 \cdot \|\hat{M}\|} \|C_k^{-1} r_0\|^2 \\
&\leq -\frac{1}{\sigma_2^2 \cdot \chi_{\hat{M}}} \|C_k^{-1} r_0\|^2 \\
&\leq -\frac{1}{\sigma_2^2 \cdot \chi_{\hat{M}} \cdot \|C_k\|^2} \|r_0\|^2.
\end{aligned}
$$

The second to last step follows because $\|\hat{M}_k\|_2 \leq \chi_{\hat{M}}$ (see Section 3.1). Since $r_0 = \hat{g}_k = D_k^{-1} g_k$ and $\|C_k\| \leq \sigma_1$, we have

$$
s_k^T g_k \ \leq -\frac{1}{\sigma_2^2 \cdot \chi_{\hat{M}} \cdot \sigma_1^2} \|D_k^{-1} g_k\|^2
$$

which implies

$$
-g_k^T s_k \geq \frac{1}{\sigma_2^2 \cdot \chi_{\hat{M}} \cdot \sigma_1^2} \|D_k^{-1} g_k\|^2
$$

The result follows. □

We have no proof of consistency for a sequence of negative curvature directions resulting from the MPCG algorithm. The difficulty in proving such a result lies in bounding the size of the residuals in MPCG. The residuals do not decrease monotonically, and we cannot even guarantee that all residuals are less than or equal to $r_0 = \hat{g}_k$. We shall see in the next section, however, that we can achieve consistency using the negative curvature direction by solving the subspace trust-region subproblem.

4.3 Subspace determination

In this section we discuss how we can compute a step that, using the subspace trust-region subproblem (3.15) and MPCG, has large-step-size and satisfies consistency. A procedure to compute such a step is shown in Figure 4.2. We need the following definition.

DEFINITION 4.3.1 *Let $\mathcal{A}$ be a subspace and w a vector. Define $r(\mathcal{A}, w)$ to be the* residual vector *of the orthogonal projection of w onto $\mathcal{A}$. If the columns of matrix Y form an orthonormal basis for $\mathcal{A}$, then $r(\mathcal{A}, w) = w - YY^T w$. We write $r(v,\ w)$ for $r(\text{span}\{v\},\ w)$.*

Theorem 5 in [CL94], given below, establishes the large-step-size property for several useful sequences of vectors.

Theorem 4.3.1 *Under the compactness and smoothness assumptions (AS.1) and (AS.2), the following definitions yield large-step-size sequences $\{s_k\}$:*

1. $s_k = -D_k^{-2} g_k$;
2. $s_k = -D_k^{-2}\text{sgn}(g_k)$.

The key to showing that steps computed from the subspace trust-region subproblem have large-step-size is contained in the following lemma, which is Lemma 18 in [CL94].

Lemma 4.3.2 *Let a_k be a unit vector and suppose the sequences $\{D_k^{-1}a_k\}$ and $\{D_k^{-1}b_k\}$ both have large-step-size; assume there exists a constant $\tau > 0$ such that $r(a_k\ , b_k) > \tau$ for all k. Then if $u_k \in \mathcal{S}_k = \text{span}\{D_k^{-1}a_k,\ D_k^{-1}b_k\}$ and $\{D_k u_k\}$ is bounded, then $\{u_k\}$ has large-step-size.*

Clearly, if s_k is the subspace trust-region problem solution, then $s_k \in \mathcal{S}_k$ and $\|D_k s_k\| \leq \Delta_k$. Thus to use this lemma we construct our subspace to span two vectors that also have large-step-size (and are "sufficiently" orthogonal to each other).

In the case where our subspace is spanned by only one vector $D_k^{-1}a_k$ that has large-step-size, the solution s_k of the subspace trust-region subproblem will also have large-step-size if the length of a_k is bounded away from zero. Say $\mathcal{S}_k = \text{span}\{D_k^{-1}a_k\}$, $\|a_k\|$ is bounded away from zero, and $\{D_k^{-1}a_k\}$ has large-step-size. Then $s_k = D_k^{-1}a_k \cdot \frac{\tau}{\|a_k\|}$ where $0 < \tau \leq \Delta_k$, and so $\|D_k^2 s_k\| = \|D_k a_k\| \cdot \frac{\tau}{\|a_k\|}$.

Descent direction determination: [Let τ be a small positive constant.]

Case 0: $\hat{M}_k$ is positive definite and $\|\hat{s}_k^{IN}\| \le \Delta_k$.
Set $s_k = s_k^{IN} = D_k^{-1}\hat{s}_k^{IN}$.

Case 1: $\hat{M}_k$ is positive definite and $\|\hat{s}_k^{IN}\| > \Delta_k$.
if $\|r(\hat{s}_k^{IN},\ \hat{g}_k)\| > \tau$
$\mathcal{S}_k = \text{span}\{D_k^{-2}g_k, s_k^{IN}\}$, solve (3.15) to get s_k.
else
set $s_k = -D_k^{-2}g_k$.
end

Case 2: $\hat{M}_k$ is not positive definite.
Compute a negative curvature direction d ($d^T\hat{M}_k d < 0$) such that d has large-step-size; let $\hat{w}_k = d$ and $\hat{z}_k = \frac{D_k^{-1}\text{sgn}(g_k)}{\|D_k^{-1}\text{sgn}(g_k)\|}$.
if $\|r(\hat{z}_k,\ \hat{w}_k)\| > \tau$
$\mathcal{S}_k = \text{span}\{D_k^{-2}\text{sgn}(g_k), D_k^{-1}\hat{w}_k\}$, solve (3.15) to get s_k.
else
$\mathcal{S}_k = \text{span}\{D_k^{-2}\text{sgn}(g_k)\}$, solve (3.15) to get s_k.
end

Figure 4.2: Determining a descent direction with large-step-size and consistency.

But τ is bounded above, $\|a_k\|$ is bounded away from zero, and $\|D_k a_k\|$ is bounded because $\{D_k^{-1} a_k\}$ has large-step-size. Thus s_k has large-step-size.

From Theorem 6 in [CL94], stated below, we know the following sequences of vectors satisfy the consistency condition.

Theorem 4.3.3 *Under the compactness and smoothness assumptions (AS.1) and (AS.2), the following definitions yield sequences* $\{s_k\}$ *satisfying the consistency condition:*

1. $s_k = -D_k^{-2} g_k$;
2. $s_k = -D_k^{-2} sgn(g_k)$;
3. s_k *is a solution to (3.15) where* $\mathcal{S}_k$ *is a subspace of* $\mathcal{R}^n$ *with the property that* $w_k = D_k^{-1} \hat{w}_k \in \mathcal{S}_k$ *for some vector* $\hat{w}_k$ *such that* $\{\|\hat{w}_k\|\}$ *is bounded away from zero and* $\{w_k\}$ *is consistent.*

Now we justify that for every step computed by the algorithm in Figure 4.2, that step has large-step-size and is consistent. Assume that $\|D_k^{-1}\mathrm{sgn}(g_k)\|$ is bounded away from zero when $\hat{M}_k$ is not positive definite (we shall prove this holds in a later lemma). Also assume that $\hat{s}_k^{IN}$ in Case 0 or Case 1 is computed from MPCG when MPCG does not detect negative curvature (MPCG exits in Step 6, Step 7, or in Step 5 under condition (4.3)); that is $\hat{s}_k^{IN} = p$ where p is returned by MPCG. And if negative curvature is detected, then the direction of negative curvature $\hat{w}_k$ in Figure 4.2 Case 2 is the negative curvature direction found by MPCG in Step 5 under condition (4.2); that is $\hat{w}_k = d$ where d is returned by MCPG.

Case 0: In this case, $s_k = D_k^{-1}\hat{s}_k^{IN}$ where $\hat{s}_k^{IN} = p$ from the MPCG algorithm. Then the large-step-size is guaranteed by Theorem 4.2.1 (2), and Theorem 4.2.2 provides consistency.

Case 1: When the two-dimensional subspace is chosen, then $D_k^{-2} g_k$ has large-step-size by Theorem 4.3.1 (1) and s_k^{IN} has large-step-size again by Theorem 4.2.1 (2). Since $\|\hat{s}_k^{IN}\| > \Delta_k > \Delta_L$, we have that $\hat{s}_k^{IN}/\|\hat{s}_k^{IN}\|$ is a well-defined unit vector. Thus both the vectors in our subspace have large-step-size and by Lemma 4.3.2, the solution to the subspace trust-region subproblem also has large-step-size. Also, we know that since $\hat{s}_k^{IN} = p$ from the MPCG algorithm, Theorem 4.2.2 says that s_k^{IN} is consistent. Furthermore, $\|\hat{s}_k^{IN}\| > \Delta_k \geq \Delta_L$, and so we can use Theorem 4.3.3 (3) to ensure consistency of the solution to the subspace trust-region subproblem.

When $s_k = -D_k^{-2} g_k$, Theorem 4.3.1 (1) gives us large-step-size and Theorem 4.3.3 (1) guarantees consistency.

Case 2: Finally for Case 2, consider when a two-dimensional subspace is selected: first we know that $D_k^{-1}\mathrm{sgn}(g_k)/\|D_k^{-1}\mathrm{sgn}(g_k)\|$ is well-defined (the denominator is bounded away from zero) and also $D_k^{-2}\mathrm{sgn}(g_k)/\|D_k^{-1}\mathrm{sgn}(g_k)\|$ has large-step-size by Theorem 4.3.1 (2). Also, $\hat{w}_k = d$ from the MPCG algorithm, where d is a direction of negative curvature with respect to $\hat{M}_k$. Then $D_k^{-1}\hat{w}_k$ has large-step-size by Theorem 4.2.1 (1). Thus by Lemma 4.3.2, large-step-size follows for the solution to the subspace trust-region subproblem. Consistency follows from Theorem 4.3.3 (3) since we assume $\|D_k^{-1}\mathrm{sgn}(g_k)\|$ is bounded away from zero and we know $D_k^{-2}\mathrm{sgn}(g_k)$ is consistent by Theorem 4.3.3 (2).

Last, if only an one-dimensional subspace is computed, then large-step-size follows from the comments after Lemma 4.3.2 since $D_k^{-2}\mathrm{sgn}(g_k)$ has large-step-size by Theorem 4.3.1 (2) and $\|D_k^{-1}\mathrm{sgn}(g_k)\|$ is bounded away from zero. Consistency holds because $\|D_k^{-1}\mathrm{sgn}(g_k)\|$ is bounded away from zero and we know $D_k^{-2}\mathrm{sgn}(g_k)$ is consistent by Theorem 4.3.3 (2); and so by Theorem 4.3.3 (3), consistency of the subspace solution follows.

4.4 First-order convergence

We now invoke the following theorem from Coleman and Li [CL94] to achieve first-order convergence.

Theorem 4.4.1 *Assume that $\{x_k\}$ is a sequence generated by the reflective path algorithm (Algorithm 1 in Figure 3.4 with the line search conducted as described in Figure 3.5) and that $\{s_k\}$ is the corresponding sequence satisfying both the consistency and large-step-size conditions. Then the corresponding sequences $\{D_k^{-2}g_k\}$ and $\{\alpha_k^2 \min(s_k^T H_k s_k, 0)\}$ converge to zero.*

Thus by using descent directions that satisfy large-step-size and consistency, as described in Figure 4.2, Algorithm 1 in Figure 3.4 (and Figure 3.5) achieves first-order convergence.

4.5 Rate of convergence

To establish convergence of the entire sequence $\{x_k\}$, we examine a local method and its rate of convergence. We show that in the neighborhood of a local minimizer, the global method reduces to the local method.

For the inexact Newton method MPCG described in Section 4.1, the rate of convergence is related to the size of the residual of the Newton equations. The next two theorems, from [DES82], specify this relationship for the following (unconstrained) inexact Newton method:

Given an initial guess x_0, generate a sequence $\{x_k\}$ of approximations to x_*:

For $k = 0, 1, 2, ...$ until convergence

- Find some step s_k which satisfies

$$H_k s_k = -g_k + r_k, \quad \text{where} \quad \|r_k\| \leq \eta_k \|g_k\|,$$

- Set $x_{k+1} = x_k + s_k$.

Here η_k may depend on x_k.

Theorem 4.5.1 *Assume that the smoothness assumption (AS.2) holds, that there exists x_* such that $g(x_*) = 0$, $H(x_*)$ is nonsingular, $\eta_k \leq \eta_{max} < t < 1$, and $\|r_k\| \leq \eta_k \|g_k\|$. There exists $\epsilon > 0$ such that, if $\|x_0 - x_*\| \leq \epsilon$, then the sequence of inexact Newton iterates $\{x_k\}$ converges to x_*. Moreover, the convergence is linear in the sense that*

$$\|x_{k+1} - x_*\|_* \leq t\|x_k - x_*\|_* \tag{4.20}$$

*where $\|y\|_* \equiv \|H(x_*)y\|$, and $\|\cdot\|$ is an arbitrary norm on R^n. We shall refer to this as *linear convergence.*

Theorem 4.5.2 *Assume that the smoothness assumption (AS.2) holds, there exists x_* such that $g(x_*) = 0$, $H(x_*)$ is nonsingular, and the inexact Newton iterates $\{x_k\}$ converge to x_*. Then*

1. $\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)$ *as* $k \to \infty$ *if and only if*

$$\|r_k\| = o(\|g(x_k)\|) \text{ as } k \to \infty.$$

2. *Moreover, if H is Lipschitz continuous at x_*, then $\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^q)$ as $k \to \infty$ if and only if*

$$\|r_k\| = O(\|g(x_k)\|^q) \text{ as } k \to \infty.$$

We shall assume the compactness and smoothness assumptions (AS.1) and (AS.2) hold for the remainder of this section.

We establish that there is a natural (local) inexact Newton process for problem (1.1). This view is used in Coleman and Li [CL94] to show convergence. To do this, we must show that

- if $\hat{s}_k^{IN}$ is an inexact Newton step in the scaled space, that is it satisfies the residual condition $\|\hat{r}_k\|/\|\hat{g}_k\| < \eta$, then $s_k^{IN} = D_k^{-1}\hat{s}_k^{IN}$ is an inexact Newton step in the original space and satisfies $\|r_k\|/\|g_k\| < \eta$

- the step size along the reflective path (see Figure A.1) goes to unity, and that the step to the first breakpoint goes to unity.

Let x_* be a specified nondegenerate point satisfying the second-order sufficiency conditions. Consider a finite set $\mathcal{V}$ of functions defined with respect to x_*:

$$F_\nu(x) = D_\nu(x)g(x) \tag{4.21}$$

where $D_\nu(x) = \mathrm{diag}(\nu(x))$ and $\nu(x)$ is a vector defined by

$$\nu_i = \begin{cases} +1 \text{ or } -1 \text{ or } x_i - u_i \text{ or } x_i - l_i & \text{if } g_i^* = 0 \\ -1 \text{ or } x_i - u_i & \text{if } g_i^* < 0 \\ +1 \text{ or } x_i - l_i & \text{if } g_i^* > 0. \end{cases} \tag{4.22}$$

Note: The choice $\nu_i = x_i - u_i$ is valid only when u_i is finite; the choice $\nu_i = x_i - l_i$ is valid only when l_i is finite.

The following theorem is analogous to Theorem 11 in [CL94]

Theorem 4.5.3 *Let $\mathcal{V} = \{F_\nu : R^n \to R^n\}$ be a finite set of functions satisfying the following assumptions:*

- *Each F_ν is continuously differentiable in an open convex set $\mathcal{C}$.*
- *There is a x_* in $\mathcal{C}$ such that $F_\nu(x_*) = 0$ and $\nabla F_\nu(x_*)$ is nonsingular for all $F_\nu \in \mathcal{V}$.*
- *There is a constant κ_0 such that for all $F_\nu \in \mathcal{V}$,*

$$\|\nabla F_\nu(x) - \nabla F_\nu(x_*)\| \leq \kappa_0 \|x - x_*\|, \tag{4.23}$$

for $x \in \mathcal{C}$.

Let $\{x_k\}$ and $\{s_k\}$ be sequences in $\mathcal{C}$ with $\|x_1 - x_*\| \leq \epsilon$ such that $x_{k+1} = x_k + s_k$, and $s_k^{IN_{\nu_k}}$ is the inexact Newton step for one of the functions $F_{\nu_k} \in \mathcal{V}$ at x_k, i.e.,

$$s_k^{IN_{\nu_k}} = -(\nabla F_{\nu_k}(x_k))^{-1} F_{\nu_k}(x_k) + r_{\nu_k}.$$

1. *Suppose*
$$\|s_k - s_k^{IN_{\nu_k}}\| = O(\|x_k - x_*\|)^2. \tag{4.24}$$
If $\|r_{\nu_k}\| = O(\|F_{\nu_k}\|^2)$, then for ϵ sufficiently small, $\{x_k\}$ converges quadratically to x_*: $\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^2)$ as $k \to \infty$.

2. *Suppose*
$$\|s_k - s_k^{IN_{\nu_k}}\| = o(\|x_k - x_*\|). \tag{4.25}$$
If $\|r_{\nu_k}\| = o(\|F_{\nu_k}\|)$, then for ϵ sufficiently small, $\{x_k\}$ converges superlinearly to x_*: $\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)$ as $k \to \infty$.

3. *Suppose*

$$\|s_k - s_k^{IN_{\nu_k}}\|_* \le \tilde{t}(\|x_k - x_*\|_*), \tag{4.26}$$

for $0 \le \tilde{t} < \frac{1}{2}$. *If* $\|r_{\nu_k}\| \le \eta_k \cdot \|F_{\nu_k}\|$, *where* $\eta_k \le \eta_{max} < \tilde{t} < \frac{1}{2}$, *then for* ϵ *sufficiently small,* $\{x_k\}$ *converges *linearly to* x_*: $\|x_{k+1} - x_*\|_* \le t\|x_k - x_*\|_*$, *for some* $0 \le t < 1$.

Proof.

$$\begin{aligned} \|x_{k+1} - x_*\| &= \|x_k + s_k - x_*\| \\ &= \|x_k + s_k^{IN_{\nu_k}} - x_* + s_k - s_k^{IN_{\nu_k}}\| \\ &\le \|(x_k + s_k^{IN_{\nu_k}}) - x_*\| + \|s_k - s_k^{IN_{\nu_k}}\| \end{aligned} \tag{4.27}$$

But $x_k + s_k^{IN_{\nu_k}}$ is an inexact Newton step for some function F_ν. Thus we can use Theorems 4.5.1 and 4.5.2 to bound $\|(x_k + s_k^{IN_{\nu_k}}) - x_*\|$. In the last case, $\|r_{\nu_k}\| \le \eta_k\|F_{\nu_k}\|$ implies

$$\|(x_k + s_k^{IN_{\nu_k}}) - x_*\|_* \le \tilde{t}\|x_k - x_*\|_*. \tag{4.28}$$

By assumption $\|s_k - s_k^{IN_{\nu_k}}\|_* \le \tilde{t}(\|x_k - x_*\|_*)$, so with (4.27) this implies $\|x_{k+1} - x_*\|_* \le t\|x_k - x_*\|_*$ for $t = 2\tilde{t} \in [0,\ 1)$.

For the quadratic case, Theorem 4.5.2 and $\|r_{\nu_k}\| = O(\|F_{\nu_k}\|^2)$ imply

$$\|(x_k + s_k^{IN_{\nu_k}}) - x_*\| = O(\|x_k - x_*\|^2). \tag{4.29}$$

By assumption $\|s_k - s_k^{IN_{\nu_k}}\| = O(\|x_k - x_*\|^2)$, so with (4.27) this implies $\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^2)$. The superlinear case follows similarly. □

Our Local Reflective Inexact Newton method is shown as Algorithm 2. In a neighborhood of x_*, $\hat{M}_k$ is positive definite. We know $\hat{g}_k \ne 0$ if $x_k \ne x_*$. Thus, s_k in Algorithm 2 is well defined when x_k is sufficiently close to x_* and $x_k \ne x_*$. Furthermore, in a neighborhood of x_*, we will show $\alpha_k = 1$ satisfies the stepsize condition (3.16), unless $x_k + p(1)$ is on the boundary; in this case, we choose α_k slightly smaller such that $|\alpha_k - 1| = O(\|x_k - x_*\|)$ to ensure strict feasibility. (We can compute $\|D_k^{-1}g_k\|$ and we know $\|D_k^{-1}g_k\| = O(\|x_k - x_*\|)$ by using Taylor's theorem and the definition of D_k^{-1}.)

We assume x_* satisfies nondegeneracy and second-order sufficiency. Thus when x_1 is sufficiently close to x_*, the step s_k that our local method, Algorithm 2, generates is an inexact Newton step for one of the functions in $\mathcal{V}$. Then to use Theorem 4.5.3, we need that $\|p_k(\alpha_k) - s_k\|$ is bounded as in (4.24), (4.25) or (4.26). Recall that $p_k(\alpha_k)$ is the step found by minimizing along the piecewise linear reflective path. Thus we can achieve this bound by bounding the distance between breakpoints. Lemma 4.5.4 below gives us the bound on breakpoint separation,

SLIR: Algorithm 2

Choose $x_1 \in int(\mathcal{F})$.

For $k = 1, 2, ...,$

1. Solve $\hat{M}_k \hat{s}_k^{IN} = -\hat{g}_k = -D_k^{-1} g_k$ such that the appropriate residual bound (4.30), (4.31), or (4.32) is met. Set $s_k = D_k^{-1} \hat{s}_k^{IN}$.
2. Determine α_k s.t. $|\alpha_k - 1| = O(\|x_k - x_*\|)$ and $x_k + p_k(\alpha_k) \in int(\mathcal{F})$.
3. $x_{k+1} = x_k + p_k(\alpha_k)$.

Figure 4.3: A *local* inexact reflective Newton method.

and Theorem 4.5.5 uses this bound with Theorem 4.5.3 to give us local quadratic convergence for Algorithm 2.

The usual inexact Newton residual condition $\frac{\|\hat{r}\|}{\|\hat{g}\|} = O(\|\hat{g}\|)$ (omitting the iteration subscripts) used in unconstrained minimization for quadratic convergence is not sufficient to achieve the rates of convergence specified in Theorems 4.5.1 and 4.5.2. (This is true for the corresponding conditions for superlinear and linear convergence as well.) We need more component-wise control so that we make sufficient progress in each dimension. We need a stopping condition that implies all three of the following conditions (assume $\|\cdot\| \stackrel{\text{def}}{=} \|\cdot\|_2$):

- for quadratic convergence,

$$\frac{\|\hat{r}\|}{\|\hat{g}\|} = O(\|\hat{g}\|), \quad \frac{\|r\|}{\|g\|} = O(\|g\|), \quad \frac{\|r\|}{\|g\|} = O(\|\hat{g}\|)$$

- for superlinear convergence,

$$\|\hat{r}\| = o(\|\hat{g}\|), \quad \|r\| = o(\|g\|), \quad \|r\| = o(\|\hat{g}\|)$$

- for *linear convergence,

$$\|\hat{r}\| \leq \eta(\|\hat{g}\|), \quad \|r\| \leq \eta(\|g\|), \quad \|r\| \leq \eta(\|\hat{g}\|)$$

  where $0 < \eta < \frac{1}{2}$.

These conditions tell us that first, the step computed is an inexact Newton step in both the s_k and the $\hat{s}_k$ space. Also, that every component of the residual r is decreasing with $\|\hat{g}\|$, that is even for the components where v_j, but not g_j, is going to zero (that is, we are approaching a bound) the residual is shrinking at each step.

The following conditions are sufficient stopping criteria for our MPCG algorithm, that is, it is straightforward to show they imply the desired conditions above:

- quadratic convergence:

$$\frac{|r_j|}{|g_j|} = O(\min(|\hat{g}_j|,\ |g_j|\ )),\quad j = 1, ..., n, \tag{4.30}$$

- superlinear convergence:

$$|r_j| = o(\min(|\hat{g}_j|,\ |g_j|\ )),\quad j = 1, ..., n, \tag{4.31}$$

- *linear convergence:

$$|r_j| \leq \eta(\min(|\hat{g}_j|,\ |g_j|\ )),\quad j = 1, ..., n, \tag{4.32}$$

where the j subscripts refer to the components of the vectors. Note that $\frac{|r_j|}{|g_j|} = \frac{|\hat{r}_j|}{|\hat{g}_j|} = \frac{\sqrt{|v_j|}\ |r_j|}{\sqrt{|v_j|}\ |g_j|}$.

We can further support this new stopping criteria by pointing out that although

$$\frac{\|r\|}{\|g\|} = O(\|g\|) \not\Leftrightarrow \frac{\|\hat{r}\|}{\|\hat{g}\|} = O(\|\hat{g}\|),$$

it is easy to see that

$$\frac{|r_j|}{|g_j|} = O(\min(|\hat{g}_j|,\ |g_j|\ )) \iff \frac{|\hat{r}_j|}{|\hat{g}_j|} = O(\min(|\hat{g}_j|,\ |g_j|\ )).$$

Of course, we want this implication to be true so that an inexact Newton step in the scaled space implies an inexact Newton step in the original space, and vice versa.

For shorthand notation instead of $\frac{|r_j|}{|g_j|} = O(\min(|\hat{g}_j|,\ |g_j|)),\quad j = 1, ..., n$, we shall write:

$$|r_k| \cdot/ |g_k| \ \ \cdot= \ \ O(\min(|D_k^{-1} g_k|,\ |g_k|))$$

or

$$|r_k| \cdot/ |g_k| \ \ \cdot\leq \ \ \eta(\min(|D_k^{-1} g_k|,\ |g_k|))$$

so that we may include the iteration subscript. The $\cdot/$ represents component-wise division and $\cdot=$ ($\cdot\leq$) represents component-wise equality (inequality) between two vectors. Then the stopping conditions become

- quadratic convergence:
  $|r_k| \cdot/ |g_k| \ \ \cdot= \ \ O(\min(|D_k^{-1} g_k|,\ |g_k|))$

- superlinear convergence:
  $|r_k| \ \ \cdot= \ \ o(\min(|D_k^{-1} g_k|,\ |g_k|))$

- *linear convergence:
  $|r_k| \;\cdot\!\leq\; \eta(\min(|D_k^{-1}g_k|,\ |g_k|))$

for $0 < \eta < \frac{1}{2}$.

We now prove the bound on breakpoint separation.

Lemma 4.5.4 *Let x_* be a nondegenerate point satisfying second-order sufficiency conditions. Assume that $\nu(x)$ is chosen such that $\nu(x) = v(x)$. Let $s^{IN}(x)$ be the corresponding inexact Newton step, i.e.,*

$$s^{IN}(x) \;=\; (D^{-2}H + J^v D^g)^{-1}(-D^{-2}g + D^{-2}r) \tag{4.33}$$

where $g = g(x) = \nabla f(x)$, $H = H(x) = \nabla^2 f(x)$, $D^g = D^g(x) = \mathrm{diag}(g)$, $D^{-2} = D^{-2}(x) = \mathrm{diag}(|v(x)|)$, $J^v = J^v(x)$ is the diagonal Jacobian[1] matrix of v, and r is the residual of the inexact Newton method, i.e. $Ms = -g + r$ (or $\hat{M}\hat{s} = -\hat{g} + \hat{r}$ where $\hat{r} = D^{-1}r$). There exists an open neighborhood $\mathcal{C}$ containing x_* such that for all $x \in int(\mathcal{F}) \bigcap \mathcal{C}$, $s^{IN}(x)$ is well-defined and for each $j \notin Free_*$,

1. *If $|r_k| \cdot / |g_k| \;\cdot\!=\; O(\min(|D_k^{-1}g_k|,\ |g_k|))$ then*
$$|1 - \beta_j^{IN}(x)| = O(\|x_* - x\|)$$
2. *If $|r_k| \;\cdot\!=\; o(\min(|D_k^{-1}g_k|,\ |g_k|))$ then*
$$|1 - \beta_j^{IN}(x)| = O(\|x_* - x\|)$$
3. *If $|r_k| \;\cdot\!\leq\; \eta(\min(|D_k^{-1}g_k|,\ |g_k|))$ then*
$$|1 - \beta_j^{IN}(x)| = O(\|x_* - x\|_*)$$

where $\beta_j^{IN} = \frac{|v_j(x)|}{|s_j^{IN}(x)|}$.

