Convexification in Unconstrained Continuous Optimization

by

Lingzhou Hong

A thesis submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Engineering.

Baltimore, Maryland

December, 2014

Abstract

In this Master's thesis, we study the role of convexification as it is used in unconstrained optimization of smooth functions. Many variants of convexification exist, but a detailed study of their practical performance has not been performed. We complete such a study in this thesis since the performance of an optimization algorithm is greatly affected by the convexification used. We also propose and validate a new convexification procedure by comparing it with commonly used schemes through a series of extensive numerical experiments; the new procedure performs the best. The results we obtained will likely aid in the design of future optimization algorithms.

Advisor: Dr. Daniel P. Robinson

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Professor Daniel P. Robinson for his continuous support of my research, for his patience, motivation, enthusiasm, and most importantly, his friendship during my study in Johns Hopkins. He has helped in my research and the writing of this thesis. He encouraged me to not only to grow as a an optimization person, but also an independent thinker.

I would also like to gratefully and sincerely thank Prof. Donniell E. Fishkind for his help and encouragement.

Last, but not the least, I would like to thank my parents, my sister, and my boyfriend, for supporting me spiritually.

Dedication

I dedicate my thesis to my family and my boyfriend. A special feeling of gratitude to my parents, Zhi Hong and Ya TU. I will always appreciate everything that they have done for me.

Contents

Abstract **ii**

Acknowledgments **iii**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 The Problem Formulation 1

1.2 Contributions 3

1.3 Notation 4

1.4 Definitions and Optimality Conditions 5

2 Popular Convexification Schemes **8**

2.1 Steepest Descent 12

2.2 Shifted Spectra 13

2.3 Modified Newton 15
2.4 Quasi-Newton 17
2.4.1 BFGS 17
2.4.2 LBFGS 18
2.5 Linear Conjugate Gradient Method 20
2.6 Tensor Model Based Methods 23
2.6.1 Tensor Model Minimization Methods 23
2.6.2 Tensor Model Based Modified LBFGS 25

3 A New Convexification Procedure **28**
3.1 The Derivation of MN-CG 29
3.2 The Formal Statement of MN-CG 33

4 Numerical Experiments **34**
4.1 Popular Convexification Schemes 36
4.2 Various Instances of MN-CG 41
4.3 The Best Overall Methods 44

5 Conclusions **47**

Bibliography **49**

List of Tables

4.1 Results for popular convexification methods on the CUTEst problems. 38
4.2 Results for various instances of MN-CG on the CUTEst problems. . . 42

List of Figures

4.1 Performance profile for iterations on the CUTEst problems. 39
4.2 Performance profile for function evaluations on the CUTEst problems. 39
4.3 Performance profile for iterations on the CUTEst problems. 43
4.4 Performance profile for function evaluations on the CUTEst problems. 43
4.5 Performance profiles for iterations on the CUTEst problems. 45
4.6 Performance profile for function evaluations on the CUTEst problems. 46

Chapter 1

Introduction

In this chapter we lay the foundation for the thesis by first introducing in Section 1.1 the optimization problem that we study. In Section 1.2 we give the contributions of this thesis and explain why they are important. In Section 1.3 we summarize the notation used throughout, and in Section 1.4 provide definitions and state well-known results related to solutions of the optimization problem under study.

1.1 The Problem Formulation

We consider the unconstrained optimization problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \;\; f(x) \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a twice-continuously differentiable function, i.e., f and its first two derivatives are continuous. Problems of this type include nonlinear least-squares,[1,2] polynomial functions,[3] among others. It is worth mentioning that unconstrained optimization problems arise in both academic and real-world problems.

For example, least-squares regression models obtain parameters by minimizing the squared error between the values predicted by the model and the observed values.[4] Consider the linear regression model in matrix form given by $y = Xb + e$, where X is the matrix of independent variables, y is a vector of dependent variables, b is the unknown parameter vector, and e is an error residual vector. In least squares regression, the parameter vector b is chosen to produce the "best fit" model, which means the residuals should be as small as possible, i.e., we can minimize the squared error function $f(b) = \|y - Xb\|_2^2$. It is well-known that the vector that minimizes $f(b)$ is, when X^TX is invertible, given by $b = (X^TX)^{-1}X^Ty$. In the real world, many business decisions are based on optimization problems of this form, e.g., maximizing profit,[5] minimizing lost,[6] constructing optimal investment portfolios,[7] and designing supply chains.[8]

Many optimization problems involve objective functions that are not twice continuously differentiable; problems of that type will not be addressed in this thesis.

1.2 Contributions

Problem (1.1) is usually solved by an iterative algorithm, i.e., a method that uses an initial estimate of a solution x_0 to generates a sequence of iterates $\{x_k\}$ that hopefully converges to a solution of x^*. The various known algorithms may broadly be classified as either line-search or trust-region methods. These two classes of algorithms are similar in many aspects, but do differ in certain key ways. One key difference concerns convexification, which is a major contributor to the overall efficiency of the algorithm. Broadly speaking, convexification is a notion used by every line-search and trust-region method to ensure that each subproblem used during the solution process is well-defined and emits a useful solution (perhaps an approximate solution). In this thesis, we will only consider convexification as it pertains to line-search methods.