Proof. Since x_* satisfies nondegeneracy and second-order sufficiency, it follows that the matrix $D^{-2}H + J^v D^g$ is nonsingular in a neighborhood of x_* and so $s^{IN}(x)$ is well-defined. From the definition of the inexact Newton step (4.33) it follows that

$$D^{-2}Hs^{IN}(x) + J^v D^g s^{IN}(x) = -D^{-2}g + D^{-2}r.$$

Then for $j \notin Free_*$,

$$|v_j|(Hs^{IN})_j + |g_j|s_j^{IN} = -|v_j| \cdot g_j + |v_j|r_j$$

[1] Matrix J^v is a diagonal matrix with each diagonal component equal to zero or unity. For example, if all the components of u and v are finite then $J^v = I$. If variable x_i has a finite lower bound and an infinite upper bound (or vice-versa) then strictly speaking v_i is not differentiable at a point $g_i = 0$; we define $J_{ii}^v = 0$ at such a point. Note that v_i is discontinuous at such a point but $v_i \cdot g_i$ is continuous.

As in Lemma 12 in [CL94], we derive that

$$1 - \left(\frac{|(Hs^{IN})_j|}{|g_j|} + \frac{|r_j|}{|g_j|} \right) \leq \frac{|s_j^{IN}|}{|v_j|} \leq 1 + \left(\frac{|(Hs^{IN})_j|}{|g_j|} + \frac{|r_j|}{|g_j|} \right) \tag{4.34}$$

and $Hs^{IN} \to 0$ as $x \to x_*$. But, by nondegeneracy and continuity, $|g_j|$ is bounded away from zero in a neighborhood of x_*; also H is bounded.

In the first case, $|r_k| \cdot / |g_k| \ \cdot = \ O(\min(|D_k^{-1} g_k|,\ |g_k|))$ which implies $\|r\| = O(\|g\|^2)$. In the second case, $|r_k| \ \cdot = \ o(\min(|D_k^{-1} g_k|,\ |g_k|))$ which implies $\|r\| = o(\|g\|)$. From this, using Theorem 4.5.2, it is straightforward to deduce that $\|s^{IN}\| = O(\|x - x_*\|)$. In the last case, $|r_k| \ \cdot \leq \ \eta(\min(|D_k^{-1} g_k|,\ |g_k|))$ which implies $\|r\| = \eta \|g\|$. Then using Theorem 4.5.1, it is straightforward to deduce that $\|s^{IN}\|_* = O(\|x - x_*\|_*)$. Also, all three cases imply $|r_k| = O(\|D^{-1} g\|)$ when $j \notin Free_*$. We know $\|D^{-1} g\| = O(\|x - x_*\|)$; therefore, from (4.34) it is easy to show that $|1 - \beta_j^{IN}| = O(\|x - x_*\|)$ (or $|1 - \beta_j^{IN}| = O(\|x - x_*\|_*)$ in the linear case). □

Theorem 4.5.5 *Let x_* be a nondegenerate point satisfying the second-order sufficiency conditions. Assume that $\{x_k\}$ is generated by Algorithm 2. Then, for $x_1 \in int(\mathcal{F})$ and sufficiently close to x_*, $\{x_k\} \in int(\mathcal{F})$ and $\{x_k\}$ converges to x_*.*

1. *If $|r_k| \cdot / |g_k| \ \cdot = \ O(\min(|D_k^{-1} g_k|,\ |g_k|))$ as $k \to \infty$ then $x_k \to x_*$ quadratically, $\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^2)$.*
2. *If $|r_k| \ \cdot = \ o(\min(|D_k^{-1} g_k|,\ |g_k|))$ then as $k \to \infty$ then $x_k \to x_*$ superlinearly, $\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)$ as $k \to \infty$.*
3. *If $|r_k| \ \cdot \leq \ \eta(\min(|D_k^{-1} g_k|,\ |g_k|))$ then as $k \to \infty$ then $x_k \to x_*$ *linearly*

$$\|x_{k+1} - x_*\|_* \leq t \|x_k - x_*\|_*. \tag{4.35}$$

Proof. We follow the proof of Theorem 13 in [CL94] using Theorem 4.5.3 and Lemma 4.5.4.

Let β_k^1 be the steplength to the first breakpoint along direction s_k. If $\alpha_k < \beta_k^1$ then $p_k(\alpha_k) = \alpha_k s_k$ where s_k is the inexact Newton step. However, $|\alpha_k - 1| = O(\|x_k - x_*\|)$ and so $\|p_k(\alpha_k) - s_k\| = \|s_k\| \cdot |\alpha_k - 1| = O(\|s_k\| \cdot \|x_k - x_*\|)$. Then since s_k is the inexact Newton step for some function in $\mathcal{F}$, Theorem 4.5.1 says that $\|s_k\| = O(\|x_k - x_*\|_*)$, which implies $\|p_k(\alpha_k) - s_k\| = O(\|x_k - x_*\|_*^2)$. And Theorem 4.5.2 says that $\|s_k\| = O(\|x_k - x_*\|)$, which implies $\|p_k(\alpha_k) - s_k\| = O(\|x_k - x_*\|^2)$. Then one of the cases of Theorem 4.5.3 holds and the result follows.

Assume that $\beta_k^{t_k} < \alpha_k < \beta_k^{t_k+1}$. From the definition of the reflective process, we can write

$$p_k(\alpha_k) - s_k = \sum_{i=2}^{t_k} (\beta_k^i - \beta_k^{i-1}) p_k^i + (\alpha_k - \beta_k^{t_k}) p_k^{t_k+1} + \beta_k^1 s_k - s_k.$$

Note that $p_k^{i+1} = D_k^{\sigma_{i+1}} s_k$ where $D_k^{\sigma_{i+1}}$ is a diagonal matrix with each diagonal entry equal to ± 1; therefore $\|p_k^{i+1}\| = O(\|s_k\|)$. Applying Lemma 4.5.4 and that $|\alpha_k - 1| = O(\|x_k - x_*\|)$,

$$\|p_k(\alpha_k) - s_k\| = O(\|s_k\| \cdot \|x_k - x_*\|).$$

Again, we can bound $\|s_k\|$ using either Theorem 4.5.1 or Theorem 4.5.2, then one of the cases of Theorem 4.5.3 holds and the result follows. □

We have an inexact local method with the desired convergence rate. We now specify our global algorithm, the Inexact Reflective Newton Algorithm (Algorithm 3), which is Algorithm 1 with the line search conditions further specified; in particular, we take steps with $\alpha_k = 1$ if this step size satisfies the line search conditions. Recall that if α_k is accepted by the line search but corresponds to a breakpoint, we choose $\alpha_k = \tilde{\alpha}_k = 1 - \epsilon_k$ such that $\tilde{\alpha}_k$ does not correspond to a breakpoint, $\tilde{\alpha}_k$ satisfies (3.16), and $\epsilon_k < \chi_\alpha \|D_k^{-1} g_k\|$ for some $\chi_\alpha > 0$. Thus the global convergence will follow from the global convergence of Algorithm 1 shown by Theorem 4.4.1.

SLIR: Algorithm 3

Choose $x_1 \in int(\mathcal{F})$.

For $k = 1, 2, \ldots,$

1. Determine initial descent direction s_k for f at x_k: Solve the system $\hat{M}_k \hat{s}_k = -\hat{g}_k$ for an inexact Newton step $\hat{s}_k^{IN}$. If $\hat{M}_k$ is positive definite and $\|\hat{s}_k^{IN}\| \leq \Delta_k$ then choose $s_k = D_k^{-1} \hat{s}_k^{IN}$. Otherwise, choose $\Delta_k \in [\Delta_L, \Delta_U]$, choose subspace $\mathcal{S}_k$, and solve (3.15) to get s_k.
2. Determine α_k: *If* no negative curvature was detected by the MPCG algorithm, and $x_k + p_k(1)$ satisfies (3.16), then set $\alpha_k = 1$; *otherwise*, perform an approximate piecewise line minimization of $f(x_k + p_k(\alpha))$, with respect to α, to determine α_k such that
   (a) α_k is not a breakpoint;
   (b) α_k satisfies (3.16) and (3.17).
3. $x_{k+1} = x_k + p_k(\alpha_k)$.

Figure 4.4: A *global* inexact Newton reflective path algorithm.

We need to show that in a neighborhood of x_* satisfying nondegeneracy and second-order sufficiency conditions, Algorithm 3 reduces to the local method Algorithm 2. To do this, we show that in a neighborhood of x_*, an inexact Newton step will satisfy (3.16), that is, for Algorithm 2, α_k goes to 1. From this we can conclude global and quadratic (superlinear, *linear) convergence of Algorithm 3.

Theorem 4.5.6 *Assume x_* is a nondegenerate point satisfying second-order sufficiency conditions and τ_ϵ is sufficiently small* [2]. *Let $0 < \sigma_L < \frac{1}{2}$. Suppose $\{x_k\}$ is generated by Algorithm 2, and that the inexact Newton step satisfies*

$$\lim_{k\to\infty} \frac{\hat{s}_k^T(\hat{g}_k + \hat{M}_k\hat{s}_k)}{\hat{s}_k^T\hat{s}_k} = 0. \tag{4.36}$$

Then for x_1 sufficiently close to x_* and k sufficiently large,

$$f(x_k + p_k(\alpha_k)) < f(x_k) + \sigma_L(g_k^T s_k + \frac{1}{2}\min(s_k^T H_k s_k, 0)). \tag{4.37}$$

Proof. Suppose there are $t_k - 1$ breakpoints $b_1, b_2, ..., b_{t_k-1}$, to the left of α_k, corresponding to step lengths $\beta_k^1, \beta_k^2, ..., \beta_k^{t_k-1}$. For notational simplicity let us label $p_k(\alpha_k)$ with $b_k^{t_k}$.

As in Theorem 14 in [CL94], we may derive from Lemma 4.5.4 that

$$f(x_k + p_k(\alpha_k)) - f_k = f(b_k^1) - f_k + \sum_{i=1}^{t_k-1}[f(b_k^{i+1}) - f(b_k^i)] \tag{4.38}$$

and

$$f(b_k^1) - f_k \quad = \quad g_k^T s_k + \frac{1}{2}s_k^T H_k s_k + o(|g_k^T s_k|) + o(\|s_k\|^2).$$

and

$$f(b_k^{i+1}) - f(b_k^i) \quad = \quad (\beta_k^{i+1} - \beta_k^i)g_k^T p_k^{i+1} + o(\|s_k\|^2).$$

The most difficult term to deal with is $(\beta_k^{i+1} - \beta_k^i)g_k^T p_k^{i+1}$; however, we can show that this term is $o(-g_k^T s_k)$ and this leads the way to the final result. To show this we use the fact that, due to second-order sufficiency, there exists $\mu > 0$ such that for all k sufficiently large,

$$s_k^T M_k s_k \geq \mu\|s_k\|^2, \tag{4.39}$$

and

$$\hat{s}_k^T \hat{M}_k \hat{s}_k \geq \mu\|\hat{s}_k\|^2. \tag{4.40}$$

But since we assume (4.36), there exists $\delta > 0$ and k_0 such that for all $k \geq k_0$

$$|\hat{s}_k^T(\hat{g}_k + \hat{M}_k\hat{s}_k)| \leq \delta\|\hat{s}_k\|^2.$$

[2] τ_ϵ is used in the definition of the extended gradient (3.12).

Then

$$\begin{aligned} g_k^T s_k &= \hat{s}_k^T \hat{g}_k & (4.41)\\ &= \hat{s}_k^T(\hat{g}_k + \hat{M}_k \hat{s}_k) - \hat{s}_k^T \hat{M}_k \hat{s}_k & (4.42)\\ &\leq \delta\|\hat{s}_k\|^2 - \mu\|\hat{s}_k\|^2 & (4.43)\\ &= -(\mu - \delta)\|\hat{s}_k\|^2. & (4.44) \end{aligned}$$

Thus

$$\|\hat{s}_k\|^2 = O(-g_k^T s_k). \tag{4.45}$$

Furthermore,

$$\|s_k\|^2 \leq \|D_k^{-1}\|^2 \cdot \|\hat{s}_k\|^2 = O(-g_k^T s_k). \tag{4.46}$$

But $p_k^{i+1} = D_k^{\sigma_{i+1}} s_k$ where $D_k^{\sigma_{i+1}}$ is a diagonal matrix with each diagonal element equal to ± 1. Hence, using the boundedness of $\{\hat{M}_k\}$ and that $\|\hat{r}_k\| = O(\|\hat{g}_k\|)$ (for all three cases)

$$\begin{aligned} |-g_k^T p_k^{i+1}| &= |s_k^T D_k^{\sigma_{i+1}}(M_k s_k + r_k)|\\ &= |\hat{s}_k^T D_k^{\sigma_{i+1}} \hat{M}_k \hat{s}_k + \hat{s}_k^T D_k^{\sigma_{i+1}} \hat{r}_k|\\ &= O(\|\hat{s}_k\|^2) + O(\|\hat{s}_k\| \cdot \|\hat{g}_k\|) \end{aligned}$$

Therefore by applying Lemma 4.5.4 and (4.45) we have,

$$\begin{aligned} |\beta_k^{i+1} - \beta_k^i| \cdot |g_k^T p_k^{i+1}| &= O(\|s_k\|)\ |g_k^T p_k^{i+1}|\\ &= O(\|s_k\| \cdot \|\hat{s}_k\|^2) + O(\|s_k\| \cdot \|\hat{s}_k\| \cdot \|\hat{g}_k\|)\\ &= O(\|s_k\| \cdot \|\hat{s}_k\|^2) + O(\|\hat{s}_k\|^2 \cdot \|\hat{g}_k\|)\\ &= O(\|s_k\| \cdot (-g_k^T s_k)) + O((-g_k^T s_k) \cdot \|\hat{g}_k\|)\\ &= o(-g_k^T s_k) \end{aligned}$$

Collecting together the terms above, (4.38) becomes

$$f(x_k + p_k(\alpha_k)) - f(x_k) = g_k^T s_k + \frac{1}{2} s_k^T H_k s_k + o(|g_k^T s_k|) + o(\|s_k\|^2).$$

From here we follow the proof of Theorem 14 in [CL94] to conclude

$$f(x_k + p_k(\alpha_k)) - f(x_k) < \sigma_L(s_k^T g_k + \frac{1}{2}\min(s_k^T H_k s_k, 0)).$$

□

The above proof does not depend specifically on the MPCG algorithm, only on the condition (4.36). Thus the theorem holds for any iterative method applied to the Newton equations as long as (4.36) holds. Note that this condition is trivially true for MPCG since by relation (4.16) in Theorem 4.1.1 we know

$$0 = \hat{s}_k^T \hat{r}_k = \hat{s}_k^T(\hat{g}_k + \hat{M}_k \hat{s}_k).$$

A similar condition is used in Theorem A.4 in [DS83] for inexact Newton methods with line search conditions in the case of unconstrained minimization.

Theorem 4.5.7 *Assume $\{x_k\}$ is generated by Algorithm 3 and τ_ϵ is sufficiently small. Let $\{s_k\}$ satisfy large-step-size and consistency. Then,*

- *Every limit point of $\{x_k\}$ is a first-order point.*
- *If a nondegenerate limit point x_* satisfies second-order sufficiency conditions then, provided τ_ϵ is sufficiently small, $\{x_k\}$ is convergent to x_*.*

  1. *If $|r_k| \cdot / |g_k| \quad \cdot = \quad O(\min(|D_k^{-1} g_k|,\ |g_k|))$ as $k \to \infty$ then $x_k \to x_*$ quadratically, $\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^2)$.*
  2. *If $|r_k| \quad \cdot = \quad o(\min(|D_k^{-1} g_k|,\ |g_k|))$ then as $k \to \infty$ then $x_k \to x_*$ superlinearly, $\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)$ as $k \to \infty$.*
  3. *If $|r_k| \quad \cdot \leq \quad \eta(\min(|D_k^{-1} g_k|,\ |g_k|))$ then as $k \to \infty$ then $x_k \to x_*$ *linearly,*

$$\|x_{k+1} - x_*\|_* \leq t \|x_k - x_*\|_*. \tag{4.47}$$

Proof. As in the proof of Theorem 15 in [CL94], we use Theorems 4.4.1 and 4.5.6. By Theorem 4.4.1, every limit point satisfies first-order necessary conditions. Let x_* be a limit point satisfying nondegeneracy and second-order sufficiency conditions. By Theorem 4.5.6, a unit step size [3], for some constant $\chi_\alpha > 0$, will satisfy (3.16) for $\|x_k - x_*\|$ sufficiently small. Therefore, for $\|x_k - x_*\|$ sufficiently small, Algorithm 3 reduces to Algorithm 2. Then one of the cases in Theorem 4.5.5 holds and the corresponding convergence rate follows. □

Again, from [CL94] we have the following lemma.

Lemma 4.5.8 *Assume $\{x_k\}$ is generated by Algorithm 3 in Fig. 4.4, with $\{s_k\}$ calculated by the subspace descent direction procedure in Fig. 4.2. Then,*

- *the subsequence $\{\|D_k^{-1}\mathrm{sgn}(g_k)\| : \lambda_{\min}(\hat{M}_k) < 0\}$ is bounded away from zero.*
- *the subsequence $\{z_k = D_k^{-1}\hat{z}_k : \lambda_{\min}(\hat{M}_k) < 0\}$ has large-step size, where $\hat{z}_k = D_k^{-1}\mathrm{sgn}(g_k)/\|D_k^{-1}\mathrm{sgn}(g_k)\|$.*

Theorem 4.5.9 *Assume $\{x_k\}$ is generated by Algorithm 3 in Fig. 4.4, with $\{s_k\}$ calculated by the subspace descent direction procedure in Fig. 4.2. Then every subsequence $\{s_k\}$ satisfies the consistency condition. Moreover, for any subsequence, if either $\{\|D_k^{-1} g_k\|\}$ or $\{\max(0,\ \lambda_{\min}(\hat{M}_k))\}$ is bounded away from zero, then the corresponding subsequence $\{s_k\}$ has large-step-size.*

---

[3]If $\alpha_k = 1$ corresponds to a breakpoint then $\alpha_k = \tilde{\alpha}_k = 1 - \epsilon_k$ where $\tilde{\alpha}_k$ is not a breakpoint, $\tilde{\alpha}_k$ satisfies (3.16), and $\epsilon_k < \chi_\alpha \|D_k^{-1} g_k\|$

Proof. We have already shown that every step computed by the algorithm shown in Figure 4.2 has large-step-size and is consistent in Section 4.3. The rest follows as in Theorem 20 in [CL94]. □

Theorem 4.5.10 *Assume $\{x_k\}$ is generated by Algorithm 3 and $\{s_k\}$ calculated by the subspace descent direction procedure in Fig. 4.2. Then,*

- *Every limit point of $\{x_k\}$ is a first-order point.*
- *If a nondegenerate limit point x_* satisfies second-order sufficiency conditions, then, provided τ_ϵ is sufficiently small, $\{x_k\}$ is convergent to x_*.*

  1. *If $|r_k| \cdot / |g_k| \quad \cdot = \quad O(\min(|D_k^{-1} g_k|,\ |g_k|))$ as $k \to \infty$ then $x_k \to x_*$ quadratically, $\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^2)$.*
  2. *If $|r_k| \quad \cdot = \quad o(\min(|D_k^{-1} g_k|,\ |g_k|))$ then as $k \to \infty$ then $x_k \to x_*$ superlinearly, $\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)$ as $k \to \infty$.*
  3. *If $|r_k| \quad \cdot \leq \quad \eta(\min(|D_k^{-1} g_k|,\ |g_k|))$ then as $k \to \infty$ then $x_k \to x_*$ *linearly,*

$$\|x_{k+1} - x_*\|_* \leq t\|x_k - x_*\|_*. \tag{4.48}$$

Proof. From the previous Theorem 4.5.9 and Theorem 4.5.7, the result follows as in Theorem 21 in [CL94]. □

4.6 Second-order convergence

For second-order convergence (every nondegenerate limit point satisfies the second-order necessary conditions), we need that the negative curvature directions are sufficiently negative. Although we may not be able to guarantee this holds for the negative curvature directions generated by the MPCG algorithm, we can note that when these directions are sufficiently negative, second order convergence occurs and the convergence is as specified above (that is, there are no other unsatisfied conditions). The specific condition that does not seem provably satisfiable is: choose Y_k as a matrix with orthonormal columns such that, when $\lambda_{\min}(\hat{M}_k) \leq 0$,

$$\lambda_{\min}(Y_k^T \hat{M}_k Y_k) \leq \max(-\epsilon_{nc}, \tau\lambda_{\min}(\hat{M}_k)) \tag{4.49}$$

for some $\epsilon_{nc} > 0$, $\tau > 0$, and choose $\mathcal{S}_k = \text{span}\{Y_k\}$.

Chapter 5

Inexact STIR: Convergence results

In this chapter we show that the inexact STIR method achieves first-order convergence and we establish the rate of convergence. The method is shown in Figure 3.6 with the subspace determination described in Figure 5.1.

Determine Subspace $\mathcal{S}_k$: [Assume that $w_k = D_k^{-1}\hat{w}_k$ where $\{w_k\}$ has large-step-size. Let $0 < \tau < 1$ be a small positive constant.]

if $\hat{M}_k$ is positive definite
  $\mathcal{S}_k \stackrel{\text{def}}{=} \text{span}\{D_k^{-2}g_k, w_k\}$
else $\hat{M}_k$ is not positive definite
  if $(D_k^{-2}\text{sgn}(g_k))^T M_k (D_k^{-2}\text{sgn}(g_k)) < \tau \frac{\|D_k^{-2}g_k\|^2}{\|w_k\|^2} w_k^T M_k w_k$
    $\mathcal{S}_k \stackrel{\text{def}}{=} \text{span}\{D_k^{-2}\text{sgn}(g_k)\}$
  else
    $\mathcal{S}_k \stackrel{\text{def}}{=} \text{span}\{D_k^{-2}\text{sgn}(g_k), w_k\}$
  end
end

Figure 5.1: The STIR method: determining a subspace.

5.1 First-order convergence

We need the following two technical lemmas; the first is Lemma 3.1 in [CLa]. The second is Lemma (4.10) in [MS83].

Lemma 5.1.1 *Assume that the compactness and smoothness conditions (AS.1) and (AS.2) are satisfied. If s_k satisfies (STIR.1) then*

$$-\psi_k(s_k) \geq \tfrac{1}{2}\beta\|\hat{g}_k\| \min\{\Delta_k, \frac{\|\hat{g}_k\|}{\|\hat{M}_k\|}, \frac{\|\hat{g}_k\|}{\|g_k\|_\infty}\}.$$

Lemma 5.1.2 *Let x_* be an isolated limit point of a sequence $\{x_k\}$ in $\Re^n$. If $\{x_k\}$ does not converge then there is a subsequence $\{x_{k_j}\}$ which converges to x_* and an $\epsilon > 0$ such that $\|x_{k_{j+1}} - x_{k_j}\| \geq \epsilon$.*

Now we can state the convergence result. We call an iteration *successful* if $\rho_k > \mu$, which implies $x_{k+1} = x_k + s_k$ (see STIR Algorithm 1 in Figure 3.6). Otherwise, the iteration is *unsuccessful.*

Theorem 5.1.3 *We assume the compactness and smoothness assumptions (AS.1) and (AS.2). Let $\{x_k\}$ be the sequence generated by the STIR algorithm in Figure 3.6 and Figure 5.1 using MPCG to define w_k. Assume (STIR.1) is satisfied. Then*
A. *The sequence $\{\hat{g}_k\}$ converges to zero.*
B. *If $\hat{M}(x_*)$ is positive definite for some limit point x_* of $\{x_k\}$, then $\{x_k\}$ converges to x_*, all iterations are eventually successful, and $\{\Delta_k\}$ is bounded away from zero.*

Proof.
A. This follows from Theorem 3.10 in [CLa].
B. This is similar to Theorem 3.11 in [CLa] and Theorem 4.19 in [Mor83], but is included for completeness.

We know that $\hat{M}(x_*)$ is positive definite and that $\hat{g}_* = 0$. Since x_* is a limit point of $\{x_k\}$ and $\hat{M}(x_*)$ is nonsingular we know that x_* is an *isolated* limit point. By Lemma 5.1.2, if $\{x_k\}$ does not converge there is a convergent subsequence $\{x_{k_j}\} \rightarrow x_*$ such that $\|x_{k_{j+1}} - x_{k_j}\| \geq \epsilon$ for some $\epsilon > 0$.

We may define $B^*(\delta) = \{x : \|x - x_*\| \leq \delta\}$ for some $\delta > 0$ such that $\hat{M}(x)$ is positive definite for all $x \in B^*(\delta)$, and let $\hat{M}_k = \hat{R}_k^T \hat{R}_k$. Without loss of generality, assume $\delta \leq \epsilon$. If $\|x_k - x_*\| \leq \delta$ then because $\hat{\psi}_k(\hat{s}_k) = \frac{1}{2}\hat{s}_k^T \hat{M}_k \hat{s}_k + \hat{s}_k^T \hat{g}_k \leq 0$ and $\hat{M}_k$ is positive definite, we have

$$\tfrac{1}{2}\|\hat{s}_k^T \hat{R}_k^T \hat{R}_k \hat{s}_k\| = \tfrac{1}{2}\|\hat{R}_k \hat{s}_k\|^2 \leq \|\hat{s}_k\| \cdot \|\hat{g}_k\|.$$

Then

$$\tfrac{1}{2}\|\hat{s}_k\|^2 \leq \|\hat{R}_k^{-1}\|^2 \cdot \tfrac{1}{2}\|\hat{R}_k \hat{s}_k\|^2 \leq \|\hat{R}_k^{-1}\|^2 \|\hat{s}_k\| \cdot \|\hat{g}_k\|,$$

and so

$$\tfrac{1}{2}\|D_k s_k\| \leq \|\hat{R}_k^{-1}\|^2 \cdot \|\hat{g}_k\|.$$

Then

$$\tfrac{1}{2}\|s_k\| \le \tfrac{1}{2}\|D_k^{-1}\| \cdot \|D_k s_k\| \le \chi_D \|\hat{R}_k^{-1}\|^2 \cdot \|\hat{g}_k\|$$

and there exists ϵ_1 such that

$$\epsilon_1 \|s_k\| \le \|\hat{g}_k\|.$$

But $\{\hat{g}_k\}$ converges to zero by A above.

Let

$$\|D_k^{-1} g_k\| \le \tfrac{1}{2}\epsilon_1 \delta$$

for $k \ge k_1$, so that

$$\|x_{k+1} - x_k\| = \|s_k\| \le \tfrac{1}{2}\delta.$$

But this contradicts the existence of the subsequence $\{x_{k_j}\}$ above. Hence $\{x_k\}$ converges to x_*.

We show that all iterations are eventually successful. By assumption (STIR.1), $\|D_k s_k\| \le \beta_0 \Delta_k$. Then

$$\|s_k\| = \|D_k^{-1} D_k s_k\| \le \|D_k^{-1}\| \beta_0 \Delta_k = \chi_D \beta_0 \Delta_k$$

because $\|D_k^{-1}\| \le \chi_D$ (see Section 3.1).

Then we have from Lemma 5.1.1 that

$$\begin{aligned}
-\psi_k(s_k) &\ge \tfrac{1}{2}\beta\|\hat{g}_k\| \min\{\Delta_k, \frac{\|\hat{g}_k\|}{\|\hat{M}_k\|}, \frac{\|\hat{g}_k\|}{\|g_k\|_\infty}\} && (5.1)\\
&\ge \tfrac{1}{2}\beta\epsilon_1\|s_k\| \min\{\Delta_k, \frac{\epsilon_1\|s_k\|}{\|\hat{M}_k\|}, \frac{\epsilon_1\|s_k\|}{\|g_k\|_\infty}\} && (5.2)\\
&\ge \tfrac{1}{2}\beta\epsilon_1\|s_k\| \min\{\frac{\|s_k\|}{\chi_D\beta_0}, \frac{\epsilon_1\|s_k\|}{\|\hat{M}_k\|}, \frac{\epsilon_1\|s_k\|}{\|g_k\|_\infty}\} && (5.3)\\
&\ge \tfrac{1}{2}\beta\epsilon_1\|s_k\| \min\{\frac{\|s_k\|}{\chi_D\beta_0}, \frac{\epsilon_1\|s_k\|}{\chi_{\hat{M}}}, \frac{\epsilon_1\|s_k\|}{\chi_g}\} && (5.4)\\
&\ge \epsilon_2\|s_k\|_2^2 && (5.5)\\
&\ge 0 && (5.6)
\end{aligned}$$

for some $\epsilon_2 > 0$. Then $\frac{1}{\psi_k(s_k)}\|s_k\|^2 \ge -\frac{1}{\epsilon_2}$.

By Taylor's theorem, it follows that

$$|f(x_k+s_k) - f(x_k) + \tfrac{1}{2}s^T C_k^+ s_k - \psi_k(s_k)| \le \tfrac{1}{2}\|s_k\|^2 \max_{0\le\tau\le 1} \|\nabla^2 f(x_k+\tau s_k) - \nabla^2 f(x_k)\|.$$

The last two inequalities imply

$$\begin{aligned}
|\rho_k - 1| &\le \tfrac{1}{2}\frac{1}{|\psi_k(s_k)|}\|s_k\|^2 \max_{0\le\tau\le 1} \|\nabla^2 f(x_k+\tau s_k) - \nabla^2 f(x_k)\| && (5.7)\\
&\le \tfrac{1}{2}\frac{1}{\epsilon_2} \max_{0\le\tau\le 1} \|\nabla^2 f(x_k+\tau s_k) - \nabla^2 f(x_k)\|. && (5.8)
\end{aligned}$$

We know that $\{x_k\}$ is bounded, $\nabla^2 f$ is continuous, and $x_k \to x_*$, so $|\rho_k - 1| \to 0$. Thus all iterations are eventually successful.

Finally, the update procedure for Δ_k (see Appendix C) shows that $\{\Delta_k\}$ is bounded away from zero. $\square$

5.2 Rate of convergence

As in the SLIR method, we show that the rate of convergence is dependent upon the residual stopping criteria of the inexact Newton step in the MPCG algorithm.

The next lemma establishes that if M_k is positive definite and the inexact Newton step is within the trust-region bound, our solution to the trust-region subproblem (restricted to the two-dimensional subspace) is the inexact Newton step. We define s_k^{IN} as the inexact Newton step that solves

$$M_k s_k^{IN} = -g_k + r_k$$

for r_k that satisfies one of the residual stopping conditions (4.30), (4.31), (4.32). We assume $\hat{s}_k^{IN} = D_k s_k^{IN}$ is computed using the MPCG algorithm with the preconditioner $P_k = I$, where I is the identity matrix.

Lemma 5.2.1 *Assume $\|D_k s_k^{IN}\| < \Delta_k$, and that M_k is positive definite. If $s_k^{TR} \stackrel{\text{def}}{=} \text{argmin}\{\psi_k(s) = s^T g_k + \frac{1}{2} s^T M_k s : \|D_k s\|_2 \leq \Delta_k,\ s \in \mathcal{S}_k\}$ where $\mathcal{S}_k = \{D_k^{-2} g_k, s_k^{IN}\}$, then $s_k^{TR} = s_k^{IN}$.*

Proof.

By the definition of s_k^{TR}, we know that $\psi(s_k^{TR}) \leq \psi(s_k^{IN})$ since $\|D_k s_k^{IN}\| < \Delta_k$. Suppose that $\psi(s_k^{TR}) < \psi(s_k^{IN})$. Then this implies that $\hat{\psi}(\hat{s}_k^{TR}) < \hat{\psi}(\hat{s}_k^{IN})$. From (4.8) in Lemma 4.1.1 we have that $\hat{s}_k^{IN} = \alpha_1 d_1 + \ldots + \alpha_m d_m$ where $d_1 = \hat{g}_k$. Furthermore, for some α and some β

$$\begin{aligned} \hat{s}_k^{TR} &= \alpha \hat{g}_k + \beta \hat{s}_k^{IN} && (5.9) \\ &= \alpha \hat{g}_k + \beta \alpha_1 d_1 + \beta \sum_{i=2}^{m} \alpha_i d_i && (5.10) \\ &= (\alpha + \beta \alpha_1) d_1 + \beta \sum_{i=2}^{m} \alpha_i d_i. && (5.11) \end{aligned}$$

Thus we have that $\hat{s}_k^{TR}$ is a linear combination of the directions $\{d_1, \ldots, d_m\}$, or $\mathcal{S}_k \subseteq \{d_1, \ldots, d_m\}$.

But it is also true that $\hat{s}_k^{IN}$ is the minimizer of $\hat{\psi}$ over the subspace $\{d_1, \ldots, d_m\}$. Thus $\hat{\psi}(\hat{s}_k^{TR}) \geq \hat{\psi}(\hat{s}_k^{IN})$, a contradiction. This implies that $\hat{\psi}(\hat{s}_k^{TR}) = \hat{\psi}(\hat{s}_k^{IN})$. And since $\hat{s}_k^{IN}$ is the unique minimizer of $\hat{\psi}_k$ (since by assumption $\hat{\psi}_k$ is strictly convex), we have that $\hat{s}_k^{TR} = \hat{s}_k^{IN}$. Equivalently, we have that $s_k^{IN} = s_k^{TR}$.

□

We end by establishing the convergence rate for STIR.

Theorem 5.2.2 *We assume the compactness and smoothness assumptions (AS.1) and (AS.2). Let $\{x_k\}$ be the sequence generated by the STIR algorithm in Figure 3.6 and Figure 5.1 using MPCG to define w_k, implementing the appropriate stopping criteria (bound on the residual) as in (4.30), (4.31), or (4.32). Assume (STIR.1) is satisfied and $0 < \beta < 1$.*

*If $\hat{M}(x_*)$ is positive definite for some limit point x_* of $\{x_k\}$, then $\{x_k\}$ converges to x_* and the convergence is quadratic, superlinear, or *linear (depending on which of the stopping criteria (4.30), (4.31), or (4.32), respectively, is used in MPCG).*

Proof. We know from Theorem 5.1.3 that $\{\Delta_k\}$ is bounded away from zero, $\{D_k^{-1}g_k\}$ converges to zero, and that $\{x_k\}$ converges to x_*. Then $\{D_k s_k^{IN}\}$ converges to zero where s^{IN} is the inexact Newton step.

For some k_1, if $k \geq k_1$ then we know M_k is positive definite and also $\|D_k s_k^{IN}\| < \Delta_k$. Then by Lemma 5.2.1, $s_k^{TR} = s_k^{IN}$ and so $\alpha^*[s_k^{TR}] = \alpha^*[s_k^{IN}] = \theta_k \tau_k^* s_k^{IN}$.

We need to show that $\alpha^*[s_k^{IN}]$ satisfies (STIR.1). Since we know that $\|D_k s_k^{IN}\| < \Delta_k$ and s_k^{IN} is the minimizer of the subspace trust-region subproblem, then $\tau_k^* = 1$ unless $x_k + s_k^{IN}$ is not feasible. Then we have to scale back to be feasible, that is, τ_k^* may be less than unity to scale the direction back to the first breakpoint. But from Lemma 4.5.4 we know that $|\tau_k^* - 1| = O(\|x_k - x_*\|)$, that is, the distance to the breakpoints is going to unity. Also, from the definition (3.22), $|\theta_k - 1| = O(\|s_k^{IN}\|)$. Since $0 < \beta < 1$, for large enough k we have that $\beta < \theta_k \tau_k^* \leq 1$. Then $0 \leq \theta_k^2 \tau_k^{*2} \leq \theta_k \tau_k^*$. For sufficiently large k, we have from definition (3.23),

$$
\begin{aligned}
\psi_k(\alpha_k^*[s_k^{IN}]) &= \theta_k \tau_k^* g_k^T s_k^{IN} + \theta_k^2 \tau_k^{*2} \tfrac{1}{2} (s_k^{IN})^T M_k s_k^{IN} && (5.12) \\
&\leq \theta_k \tau_k^* (g_k^T s_k^{IN} + \tfrac{1}{2} (s_k^{IN})^T M_k s_k^{IN}) && (5.13) \\
&\leq \beta(\psi_k(s_k^{IN})) && (5.14) \\
&\leq \beta(\psi_k^*[-D_k^{-2} g_k]) && (5.15)
\end{aligned}
$$

and so we know that (STIR.1) is satisfied for sufficiently large k, and so $s_k = \alpha_k^*[s_k^{IN}]$.

Then by the definition (3.23),

$$
\begin{aligned}
s_k - s_k^{IN} &= \alpha_k^*[s_k^{IN}] - s_k^{IN} && (5.16) \\
&= \tau_k^* \theta_k s_k^{IN} - \tau_k^* s_k^{IN} + \tau_k^* s_k^{IN} - s_k^{IN} && (5.17) \\
&= |\theta_k - 1|(\tau_k^* s_k^{IN}) + |\tau_k^* - 1|(s_k^{IN}). && (5.18)
\end{aligned}
$$

where $\tau_k^* = \min\{1, \alpha_k\}$. But $|\tau_k^* - 1| = O(\|x_k - x_*\|)$ and $|\theta_k - 1| = O(\|s_k^{IN}\|)$. Then $\|s_k^{IN}\| = O\|x_k - x_*\|$. Hence $\|s_k - s_k^{IN}\| = O(\|x_k - x_*\|^2)$.

By Theorem 4.5.3 $\{x_k\}$ converges quadratically (or as the residual stopping condition predicts) to x_*. □

To simplify the presentation, we assumed that our MPCG preconditioner $P_k = I$ where I is the identity matrix. We do not need to make this assumption. If we assume $\{P_k\}$ and $\{P_k^{-1}\}$ are uniformly bounded, then the results in this chapter can be shown as in [Mor83].

Chapter 6

SLIR: Computational results for quadratic problems

We discuss the numerical performance of the inexact SLIR method solving box-constrained quadratic problems which are strictly convex. We compare the inexact method to the exact SLIR method from which it is derived. Our computational results show that the inexact method compares favorably in terms of flops, particularly for large matrices and as density increases. It is comparable to the original method in terms of the number of iterations. As for accuracy, it does as well except for some ill-conditioned problems.

6.1 Implementation

In this chapter we investigate two variations of the SLIR method. Both the exact and the inexact methods follow the same basic steps shown in Figure 3.4. The methods differ in how Step 1 is implemented. Our implementation of the (exact) SLIR method uses a Cholesky factorization to find the Newton direction $s^N = -M^{-1}g$ (as opposed to finding the inexact Newton direction s^{IN}) from which part of the subspace $\mathcal{S}$ is constructed. To improve the likelihood of generating sparse factors, a symmetric minimum degree ordering is precomputed on the Hessian. The Cholesky factorization may be prohibitedly expensive, taking from 50 to 90 percent of the total time needed to solve (1.1). The inexact Newton version, as described in Chapters 3 and 4, allows us to solve the Newton equations only approximately and thus reduce our overall time. We refer to the (exact) SLIR method as SLIR. Our inexact version employs a preconditioned conjugate gradient algorithm to solve the linear systems to find s^{IN}, and so we call it CG-SLIR.

We refer to the steps in Figure 3.4 as the *outer* or *major* iterations. For CG-SLIR, we call the iterations of the preconditioned conjugate gradient algorithm the *inner* or *minor* iterations.

These results show that SLIR and CG-SLIR are comparable in terms of major iteration count. In terms of megaflops, CG-SLIR is as fast and often much faster than SLIR. For large problems, CG-SLIR may run up to 250 times faster than SLIR. On well-conditioned problems, the relative error [1] for both algorithms is comparable to machine epsilon. For ill-conditioned problems, SLIR retains this accuracy, but CG-SLIR may only attain half as many accurate digits. Some further experimentation suggests, however, that this accuracy may be improved with only some degradation in the speed.

6.1.1 Inexact Newton step

We compute our inexact Newton step using the MPCG algorithm given in Figure 4.1 with $\eta = 0.1$ for the residual bound. Also, we terminate if we exceed $n/2$ iterations as this suggests a loss of orthogonality.

6.1.2 Preconditioning

A preconditioner for the linear conjugate gradient method is used to reduce the number of distinct eigenvalues of the matrix $\hat{M}$, reduce the condition number of the $\hat{M}$, or both. The preconditioning matrix must be positive definite, and linear systems involving it should be "easy" to solve. We choose to precondition the matrix $\hat{M}$ with a diagonal scaling matrix constructed from the diagonal of $\hat{M}$ as described in Section 4.2. Such a preconditioner may improve the clustering of $\hat{M}$'s eigenvalues. Also, from an optimization viewpoint it is clear why this scaling is valuable. Although the Newton direction is scale-invariant, the steepest descent direction is not [DS83]. Because the linear-conjugate gradient algorithm chooses a multiple of the steepest descent direction in its first iteration, if the problem is badly scaled this can be a particularly poor choice for the first conjugate direction.

Our experience has been that although the scaling may have no noticeable effect, it seems to reduce most inner (MPCG) iterations down to 6, sometimes down to 2. On ill-conditioned problems, we have seen the number of inner iterations drop from 581 to 2 just by adding diagonal scaling.

6.2 Computational results

Four types of problems were tested. The performance of SLIR and CG-SLIR on these problems is evaluated in terms of the number of major iterations, the total number of flops, and the accuracy of the computed solution.

[1] relative error is measured by $(q(x^*) - q(x))/\max(|q(x^*)|, 1)$, where x is the computed solution and x^* is the optimal solution

6.2.1 QPGEN

The first type, which we refer to as QPGEN, consists of randomly generated, box-constrained, strictly convex quadratic programming problems.

We created seven sets of problems of type QPGEN by varying the exponent of the condition number of the Hessian matrix (`CONDN`), the percentage of variables at their bounds at the solution (`PCTBND`), and the percentage of variables with their upper bounds set to infinity (`PCTINF`). The Matlab [GMS92] function `sprandsym` was used to generate the Hessian matrices. The parameter combinations are shown in Table 6.1. For each set of parameter combinations, we generated problems of size $n = 400$, 900, 1600, and 2500. For each n we varied the density of the Hessian matrix. The density was set to $frac/n$, where $frac$ took on the values 1, 5, 10, and 15.

The results for six of the seven sets of problems are hardly distinguishable. The results for QPGEN1 apply to QPGEN3, QPGEN4, QPGEN5, QPGEN6, and QPGEN7 as well. Thus we restrict our remarks to the sets QPGEN1 and QPGEN2.

For QPGEN1, the accuracy of CG-SLIR is comparable to that of SLIR (see Table 6.2), both on the order of machine epsilon. Likewise, the number of major iterations is comparable for both algorithms (see Table 6.2). These results are important as some of the desirable characteristics of SLIR is its high accuracy and low major iteration count. Table 6.3 shows the megaflops for SLIR and CG-SLIR. The last column shows the speedup, that is

$$speedup \;=\; \frac{\text{SLIR} \quad \text{Mflops}}{\text{CG-SLIR} \quad \text{Mflops}}. \tag{6.1}$$

The speedup is also graphed in Figure 6.1. For all but the most sparse problems, CG-SLIR is a win over SLIR, particularly as size and density increase.

Now we consider the QPGEN2 test set. These problems have condition number approximately 1.0×10^6. For most of these problems we have an error on the order of 1.0×10^{-8} for CG-SLIR and 1.0×10^{-16} for SLIR (Table 6.4). The number of major iterations for CG-SLIR is slightly higher than for SLIR, but still often the same or better (Table 6.4). The speedup is approximately half of what we achieved for well-conditioned problems, but still good (Table 6.5 and Figure 6.2). As before, speedup improves as size and density increase. Varying `PCTBND` and `PCTINF` did not have a noticeable effect on the results.

Further experimentation was done to see if the reduced accuracy for the QP-GEN2 problems occurred even when the MPCG algorithm was solved with a tighter bound on the residual. We reran CG-SLIR on the QPGEN2 problems, this time with a smaller η in the stopping criteria for MPCG. Instead of $\eta = 0.1$, we used $\eta = 1.0 \times 10^{-12}$ (we call this modified algorithm ExactCG-SLIR). The results are shown in Tables 6.6, 6.7, 6.8.

Table 6.1: QPGEN test problems generated with various parameters.

name	CONDN	PCTBND	PCTINF
QPGEN1	2	0.5	0.0
QPGEN2	6	0.5	0.0
QPGEN3	2	0.9	0.0
QPGEN4	2	0.1	0.0
QPGEN5	2	0.5	0.2
QPGEN6	2	0.5	0.5
QPGEN7	2	0.5	0.9

Table 6.2: QPGEN1 major iterations and error values.

		iterations		*error*	
n	*density*	*SLIR*	*CG-SLIR*	*SLIR*	*CG-SLIR*
400	0.0025	14	15	1.181e-15	1.476e-15
400	0.0125	13	14	1.109e-15	1.109e-15
400	0.0250	13	14	-1.449e-15	-4.829e-16
400	0.0375	14	16	5.165e-16	1.291e-16
900	0.0011	14	16	1.159e-15	1.883e-15
900	0.0055	15	16	-3.356e-16	1.342e-15
900	0.0011	13	15	3.456e-16	-4.608e-16
900	0.0166	15	16	1.512e-15	2.094e-15
1600	0.0006	15	16	2.610e-15	2.320e-15
1600	0.0031	15	15	-0.000	3.209e-15
1600	0.0062	14	16	-0.000	8.536e-16
1600	0.0093	15	16	-7.770e-16	2.202e-15
2500	0.0004	14	15	1.720e-15	1.720e-15
2500	0.0020	15	16	-4.220e-16	1.688e-15
2500	0.0040	16	16	-3.435e-16	1.202e-15
2500	0.0060	16	16	1.148e-15	2.952e-15

Table 6.3: QPGEN1 Mflop counts and speedup.

		total Mflops		
n	*density*	*SLIR*	*CG-SLIR*	*speedup*
400	0.0025	2.301	2.807	0.820
400	0.0125	6.944	4.186	1.659
400	0.0250	27.591	5.558	4.964
400	0.0375	52.354	7.536	6.947
900	0.0011	5.181	6.960	0.744
900	0.0055	46.376	10.772	4.305
900	0.0111	244.043	13.510	18.064
900	0.0166	546.742	17.815	30.689
1600	0.0006	9.780	12.897	0.758
1600	0.0031	232.663	17.179	13.543
1600	0.0062	1382.793	25.050	55.200
1600	0.0093	2963.446	31.225	94.906
2500	0.0004	14.562	18.071	0.806
2500	0.0020	823.791	29.045	28.363
2500	0.0040	2013.804	39.014	51.617
2500	0.0060	11596.961	48.654	238.358

Table 6.4: QPGEN2 major iterations and error values.

		iterations		*error*	
n	*density*	*SLIR*	*CG-SLIR*	*SLIR*	*CG-SLIR*
400	0.0025	12	11	1.239e-15	1.180e-08
400	0.0125	16	16	1.283e-15	1.493e-07
400	0.0250	16	18	-4.390e-16	1.375e-08
400	0.0375	19	19	5.536e-13	6.040e-10
900	0.0011	14	17	1.937e-15	3.088e-08
900	0.0055	14	22	1.436e-10	7.785e-09
900	0.0111	18	22	8.938e-16	5.333e-08
900	0.0166	18	26	7.856e-10	1.197e-08
1600	0.0006	16	15	1.591e-15	6.552e-08
1600	0.0031	21	21	-4.308e-16	5.381e-08
1600	0.0062	27	26	9.715e-16	4.632e-08
1600	0.0093	27	20	-1.280e-15	5.871e-08
2500	0.0004	15	17	1.480e-15	6.233e-08
2500	0.0020	19	20	1.788e-15	4.318e-08
2500	0.0040	25	24	1.795e-14	4.461e-08
2500	0.0060	27	35	-7.417e-16	3.978e-08

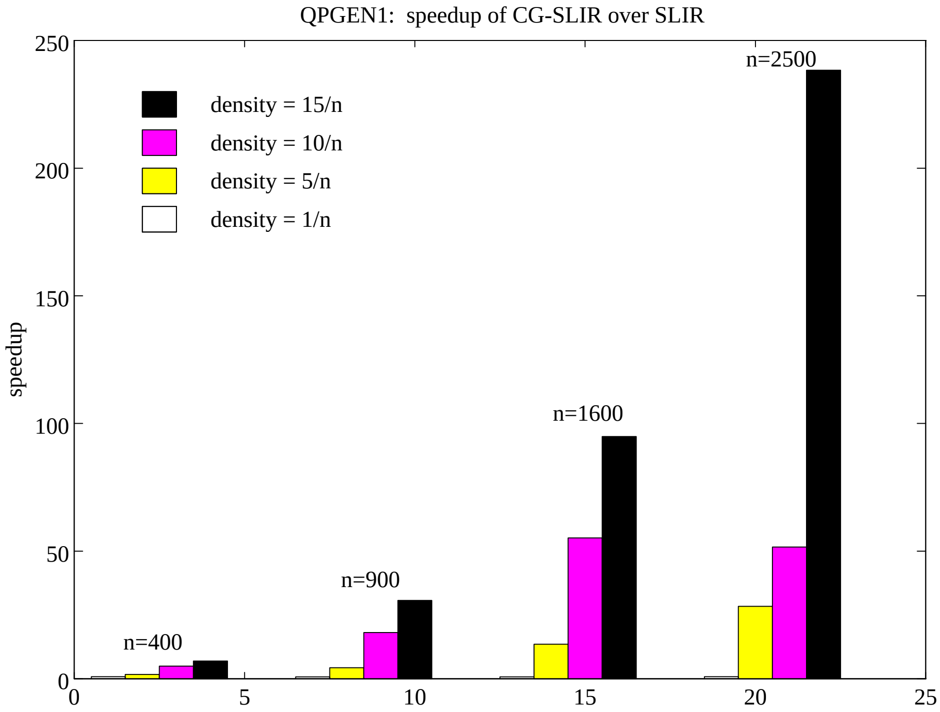

Figure 6.1: Speedup of CG-SLIR over SLIR on QPGEN1 problems.

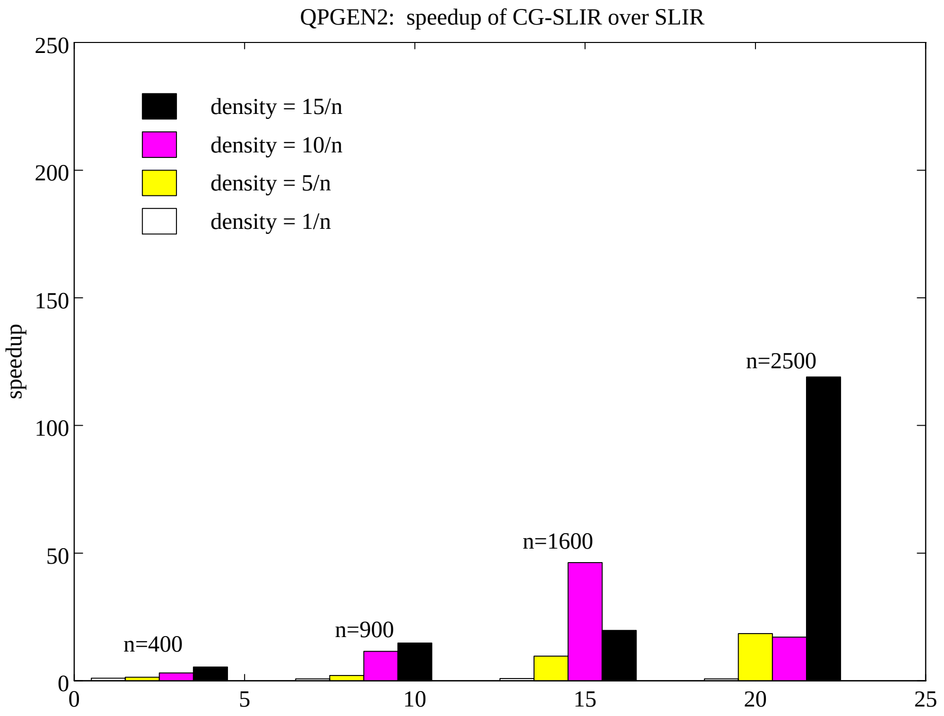

Figure 6.2: Speedup of CG-SLIR over SLIR on QPGEN2 problems.

The accuracy for ExactCG-SLIR is comparable to that of SLIR; major iteration count is also close (except one outlier for $n = 1600$, $frac = 10$). Speedup over SLIR is reduced by about a factor of 4 compared to CG-SLIR, but still up to 36 times faster than SLIR (see also Figure 6.3; note that the y-axis is on a smaller scale than the previous two graphs). These results are encouraging because they suggest that our inexact method can be be competitive with SLIR in terms of accuracy, even for ill-conditioned problems, if the stopping rule for MPCG is adjusted.

6.2.2 QP3D

Our second set of problems QP3D is similar to QPGEN, but the Hessians have the structure of a three-dimensional 7-point finite difference mesh. We generated problems for n equal to 343, 1000, 1331, and 2197 (n needed to be a cube). Density is no longer a parameter since it is determined by the mesh. Table 6.9 shows the choice of parameters for each set of problems. We did not vary PCTINF for the QP3D problems.

For all problems, including those with ill-conditioned Hessians, the outer iteration count and accuracy is almost identical for SLIR and CG-SLIR (Table 6.10). The speedup appears to be not as good as for the QPGEN problems, with a maximum of only about 11 (Table 6.11). However, if we compare the problems in QP3D to problems in QPGEN of similar size and density, then the speedup results are only slightly less for these problems. (In other words, none of these problems are as dense as the QPGEN problems where the highest speedup was achieved.) The speedup would be the same, but the Cholesky factorization in SLIR is faster on these problems (apparently due to the reordering done prior to factoring). Again, varying `PCTBND` and `PCTINF` did not have a noticeable effect on the results.

6.2.3 QP2D

Our third set of problems QP2D have Hessians with the structure of a two-dimensional k-point finite difference mesh, where k is 5, 7, and 9. Again, density is not a factor. Each type of problem was generated with n equal to 400, 900, 1600, and 2500. The choices for parameters are shown in Table 6.12. For 5-point grid problems, we used the same parameters as for the three-dimensional case. For 7-point and 9-point we only varied the condition number.

As with the three-dimensional case, the outer iteration count and accuracy is almost identical for SLIR and CG-SLIR for all problems (Table 6.13). Again, speedup is even less than it was for the QP3D problems, with a maximum around 2.5 (Table 6.14). The reduced speedup is again because the Cholesky factorization in SLIR is able to execute faster on the QP2D problems than on the QP3D problems. Density increases with k (the differencing parameter), and as expected,

Table 6.5: QPGEN2 Mflop counts and speedup.

		total Mflops		
n	*density*	*SLIR*	*CG-SLIR*	*speedup*
400	0.0025	1.961	1.949	1.007
400	0.0125	8.685	6.407	1.356
400	0.0250	34.456	11.290	3.052
400	0.0375	72.129	13.459	5.359
900	0.0011	5.219	7.244	0.720
900	0.0055	42.994	20.820	2.065
900	0.0111	345.714	29.970	11.535
900	0.0166	662.465	44.816	14.782
1600	0.0006	10.073	11.907	0.846
1600	0.0031	330.680	34.213	9.665
1600	0.0062	2763.327	59.704	46.284
1600	0.0093	1207.134	61.114	19.752
2500	0.0004	15.641	21.359	0.732
2500	0.0020	1059.426	57.405	18.455
2500	0.0040	1501.679	87.772	17.109
2500	0.0060	20099.087	168.939	118.972

Table 6.6: QPGEN2 error values with ExactCG-SLIR.

			error	
n	*density*	*SLIR*	*ExactCG-SLIR*	*CG-SLIR*
400	0.0025	1.239e-15	1.771e-15	1.180e-08
400	0.0125	1.283e-15	3.636e-15	1.493e-07
400	0.0250	-4.390e-16	1.537e-15	1.375e-08
400	0.0375	5.536e-13	7.484e-16	6.040e-10
900	0.0011	1.937e-15	2.324e-15	3.088e-08
900	0.0055	1.436e-10	2.056e-10	7.785e-09
900	0.0111	8.938e-16	4.294e-09	5.333e-08
900	0.0166	7.856e-10	1.568e-15	1.197e-08
1600	0.0006	1.591e-15	1.989e-15	6.552e-08
1600	0.0031	-4.308e-16	6.461e-16	5.381e-08
1600	0.0062	9.715e-16	1.749e-15	4.632e-08
1600	0.0093	-1.280e-15	-3.291e-15	5.871e-08
2500	0.0004	1.480e-15	1.480e-15	6.233e-08
2500	0.0020	1.788e-15	-8.253e-16	4.318e-08
2500	0.0040	1.795e-14	6.452e-14	4.461e-08
2500	0.0060	-7.417e-16	-1.236e-16	3.978e-08

Table 6.7: QPGEN2 major iterations with ExactCG-SLIR.

		iterations		
n	*density*	*SLIR*	*ExactCG-SLIR*	*CG-SLIR*
400	0.0025	12	12	11
400	0.0125	16	17	16
400	0.0250	16	16	18
400	0.0375	19	17	19
900	0.0011	14	15	17
900	0.0055	14	14	22
900	0.0111	18	16	22
900	0.0166	18	21	26
1600	0.0006	16	18	15
1600	0.0031	21	22	21
1600	0.0062	27	41	26
1600	0.0093	27	23	20
2500	0.0004	15	17	17
2500	0.0020	19	21	20
2500	0.0040	25	26	24
2500	0.0060	27	28	35

Table 6.8: QPGEN2 Mflop counts and speedup with ExactCG-SLIR.

		total Mflops		
n	*density*	*SLIR*	*ExactCG-SLIR*	*CG-SLIR*
400	0.0025	1.961	3.370	1.949
400	0.0125	8.685	22.221	6.407
400	0.0250	34.456	22.875	11.290
400	0.0375	72.129	24.011	13.459
900	0.0011	5.219	12.687	7.244
900	0.0055	42.994	33.641	20.820
900	0.0111	345.714	53.115	29.970
900	0.0166	662.465	113.322	44.816
1600	0.0006	10.073	28.309	11.907
1600	0.0031	330.680	157.862	34.213
1600	0.0062	2763.327	301.708	59.704
1600	0.0093	1207.134	270.092	61.114
2500	0.0004	15.641	54.747	21.359
2500	0.0020	1059.426	349.403	57.405
2500	0.0040	1501.679	347.886	87.772
2500	0.0060	20099.087	558.659	168.939

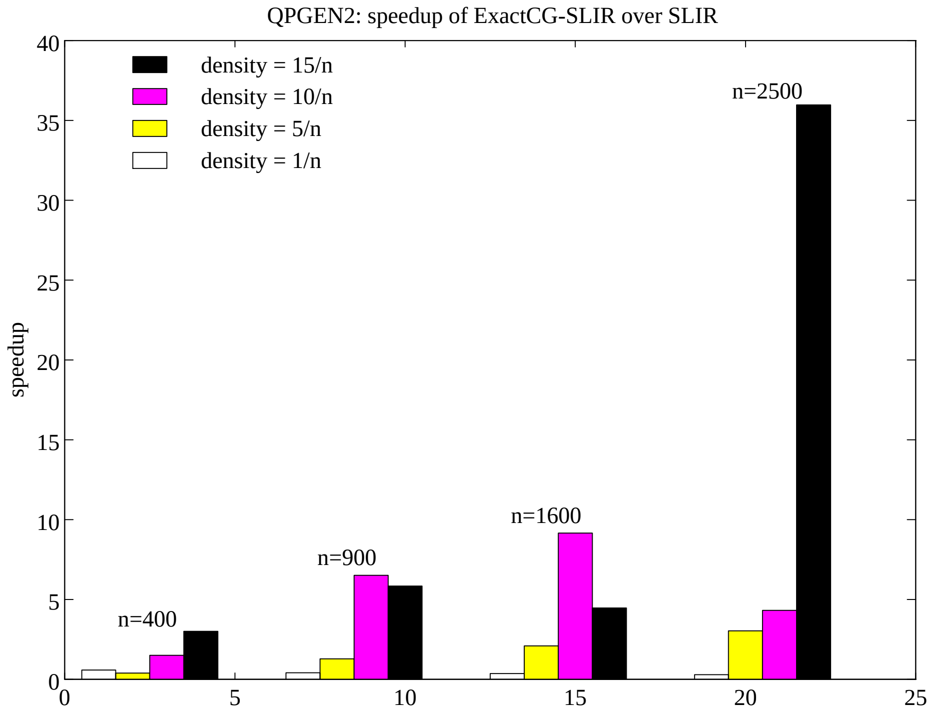

Figure 6.3: Speedup of ExactCG-SLIR over SLIR on QPGEN2 problems.

Table 6.9: QP3D test problems generated with various parameters.

name	CONDN	PCTBND	PCTINF
QP3D1	2	0.5	0.0
QP3D2	6	0.5	0.0
QP3D3	2	0.9	0.0
QP3D4	2	0.1	0.0

Table 6.10: QP3D major iterations and error values.

			iterations		*error*	
problem	*n*	*density*	*SLIR*	*CG-SLIR*	*SLIR*	*CG-SLIR*
QP3D1	343	0.0179	13	14	1.082e-15	1.353e-16
	1000	0.0064	15	15	2.181e-15	2.036e-15
	1331	0.0048	15	15	-1.060e-15	-4.713e-16
	2197	0.0030	15	15	5.380e-16	-5.380e-16
QP3D2	343	0.0179	13	14	1.058e-15	1.586e-15
	1000	0.0064	15	15	1.277e-15	3.101e-15
	1331	0.0048	14	14	1.874e-15	1.363e-15
	2197	0.0030	14	14	6.036e-16	3.420e-15
QP3D3	343	0.0179	14	14	1.057e-15	8.458e-16
	1000	0.0064	16	16	-0.000	-2.948e-16
	1331	0.0048	17	17	-3.981e-16	-3.981e-16
	2197	0.0030	16	16	2.060e-15	1.945e-15
QP3D4	343	0.0179	14	14	1.521e-15	2.705e-15
	1000	0.0064	14	14	-1.736e-16	3.473e-16
	1331	0.0048	13	14	-4.124e-16	1.787e-15
	2197	0.0030	13	14	1.558e-16	4.986e-15

Table 6.11: QP3D Mflops counts and speedup.

			total Mflops		
problem	*n*	*density*	*SLIR*	*CG-SLIR*	*speedup*
QP3D1	343	0.0179	5.923	3.246	1.824
	1000	0.0064	44.309	10.405	4.259
	1331	0.0048	98.791	13.895	7.110
	2197	0.0030	248.552	22.950	10.830
QP3D2	343	0.0179	5.911	3.304	1.789
	1000	0.0064	44.317	10.656	4.159
	1331	0.0048	91.708	13.137	6.981
	2197	0.0030	230.759	21.896	10.539
QP3D3	343	0.0179	6.418	3.231	1.986
	1000	0.0064	47.449	11.098	4.276
	1331	0.0048	112.830	15.755	7.162
	2197	0.0030	266.281	24.567	10.839
QP3D4	343	0.0179	6.364	3.351	1.899
	1000	0.0064	41.074	9.759	4.209
	1331	0.0048	84.627	13.142	6.439
	2197	0.0030	212.905	21.859	9.740

speedup increases with k as well (see Table 6.14). Varying PCTBND, PCTINF, and CONDN did not have any noticeable effect on the results.

6.2.4 OBS

The last set of problems (OBS) are from [Moré and Toraldo]. Problem 1 is an obstacle problem with lower bound surface only. Problem 2 is an obstacle problem with lower and upper bound surfaces. Problem 3 is an elastic-plastic torsion problem. For these problems only the size was varied. For problems 1, 2, and 3 we used $n = 900$, 1600, and 2500. These problems also have Hessian matrices with the structure of a two-dimensional 5-point finite difference mesh.

Comments on accuracy will not be made as we do not have the "exact" solution to compare to computed solutions. As for major iteration counts, CG-SLIR was close to SLIR, but sometimes as much as four iterations more (Table 6.15). In terms of flops, CG-SLIR did slightly worse on about half the problems, and slightly better on the other half (speedup ranged from 0.766 to 1.330). Table 6.16 shows the flop counts and speedup for each problem. Again, the Cholesky factorization in SLIR is particularly efficient on these problems, explaining the low speedup.

6.3 Discussion

The computational results show that CG-SLIR is promising. It maintains the high accuracy and low iteration count of SLIR in most cases, with an impressive reduction in flop counts. The speedup of the algorithm is improved when the size grows and as density increases. Thus, while the Cholesky factorization in SLIR sometimes performs well on large problems, CG-SLIR always does at least as well as SLIR, and much of the time executes much faster. As noted before, the accuracy of CG-SLIR on the ill-conditioned problems appears improvable if the stopping criteria for MPCG is more stringent.

Table 6.12: QP2D test problems generated with various parameters.

name	*k-point*	`CONDN`	`PCTBND`	`PCTINF`
QP2D1	5	2	0.5	0.0
QP2D2	5	6	0.5	0.0
QP2D3	5	2	0.9	0.0
QP2D4	5	2	0.1	0.0
QP2D5	7	2	0.5	0.0
QP2D6	7	6	0.5	0.0
QP2D7	9	6	0.5	0.0
QP2D8	9	2	0.5	0.0

Table 6.13: QP2D major iterations and error values.

				iterations		*error*	
problem	*k*	*n*	*density*	*SLIR*	*CG-SLIR*	*SLIR*	*CG-SLIR*
QP2D1	5	400	0.0120	13	14	5.097e-16	2.549e-16
	5	900	0.0054	14	14	-3.402e-16	1.247e-15
	5	1600	0.0031	14	14	1.289e-15	-1.523e-15
	5	2500	0.0020	14	14	-8.650e-16	-1.586e-15
QP2D5	7	400	0.0165	13	13	-1.927e-16	7.707e-16
	7	900	0.0075	15	15	2.878e-16	7.194e-16
	7	1600	0.0043	14	15	2.053e-15	8.554e-16
	7	2500	0.0027	15	15	3.053e-15	2.181e-16
QP2D7	9	400	0.0210	14	14	1.163e-15	1.018e-15
	9	900	0.0096	14	13	3.716e-16	8.671e-16
	9	1600	0.0054	15	15	5.290e-16	2.909e-15
	9	2500	0.0035	15	14	1.750e-16	-2.625e-15

Table 6.14: QP2D Mflops counts and speedup.

				total Mflops		
problem	*k*	*n*	*density*	*SLIR*	*CG-SLIR*	*speedup*
QP2D1	5	400	0.0120	3.755	3.483	1.078
	5	900	0.0054	10.435	7.907	1.320
	5	1600	0.0031	21.724	14.173	1.533
	5	2500	0.0020	38.007	22.052	1.724
QP2D5	7	400	0.0165	4.802	3.611	1.330
	7	900	0.0075	15.191	9.515	1.597
	7	1600	0.0043	29.140	16.967	1.717
	7	2500	0.0027	57.542	26.527	2.169
QP2D7	9	400	0.0210	6.516	4.406	1.479
	9	900	0.0096	17.972	9.242	1.945
	9	1600	0.0054	51.213	19.126	2.678
	9	2500	0.0035	77.010	27.779	2.772

Table 6.15: OBS major iterations.

			iterations	
problem	*n*	*density*	*SLIR*	*CG-SLIR*
OBS1	900	0.0054	12	16
	1600	0.0031	13	16
	2500	0.0020	14	17
OBS2	900	0.0054	11	13
	1600	0.0031	12	14
	2500	0.0020	13	14
OBS3	900	0.0054	12	13
	1600	0.0031	12	11
	2500	0.0020	12	14

Table 6.16: OBS Mflops counts and speedup.

			total Mflops		
problem	*n*	*density*	*SLIR*	*CG-SLIR*	*speedup*
OBS1	900	0.0054	8.393	10.957	0.766
	1600	0.0031	18.549	21.524	0.862
	2500	0.0020	35.478	38.339	0.925
OBS2	900	0.0054	8.064	8.102	0.995
	1600	0.0031	18.338	16.381	1.119
	2500	0.0020	35.152	26.437	1.330
OBS3	900	0.0054	8.932	10.067	0.887
	1600	0.0031	18.433	15.783	1.168
	2500	0.0020	32.111	36.845	0.872

Chapter 7

STIR: Computational results for nonlinear problems

We demonstrate the computational performance of our STIR method given in Figure 3.6 with the subspace determined as in Figure 5.1. We compare our experience with the modified Cholesky and the conjugate gradient (MPCG) implementations. We examine the sensitivity of the STIR method to a starting point. Finally, some limited comparisons with SBMIN of LANCELOT [CGT92] are also made.

7.1 Implementation

In the implementation of STIR, we compute s_k using a reflective technique as in Figure 3.6. The inexact Newton direction is computed using MPCG in Figure 4.1. The vector w_k in Figure 5.1 is the vector returned by MPCG (either an inexact Newton direction or a direction of negative curvature). The exact trust-region updating procedure is given in Figure 7.1.

For the modified Cholesky implementation of STIR, the vector w_k in Figure 5.1 is the Newton step computed from the Cholesky factorization when the Hessian is positive definite; when the Hessian is indefinite, a direction of negative curvature is computed from the incomplete Cholesky factorization (see [GMW81], page 111).

Our experiments were carried out on a Sun Sparc workstation using the Matlab environment.

The stopping criteria used are as follows. We stop if

$$\begin{array}{ll} \text{either} & f(x_k) - f(x_{k+1}) \leq \tau_1(1 + |f(x_k)|) \\ \text{or} & \|x_{k+1} - x_k\|_2 \leq \tau_2 \\ \text{or} & \text{no negative curvature has been detected for } \hat{M}_k \textit{ and } \|D_k g_k\|_\infty \leq \tau_1. \end{array}$$

We define $\tau_1 = 10^{-10}$ and $\tau_2 = \sqrt{\tau_1}/10 = 10^{-6}$. We also impose an upper bound of 600 on the number of iterations.

Updating Trust-Region Size Δ_k

Let $\mu = 0.25,\ \eta = 0.75,\ \Lambda_l = 1,$
$\Lambda_u = \max(\sqrt{\sum_i \min((u_i - l_i)^2, 1000)},\ 1),\ \Delta_1 = \min(0.1\|g\|, \Lambda_u),$
$\gamma_0 = 0.0625,\ \gamma_1 = 0.5,\ \gamma_2 = 2$ be given.

1. If $\rho_k \leq 0$ then set $\Delta_{k+1} = \gamma_0 \Delta$.
2. If $\rho_k \in (0, \mu]$ then set $\Delta_{k+1} = \max(\gamma_0 \Delta_k, \gamma_1 \|D_k s_k\|)$.
3. If $\rho_k \in (\mu, \eta)$ then set $\Delta_{k+1} = \Delta_k$.
4. If $\rho_k \geq \eta$ then
   if $\Delta_k > \Lambda_l$
   $\Delta_{k+1} = \gamma_2 \Delta_k$
   otherwise
   $\Delta_{k+1} = \min(\max(\Delta_k, \gamma_2 \|D_k s_k\|), \Lambda_u)$.

Figure 7.1: Updating the trust-region size.

7.2 Computational experience

We first report the results of the STIR method using the modified Cholesky factorization. Table 7.1 lists the number of iterations required for some standard testing problems (for details of these problems see [CGT88b]). We do note that, for example, "GENROSE U" implies the solution of the problem has no active constraints, although bound constraints are present in the problem, whereas "GENROSE C" implies that there exist active constraints at the solution. (For all the results in this paper, the number of iterations is the same as the number of objective function evaluations.) The problem sizes vary from 100 to 10,000. The results in Table 7.1 indicate that, for these test problems at least, the number of iterations increases only slightly, if at all, with the problem size. Moreover, in comparison to the unconstrained problems, the presence of the bound restrictions does not seem to increase the number of iterations. This is depicted pictorially in Figure 7.2. In this graph, the problem size is plotted versus iteration count. For each problem, the corresponding points have been connected to show how the iteration count relates to the problem size.

Our second set of results are for the STIR algorithm but using a conjugate gradient implementation. We use the algorithm MPCG in Figure 4.1 to find the directions needed to form the subspace $\mathcal{S}_k$. The stopping condition applied to the relative residual in MPCG is $\eta = 0.1$. The results are shown in Table 7.2 and Figure 7.3. Again, for these problems the iteration counts are low and steady. The

Table 7.1: STIR method with exact Newton steps: number of iterations.

Problem	n 100	200	500	1000	10000
GENROSE U	25	25	25	25	25
GENROSE C	11	11	11	11	10
GENSING U	24	25	25	26	27
GENSING C	18	19	20	20	21
CHAINSING U	23	23	23	23	23
CHAINSING C	16	16	16	16	19
DEGENSING U	22	23	23	40	39
DEGENSING C	28	28	28	28	29
GENWOOD C	9	10	10	10	11
CHAINWOOD C	9	10	10	10	11
BROYDEN1A U	12	12	13	13	14
BROYDEN1A C	11	11	11	11	11
BROYDEN1B U	7	7	7	7	7
BROYDEN1B C	8	8	8	8	8
BROYDEN2A U	13	13	13	14	14
BROYDEN2A C	14	19	17	19	19
BROYDEN2B U	9	9	9	9	9
BROYDEN2B C	13	11	15	14	15
TOINTBROY U	8	8	8	8	8
TOINTBROY C	9	9	9	9	9
CRAGGLEVY U	16	14	15	16	15
CRAGGLEVY C	29	29	30	30	31
AUGMLAGN C	38	32	35	36	37
BROWN3 U	8	8	8	8	8
BROWN3 C	17	10	11	9	11
BVP U	9	10	9	8	8
BVP C	11	11	10	10	7
VAR U	9	9	10	12	15
VAR C	18	18	23	45	38

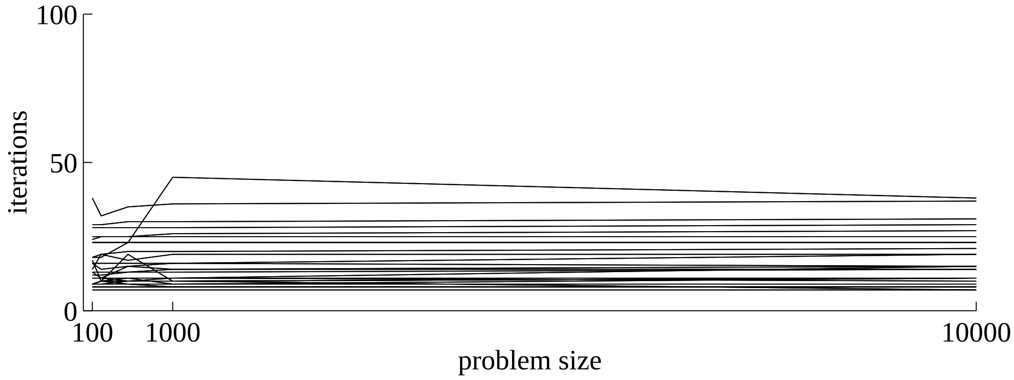

Figure 7.2: STIR performance with exact Newton steps.

exception is for the problem VAR C with 10,000 variables, where the iteration count jumps to 75. In Figure 7.5, VAR C is shown with a dotted line. In this case, the MPCG algorithm seems to be at fault since it takes the maximum number of MPCG iterations ($n/2$) each time. A better preconditioner might resolve this behavior.

Next we include some results which indicate that our STIR method is fairly insensitive to the starting point. The results in Table 7.3 were obtained using exact Newton steps on problems of dimension 1000. The results in Table 7.4 were obtained using the conjugate gradient implementation, also on problems with 1000 variables. The starting points are as follows: *original* is the suggested starting point according to [CGT88b]; *upper* starts all variables at upper bounds; *lower* starts all variables at the lower bounds; *middle* starts at the midpoint between bounds; *zero* starts each variable at zero (the origin); *upper-lower* starts the odd variables at the upper and the even variables at the lower bounds; *lower-upper* is the reverse of this. For all of these, we perturb the starting point slightly if necessary to be strictly feasible. Note that for the problem BROWN3 C, the iteration count is not shown starting at *middle* and at *origin* as the gradient is undefined at both these starting points. These results are also shown graphically in Figure 7.4 and Figure 7.5. From these graphs it is clear that both implementations of STIR are fairly robust when it comes to starting points. This is in contrast to active-set methods where the starting point can have a more dramatic effect on the iteration count.

Last we contrast the performance of the STIR method using the conjugate gradient option with the SBMIN algorithm, a gradient projection active-set method, in the LANCELOT software package [CGT92]. In particular, we choose problems where negative curvature is present or where it appears that the "active set" at

Table 7.2: STIR method with inexact Newton steps, $\eta = 0.1$: number of iterations.

	n				
Problem	100	200	500	1000	10000
GENROSE U	23	23	23	23	23
GENROSE C	10	10	10	10	9
GENSING U	23	23	23	24	25
GENSING C	16	16	16	16	16
CHAINSING U	21	21	21	21	21
CHAINSING C	16	16	16	17	17
DEGENSING U	31	33	32	33	35
DEGENSING C	27	27	28	29	30
GENWOOD C	7	7	7	7	7
CHAINWOOD C	7	7	7	7	7
BROYDEN1A U	11	11	11	11	12
BROYDEN1A C	8	8	8	8	8
BROYDEN1B U	6	6	6	6	6
BROYDEN1B C	7	7	7	7	7
BROYDEN2A U	15	15	19	17	20
BROYDEN2A C	10	10	10	10	10
BROYDEN2B U	8	8	8	8	8
BROYDEN2B C	9	9	9	9	9
TOINTBROY U	7	7	7	7	6
TOINTBROY C	8	8	8	8	8
CRAGGLEVY U	26	26	27	27	29
CRAGGLEVY C	26	26	26	26	27
AUGMLAGN C	29	28	30	28	28
BROWN3 U	7	7	7	7	7
BROWN3 C	7	7	7	7	8
BVP U	13	13	12	13	25
BVP C	15	15	14	14	15
VAR U	34	35	35	36	37
VAR C	20	22	25	41	75

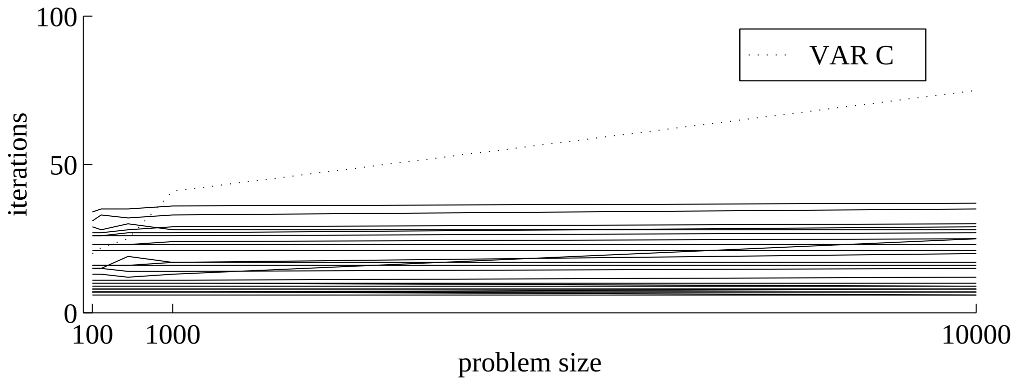

Figure 7.3: STIR method with inexact Newton steps.

Table 7.3: STIR method with exact Newton steps for $n = 1000$: number of iterations.

	Starting Point						
Problem	original	upper	lower	middle	zero	up-low	low-up
GENROSE C	11	27	33	15	16	43	27
GENSING C	20	31	45	25	22	31	32
CHAINSING C	16	29	33	13	11	32	30
DEGENSING C	28	47	39	52	42	39	36
GENWOOD C	10	18	14	13	10	17	17
CHAINWOOD C	10	17	14	13	10	17	16
BROYDEN1A C	11	24	25	13	12	25	24
BROYDEN1B C	8	22	19	18	9	19	21
BROYDEN2A C	19	38	38	13	9	38	38
BROYDEN2B C	14	30	34	12	8	33	30
CRAGGLEVY C	30	38	33	26	26	34	37
AUGMLAGN C	36	40	26	36	15	23	37
BROWN3 C	9	28	14	*	*	28	14
BVP C	10	17	8	9	10	11	17
VAR C	45	9	32	18	21	23	17

Table 7.4: STIR method with inexact Newton steps for $n = 1000$: number of iterations.

Problem	Starting Point original	upper	lower	middle	zero	up-low	low-up
GENROSE C	10	22	34	18	20	38	22
GENSING C	16	27	43	25	23	29	29
CHAINSING C	17	27	33	12	9	32	27
DEGENSING C	29	43	36	44	37	36	44
GENWOOD C	7	13	9	10	7	13	13
CHAINWOOD C	7	13	9	10	7	13	13
BROYDEN1A C	8	23	21	12	8	21	23
BROYDEN1B C	7	20	16	13	7	16	20
BROYDEN2A C	10	35	34	12	7	36	35
BROYDEN2B C	9	28	30	11	7	30	28
CRAGGLEVY C	26	42	43	26	24	34	39
AUGMLAGN C	28	58	53	28	10	36	58
BROWN3 C	7	29	53	*	*	29	53
BVP C	14	21	14	13	14	14	21
VAR C	41	7	35	30	37	29	8

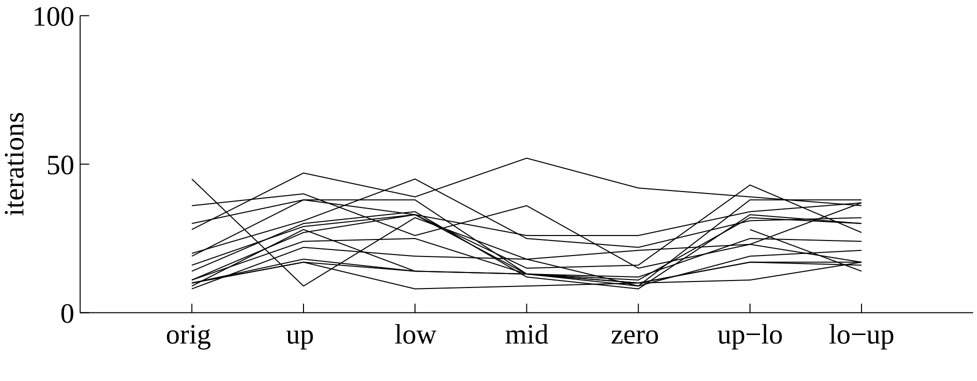

Figure 7.4: STIR method with exact Newton steps at varied starting points.

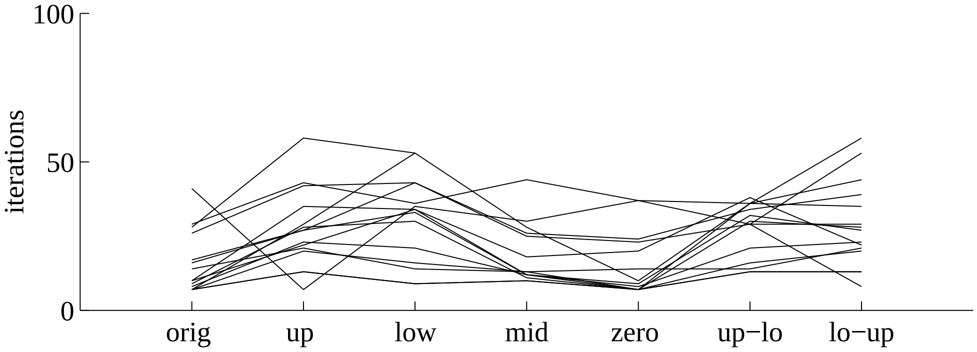

Figure 7.5: STIR method with inexact Newton steps at varied starting points.

the solution may be difficult to find. We expect our STIR method to outperform an active-set method in these situations; indeed, we have found this to be the case. For these problems, we use the default settings for LANCELOT and adjusted our STIR stopping conditions to be comparable if not more stringent.

First consider a constrained convex quadratic problem. The results, given in Table 7.5, show that our proposed STIR method is markedly superior (by an order of magnitude) to SBMIN on this problem (*c.g. it* is the total number of conjugate gradient iterations). SBMIN takes many iterations on this problem when the starting point is near some of the bounds — the method mis-identifies the correct active set at the solution and takes many iterations to recover. Our proposed STIR method, a strictly interior method, moves directly to the solution without faltering when started at the same point.

Table 7.6 summarizes the performances of STIR and SBMIN on a set of constrained problems exhibiting negative curvature. (These problems are also from [CGT88b] except that the last two have been constrained differently to display negative curvature.) STIR is significantly better on these problems — this is probably due to the fact that negative curvature is better exploited in our subspace trust-region approach than in the Steihaug trust-region method, which SBMIN employs. This is consistent with results presented in Chapter 8, for example see Table 8.2.

Table 7.5: STIR with inexact Newton steps versus LANCELOT SBMIN on a convex quadratic: number of iterations.

	inexact STIR		SBMIN	
$n = 800$	iteration	c.g. it	iteration	c.g. it
BIGGSB2	16	5551	281	53157

Table 7.6: STIR with inexact Newton steps versus LANCELOT SBMIN when negative curvature exists: number of iterations.

	inexact STIR			SBMIN		
		n			n	
Problem	100	1000	10000	100	1000	10000
AUGMLAGN U	34	30	37	29	38	46
CHAINWOOD U	106	1004	8953	6594	> 10000	> 10000
GENWOOD U	81	89	72	439	952	554
GENROSE U	23	23	23	76	76	76
CHAINWOOD NC	17	31	16	54	48	61
GENWOOD NC	16	24	23	47	> 1000	60

Chapter 8

Solving the trust-region subproblem

We compare four trust-region methods for unconstrained minimization problems. Two are based on subspace trust-region ideas. The third is an approach developed by Steihaug [Ste83]. The last is based on a more traditional dogleg strategy. Each method is implemented using a preconditioned conjugate gradient method to find the inexact Newton direction or a direction of negative curvature. Computationally, we are interested in contrasting the subspace and the Steihaug methods. We believe that a subspace trust-region approach better captures the negative curvature information compared to the Steihaug approach [Ste83]. The other methods (three-dimensional subspace and the dogleg strategy) are included as they help explain the behavior of the subspace and Steihaug algorithms. Results from numerical experiments supporting our preference for the subspace strategy are discussed.

8.1 Introduction

In this chapter we consider in more detail the *unconstrained* subproblem

$$\min_s \{s^T g + \frac{1}{2} s^T H s : \ \|s\| \leq \Delta\} \tag{8.1}$$

where $g \in R^n$, $H \in R^{n \times n}$ is symmetric, and Δ is positive.

We compare four algorithms which we shall refer to as SUBSP2, SUBSP3, STEIH, and DOGLEG. The first and second are due to Byrd, Schnabel, and Schultz [SSB85]. These algorithms compute an approximate solution to (8.1) by solving

$$\min_s \{s^T g + \frac{1}{2} s^T H s : \ \|s\|_2 \leq \Delta, s \in \mathcal{S}\} \tag{8.2}$$

where $\mathcal{S}$ is a low-dimensional subspace. For SUBSP2, $\mathcal{S}$ is 2-dimensional. For SUBSP3, $\mathcal{S}$ is 3-dimensional. When H is positive definite, $\mathcal{S}$ is the subspace spanned by the inexact Newton direction (an approximation to $-H^{-1}g$) and the steepest descent direction $-g$ for both methods. When H is indefinite, $\mathcal{S}$ is chosen to be the subspace containing $-g$ and a direction of negative curvature d (SUBSP2), or these two directions plus a descent direction p (SUBSP3). The inexact Newton direction is found using a preconditioned conjugate gradient (PCG) algorithm (see Figure 8.1). If a direction of negative curvature d is found, that direction, and possibly the current direction p maintained by the PCG algorithm (the current estimate to the inexact Newton direction), are used. See Figures 8.2 and 8.3.

The third method STEIH is due to Steihaug [Ste83]. This method computes an approximate solution to (8.1) by solving

$$\min_s \{s^T g + \frac{1}{2} s^T H s : \ \|s\|_C \leq \Delta\} \tag{8.3}$$

where C is the preconditioning matrix of the conjugate gradient algorithm, and $\|s\|_C^2 \stackrel{\text{def}}{=} s^T C s$. The Steihaug applies the method of preconditioned conjugate gradients (PCG) to the current Newton system until either negative curvature is revealed, the current approximate solution reaches the boundary of the trust region, or the Newton system residual is sufficiently reduced. The algorithm is displayed in Figure 8.4.

The last algorithm DOGLEG is a version of the single dogleg method. This algorithm approximately solves (8.1) as follows. If the Cauchy point is outside the trust region, or the Cauchy is not defined (because $g^T H g < 0$), then our solution is the point where the steepest descent direction meets the trust-region bound. If the Hessian is positive definite and the inexact Newton step is within, or on, the trust-region bound, our solution is the inexact Newton step. Otherwise, we take the path from our current point to the Cauchy point, and then move in the inexact Newton direction until the trust region bound is met. Finally, if H is indefinite, and the Cauchy point is inside the trust region, we consider the path formed from our current point to the Cauchy point, and then in a direction of negative curvature d. We find the point along this path that meets the boundary. Again, we use a preconditioned conjugate gradient algorithm to find the inexact Newton direction or a direction of negative curvature. See Figure 8.5.

All four algorithms are wrapped in a standard trust-region framework for the unconstrained minimization problem. Also, all four algorithms use the preconditioned conjugate gradient method as given in [GVL89], p. 529 (see Figure 8.1), with a modification to detect negative curvature. The preconditioning matrix used is $C = R^2$ where R is the diagonal matrix computed from $R_{ii} = \sqrt{|H_{ii}|}$ for $H_{ii} \neq 0$ and $R_{ii} = 1$ otherwise.

If the inexact Newton direction is computed, the accuracy with which it is

```
function [p, d, posdef] = PCG(H, g, C, α)

% Note: C is some preconditioning matrix for H.
% C must be positive definite. C = R^2.

n = length(g);
k = 0; p_0 = 0; r_0 = -g;
kmax = n/2; posdef = 1;

while k < kmax
    Step 1:  Solve C z_k = r_k;
    Step 2:  k = k + 1
    Step 3:  if k = 1
               d_1 = z_0
             else
               β_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}
               d_k = z_{k-1} + β_k d_{k-1}
             end
    Step 4:  γ_k = d_k^T H d_k
    Step 5:  if γ_k ≤ 0
               d = d_k, p = p_{k-1}, posdef = 0, return
             else
               α_k = r_{k-1}^T z_{k-1} / γ_k
               p_k = p_{k-1} + α_k d_k
               r_k = r_{k-1} - α_k H d_k
             end
    Step 6:  if ||R^{-1}|| · ||r_k|| ≤ η||R^{-1} g||
               p = p_k, d = 0, return
             end

end
p = p_k, d = 0, return
```

Figure 8.1: The preconditioned conjugate gradient algorithm.

```
function [s] = trustS2(H, g, C, α, Δ)

% Note: C is some preconditioning matrix for H.
% C must be positive definite.


Step 1: [p, d, posdef] = PCG (H, g , C, α)
Step 2: if posdef
            solve:  min_s{s^T g + ½ s^T H s :  ||s||_2 ≤ Δ, s ∈ S} where S = [g, p]
        else
            solve:  min_s{s^T g + ½ s^T H s :  ||s||_2 ≤ Δ, s ∈ S} where S = [g, d]
        end
```

Figure 8.2: The 2-dimensional subspace algorithm.

```
function [s] = trustS3(H, g, C, α, Δ)

% Note: C is some preconditioning matrix for H.
% C must be positive definite.


Step 1: [p, d, posdef] = PCG (H, g , C, α)
Step 2: if posdef
            solve:  min_s{s^T g + ½ s^T H s :  ||s||_2 ≤ Δ, s ∈ S} where S = [g, p]
        else
            solve:  min_s{s^T g + ½ s^T H s :  ||s||_2 ≤ Δ, s ∈ S} where S = [g, p, d]
        end
```

Figure 8.3: The 3-dimensional subspace algorithm.

function $[s] = Steihaug(H,\ g,\ C,\ \alpha,\ \Delta)$

% Note: C is some preconditioning matrix for H.
% C must be positive definite. $C = R^2$.

$n = length(g);$
$k = 0;\ p_0 = 0;\ r_0 = -g;$
$kmax = n/2;$

while $k < kmax$
  Step 1: Solve $Cz_k = r_k$;
  Step 2: $k = k + 1$
  Step 3: **if** $k = 1$
    $d_1 = z_0$
   else
    $\beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
    $d_k = z_{k-1} + \beta_k d_{k-1}$
   end
  Step 4: $\gamma_k = d_k^T H d_k$
  Step 5: **if** $\gamma_k \leq 0$
    compute $\tau > 0$ so that $\|p_{k-1} + \tau d_k\|_C = \Delta$,
    $s = p_{k-1} + \tau d_k$, **return**
   else
    $\alpha_k = r_{k-1}^T z_{k-1} / \gamma_k$
    $p_k = p_{k-1} + \alpha_k d_k$
    $r_k = r_{k-1} - \alpha_k H d_k$
   end
  Step 6: **if** $\|p_k\| \geq \Delta$
    compute $\tau > 0$ so that $\|p_{k-1} + \tau d_k\|_C = \Delta$,
    $s = p_{k-1} + \tau d_k$, **return**
  Step 7: **if** $\|R^{-1}\| \cdot \|r_k\| \leq \eta \|R^{-1} g\|$
    $s = p_k$, **return**
   end

end
$s = p_k$, **return**

Figure 8.4: The Steihaug algorithm.

function $[s] = Dogleg(H,\ g,\ C,\ \alpha,\ \Delta)$

% Note: C is some preconditioning matrix for H.
% C must be positive definite. $C = R^2$.

$n = length(g)$;
$k = 0$; $p_0 = 0$; $r_0 = -g$;
$kmax = n/2$;

Step 1: Compute $\sigma \;=\; g^T H g$
Step 2: **if** $\sigma <= 0$
    $s = -\tau g$ where $\|s\| = \Delta, \tau > 0$, **return**
  end
Step 3: compute $cauchylen = \frac{\|g\|^3}{\sigma}$
  if $cauchylen \geq \Delta$
    $s = -\tau g$ where $\|s\| = \Delta, \tau > 0$, **return**
  end
Step 4: $cauchy\_point = -\frac{\|g\|^2}{\sigma} g$
Step 5: $[p,\ d,\ posdef] = PCG\ (H,\ g\ ,\ C,\ \alpha)$
Step 6: **if** $posdef$
    if $\|p\| \leq \Delta$
      $s = p$, **return**
    else
      compute $\tau > 0$ so that $\|cauchy\_point + \tau p\| = \Delta$,
      $s = cauchy\_point + \tau p$, **return**
    end
Step 7: **else** % not positive definite
    if $d^T g > 0$, $d = -d$, **end**
    compute $\tau > 0$ so that $\|cauchy\_point + \tau d\| = \Delta$,
    $s = cauchy\_point + \tau d$, **return**
  end

Figure 8.5: The modified dogleg algorithm.

computed is based on the following rule. If the linear system we are solving is

$$Hp = -g. \tag{8.4}$$

then our stopping condition for our conjugate gradient algorithm is

$$\|r\| \le \eta \cdot \|g\|, \tag{8.5}$$

for some η. With preconditioning, the stopping condition is

$$\|R^{-1}\| \cdot \|r\| \le \eta \cdot \|R^{-1}g\|. \tag{8.6}$$

The same basic strategy is used to update Δ in all four methods. For the first iteration, we let $\Delta = \|g\|$ where the norm used is $\|\cdot\|_2$ for the dogleg and subspace methods and $\|\cdot\|_C$ for the Steihaug method.

8.2 Computational results

For each experiment we used twenty different test problems. All but four are test problems described in [CGT88b], but with all the bound constraints removed. The problems EROSENBR and EPOWELL are taken from [MGH81]. The last two problems, molecule problems MOLE1 and MOLE3, are described in [Hen91, Col93]. For all problems, the number of variables n is 260. The minimization algorithm terminates when $\|g\|_2 \le 10^{-6}$.

We first focus on comparing the SUBSP2 method and the STEIH method. Tables 8.1 and 8.2 compare the Steihaug and subspace methods described above in terms of the number of minimization iterations and the total number of conjugate gradient (CG) iterations. We also include *CG:IT==1*, which counts the number of times the method performs only one iteration of the conjugate gradient method to determine the solution of the trust-region problem. Table 8.1 shows problems for which negative curvature was not detected, and Table 8.2 shows problems for which negative curvature was detected. Although not included here, the function values and gradient norms (upon termination) were virtually the same for both methods for all problems. Since these values were essentially the same among the two methods, we only discuss the difference in iterations counts. The difference in minimization and CG iteration counts is plotted in Figure 8.6 and Figure 8.7.

Most notable in Table 8.1 and the graphs of Figure 8.6 is how strikingly similar the results are for the Steihaug and subspace methods; the minimization with each method stops within two iterations of the other in all cases. Furthermore, both methods take an *identical* number of total CG iterations except for the problem BROWN1 where the Steihaug method takes two more iterations. When negative curvature is encountered, shown in Table 8.2 and in Figure 8.7, the iteration counts

Table 8.1: Comparison when only positive curvature is encountered: number of iterations.

	Problem	Minimization SUBSP2	Minimization STEIH	CG SUBSP2	CG STEIH	CG:IT==1 SUBSP2	CG:IT==1 STEIH
1.	BROWN1	27	29	39	43	15	15
2.	BROWN3	6	6	6	6	6	6
3.	BROYDEN1A	11	11	81	81	0	0
4.	BROYDEN1B	5	5	34	34	0	0
5.	BROYDEN2B	7	7	71	71	0	0
6.	CHAINSING	22	22	188	188	0	0
7.	CRAGGLEVY	21	21	125	125	0	0
8.	DEGENSING	22	22	188	188	0	0
9.	EPOWELL	18	18	72	72	0	0
10.	GENSING	22	22	83	83	0	0
11.	TOINTBROY	7	7	58	58	0	0
12.	VAR	43	43	5590	5590	0	0

Table 8.2: Comparison when negative curvature is encountered: number of iterations

	Problem	Minimization SUBSP2	Minimization STEIH	CG SUBSP2	CG STEIH	CG:IT==1 SUBSP2	CG:IT==1 STEIH
1.	AUGMLAGN	36	29	267	228	0	1
2.	BROYDEN2A	22	19	247	196	0	0
3.	CHAINWOOD	156	988	3905	3878	5	244
4.	EROSENBR	44	46	52	86	0	522
5.	GENROSE	23	33	166	165	1	16
6.	GENWOOD	58	63	304	275	2	2
7.	MOLE1	46	119	460	376	4	80
8.	MOLE3	125	186	6311	5356	0	32

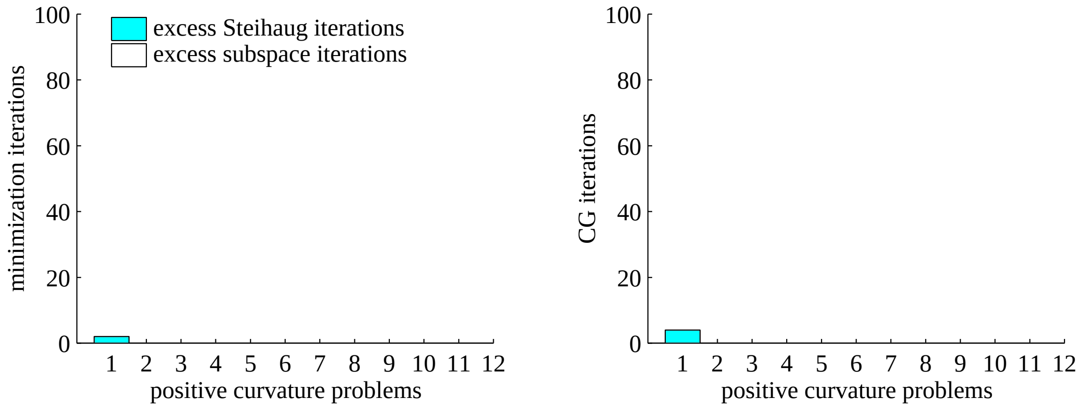

Figure 8.6: Comparison of SUBSP2 and STEIH trust-region methods for unconstrained problems.

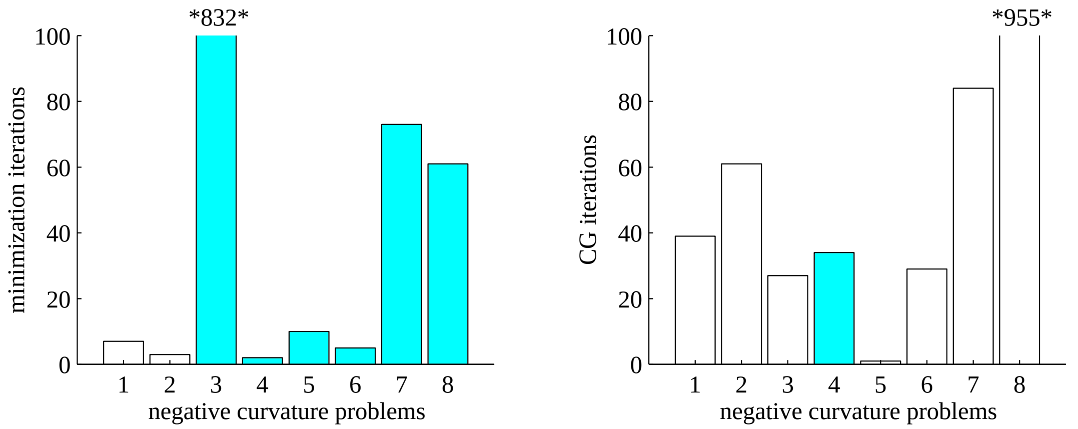

Figure 8.7: Comparison of SUBSP2 and STEIH trust-region methods for unconstrained problems.

for each method are again similar for a few problems. For most problems, however, the Steihaug method takes more iterations, and for some problems the difference is substantial. This is particularly true for the problems CHAINWOOD, MOLE1 and MOLE3 (for CHAINWOOD, problem 3 in Figure 8.7, the total difference in iteration counts is explicitly noted as it is beyond the scale of the graph). In general the subspace method does take more CG iterations on problems with negative curvature, but it is these extra, relatively inexpensive, CG iterations that reduce the total number of minimization iterations. (Again, for the problem MOLE3 the difference in CG iterations is explicitly noted in Figure 8.7 as it is beyond the scale of the graph.) Note that for all the problems in Table 8.2 except MOLE3, when the subspace method does take more CG iterations, it is only one or two CG iterations more per minimization iteration than the Steihaug method. For MOLE3, the cost is approximately ten extra CG iterations per minimization iteration. In other words, when negative curvature is encountered, the amortized cost of the extra CG iterations is relatively small in most cases.

In Tables 8.3 and 8.4, we compare the four algorithms described above in terms of number of iterations taken: *iter* is the number of minimization iterations, and *pcg* is the total number of iterations taken by the preconditioned conjugate gradient algorithm. Also, *neg.c* denotes if negative curvature was encountered, *cg tol* is the value of η used. We varied the accuracy of the inexact Newton direction by using $\eta = 0.5$, 0.05, 0.005, and 0.0005 (see (8.6)).

The remaining tables contain more detailed information for each method: *fun val* is the function value at the final point, and *grad nrm* is the norm of the gradient at the final point. Tables 8.5 and 8.6 give results for the 2-dimensional subspace method; tables 8.7 and 8.8 describe the 3-dimensional subspace method results; tables 8.9 and 8.10 list the Steihaug method results; and tables 8.11 and 8.12 describe the dogleg method.

8.3 Discussion

A closer examination of the behavior of the two algorithms SUBSP2 and STEIH indeed shows that when negative curvature is *not* encountered, both methods take similar steps. (In this case, if the trust region is large enough, both methods in Figure 8.4 and Figure 8.1 will stop under the same conditions after the same number of CG iterations, as displayed in Table 8.1.) By the nature of the algorithms, if the Steihaug method detects negative curvature, then so will the subspace approach. However if the subspace algorithm detects negative curvature, the Steihaug method may terminate before it finds negative curvature; and then it does not converge (to a local minimizer) as quickly as the subspace method. Note how often the Steihaug method takes but one CG iteration before stopping (see *CGIT==1* in

Table 8.3: Comparison of iteration counts for four methods.

			STEIH		SUBSP2		SUBSP3		DOGLEG	
Problem	cg tol	neg.c	iter	pcg	iter	pcg	iter	pcg	iter	pcg
AUGMLAGN	0.5	√	80	269	63	198	60	196	79	306
AUGMLAGN	0.05	√	34	182	45	227	29	129	46	276
AUGMLAGN	0.005	√	25	169	38	260	23	146	41	274
AUGMLAGN	0.0005	√	29	228	36	267	24	155	40	278
BROWN1	0.5		31	37	27	31	27	31	28	35
BROWN1	0.05		29	38	27	35	27	35	27	37
BROWN1	0.005		29	41	27	37	27	37	27	39
BROWN1	0.0005		29	43	27	39	27	39	27	41
BROWN3	0.5		6	6	6	6	6	6	6	6
BROWN3	0.05		6	6	6	6	6	6	6	6
BROWN3	0.005		6	6	6	6	6	6	6	6
BROWN3	0.0005		6	6	6	6	6	6	6	6
BROYDEN1A	0.5		13	24	15	26	15	26	13	24
BROYDEN1A	0.05		11	26	11	26	11	26	11	26
BROYDEN1A	0.005		11	52	11	52	11	52	11	52
BROYDEN1A	0.0005		11	81	11	81	11	81	11	81
BROYDEN1B	0.5		15	22	15	22	15	22	15	22
BROYDEN1B	0.05		7	21	7	21	7	21	7	21
BROYDEN1B	0.005		6	28	6	28	6	28	6	28
BROYDEN1B	0.0005		5	34	5	34	5	34	5	34
BROYDEN2A	0.5		17	39	17	37	17	37	17	39
BROYDEN2A	0.05		13	60	13	70	13	70	13	60
BROYDEN2A	0.005	√	19	147	22	190	23	190	22	212
BROYDEN2A	0.0005	√	19	196	22	247	22	251	22	290
BROYDEN2B	0.5		19	41	19	43	19	43	19	41
BROYDEN2B	0.05		10	50	10	52	10	52	10	50
BROYDEN2B	0.005		8	62	8	62	8	62	8	62
BROYDEN2B	0.0005		7	71	7	71	7	71	7	71
CHAINSING	0.5		27	71	26	69	26	69	27	71
CHAINSING	0.05		26	103	23	121	23	121	26	112
CHAINSING	0.005		22	148	22	148	22	148	22	148
CHAINSING	0.0005		22	188	22	188	22	188	22	188
CHAINWOOD	0.5	√	600	1202	600	2998	600	2372	600	3693
CHAINWOOD	0.05	√	600	1773	180	2763	316	3835	225	4792
CHAINWOOD	0.005	√	600	2218	178	3012	287	5018	161	4441
CHAINWOOD	0.0005	√	600	1675	160	4312	219	3815	186	5350
CRAGGLEVY	0.5		23	44	27	44	27	44	23	44
CRAGGLEVY	0.05		23	82	24	86	24	86	23	82
CRAGGLEVY	0.005		21	88	21	89	21	89	21	88
CRAGGLEVY	0.0005		21	125	21	125	21	125	21	125

Table 8.4: Comparison of iteration counts for four methods.

			STEIH		SUBSP2		SUBSP3		DOGLEG	
Problem	cg tol	neg.c	iter	pcg	iter	pcg	iter	pcg	iter	pcg
DEGENSING	0.5		27	71	26	69	26	69	27	71
DEGENSING	0.05		26	103	23	46	23	46	26	112
DEGENSING	0.005		22	148	22	148	22	148	22	148
DEGENSING	0.0005		22	188	22	188	22	188	22	188
EPOWELL	0.5		23	47	21	42	21	42	23	47
EPOWELL	0.05		18	63	18	66	18	66	18	63
EPOWELL	0.005		18	72	18	72	18	72	18	72
EPOWELL	0.0005		18	72	18	72	18	72	18	72
EROSENBR	0.5	$\surd$	85	135	74	48	74	48	93	140
EROSENBR	0.05	$\surd$	36	62	72	72	76	75	45	81
EROSENBR	0.005	$\surd$	600	671	74	84	75	86	88	167
EROSENBR	0.0005	$\surd$	600	678	74	84	75	86	88	167
GENROSE	0.5	$\surd$	54	82	35	63	37	71	47	86
GENROSE	0.05	$\surd$	28	106	24	87	26	91	29	120
GENROSE	0.005	$\surd$	31	119	24	135	23	132	21	139
GENROSE	0.0005	$\surd$	33	165	23	166	22	168	21	184
GENSING	0.5		41	82	31	64	31	64	42	100
GENSING	0.05		21	63	21	61	21	61	21	63
GENSING	0.005		22	78	22	76	22	76	22	78
GENSING	0.0005		22	83	22	83	22	83	22	83
GENWOOD	0.5	$\surd$	150	285	113	236	121	243	122	332
GENWOOD	0.05	$\surd$	95	254	59	210	58	177	62	256
GENWOOD	0.005	$\surd$	69	278	66	326	52	253	63	390
GENWOOD	0.0005	$\surd$	63	275	61	292	53	282	74	478
MOLE1	0.5	$\surd$	138	281	70	245	65	222	76	292
MOLE1	0.05	$\surd$	120	341	60	435	56	269	60	407
MOLE1	0.005	$\surd$	119	348	48	423	63	546	74	758
MOLE1	0.0005	$\surd$	119	376	46	460	65	635	70	857
MOLE3	0.5	$\surd$	341	2313	245	3703	258	3594	155	2664
MOLE3	0.05	$\surd$	196	2416	133	3566	177	6231	103	3320
MOLE3	0.005	$\surd$	184	4244	129	5121	160	8647	101	4939
MOLE3	0.0005	$\surd$	186	5356	131	6661	159	10655	98	6130
TOINTBROY	0.5		19	35	17	25	17	25	19	35
TOINTBROY	0.05		8	30	8	29	8	29	8	30
TOINTBROY	0.005		7	39	7	39	7	39	7	39
TOINTBROY	0.0005		7	58	7	58	7	58	7	58
VAR	0.5		58	3813	39	2392	39	2392	58	3813
VAR	0.05		51	6519	51	6519	51	6519	51	6519
VAR	0.005		43	5590	43	5590	43	5590	43	5590
VAR	0.0005		43	5590	43	5590	43	5590	43	5590

Table 8.5: Results for the SUBSP2 method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
AUGMLAGN	0.5	63	7	198	2.25e+01	8.39e-07
AUGMLAGN	0.05	45	15	227	2.25e+01	4.02e-07
AUGMLAGN	0.005	38	12	260	2.25e+01	6.15e-08
AUGMLAGN	0.0005	36	12	267	2.25e+01	4.44e-09
BROWN1	0.5	27	0	31	2.60e+01	1.14e-10
BROWN1	0.05	27	0	35	2.60e+01	1.14e-10
BROWN1	0.005	27	0	37	2.60e+01	1.14e-10
BROWN1	0.0005	27	0	39	2.60e+01	1.14e-10
BROWN3	0.5	6	0	6	1.55e-11	6.93e-07
BROWN3	0.05	6	0	6	1.55e-11	6.93e-07
BROWN3	0.005	6	0	6	1.55e-11	6.93e-07
BROWN3	0.0005	6	0	6	1.55e-11	6.93e-07
BROYDEN1A	0.5	15	0	26	1.00e+00	6.29e-07
BROYDEN1A	0.05	11	0	26	1.00e+00	4.41e-07
BROYDEN1A	0.005	11	0	52	1.00e+00	3.89e-07
BROYDEN1A	0.0005	11	0	81	1.00e+00	4.67e-07
BROYDEN1B	0.5	15	0	22	1.00e+00	4.36e-07
BROYDEN1B	0.05	7	0	21	1.00e+00	6.94e-07
BROYDEN1B	0.005	6	0	28	1.00e+00	7.21e-08
BROYDEN1B	0.0005	5	0	34	1.00e+00	7.10e-07
BROYDEN2A	0.5	17	0	37	1.36e-11	6.14e-07
BROYDEN2A	0.05	13	0	70	1.21e-12	4.31e-07
BROYDEN2A	0.005	22	7	190	4.18e-12	9.79e-07
BROYDEN2A	0.0005	22	9	247	1.58e-12	5.18e-07
BROYDEN2B	0.5	19	0	43	1.56e-13	6.24e-07
BROYDEN2B	0.05	10	0	52	1.57e-13	5.28e-07
BROYDEN2B	0.005	8	0	62	6.59e-16	4.86e-08
BROYDEN2B	0.0005	7	0	71	1.38e-13	6.15e-07
CHAINSING	0.5	26	0	69	1.38e-11	5.07e-07
CHAINSING	0.05	23	0	121	1.27e-11	3.03e-07
CHAINSING	0.005	22	0	148	1.46e-11	3.71e-07
CHAINSING	0.0005	22	0	188	1.55e-11	3.32e-07
CHAINWOOD	0.5	600	316	2998	2.08e+02	2.84e+00
CHAINWOOD	0.05	180	85	2763	7.93e+01	3.04e-07
CHAINWOOD	0.005	178	132	3012	1.17e+01	9.37e-08
CHAINWOOD	0.0005	160	115	4312	7.43e+01	1.58e-08
CRAGGLEVY	0.5	27	0	44	3.33e-09	3.75e-07
CRAGGLEVY	0.05	24	0	86	4.56e-09	6.65e-07
CRAGGLEVY	0.005	21	0	89	2.84e-07	7.89e-07
CRAGGLEVY	0.0005	21	0	125	2.84e-07	7.88e-07

Table 8.6: Results for the SUBSP2 method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
DEGENSING	0.5	26	0	69	9.49e-07	5.07e-07
DEGENSING	0.05	23	0	46	1.76e-01	2.50e+00
DEGENSING	0.005	22	0	148	7.70e-07	3.71e-07
DEGENSING	0.0005	22	0	188	7.15e-07	3.32e-07
EPOWELL	0.5	21	0	42	3.66e-08	4.64e-07
EPOWELL	0.05	18	0	66	3.19e-08	8.29e-07
EPOWELL	0.005	18	0	72	3.29e-08	8.50e-07
EPOWELL	0.0005	18	0	72	3.29e-08	8.50e-07
EROSENBR	0.5	74	51	48	5.77e-29	1.33e-15
EROSENBBR	0.05	72	49	72	9.71e-26	2.22e-14
EROSENBR	0.005	74	51	84	2.72e-27	3.55e-14
EROSENBR	0.0005	74	51	84	2.72e-27	3.55e-14
GENROSE	0.5	35	5	63	1.00e+00	9.48e-07
GENROSE	0.05	24	8	87	1.00e+00	9.62e-07
GENROSE	0.005	24	8	135	1.00e+00	1.99e-08
GENROSE	0.0005	23	8	166	1.00e+00	3.70e-07
GENSING	0.5	31	0	64	3.81e-07	7.60e-07
GENSING	0.05	21	0	61	1.89e-08	7.35e-07
GENSING	0.005	22	0	76	6.31e-10	3.19e-07
GENSING	0.0005	22	0	83	5.06e-10	3.27e-07
GENWOOD	0.5	113	35	236	1.00e+00	1.13e-07
GENWOOD	0.05	59	17	210	1.00e+00	2.06e-10
GENWOOD	0.005	66	20	326	1.00e+00	2.43e-07
GENWOOD	0.0005	61	22	292	1.00e+00	1.58e-09
MOLE1	0.5	70	38	245	1.13e+01	6.46e-07
MOLE1	0.05	60	38	435	1.14e+01	3.35e-07
MOLE1	0.005	48	32	423	1.14e+01	3.04e-09
MOLE1	0.0005	46	32	460	1.14e+01	6.82e-09
MOLE3	0.5	245	80	3703	7.26e+00	7.74e-07
MOLE3	0.05	133	33	3566	6.28e+00	7.10e-07
MOLE3	0.005	129	59	5121	4.48e+00	1.42e-07
MOLE3	0.0005	131	57	6661	4.48e+00	6.59e-09
TOINTBROY	0.5	17	0	25	8.83e+01	9.99e-07
TOINTBROY	0.05	8	0	29	8.83e+01	9.07e-07
TOINTBROY	0.005	7	0	39	8.83e+01	5.58e-08
TOINTBROY	0.0005	7	0	58	8.83e+01	2.65e-10
VAR	0.5	39	0	2392	-8.52e+00	8.06e-07
VAR	0.05	51	0	6519	-8.52e+00	8.88e-07
VAR	0.005	43	0	5590	-8.52e+00	8.67e-07
VAR	0.0005	43	0	5590	-8.52e+00	8.67e-07

Table 8.7: Results for the SUBSP3 method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
AUGMLAGN	0.5	60	9	196	2.25e+01	5.92e-07
AUGMLAGN	0.05	29	9	129	2.25e+01	2.13e-07
AUGMLAGN	0.005	23	9	146	2.25e+01	2.24e-07
AUGMLAGN	0.0005	24	9	155	2.25e+01	5.74e-11
BROWN1	0.5	27	0	31	2.60e+01	1.14e-10
BROWN1	0.05	27	0	35	2.60e+01	1.14e-10
BROWN1	0.005	27	0	37	2.60e+01	1.14e-10
BROWN1	0.0005	27	0	39	2.60e+01	1.14e-10
BROWN3	0.5	6	0	6	1.55e-11	6.93e-07
BROWN3	0.05	6	0	6	1.55e-11	6.93e-07
BROWN3	0.005	6	0	6	1.55e-11	6.93e-07
BROWN3	0.0005	6	0	6	1.55e-11	6.93e-07
BROYDEN1A	0.5	15	0	26	1.00e+00	6.29e-07
BROYDEN1A	0.05	11	0	26	1.00e+00	4.41e-07
BROYDEN1A	0.005	11	0	52	1.00e+00	3.89e-07
BROYDEN1A	0.0005	11	0	81	1.00e+00	4.67e-07
BROYDEN1B	0.5	15	0	22	1.00e+00	4.36e-07
BROYDEN1B	0.05	7	0	21	1.00e+00	6.94e-07
BROYDEN1B	0.005	6	0	28	1.00e+00	7.21e-08
BROYDEN1B	0.0005	5	0	34	1.00e+00	7.10e-07
BROYDEN2A	0.5	17	0	37	1.36e-11	6.14e-07
BROYDEN2A	0.05	13	0	70	1.21e-12	4.31e-07
BROYDEN2A	0.005	23	9	190	1.69e-13	2.17e-07
BROYDEN2A	0.0005	22	9	251	2.63e-12	8.89e-07
BROYDEN2B	0.5	19	0	43	1.56e-13	6.24e-07
BROYDEN2B	0.05	10	0	52	1.57e-13	5.28e-07
BROYDEN2B	0.005	8	0	62	6.59e-16	4.86e-08
BROYDEN2B	0.0005	7	0	71	1.38e-13	6.15e-07
CHAINSING	0.5	26	0	69	1.38e-11	5.07e-07
CHAINSING	0.05	23	0	121	1.27e-11	3.03e-07
CHAINSING	0.005	22	0	148	1.46e-11	3.71e-07
CHAINSING	0.0005	22	0	188	1.55e-11	3.32e-07
CHAINWOOD	0.5	600	329	2372	4.59e+02	1.37e+01
CHAINWOOD	0.05	316	214	3835	7.93e+01	4.95e-07
CHAINWOOD	0.005	287	218	5018	1.00e+00	7.55e-08
CHAINWOOD	0.0005	219	163	3815	1.53e+01	4.03e-08
CRAGGLEVY	0.5	27	0	44	3.33e-09	3.75e-07
CRAGGLEVY	0.05	24	0	86	4.56e-09	6.65e-07
CRAGGLEVY	0.005	21	0	89	2.84e-07	7.89e-07
CRAGGLEVY	0.0005	21	0	125	2.84e-07	7.88e-07

Table 8.8: Results for the SUBSP3 method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
DEGENSING	0.5	26	0	69	9.49e-07	5.07e-07
DEGENSING	0.05	23	0	46	1.76e-01	2.50e+00
DEGENSING	0.005	22	0	148	7.70e-07	3.71e-07
DEGENSING	0.0005	22	0	188	7.15e-07	3.32e-07
EPOWELL	0.5	21	0	42	3.66e-08	4.64e-07
EPOWELL	0.05	18	0	66	3.19e-08	8.29e-07
EPOWELL	0.005	18	0	72	3.29e-08	8.50e-07
EPOWELL	0.0005	18	0	72	3.29e-08	8.50e-07
EROSENBR	0.5	74	51	48	5.77e-29	1.33e-15
EROSENBR	0.05	76	53	75	1.94e-28	2.44e-15
EROSENBR	0.005	75	52	86	1.30e-28	2.00e-15
EROSENBR	0.0005	75	52	86	1.30e-28	2.00e-15
GENROSE	0.5	37	7	71	1.00e+00	5.79e-07
GENROSE	0.05	26	9	91	1.00e+00	3.96e-08
GENROSE	0.005	23	8	132	1.00e+00	5.26e-09
GENROSE	0.0005	22	8	168	1.00e+00	4.04e-09
GENSING	0.5	31	0	64	3.81e-07	7.60e-07
GENSING	0.05	21	0	61	1.89e-08	7.35e-07
GENSING	0.005	22	0	76	6.31e-10	3.19e-07
GENSING	0.0005	22	0	83	5.06e-10	3.27e-07
GENWOOD	0.5	121	30	243	1.00e+00	4.78e-07
GENWOOD	0.05	58	18	177	1.00e+00	4.57e-11
GENWOOD	0.005	52	9	253	1.00e+00	1.23e-11
GENWOOD	0.0005	53	21	282	1.00e+00	3.31e-07
MOLE1	0.5	65	45	222	1.13e+01	4.95e-07
MOLE1	0.05	56	47	269	1.13e+01	1.39e-07
MOLE1	0.005	63	49	546	1.13e+01	5.10e-09
MOLE1	0.0005	65	52	635	1.13e+01	9.92e-09
MOLE3	0.5	258	82	3594	7.26e+00	7.74e-07
MOLE3	0.05	177	53	6231	7.26e+00	1.95e-07
MOLE3	0.005	160	49	8647	7.26e+00	8.19e-09
MOLE3	0.0005	159	50	10655	7.26e+00	3.00e-07
TOINTBROY	0.5	17	0	25	8.83e+01	9.99e-07
TOINTBROY	0.05	8	0	29	8.83e+01	9.07e-07
TOINTBROY	0.005	7	0	39	8.83e+01	5.58e-08
TOINTBROY	0.0005	7	0	58	8.83e+01	2.65e-10
VAR	0.5	39	0	2392	-8.52e+00	8.06e-07
VAR	0.05	51	0	6519	-8.52e+00	8.88e-07
VAR	0.005	43	0	5590	-8.52e+00	8.67e-07
VAR	0.0005	43	0	5590	-8.52e+00	8.67e-07

Table 8.9: Results for the STEIH method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
AUGMLAGN	0.5	80	6	269	2.25e+01	9.94e-07
AUGMLAGN	0.05	34	3	182	2.25e+01	6.71e-09
AUGMLAGN	0.005	25	3	169	2.25e+01	5.17e-07
AUGMLAGN	0.0005	29	3	228	2.25e+01	2.37e-08
BROWN1	0.5	31	0	37	2.60e+01	4.03e-11
BROWN1	0.05	29	0	38	2.60e+01	4.26e-08
BROWN1	0.005	29	0	41	2.60e+01	1.05e-08
BROWN1	0.0005	29	0	43	2.60e+01	9.80e-09
BROWN3	0.5	6	0	6	1.55e-11	6.93e-07
BROWN3	0.05	6	0	6	1.55e-11	6.93e-07
BROWN3	0.005	6	0	6	1.55e-11	6.93e-07
BROWN3	0.0005	6	0	6	1.55e-11	6.93e-07
BROYDEN1A	0.5	13	0	24	1.00e+00	3.89e-07
BROYDEN1A	0.05	11	0	26	1.00e+00	4.70e-07
BROYDEN1A	0.005	11	0	52	1.00e+00	4.08e-07
BROYDEN1A	0.0005	11	0	81	1.00e+00	4.67e-07
BROYDEN1B	0.5	15	0	22	1.00e+00	9.54e-07
BROYDEN1B	0.05	7	0	21	1.00e+00	8.73e-07
BROYDEN1B	0.005	6	0	28	1.00e+00	6.75e-08
BROYDEN1B	0.0005	5	0	34	1.00e+00	7.92e-07
BROYDEN2A	0.5	17	0	39	8.64e-12	9.83e-07
BROYDEN2A	0.05	13	0	60	5.16e-13	1.86e-07
BROYDEN2A	0.005	19	6	147	4.22e-13	4.06e-07
BROYDEN2A	0.0005	19	6	196	4.03e-13	4.01e-07
BROYDEN2B	0.5	19	0	41	2.33e-13	5.82e-07
BROYDEN2B	0.05	10	0	50	2.32e-15	6.97e-08
BROYDEN2B	0.005	8	0	62	7.66e-16	5.27e-08
BROYDEN2B	0.0005	7	0	71	1.38e-13	6.13e-07
CHAINSING	0.5	27	0	71	1.04e-11	4.12e-07
CHAINSING	0.05	26	0	103	1.65e-11	5.82e-07
CHAINSING	0.005	22	0	148	1.46e-11	3.70e-07
CHAINSING	0.0005	22	0	188	1.55e-11	3.32e-07
CHAINWOOD	0.5	600	105	1202	7.13e+02	2.46e+01
CHAINWOOD	0.05	600	111	1773	3.91e+02	2.31e+01
CHAINWOOD	0.005	600	109	2218	4.07e+02	8.58e+00
CHAINWOOD	0.0005	600	135	1675	5.21e+02	1.43e+01
CRAGGLEVY	0.5	23	0	44	9.97e-08	5.16e-07
CRAGGLEVY	0.05	23	0	82	2.01e-08	4.99e-07
CRAGGLEVY	0.005	21	0	88	2.84e-07	7.89e-07
CRAGGLEVY	0.0005	21	0	125	2.84e-07	7.89e-07

Table 8.10: Results for the STEIH method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
DEGENSING	0.5	27	0	71	8.26e-07	4.12e-07
DEGENSING	0.05	26	0	103	1.04e-06	5.82e-07
DEGENSING	0.005	22	0	148	7.69e-07	3.70e-07
DEGENSING	0.0005	22	0	188	7.15e-07	3.32e-07
EPOWELL	0.5	23	0	47	8.03e-09	6.76e-07
EPOWELL	0.05	18	0	63	3.31e-08	8.55e-07
EPOWELL	0.005	18	0	72	3.29e-08	8.50e-07
EPOWELL	0.0005	18	0	72	3.29e-08	8.50e-07
EROSENBR	0.5	85	45	135	1.14e-10	1.00e-06
EROSENBR	0.05	36	18	62	2.08e-11	4.28e-07
EROSENBR	0.005	600	577	671	0.00e+00	0.00e+00
EROSENBR	0.0005	600	579	678	0.00e+00	0.00e+00
GENROSE	0.5	54	17	82	1.00e+00	1.38e-06
GENROSE	0.05	28	7	106	1.00e+00	5.50e-08
GENROSE	0.005	31	6	119	1.00e+00	6.32e-07
GENROSE	0.0005	33	6	165	1.00e+00	7.03e-10
GENSING	0.5	41	0	82	2.65e-08	7.09e-07
GENSING	0.05	21	0	63	2.90e-08	7.61e-07
GENSING	0.005	22	0	78	6.42e-10	3.21e-07
GENSING	0.0005	22	0	83	5.06e-10	3.27e-07
GENWOOD	0.5	150	26	285	1.00e+00	1.32e-07
GENWOOD	0.05	95	19	254	1.00e+00	8.37e-08
GENWOOD	0.005	69	12	278	1.00e+00	7.62e-12
GENWOOD	0.0005	63	13	275	1.00e+00	3.46e-08
MOLE1	0.5	138	27	281	1.10e+01	3.14e-07
MOLE1	0.05	120	24	341	1.10e+01	5.26e-07
MOLE1	0.005	119	24	348	1.10e+01	4.57e-07
MOLE1	0.0005	119	24	376	1.10e+01	3.21e-07
MOLE3	0.5	341	3	2313	6.71e+00	6.77e-07
MOLE3	0.05	196	3	2416	6.50e+00	5.24e-07
MOLE3	0.005	184	5	4244	6.50e+00	1.54e-08
MOLE3	0.0005	186	3	5356	6.50e+00	1.72e-09
TOINTBROY	0.5	19	0	35	8.83e+01	4.82e-07
TOINTBROY	0.05	8	0	30	8.83e+01	1.26e-07
TOINTBROY	0.005	7	0	39	8.83e+01	5.38e-08
TOINTBROY	0.0005	7	0	58	8.83e+01	2.69e-10
VAR	0.5	58	0	3813	-8.52e+00	8.14e-07
VAR	0.05	51	0	6519	-8.52e+00	8.90e-07
VAR	0.005	43	0	5590	-8.52e+00	9.66e-07
VAR	0.0005	43	0	5590	-8.52e+00	9.66e-07

Table 8.11: Results for the DOGLEG method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
AUGMLAGN	0.5	79	17	306	2.25e+01	8.61e-07
AUGMLAGN	0.05	46	17	276	2.25e+01	4.63e-07
AUGMLAGN	0.005	41	21	274	2.25e+01	2.27e-09
AUGMLAGN	0.0005	40	22	278	2.25e+01	2.63e-08
BROWN1	0.5	28	0	35	2.60e+01	1.78e-10
BROWN1	0.05	27	0	37	2.60e+01	2.97e-10
BROWN1	0.005	27	0	39	2.60e+01	1.30e-10
BROWN1	0.0005	27	0	41	2.60e+01	1.16e-10
BROWN3	0.5	6	0	6	1.55e-11	6.93e-07
BROWN3	0.05	6	0	6	1.55e-11	6.93e-07
BROWN3	0.005	6	0	6	1.55e-11	6.93e-07
BROWN3	0.0005	6	0	6	1.55e-11	6.93e-07
BROYDEN1A	0.5	13	0	24	1.00e+00	3.89e-07
BROYDEN1A	0.05	11	0	26	1.00e+00	4.70e-07
BROYDEN1A	0.005	11	0	52	1.00e+00	4.08e-07
BROYDEN1A	0.0005	11	0	81	1.00e+00	4.67e-07
BROYDEN1B	0.5	15	0	22	1.00e+00	9.54e-07
BROYDEN1B	0.05	7	0	21	1.00e+00	8.73e-07
BROYDEN1B	0.005	6	0	28	1.00e+00	6.75e-08
BROYDEN1B	0.0005	5	0	34	1.00e+00	7.92e-07
BROYDEN2A	0.5	17	0	39	8.64e-12	9.83e-07
BROYDEN2A	0.05	13	0	60	5.16e-13	1.86e-07
BROYDEN2A	0.005	22	8	212	3.52e-13	3.28e-07
BROYDEN2A	0.0005	22	8	290	4.26e-13	4.37e-07
BROYDEN2B	0.5	19	0	41	2.33e-13	5.82e-07
BROYDEN2B	0.05	10	0	50	2.32e-15	6.97e-08
BROYDEN2B	0.005	8	0	62	7.66e-16	5.27e-08
BROYDEN2B	0.0005	7	0	71	1.38e-13	6.13e-07
CHAINSING	0.5	27	0	71	1.04e-11	4.12e-07
CHAINSING	0.05	26	0	112	1.28e-11	4.81e-07
CHAINSING	0.005	22	0	148	1.46e-11	3.70e-07
CHAINSING	0.0005	22	0	188	1.55e-11	3.32e-07
CHAINWOOD	0.5	600	309	3693	3.35e+02	9.51e+00
CHAINWOOD	0.05	225	130	4792	1.11e+02	1.29e-07
CHAINWOOD	0.005	161	102	4441	2.22e+01	1.75e-07
CHAINWOOD	0.0005	186	126	5350	1.00e+00	4.79e-08
CRAGGLEVY	0.5	23	0	44	9.97e-08	5.16e-07
CRAGGLEVY	0.05	23	0	82	2.01e-08	4.99e-07
CRAGGLEVY	0.005	21	0	88	2.84e-07	7.89e-07
CRAGGLEVY	0.0005	21	0	125	2.84e-07	7.89e-07

Table 8.12: Results for the DOGLEG method: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
DEGENSING	0.5	27	0	71	8.26e-07	4.12e-07
DEGENSING	0.05	26	0	112	9.16e-07	4.81e-07
DEGENSING	0.005	22	0	148	7.69e-07	3.70e-07
DEGENSING	0.0005	22	0	188	7.15e-07	3.32e-07
EPOWELL	0.5	23	0	47	8.03e-09	6.76e-07
EPOWELL	0.05	18	0	63	3.31e-08	8.55e-07
EPOWELL	0.005	18	0	72	3.29e-08	8.50e-07
EPOWELL	0.0005	18	0	72	3.29e-08	8.50e-07
EROSENBR	0.5	93	51	140	8.78e-12	2.90e-07
EROSENBR	0.05	45	22	81	3.17e-12	1.67e-07
EROSENBR	0.005	88	71	167	1.60e-30	2.22e-16
EROSENBR	0.0005	88	71	167	1.60e-30	2.22e-16
GENROSE	0.5	47	16	86	1.00e+00	2.98e-07
GENROSE	0.05	29	14	120	1.00e+00	6.20e-08
GENROSE	0.005	21	7	139	1.00e+00	8.89e-07
GENROSE	0.0005	21	7	184	1.00e+00	3.67e-09
GENSING	0.5	42	0	100	1.01e-08	4.77e-07
GENSING	0.05	21	0	63	2.90e-08	7.61e-07
GENSING	0.005	22	0	78	6.42e-10	3.21e-07
GENSING	0.0005	22	0	83	5.06e-10	3.27e-07
GENWOOD	0.5	122	33	332	1.00e+00	7.18e-07
GENWOOD	0.05	62	19	256	1.00e+00	4.74e-08
GENWOOD	0.005	63	21	390	1.00e+00	9.58e-07
GENWOOD	0.0005	74	32	478	1.00e+00	6.30e-10
MOLE1	0.5	76	38	292	1.11e+01	7.30e-07
MOLE1	0.05	60	36	407	1.11e+01	9.59e-08
MOLE1	0.005	74	44	758	1.11e+01	1.07e-08
MOLE1	0.0005	70	40	857	1.11e+01	6.33e-09
MOLE3	0.5	155	53	2664	6.02e+00	9.07e-07
MOLE3	0.05	103	40	3320	5.35e+00	7.31e-08
MOLE3	0.005	101	44	4939	5.35e+00	6.79e-08
MOLE3	0.0005	98	40	6130	5.36e+00	1.69e-08
TOINTBROY	0.5	19	0	35	8.83e+01	4.82e-07
TOINTBROY	0.05	8	0	30	8.83e+01	1.26e-07
TOINTBROY	0.005	7	0	39	8.83e+01	5.38e-08
TOINTBROY	0.0005	7	0	58	8.83e+01	2.69e-10
VAR	0.5	58	0	3813	-8.52e+00	8.14e-07
VAR	0.05	51	0	6519	-8.52e+00	8.90e-07
VAR	0.005	43	0	5590	-8.52e+00	9.66e-07
VAR	0.0005	43	0	5590	-8.52e+00	9.66e-07

Table 8.2). Furthermore, it is when the trust-region radius Δ_k is small that the Steihaug method is most likely to stop early and miss negative curvature. Thus it appears that the effectiveness of the Steihaug algorithm decreases as problem nonlinearity increases.

We can further support these claims by looking at all four algorithms. It appears that the Steihaug method suffers from two shortcomings when negative curvature exists. When negative curvature is not encountered, the algorithms behave similarly: they take similar size steps in similar directions. If the subspace or dogleg algorithms detect negative curvature, the Steihaug method may terminate before it finds negative curvature. This seems to cause it to take worse steps than the other methods. Is the negative curvature direction so important? Recall that SUBSP3 minimizes over the subspace that includes the current estimate to the inexact Newton direction, which is similar to the direction the STEIH method computes, *and* the direction of negative curvature when it exists. But when presented with a subspace containing both these directions, the 3-dimensional subspace method SUBSP3 often moves in a direction with a substantial component in the negative curvature direction d at the iterations when the Steihaug method did not find the negative curvature direction d (when, up to this iteration, the algorithms were moving in the same directions at about the same rate). Also, if not detecting the negative curvature direction is a problem with the STEIH method, then using a smaller η (i.e. *cg tol*) should sometimes improve the Steihaug method and make it comparable to the other methods (as then the negative curvature directions are more likely to be found) – this indeed does happen (see problems AUGMLAGN, GENROSE, and GENWOOD in Tables 8.3 and 8.4).

The second problem is that even when the Steihaug method finds a direction of negative curvature, it chooses a step that may be longer or shorter than the other methods; and this stepsize is a less desirable choice than the steps selected by the other methods. At this time, it is not clear why the Steihaug method's metric for step length is lacking. To verify that the metric causes difficulty, one more experiment was conducted. For this experiment, the dogleg algorithm was used with the stepsize chosen according to $\|\cdot\|_C$ instead of $\|\cdot\|_2$ (i.e., the same metric as the Steihaug algorithm; also Δ is initialized according to this norm like the Steihaug method). The results are shown in Tables 8.13 and 8.14. Note that this method, for problems with negative curvature, is usually worse than the dogleg method using the 2-norm, but better than the Steihaug method (it only has one of the shortcomings of the Steihaug method, not both).

We briefly comment on the SUBSP2 and SUBSP3 methods. In comparing the SUBSP2 and SUBSP3 methods, we can see that there is no clear "winner". Sometimes SUBSP2 takes fewer iterations (e.g. CHAINWOOD), but then SUBSP3 may take fewer iterations on other problems (e.g. AUGMLAGN). Since the subspace

Table 8.13: Results for the DOGLEG method using $\|\cdot\|_C$: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
AUGMLAGN	0.5	83	15	252	2.25e+01	1.23e-06
AUGMLAGN	0.05	46	16	253	2.33e+01	6.74e-07
AUGMLAGN	0.005	39	16	248	2.33e+01	8.96e-07
AUGMLAGN	0.0005	46	19	339	2.33e+01	7.92e-08
BROWN1	0.5	31	0	34	2.60e+01	4.04e-11
BROWN1	0.05	29	0	35	2.60e+01	4.26e-08
BROWN1	0.005	29	0	38	2.60e+01	1.05e-08
BROWN1	0.0005	29	0	40	2.60e+01	9.80e-09
BROWN3	0.5	6	0	6	1.55e-11	6.93e-07
BROWN3	0.05	6	0	6	1.55e-11	6.93e-07
BROWN3	0.005	6	0	6	1.55e-11	6.93e-07
BROWN3	0.0005	6	0	6	1.55e-11	6.93e-07
BROYDEN1A	0.5	13	0	24	1.00e+00	3.89e-07
BROYDEN1A	0.05	11	0	26	1.00e+00	4.70e-07
BROYDEN1A	0.005	11	0	52	1.00e+00	4.08e-07
BROYDEN1A	0.0005	11	0	81	1.00e+00	4.67e-07
BROYDEN1B	0.5	15	0	22	1.00e+00	9.54e-07
BROYDEN1B	0.05	7	0	21	1.00e+00	8.73e-07
BROYDEN1B	0.005	6	0	28	1.00e+00	6.75e-08
BROYDEN1B	0.0005	5	0	34	1.00e+00	7.92e-07
BROYDEN2A	0.5	17	0	39	8.64e-12	9.83e-07
BROYDEN2A	0.05	13	0	60	5.16e-13	1.86e-07
BROYDEN2A	0.005	19	6	147	2.47e-13	2.10e-07
BROYDEN2A	0.0005	19	6	190	2.31e-13	2.11e-07
BROYDEN2B	0.5	19	0	41	2.33e-13	5.82e-07
BROYDEN2B	0.05	10	0	50	2.32e-15	6.97e-08
BROYDEN2B	0.005	8	0	62	7.66e-16	5.27e-08
BROYDEN2B	0.0005	7	0	71	1.38e-13	6.13e-07
CHAINSING	0.5	27	0	71	1.04e-11	4.12e-07
CHAINSING	0.05	24	0	122	1.93e-11	4.59e-07
CHAINSING	0.005	22	0	148	1.46e-11	3.70e-07
CHAINSING	0.0005	22	0	188	1.55e-11	3.32e-07
CHAINWOOD	0.5	600	333	2913	4.41e+02	9.27e+00
CHAINWOOD	0.05	310	143	4126	1.00e+00	1.29e-07
CHAINWOOD	0.005	436	235	4158	1.00e+00	2.49e-07
CHAINWOOD	0.0005	448	212	4791	1.00e+00	1.34e-09
CRAGGLEVY	0.5	23	0	44	9.97e-08	5.16e-07
CRAGGLEVY	0.05	23	0	82	2.01e-08	4.99e-07
CRAGGLEVY	0.005	21	0	88	2.84e-07	7.89e-07
CRAGGLEVY	0.0005	21	0	125	2.84e-07	7.89e-07

Table 8.14: Results for the DOGLEG method using $\|\cdot\|_C$: n=260.

Problem	cg tol	iter	neg.c	pcg	fun val	grad nrm
DEGENSING	0.5	27	0	71	8.26e-07	4.12e-07
DEGENSING	0.05	24	0	47	1.99e-01	2.58e+00
DEGENSING	0.005	22	0	148	7.69e-07	3.70e-07
DEGENSING	0.0005	22	0	188	7.15e-07	3.32e-07
EPOWELL	0.5	23	0	47	8.03e-09	6.76e-07
EPOWELL	0.05	18	0	63	3.31e-08	8.55e-07
EPOWELL	0.005	18	0	72	3.29e-08	8.50e-07
EPOWELL	0.0005	18	0	72	3.29e-08	8.50e-07
EROSENBR	0.5	67	37	95	3.77e-11	5.76e-07
EROSENBR	0.05	48	28	82	3.38e-12	1.72e-07
EROSENBR	0.005	600	579	159	0.00e+00	0.00e+00
EROSENBR	0.0005	600	579	159	0.00e+00	0.00e+00
GENROSE	0.5	64	22	98	1.00e+00	4.16e-07
GENROSE	0.05	26	9	129	1.00e+00	3.32e-07
GENROSE	0.005	25	9	149	1.00e+00	7.88e-08
GENROSE	0.0005	24	7	205	1.00e+00	5.02e-10
GENSING	0.5	37	0	84	3.38e-08	7.32e-07
GENSING	0.05	21	0	63	2.90e-08	7.61e-07
GENSING	0.005	22	0	78	6.42e-10	3.21e-07
GENSING	0.0005	22	0	83	5.06e-10	3.27e-07
GENWOOD	0.5	140	56	371	1.00e+00	8.91e-10
GENWOOD	0.05	76	26	362	1.00e+00	6.99e-07
GENWOOD	0.005	68	16	430	1.00e+00	1.79e-09
GENWOOD	0.0005	71	26	433	1.00e+00	4.71e-13
MOLE1	0.5	97	34	294	1.12e+01	8.03e-07
MOLE1	0.05	88	40	452	1.12e+01	6.32e-07
MOLE1	0.005	97	48	572	1.12e+01	6.90e-08
MOLE1	0.0005	97	48	647	1.12e+01	4.81e-09
MOLE3	0.5	164	46	2180	5.40e+00	4.46e-07
MOLE3	0.05	112	41	2958	5.40e+00	5.55e-07
MOLE3	0.005	107	42	4180	5.40e+00	2.71e-07
MOLE3	0.0005	103	39	5132	5.40e+00	1.69e-07
TOINTBROY	0.5	19	0	35	8.83e+01	4.82e-07
TOINTBROY	0.05	8	0	30	8.83e+01	1.26e-07
TOINTBROY	0.005	7	0	39	8.83e+01	5.38e-08
TOINTBROY	0.0005	7	0	58	8.83e+01	2.69e-10
VAR	0.5	50	0	3320	-8.52e+00	9.49e-07
VAR	0.05	51	0	6519	-8.52e+00	8.79e-07
VAR	0.005	45	0	5850	-8.52e+00	5.17e-07
VAR	0.0005	45	0	5850	-8.52e+00	5.17e-07

of SUBSP2 is contained in the one used by SUBSP3, it would seem reasonable that SUBSP3 would converge in fewer iterations. However, sometimes SUBSP3 appears to encounter more nonlinearity with its choice of steps, and so shrinks the trust-region bound Δ_k more than SUBSP2. This appears to allow SUBSP2 to take larger steps as a minimizer is approached, since its Δ_k is larger, and so converge quicker.

We are left with the following questions. First, is it possible to prove that these methods, particularly the STEIH and SUBSP2 algorithms, are "equivalent" when only positive curvature is encountered? Certainly when the conditions in Lemma 5.2.1 hold, that is if the trust-region bound is large enough and our Hessian is positive definite, then the trust-region step computed by the SUBSP2 method is exactly the inexact Newton step, which is also the step computed by the Steihaug method (assuming again that the trust-region bound is large enough). Also, what causes the Steihaug method to take undesirable step sizes when negative curvature is found, that is, why is the norm it uses a bad metric for negative curvature? The norm is based on the preconditioning matrix; would a different preconditioner result in better step sizes? Finally, can we more precisely explain why SUBSP3 is sometimes deficient compared to SUBSP2? And can this deficiency be rectified in a way that using more directions in our subspace trust-region algorithm is always an advantage?

Chapter 9

Parallel STIR

We investigate a parallel implementation (PSTIR) of the STIR algorithm. The algorithm behavior, in terms of speed, scalability and efficiency, depends mainly on the parallel matrix-vector multiplication code. Given good behavior in the multiplication code, we can expect good overall behavior of PSTIR.

To demonstrate this claim, we shall present an efficient matrix-vector multiply code for a Hessian of specific form. We shall see that this code is both efficient and scalable. We also consider the efficiency and scalability of a distributed dot product computation. Since the matrix-vector and dot products are the bulk of the computation in our conjugate gradient routine MPCG, our parallel MPCG code has scalability and efficiency in accordance with these two operations. Also, because the MPCG code dominates the optimization (PSTIR) code, we expect our overall optimization algorithm to also have speedup and efficiency proportional to the matrix-vector and dot product timings.

The PSTIR algorithm is equivalent to STIR in that it finds the same minimizer given the same stopping conditions. Also, the implementation is straightforward; unlike active-set or projected gradient methods, there is no need to load-balance the processors as the working set changes. Furthermore, the characteristic low number of optimization iterations for STIR (PSTIR) ensures scalable performance even as the number of variables and constraints increases. In other words, as long as the matrix-vector multiply and dot products scale well, we can expect PSTIR to scale well since the number of optimization iterations is unlikely to increase substantially with problem size.

Finally we will compare PSTIR as implemented on the IBM SP2 to competing algorithms on the CM-2, Intel iPSC/860 and the IBM 3090, as reported in [MMZ95].

9.1 Computational issues

The main computation time for PSTIR is spent in the conjugate gradient iterations. These iterations are composed mainly of dot products, SAXPY's, and matrix-vector multiplies. (A SAXPY operation computes $z = \alpha x + y$ where z, x, and y are vectors and α is a scalar.) Only the dot products and matrix-vector multiplies require communication when distributed among different processors, so we focus our attention on these operations.

The rest of the PSTIR code is made up of either "embarrassingly parallel" operations, such as SAXPY's, or more dot products and matrix-vector multiplies. Although less dominant in the overall computation time, we briefly review the computation needed in the non-MPCG part of PSTIR.

Once a direction is determined from the MPCG algorithm, we solve the two-dimensional subspace problem. We choose to solve this problem sequentially due to its trivial size and time requirements; in fact, we actually call a Matlab M-file that solves this subspace trust-region problem within a background Matlab process running under control of the root process.

The next step is to determine the minimizer along the reflected-dogleg path (see Section 3.3.2). This requires evaluating the quadratic model once for each segment of this path. Thus we compute several dot products and matrix-vector multiplies to evaluate the quadratic model.

The other communication necessary is the broadcasting or "reduction" of various scalar quantities needed to determine the next step in the algorithm.

9.2 Parallel architecture and software specifications

The code is written in XLF fortran version 3.2, compiled with the optimization flags `-O2` and `-qarch=pwr2`. Whenever possible we use the ESSL (Engineering and Scientific Subroutine Library [IBM92]) routines provided by IBM. For the parallel communication, we use the Message Passing Interface library MPI-F [Fra95,Mes93] version 1.41. As discussed above, we call Matlab 4.2c in the background by the root process.

The IBM SP2 used in our experimental runs consists of 512 nodes that are standard POWER2 architecture RS/6000 processors. These processors have superscalar pipelined chips capable of executing 4 floating point operations per cycle. The processors run at 66.7 MHz, giving a peak performance per processor of 266 MFLOPS. The SP2 configuration includes two types of nodes, known as thin nodes and wide nodes. Thin nodes, roughly equivalent to an RS/6000 Model 390, have 128 MBytes memory (a few have 256 MBytes) and 64 KBytes data cache, with 2

GBytes disk space each. Wide nodes have memories that range from 256 MBytes to 2GBytes, 256 KBytes data cache, and up to 18 GBytes of disk space. Although the clock speeds for the two types of nodes are the same, the differences in cache and bus sizes result in slower performance on thin nodes than on wide nodes for computationally intensive applications. For our experiments, we note that we use the thin nodes (mainly due to greater availability).

9.3 Data distribution and computation

We discuss how the data is distributed and how the two main computations are implemented.

Our communication model is that the processors are laid out in a linear array, with each processor having a left and right neighbor. Our Hessian-vector multiply is implemented without forming the Hessian, so this matrix distribution is avoided. For all vector operations, including dot product, each vector is distributed so that each section of n/p elements is local to a given processor. That is, if vector x has n elements and the processors are numbered from 0 to $(p-1)$, then processor k has elements $k \cdot \frac{n}{p} + 1$ through $(k+1) \cdot \frac{n}{p}$.

9.3.1 Dot product

To compute the dot product $x^T y$ we first compute the dot product of x and y of the elements local to processor p. Then we compute a global sum (using `MPI_REDUCE`) of each local dot product to give the full dot product $x^T y$. Each local dot product is computed using the IBM ESSL routine `DDOT`.

9.3.2 Matrix-vector multiply

Each of the problems we test is a quadratic problem and so has a constant Hessian matrix. These problems have a Hessian matrix H with a special structured block tridiagonal form. The superdiagonal and subdiagonal blocks of H are the identity matrix $I \in \Re^{m \times m}$. The diagonal blocks are tridiagonal matrices with each diagonal element equal to 4, and each superdiagonal and subdiagonal element equal to -1. All blocks of H are m by m where $n = m^2$. The bandwidth of the matrix H is m.

Since this Hessian matrix has a small number of nonzeros per row (at most five), a general matrix-vector multiplication routine does not give the full scalability available if this structure is exploited. We choose not to form this Hessian matrix due to its regular structure (it is derived from a finite element approximation using a 5-point grid), but provide instead a subroutine which computes the product of the Hessian with a given vector. For each element of the product $(Hx)_i$, we need at most five elements of x: x_i, x_{i-1}, x_{i+1}, x_{i-m}, and x_{i+m}.

Thus the communication needed to compute the Hessian vector product is m elements of x from the right neighbor and m elements from the left neighbor processors, where m is the bandwidth of the matrix. Note that no matter how many processors we have, each processor only communicates with its two neighbors. And the size of data to be exchanged with each neighbor is always m. This allows our Hessian-vector multiplication routine good scalability and efficiency.

9.4 Obstacle problem

We test one of the obstacle problems as described in [MT89a]. These problems are quadratic

$$q(x) = \tfrac{1}{2}x^T Hx + c^T x$$

where H is as described in Section 9.3.2 and $c = -\left(\frac{1}{m+1}\right)^2$, where $m^2 = n$.

For obstacle problem I the bounds are

$$l_i = (\sin(9.2\alpha_i) \cdot \sin(9.3\gamma_i))^3$$

$$u_i = (\sin(9.2\alpha_i) \cdot \sin(9.3\gamma_i))^2 + 0.02.$$

These are based on the coefficients defined by

$$\alpha_i = (i - \lfloor \frac{i-1}{m} \rfloor \cdot m) \cdot (\frac{1}{m+1})$$

$$\gamma_i = \lceil \frac{i}{m} \rceil \cdot (\frac{1}{m+1})$$

for $i = 1, 2, ..., n$.

Our starting point x_0 is the midpoint of l and u:

$$(x_0)_i = (l_i + u_i)/2.$$

9.5 Computational results

For all timings we give wallclock time. The observed speedup is defined as

$$\textit{speedup} = \frac{\text{time for serial program}}{\text{time for parallel program}}$$

and we also define efficiency as

$$\textit{efficiency} = \frac{\text{speedup}}{\text{number of processors}}.$$

First we show the scalability of the of the dot product and matrix-vector product for problems of size $n = 160000$ and $n = 2560000$. We note that the dot product requires $2n$ floating point operations (flops). (We count adds and multiplications separately.)

For the matrix-vector multiply, the number of flops is nnz where nnz is the number of nonzeros in our Hessian matrix. Since each row of H consists of one 4 and three or four -1's, a way to view this flop count is that to compute each element $(Hx)_i$, we do one multiplication followed by three or four subtractions, depending on if H has 4 or 5 elements in that row. Thus the number of flops "per row" of H is the number of nonzeros in that row. Then the total number of flops is nnz.

Table 9.1 gives the expected scalability of the dot product: the megaflop rate, speedup and efficiency are shown for two problem sizes. The flop rate is giving in megaflops per second. We see that the performance is much better for the larger problem since more computation is done for each communication (global sum). Also, the efficiency deteriorates as the number of processors goes up due to the global sum.

Table 9.1: Dot product: wallclock timings on IBM SP2.

n	p	Mflops	speedup	efficiency
160000	1	27	1	1
	2	47	1.74	0.87
	4	93	3.44	0.86
	8	155	5.74	0.72
	16	230	8.51	0.53
	32	355	13.15	0.41
	64	422	15.63	0.24
	128	223	8.26	0.06
2560000	1	26	1	1
	2	49	1.88	0.94
	4	102	3.92	0.98
	8	194	7.46	0.93
	16	400	15.38	0.96
	32	694	26.69	0.83
	64	796	18.30	0.29
	128	1380	53.08	0.41

Table 9.2 gives the expected scalability of the matrix-vector product. Again, the megaflop rate, speedup and efficiency are shown for two problem sizes. The

overall efficiency is better than for the dot product, and it deteriorates less as processor number grows. Note for the larger problem, we get efficiency greater than unity for all processor counts greater than one. It is likely that once we go to two or more processors, the local data fits into cache and performance improves for each local processor computation.

Table 9.2: Hessian-vector product: wallclock timings on IBM SP2 for obstacle problem I Hessian.

n	p	Mflops	speedup	efficiency
160000	1	14	1	1
	2	27	1.93	0.96
	4	54	3.86	0.96
	8	102	7.29	0.91
	16	178	12.71	0.79
	32	310	22.14	0.69
	64	478	34.14	0.53
	128	545	38.93	0.30
2560000	1	9	1	1
	2	29	3.22	1.61
	4	59	6.56	1.64
	8	112	12.44	1.56
	16	244	27.11	1.69
	32	398	44.22	1.38
	64	713	79.22	1.24
	128	1204	133.78	1.05

Next we give some timings for the MPCG parallel code and the PSTIR algorithm. To make comparisons against several other implementations, we use the following stopping condition for PSTIR:

$$\|D_k \; g(x_k)\|_\infty \leq \tau_1 \cdot \|D_0 \; g(x_0)\|_\infty,$$

where $\tau_1 = 10^{-5}$ and $g(x_k)$ is the gradient of our quadratic $g(x_k) = H_k x_k + c$, and x_0 is our starting point. Also, for the stopping criteria for MPCG (see (4.4) in Figure 4.1) we use $\eta = 0.5$.

Table 9.3 shows the scalability of the conjugate gradient routine. This table gives the total (wallclock) time in seconds for PSTIR, total speedup, the average time for one MPCG iteration and speedup, and the percentage of the total time of PSTIR spent in the MPCG routine. (Speedup is not given for the larger problem as it exceeded memory requirements when run on one processor.) Here we can see the correlation between speedup of the total PSTIR algorithm and the speedup of the average time for one MPCG iteration.

Table 9.3: Parallel MPCG: wallclock timings on IBM SP2 for obstacle problem I. For $n = 160000$, solved in 14 iterations and 254 CG iterations; for $n = 2560000$, solve in 19 iterations and 1007 CG iterations.

n	p	total time	total speedup	avg time 1 MPCG iter	MPCG speedup	% time in MPCG
160000	1	245.731	1.00	.278	1.00	72.447 %
	2	128.304	1.92	.146	1.90	72.577 %
	4	66.576	3.69	.074	3.76	71.271 %
	8	34.713	7.08	.038	7.31	69.418 %
	16	19.358	12.69	.020	13.90	66.370 %
	32	12.238	20.08	.012	23.17	63.149 %
	64	8.116	30.28	.007	39.71	56.581 %
	128	19.315	12.72	.020	13.90	65.729 %
2560000	8	3722.260		1.214		32.834 %
	16	354.077		.294		83.509 %
	32	189.293		.157		83.677 %
	64	105.405		.086		81.748 %
	128	62.168		.051		82.323 %

We further explore the relationship between the scalability of PSTIR, MPCG, the matrix-vector product routine and the dot product routine in Figures 9.1 and 9.2. For both speedup and efficiency, the correlation between PSTIR and MPCG is clear. And the combined behavior of the dot and matrix-vector products corresponds to MPCG's behavior; that is as the dot product speedup decreases but the matrix-vector product speedup improves, MPCG is dominated by the dot-product behavior for problems in this size range of $n = 160000$. For larger problems since the dot-product efficiency is better, we would expect better efficiency for MPCG and thus PSTIR.

Next in Table 9.4, we give the timings in wallclock seconds for various size problems all run with 16 processors. We also include the number of PSTIR iterations and the number of MPCG iterations required. For the two largest problems, the number of processors was increased to 32 and 64 respectively to allow enough memory for the problem to be solved. Note that from the last three lines in the table we can see that if processor number grows with problem size, the total execution time remains fairly close.

We now compare our method to results in the paper [MMZ95] for this obstacle problem solved by other box-constrained quadratic (BQP) algorithms on

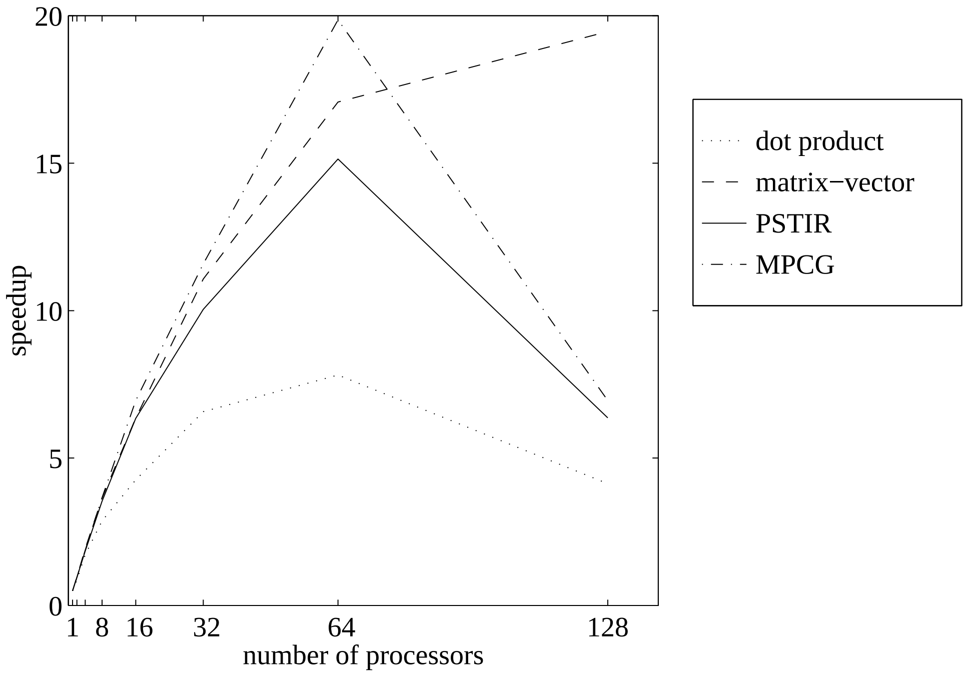

Figure 9.1: Comparison of speedup for n= 160000.

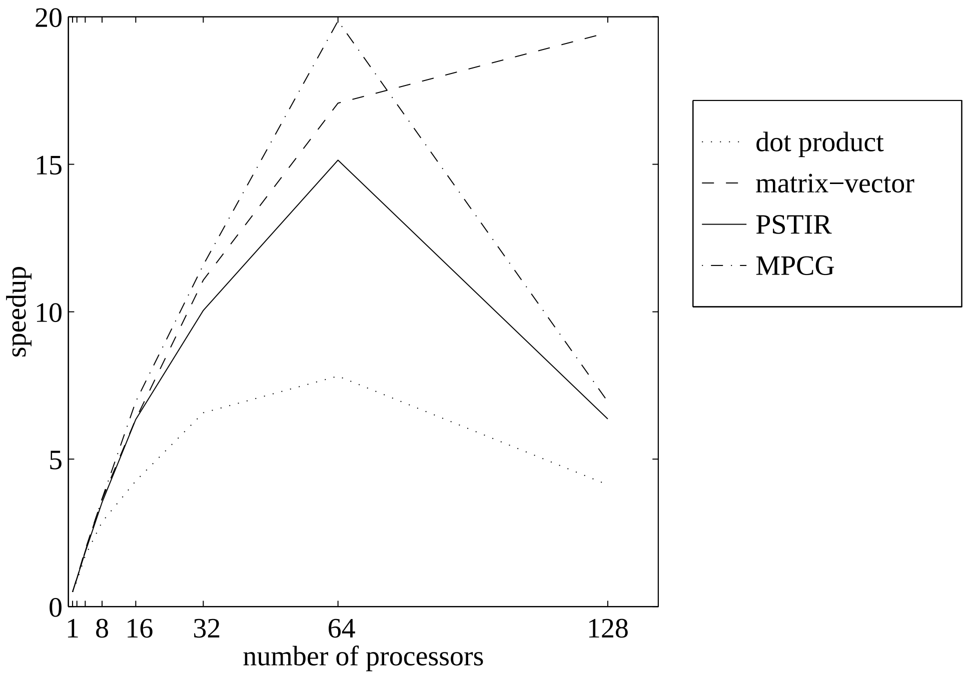

Figure 9.2: Comparison of efficiency for n= 160000.

various other architectures. The first algorithm is a parallel implementation of GPCG [MMZ95] based on the GPCG (projected gradient) algorithm by Moré and Toraldo [MT89a]. The second is a specialized iterative successive over-relaxation (SOR) method for BQP developed by R. De Leone [MMZ95] implemented on an Intel iPSC/860. The last algorithm is an primal-dual interior point method for BQP by Han, Pardalos, and Ye [HPY90] run on an IBM 3090-600S vector supercomputer.

We first give the performance characteristics of each machine in terms of (approximate) cycle time and peak computing rate. The IBM SP2 has a clock cycle time of 15 nsec, and each node has a peak computing rate of 266 MFLOPS. The CM-2 has a cycle time of 140 nsec and a peak computing rate of 14 MFLOPS per node. The IBM 3090 with vector units has a cycle time of 15 nsec. Each of its six processors has a peak computing rate of 116 MFLOPS. A single node of the Intel iPSC/860 has a peak computing rate of 40 MFLOPS.

It is difficult to compare different algorithms across different architectures. Certainly, we have the advantage since the SP2 would be considered the more powerful of the four architectures. We choose to include this comparison to first confirm that PSTIR on the SP2 is faster, as would be expected if the implementation is good, and also to display the relative speedup of the various algorithms and architectures. We do note, however, that the 8K CM-2 has 256 floating point processors, while in this comparison we use only 16 nodes of the SP2, each with exactly one floating point processor.

The results are given in Table 9.5. It is important to note that while the IBM SP2 timings are given in wallclock time, the other timings are given in CM or CPU seconds, which is usually less than the actual elapsed (wallclock) time. Despite this, the IBM SP2 timings for only 16 processors are consistently better than for the other architectures and algorithms. Clearly, we could have chosen a higher number of processors such as 64 and come out further ahead.

9.6 Discussion

The core of the PSTIR algorithm is the parallel MPCG algorithm, whose performance is guided by the matrix-vector and dot product computation performance. We have seen that the scalability of PSTIR depends on these sub-computations. Also, though difficult to compare across different architectures, we can see that we are quite competitive with other existing algorithms and their parallel/vector implementations even with just 16 processors.

Furthermore, the PSTIR algorithm implementation is straightforward. Only the matrix-vector multiply requires careful programming design to ensure scalability.

Table 9.4: PSTIR: wallclock timings on IBM SP2 with 16 processors for obstacle problem I.

n	p	iter	total time	avg time 1 MPCG iter	MPCG iter	% time in MPCG
10000	16	15	1.672	.510	171	26.667 %
40000	16	16	2.990	.006	271	51.951 %
90000	16	22	9.022	.013	438	62.395 %
115600	16	19	9.783	.016	418	66.373 %
160000	16	26	19.358	.020	638	70.209 %
250000	16	29	34.334	.031	830	74.204 %
360000	16	27	49.479	.044	870	76.970 %
2890000	16	18	357.349	.313	963	84.263 %
8000000	16	17	9124.672	2.685	1040	30.602 %
16000000	32	17	8759.070	2.468	1172	33.018 %
32000000	64	17	9450.211	2.585	1262	34.523 %

Table 9.5: Solving the obstacle problem I with PSTIR on the IBM SP2 with 16 processors, with the massively parallel implementation of GPCG on the CM-2, with the parallel successive over-relaxation of De Leone on the Intel iPSC/860, and the vector implementation of interior point algorithms of Han et al. on the IBM 3090-600S. Times are in wallclock seconds for the SP2, CM seconds for the CM-2, and CPU seconds for the iPSC and IBM 3090. * was solved on a 32K CM-2. NA: not available.

n	IBM SP2 (p=16) wallclock secs	CM-2 (8K) CM secs	Intel iPSC CPU secs	IBM 3090 CPU secs
10000	1.672	2.39	10.5	16.3
40000	2.990	5.03	49.1	131.1
90000	9.022	11.12	112.8	437.6
115600	9.783	15.34	156.7	700.3
160000	18.405	22.97	245.9	1035.8
250000	34.334	39.86	518.9	2110.5
360000	49.479	59.87	1047.6	4090.3
2890000	357.349	*482.00	NA	NA

We implemented a matrix-vector multiply that was specific to a particular Hessian matrix. However, even if the matrix-vector multiply was more general, and thus less scalable, the PSTIR performance would still be dependent on the performance of the multiply code. In fact, it would be more dependent since the extra communication necessary for a more general multiply would dominate the time as problem size grows.

The scalability is possible because our iteration count stays low. In other words, since the iteration count does not grow with problem size, it does not interfere with the scalability of the algorithm. Furthermore, since we have consistent, low iteration counts, if the processor count is increased with problem size, we can keep approximately the same execution time.

Finally, we note that had we implemented SLIR instead of STIR, we believe that due to the similar computation required in the reflective line search for SLIR (again, dot products and matrix-vector multiplies) the performance results, i.e. scalability, would be quite similar to that shown here for STIR.

Chapter 10

Conclusion

The inexact reflective Newton methods are computationally strong. These methods compare very well to their "exact" counterparts. For both the exact and the inexact methods, our experimental evidence shows

- the number of iterations is typically uncorrelated with the problem size,
- the number of iterations stays low when the starting point is varied.

Also, in comparing the inexact to the exact methods

- the speedup in floating point operations can be quite high, depending on the structure and size of the Hessian.

Furthermore, the nature of the computation in these methods permits porting to a parallel architecture such as the IBM SP2 with good results: the speedup is favorable assuming a good parallel matrix-vector multiply is available. Moreover, the inexact reflective Newton methods compare well to gradient projection (active-set) methods that use inexact Newton ideas. In this arena, one of the strengths of the reflective Newton methods is exposed: their ability to discover and take advantage of negative curvature.

For problems where negative curvature is encountered, the inexact reflective Newton method STIR often has lower iteration counts than a comparable gradient projection method, e.g. LANCELOT [CGT92]. This behavior is certainly due in part to the subspace trust-region method used by STIR. On unconstrained problems with negative curvature, it is clear that the subspace method performs more favorably than the Steihaug inexact trust-region method [Ste83] used by LANCELOT. Furthermore, since active-set type methods carry bound constraints into their subproblems, it is not clear how they *could* use the subspace method instead.

It is important to note that the results in Chapter 8 were for *unconstrained* problems. For the Steihaug method, recall that the trust-region bound causes the conjugate gradient iteration to quit early and miss negative curvature. For subproblems of gradient projection (active-set) methods, not only will the trust-region bound reduce the chance of finding negative curvature, the presence of the bound constraints is also likely to interrupt the conjugate gradient computation early. That is, when bound constraints are also present in the subproblem, the behavior of the Steihaug method is likely to be even worse than for unconstrained problems, where only the trust-region bound is present.

Why do these inexact reflective Newton algorithms preserve so well the properties of the exact reflective Newton methods? Consider the convergence results. First, we proved first-order convergence to a local minimizer for both inexact SLIR and STIR. We showed that the steps computed by the algorithm have *large-step-size* and (for the SLIR method) *consistency*. That these properties hold is *independent* of the residual bound on the inexact Newton step; that is, these properties are no less true for the inexact methods than for the exact methods. And it is the large-step-size property that keeps our iterates centered and progressing in every dimension towards the correct bounds and a local minimizer.

For the STIR method, we only needed "sufficient Cauchy decrease" to prove first-order convergence. It may seem odd that large-step-size was not explicitly required. Certainly the Cauchy step has large-step-size. If we compute steps that are as good if not better than the (possibly truncated) Cauchy step, we are doing as well as steps that do have large-step-size and this is sufficient.

We did not show convergence to a second-order point for either method. However, for both methods, the *only* missing ingredient is that the computed steps have "sufficient" negative curvature when the Hessian is indefinite. We chose not to expend the computation to find such steps beyond what the conjugate gradient algorithm provides. But the conjugate gradient negative curvature directions do compare to what is found using a modified Cholesky decomposition (since our inexact and exact methods are comparable experimentally).

One surprising result is that despite using a weak bound on the inexact Newton step residual, a bound that theoretically only gives linear convergence, we still have iteration counts comparable to the quadratically convergent exact method.

10.1 Future work

The computational performance of the reflective Newton methods, both exact and inexact, may be improved in the case of problems with degenerate and near-degenerate solutions. For these problems, the scaling by the D_k^{-1} matrices may introduce ill-conditioning in the Hessian matrix $\hat{M}_k$ since both v_i and g_i may be going to zero (near zero) for some index i. In such cases, how do we re-scale the problem to improve conditioning without adversely changing the properties of the algorithm? And how do we tell when to introduce the re-scaling since "near degenerate" is not precisely defined?

Our inexact technique does not give us second-order convergence because our negative curvature directions are not guaranteed to be negative enough. How can we find sufficient negative curvature cheaply? A method such as Lanzcos [GVL89] will compute directions of sufficient negative curvature. But this computation can be quite expensive for large problems. Some recent work by Arioli, et al. [ACD+93] discusses ways of efficiently modifying the conjugate gradient iteration to find such negative curvature directions. Do these modifications preserve the large-step-size property? And if they do, is the extra computation required worthwhile, that is, does it computationally improve the convergence speed?

We also have some unresolved questions in regard to the subspace trust-region method. In comparing various trust-region methods in Chapter 8, we noted that using an additional (third) vector in the subspace had mixed results. Sometimes it was an improvement. However for other problems it seemed to affect the trust-region size adversely and slow the convergence. This is interesting since theoretically the solution to this three-dimensional subspace problem will always be as good, if not better, than the two-dimensional subspace solution if the trust-region size is the same. First, what can be done so that using a larger subspace is always an improvement? Do we need to change the way we decide to update the trust-region size? Also, if we do choose to use more directions, what should we use? Certainly we have vectors with large-step-size available. These include the steps taken in previous iterations of the optimization method.

Kaporin and Axelsson [KA95] propose coupling inexact Newton techniques with nonlinear minimization over low-dimensional subspaces, where the subspaces include previous minimization steps, to improve convergence. Their results suggest that such a combination reduces the preconditioning operations; that is, a new preconditioner needs to be recomputed less often. Certainly, this would make more sophisticated preconditioners, which are more expensive to compute, more viable for large problems. This would benefit the inexact reflective Newton methods since better preconditioners would likely improve computation on ill-conditioned problems (recall in Chapter 6 that some loss of accuracy was observed on ill-conditioned problems). Improved preconditioning would also improve the results on the VAR C problem in Chapter 7 where the maximum number of conjugate gradient iterations was taken at each iteration, and so the number of minimization iterations was higher than typically observed for similar sized problems.

Besides the conclusions we have made about the inexact reflective Newton methods, the results in this thesis underline the important role that negative curvature can play in large-scale optimization. This role supports the subspace trust-region method as a way to approximate the full-space trust-region solution. Thus, any improvement to efficiently find sufficient negative curvature, or to use subspace methods to reduce preconditioning requirements, is not only an improvement for the inexact reflective Newton methods, but an improvement that is likely to have an impact on many large-scale optimization methods.

Appendix A

Reflective line search

The following description of the reflective line search follows that found in [CL94].

The piecewise reflection path can be described, recursively, as follows. Given any descent direction $s_k \in \Re^n$, define the vector[1]

$$BR_k = \max[(l - x_k) \;./\; s_k, \quad (u - x_k) \;./\; s_k)], \tag{A.1}$$

where the notation " ./ " indicates componentwise division. Component i of vector BR_k records the positive stepsize from x_k to the *breakpoint* corresponding to variable x_{k_i} in the direction s_k. The piecewise linear (reflective) path is defined by the reflective path described in Figure A.1. Since only a single outer iteration is considered, we do not include the subscript k with the variables in our description of the reflective path below - dependence on k is assumed.

Let $p_k(\alpha)$ denote the reflective path as defined in Figure A.1: For $\beta_k^{i-1} \leq \alpha < \beta_k^i$,

$$p_k(\alpha) = b_k^{i-1} + (\alpha - \beta_k^{i-1}) p_k^i. \tag{A.2}$$

Note that the reflective path $p_k(\alpha)$ is defined with respect to the current point and direction under consideration. This dependence is understood and is not explicitly denoted.

[1]For the purpose of computing BR we assume the following rules regarding arithmetic with infinities. If a is a finite scalar then $a + \infty = \infty$, $a - \infty = -\infty$, $\frac{\infty}{a} = \infty \cdot \mathrm{sgn}(a)$, $\frac{-\infty}{a} = -\infty \cdot \mathrm{sgn}(a)$, $\frac{a}{0} = \mathrm{sgn}(a) \cdot \infty$, $\frac{\infty}{0} = \infty$, *and* $\frac{-\infty}{0} = -\infty$, where $\mathrm{sgn}(a) = +1$ if $a \geq 0$, $\mathrm{sgn}(a) < 0$ if $a < 0$.

The Reflective Path: [Let $\beta^0 = 0$, $p^1 = s$, set $b^0 = x_k$.]

For $i = 1 : \infty$

1. Let β^i be the distance to the nearest breakpoint along p^i:

$$\beta^i = \min\{BR : \ BR > 0\}$$

2. Define the i^{th} breakpoint: $b^i = b^{i-1} + (\beta^i - \beta^{i-1})p^i$.

3. Reflect to get the new direction and update BR:

   (a) $p^{i+1} = p^i$

   (b) For each j such that $(b^i)_j = u_j$ or $(b^i)_j = l_j$

   - $BR(j) = BR(j) + |\frac{u_j - l_j}{(s)_j}|$
   - $(p^{i+1})_j = -(p^i)_j$.

Figure A.1: Determine the linear reflective path p.

Appendix B

Preconditioners for MPCG

Recall the assumption required in Section 4.2 regarding preconditoners and the large-step-size property,

(MPCG.4) If $\{D_k^{-2} w_k\}$ has large-step-size then $\{D_k^{-1} P_k^{-1} D_k^{-1} w_k\}$ has large-step-size. Equivalently, we may say if $\{w_k\}$ is bounded then $\{D_k P_k^{-1} D_k^{-1} w_k\}$ is bounded.

In Section 4.2, we showed this condition holds for diagonal preconditioners. In this section, we show how this assumption holds for more sophisticated preconditioners.

In unconstrained minimization, the preconditioner is constructed from the Hessian matrix H_k. Assume that we have some such preconditioner $\bar{P}_k$ constructed from H_k such that $\bar{P}_k$ is positive definite, $\|\bar{P}_k\| \leq \chi_{\bar{P}}$, and $\|\bar{P}_k^{-1}\| \leq \chi_{\bar{P}}$ (recall we need to satisfy assumption (MPCG.2)).

This preconditioner, for example, might be an incomplete Cholesky preconditioner (see [GVL89], page 530).

Recall that $\hat{M}_k = D_k^{-1} H_k D_k^{-1} + J_k^v D_k^{g^+}$. Then a reasonable way to construct a preconditioner P_k, based on $\bar{P}_k$, to use in MPCG is

$$P_k = D_k^{-1} \bar{P}_k D_k^{-1} + J_k^v D_k^{g^+}.$$

For example, the diagonal preconditioner discussed in Section 4.2 has this form.

We show that preconditioners of this type satisfy the assumption (MPCG.4). First we need the following technical lemma from [CL94].

Lemma B.0.1 *Let $\{s_k\}$ be a sequence of vectors and assume $\{s_k\}$ is bounded. Assume that for each iteration k and each index i such that $0 < |v_{k_i}| < 1$,*

$$e_{k_i} s_{k_i} = |v_{k_i}| z_{k_i}, \tag{B.1}$$

where e_{k_i} satisfies $|e_{k_i}| \geq g_{k_i}^+$. Assume $\{z_k\}$ is bounded. Then $\{s_k\}$ has large-step-size.

We also need a second technical lemma.

Lemma B.0.2 *If $\{w_k\}$ is bounded, then $\{s_k\}$ is bounded where*

$$s_k = D_k^{-1} P_k^{-1} D_k^{-1} w_k$$

and P_k is of the form

$$P_k = D_k^{-1} \bar{P}_k D_k^{-1} + J_k^v D_k^{g^+}$$

for some positive definite matrix $\bar{P}_k$ such that $\|\bar{P}_k\| \leq \chi_{\bar{P}}$ and $\|\bar{P}_k^{-1}\| \leq \chi_{\bar{P}}$.

Proof. Using the definition of s_k and P_k we have that

$$\begin{aligned} s_k &= D_k^{-1} P_k^{-1} D_k^{-1} w_k \\ &= D_k^{-1} (D_k^{-1} \bar{P}_k D_k^{-1} + J_k^v D_k^{g^+})^{-1} D_k^{-1} w_k \\ &= (D_k D_k^{-1} \bar{P}_k D_k^{-1} D_k + D_k J_k^v D_k^{g^+} D_k)^{-1} w_k \\ &= (\bar{P}_k + J_k^v D_k^{g^+} D_k^2)^{-1} w_k. \end{aligned}$$

We claim that $\|(\bar{P}_k + J_k^v D_k^{g^+} D_k^2)^{-1}\|$ is bounded above. Let $C = A + B$ where $A = \bar{P}_k$ and $B = J_k^v D_k^{g^+} D_k^2$. Then C is positive definite and by Theorem 5.10 in [Ste73], if $\alpha_1 \leq \alpha_2 \leq ... \leq \alpha_n$, $\beta_1 \leq \beta_2 \leq ... \leq \beta_n$, $\gamma_1 \leq \gamma_2 \leq ... \leq \gamma_n$, are the eigenvalues of A, B, and C respectively, then

$$\gamma_i \geq \alpha_i + \beta_1$$

for $i = 1, 2, ..., n$. Thus $\gamma_1 \geq \alpha_1$, and α_1 is bounded away from zero since $\|\bar{P}_k^{-1}\| \leq \chi_{\bar{P}}$. Thus $\|(\bar{P}_k + J_k^v D_k^{g^+} D_k^2)^{-1}\| \leq \frac{1}{\alpha_1} \leq \chi_{\bar{P}}$. Then since we assumed $\{w_k\}$ is bounded, we have the desired result. □

Theorem B.0.3 *If $\{w_k\}$ is bounded and P_k is of the form*

$$P_k = D_k^{-1} \bar{P}_k D_k^{-1} + J_k^v D_k^{g^+},$$

for some positive definite matrix $\bar{P}_k$ such that $\|\bar{P}_k\| \leq \chi_{\bar{P}}$ and $\|\bar{P}_k^{-1}\| \leq \chi_{\bar{P}}$, then $\{D_k P_k^{-1} D_k^{-1} w_k\}$ is bounded.

Proof. Let $s_k = D_k^{-1} P_k^{-1} D_k^{-1} w_k$. Then

$$D_k^2 s_k = D_k P_k^{-1} D_k^{-1} w_k$$

and so

$$P_k (D_k s_k) = D_k^{-1} w_k$$

which implies

$$D_k^{-1}\bar{P}_k D_k^{-1} D_k s_k + J_k^v D_k^{g^+} D_k s_k = D_k^{-1} w_k$$

and so

$$J_k^v D_k^{g^+} D_k s_k = D_k^{-1}(-\bar{P}_k s_k + w_k).$$

Then

$$J_k^v D_k^{g^+} s_k = D_k^{-2}(-\bar{P}_k s_k + w_k).$$

Now we use Lemma B.0.1. For each index i with $|v_{k_i}| < 1$, $|J_{k_i}^v| = 1$ and so $|J_k^v D_k^{g^+}| \geq g_{k_i}^+$. Also, we know that $\|\bar{P}_k\| \leq \chi_{\bar{P}}$, and that $\{w_k\}$ is bounded. Furthermore $\{s_k\}$ is bounded by Lemma B.0.2. Thus, we have that s_k has large-step-size, that is, $\{D_k^2 s_k\} = \{D_k P_k^{-1} D_k^{-1} w_k\}$ is bounded. $\square$

Appendix C

STIR: Trust region update

We give our trust-region updating strategy for the STIR method. We assume that the ratio ρ_k has been computed (see Figure 3.6). Also recall that the step is updated only if $\rho_k > \mu$.

Updating Trust Region Size Δ_k

[Let $0 < \mu < \eta < 1$, $0 < \Lambda_L < \Lambda_U$ and $\gamma_1 < 1 < \gamma_2$ be given. Let $\Delta_1 < \Lambda_U$.]

1. If $\rho_k \leq \mu$ then set $\Delta_{k+1} \in (0, \gamma_1 \Delta_k]$.
2. If $\rho_k \in (\mu, \eta)$ then set $\Delta_{k+1} \in [\gamma_1 \Delta_k, \Delta_k]$.
3. If $\rho_k \geq \eta$ then
   if $\Delta_k > \Lambda_L$ then
   set $\Delta_{k+1} \in$ either $[\gamma_1 \Delta_k, \Delta_k]$ or $[\Delta_k, \gamma_2 \Delta_k]$,
   otherwise,
   set $\Delta_{k+1} \in [\Delta_k, \min(\gamma_2 \Delta_k, \Lambda_U)]$.

Figure C.1: STIR: updating Δ_k.

Bibliography

[ACD+93] M. Arioli, T. F. Chan, I. S. Duff, N. I. M. Gould, and J. K. Reid. Computing a search direction for large-scale linearly-constrained nonlinear optmization calculations. Technical Report RAL 93-066, Rutherford Appleton Laboratory, 1993.

[Arm66] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pac. J. Math.*, 16:1–3, 1966.

[BCL95] Mary Ann Branch, Thomas F. Coleman, and Yuying Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. Technical Report TR95-1525, Computer Science Department, Cornell University, 1995.

[Ber76] Dimitri P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, AC-21(2):174–184, 1976.

[Ber82] Dimitri P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2):221–246, 1982.

[Bjö84] Åke Björck. A general updating algorithm for constrained linear least squares problems. *SIAM Journal on Scientific and Statistical Computing*, 5(2):394–402, 1984.

[BLN93] Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. A limited-memory algorithm for bound-constrained optimization. Technical Report MCS-P404-1293, Mathematics and Computer Science Division, Argonne National Lab, 1993.

[BM88] James V. Burke and Jorge J. Moré. On the identification of active constraints. *SIAM Journal on Numerical Analysis*, 25:1197–1211, 1988.

[BSS88a] R. Byrd, R.B. Schnabel, and G. Shultz. Parallel quasi-Newton methods for unconstrained optimization. Technical report, Department of Computer Science, University of Colorado, 1988.

[BSS88b] Richard H. Byrd, Robert B. Schnabel, and Gerald A. Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Mathematical Programming*, 40:247–263, 1988.

[CGT88a] Andrew R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25(2):433–460, 1988.

[CGT88b] Andrew R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50(182):399–430, 1988.

[CGT92] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer-Verlag, 1992.

[CH89] Thomas F. Coleman and Laurie A. Hulbert. A direct active set algorithm for large sparse quadratic programs with simple bounds. *Mathematical Programming*, 45(3):373–406, 1989.

[CH93] Thomas F. Coleman and Laurie A. Hulbert. A globally and superlinearly convergent algorithm for convex quadratic programs with simple bounds. *SIAM Journal on Optimization*, 3:298–321, 1993.

[CLa] Thomas F. Coleman and Yuying Li. An interior, trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, to appear.

[CLb] Thomas F. Coleman and Yuying Li. A reflective Newton method for minimizing a quadratic function subject to bounds on the variables. *SIAM Journal on Optimization*, to appear.

[CL94] Thomas F. Coleman and Yuying Li. On the convergence of reflective Newton methods for large-scale nonlinear minimization subject to bounds. *Mathematical Programming*, 67:189–224, 1994.

[CM87] Paul H. Calamai and Jorge J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39:93–116, 1987.

[Col84] Thomas F. Coleman. *Large Sparse Numerical Optimization*. Lecture Notes in Computer Science, Volume 165, Springer-Verlag, 1984.

[Col93] Thomas F. Coleman. Large-scale numerical optimization: Introduction and overview. In Allen Kent and James G. Williams, editors, *Encyclopedia of Computer Science and Technology*, pages 167–195. Marcel Dekker, INC., 1993.

[DES82] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19:400–408, 1982.

[DS83] Ron S. Dembo and Trond Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.

[DT83] Ron S. Dembo and Ulrich Tulowitzki. On the minimization of quadratic functions subject to box constraints. Technical Report B 71, Yale University, 1983.

[Dun81] J. C. Dunn. Global and asymptotic convergence rate estimates for a class of projected gradient processes. *SIAM Journal on Control and Optimization*, 19(3):368–400, 1981.

[FJ74] R. Fletcher and M. P. Jackson. Minimization of a quadratic function of many variables subject only to lower and upper bounds. *Journal of the Institute for Mathematics and its Applications*, 14:159–174, 1974.

[FL92] F. Facchinei and S. Lucidi. A class of penaly functions for optimization problems with bound constraints. *Optimization*, 26(3-4):239–259, 1992.

[Fle87] R. Fletcher. *Practical Methods of Optimization, Second Edition.* John Wiley and Sons, 1987.

[Fra95] Hubertus Franke. MPI-F: An MPI implementation for the IBM SP-1/SP-2, version 1.41. Technical report, IBM T.J. Watson Research Center, 1995.

[GL81] Alan George and Joseph W-H Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, 1981.

[GM76] P.E. Gill and W. Murray. Minimization subject to bounds on the variables. Technical Report Report NAC 71, National Physical Laboratory, England, 1976.

[GMS92] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13:333–356, 1992.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

[Gol64] A.A. Goldstein. Convex programming in Hilbert space. *Bulletin of the American Mathematical Society*, 70:709–710, 1964.

[Gol65] A.A. Goldstein. On steepest descent. *SIAM Journal on Control*, 3:147–151, 1965.

[Gol80] D. Goldfarb. Curvilinear path steplength algorithms for minimization algorithms which use directions of negative curvature. *Mathematical Programming*, 18:31–40, 1980.

[GVL89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1989.

[Hen91] Bruce A. Hendrickson. *The molecule problem: Determining conformation from pairwise distances.* Cornell University Ph.D thesis, Computer Science, 1991.

[HPY90] Chi-Guen Han, Panos M. Pardalos, and Yinyu Ye. Computational aspects of an interior point algorithm for quadratic programming problems with box constraints. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 92–112. SIAM, 1990.

[IBM92] IBM Corporation. *Engineering and Scientific Subroutine Library, Version 2, Guide and Reference*, 1992.

[KA95] Igor E. Kaporin and Owe Axelsson. On a class of nonlinear equation solvers based on the residual norm reduction over a sequence of affine subspaces. *SIAM Journal on Scientific and Statistical Computing*, 16(1):228–249, 1995.

[Löt84] P. Lötstedt. Solving the minimal least squares problem subject to bounds on the variables. *BIT*, 24:206–224, 1984.

[LP66] E.S. Levitin and B.T. Polyak. Constrained minimization problems. *USSR Computational Mathematics and Mathematical Physics*, 6:1–50, 1966.

[McC69] Garth P. McCormick. Anti-zig-zagging by bending. *Management Science*, 15(5):315–320, 1969.

[Mes93] Message Passing Interface Forum. Document for a standard message-passing interface. Technical report, University of Tennessee, Knoxville, Tenn., 1993.

[MGH81] Jorge J. Moré, Burton S. Garbow, and Kenneth. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7:17–41, 1981.

[MMZ95] M.P. McKenna, J.P. Mesirov, and S.A. Zenios. Data parallel quadratic programming on box-constrained problems. *SIAM Journal on Optimization*, 5(3), 1995.

[Mor83] Jorge J. Moré. Recent developments in algorithms and software for trust region methods. In M. Götchel A. Bachem and B. Korte, editors, *Mathematical Programming: The State of the Art.* Springer Verlag, Berlin, 1983.

[Mor90] Jorge J. Moré. On the performance of algorithms for large-scale bound constrained problems. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 31–46. SIAM, 1990.

[MS83] Jorge J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4:553–572, 1983.

[MT89a] Jorge J. Moré, , and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. Preprint ANL/MCS-P77-0589, Argonne National Laboratory, 1989.

[MT89b] Jorge J. Moré and Gerardo Toraldo. Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55:377–400, 1989.

[Nas84] Stephen G. Nash. Newton-type minimization via the Lanczos method. *SIAM Journal on Numerical Analysis*, 21:770–788, 1984.

[Nas85] S. G. Nash. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6:599–616, 1985.

[Nas93] Stephen G. Nash. A parallel barrier algorithm for large-scale bound-constrained optimization. Technical Report 93-01, Operations Research and Engineering Department, George Mason University, 1993.

[NN91] S. G. Nash and J. Nocedal. A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization. *SIAM Journal on Optimization*, 1:358–372, 1991.

[NPS94] Stephen G. Nash, R. Polyak, and Ariela Sofer. A numerical comparison of barrier and modified barrier methods for large-scale bound-constrained optimization. In W.W. Hager, D.W. Hearn, and P.M. Pardalos, editors, *Large Scale Optimization, State of the Art*, pages 319–338. Kluwer Academic Publishers, 1994.

[NS89] Stephen G. Nash and Ariela Sofer. Block truncated-Newton methods for parallel optimization. *Mathematical Programming*, 45:529–546, 1989.

[NS91] Stephen G. Nash and Ariela Sofer. A general-purpose parallel algorithm for unconstrained optimization. *SIAM Journal on Optimization*, 1:530–547, 1991.

[NS93] Stephen G. Nash and Ariela Sofer. A barrier method for large-scale constrained optimization. *ORSA Journal on Computing*, 5(1):40–53, 1993.

[O'L80] D. P. O'Leary. A generalized conjugate gradient algorithm for solving a class of quadratic programming problems. *Linear Algebra and its Applications*, 34:371–399, 1980.

[Sor94] D.C. Sorensen. Minimization of a large scale quadratic function subject to an ellipsoidal constraint. Technical Report TR94-27, Department of Computational and Applied Mathematics, Rice University, 1994.

[SSB85] Gerald A. Shultz, R. B. Schnabel, and Richard H. Byrd. A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties. *SIAM Journal on Numerical Analysis*, 22(1):47–67, 1985.

[Ste73] G. W. Stewart. *Introduction to Matrix Computations.* Academic Press, 1973.

[Ste83] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20:626–637, 1983.

[Zen89] S.A. Zenios. Parallel optimization: current status and an annotated bibliography. *ORSA J. Computing*, 1:20–43, 1989.