This thesis has three contributions: (i) To present and summarize the most commonly used line-search procedures used by modern algorithms; (ii) To present a new convexification scheme that performs the best on a certain subclass of problems; and (iii) To provide the results of extensive numerical experiments concerning the various approaches. Since the best criteria used to compare algorithms varies based on problem size, availability and sparsity of the derivatives of the objective function, and the goal of the user, we consider multiple measures of performance.

1.3 Notation

The notation that we use is standard and listed here for convenience to the reader.

- The objective function is denoted by $f : \mathbb{R}^n \to \mathbb{R}$.
- The gradient of f is denoted by $g := \nabla_x f : \mathbb{R}^n \to \mathbb{R}^n$.
- The Hessian function of f is denoted by $H := \nabla^2_{xx} f : \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$.
- Given an iteration x_k, we let $f_k = f(x_k)$, $g_k = g(x_k)$, and $H_k = H(x_k)$ denote the objective function and its first two derivatives, respectively, evaluated at x_k.
- The kth search direction is p_k with step length α_k so that $x_{k+1} = x_k + \alpha_k p_k$.
- The matrix B_k is a symmetric positive-definite matrix that is intended to approximate (perhaps poorly) the Hessian matrix H_k.
- For any matrix M, we use $M \succ 0$ to mean that M is positive definite, $M \succeq 0$ to mean that M is positive semidefinite, $M \not\succ 0$ to mean that M is not positive definite, and $M \not\succeq 0$ to mean that M is not positive semidefinite.
- The sequence of iterates generated by an algorithm are denoted by $\{x_k\} \subset \mathbb{R}^n$.
- The vector $x^* \in \mathbb{R}^n$ is a local minimizer for problem (1.1) (see Definition 1.4.2).
- The secant equation is $B_{k+1} s_k = y_k$, where $s_k := x_{k+1} - x_k$ and $y_k := g_{k+1} - g_k$.

1.4 Definitions and Optimality Conditions

Various standard optimization terms are used in this thesis and, for ease of later reference, are stated in this section. We note that we only focus on minimization since finding a maximizer may be done by minimizing the negative of the objective function. We begin with global and local minimizers.

Definition 1.4.1 (global minimizer) *A point x^* is a global minimizer if and only if $f(x^*) \leq f(x)$ for all $x \in \mathbb{R}^n$.*

Definition 1.4.2 (local minimizer) *A point x^* is a local minimizer if and only if there exists $\varepsilon > 0$ so that $f(x^*) \leq f(x)$ for all $x \in \mathbb{B}(x^*, \varepsilon) := \{x \in \mathbb{R}^n : \|x - x^*\| \leq \varepsilon\}$.*

The previous definitions are not practical since the conditions that define them are rarely verifiable. Necessary and sufficient optimality conditions, on the other hand, provide us with a means to tell when a point is optimal (sufficient conditions) or is not optimal (necessary conditions). We begin with the first-order necessary conditions.

Theorem 1.4.1 (first-order necessary conditions) *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is once continuously differentiable. If x^* is a local minimizer of f, then $g(x^*) = 0$.*

The contrapositive of the previous theorem allows us to deduce that if $g(\hat{x}) \neq 0$, then $\hat{x}$ is not a minimizer. However, $g(\hat{x}) = 0$ does not necessarily mean that $\hat{x}$ is a minimizer, e.g., it could be a saddle point or even a maximizer. Consider the simple example $f(x) = x^3$ for which $\hat{x} = 0$ satisfies $g(\hat{x}) = 0$, but clearly $\hat{x}$ does

not minimizer $f(x)$. Since first order information cannot provide us with enough information to determine when a point is a minimizer (at least when f is nonconvex), we also consider second order information (curvature information) for the function.

Theorem 1.4.2 (second-order necessary conditions) *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. If x^* is a local minimizer of f, then $g(x^*) = 0$ and $H(x^*)$ is positive semi-definite, i.e., $s^T H(x^*)s \geq 0$ for all $s \in \mathbb{R}^n$.*

Similar to the first-order necessary condition, the contrapositive of the previous result allows us to deduce that if $g(\hat{x}) = 0$ but $H(\hat{x}) \not\geq 0$, then $\hat{x}$ is certainly not a local minimizer. However, even if the Hessian is positive semi-definite, we still can not guarantee that $\hat{x}$ is a minimizer. Again, for the problem $f(x) = x^3$, the point $\hat{x} = 0$ satisfies $g(\hat{x}) = 0$ and $H(\hat{x}) = 0 \geq 0$, but $\hat{x} = 0$ is not a minimizer.

We now develop sufficient optimality conditions. Before stating such conditions, we give two definitions associated with different types of local minimizers.

Definition 1.4.3 (isolated local minimizer) *The vector x^* is an isolated local minimizer if and only if there exists $\varepsilon > 0$ so that x^* is the only local minimizer in $\mathbb{B}(x^*, \varepsilon)$.*

Definition 1.4.4 (strict local minimizer) *The vector x^* is a strict local minimizer if and only if there exists $\varepsilon > 0$ so that $f(x^*) < f(x)$ for all $x^* \neq x \in \mathbb{B}(x^*, \varepsilon)$.*

An isolated local minimizer is a strict local minimizer, but a strict local minimizer is not necessarily an isolated local minimizer. For example, it is possible to construct

functions that oscillate infinitely often as $x \to 0$, which means that every neighborhood of 0 contains infinitely many local minimizers.

We may now state the sufficient optimality conditions. Specifically, the next result gives conditions that allow us to assert that a point is a strict local minimizer.

Theorem 1.4.3 (second-order sufficient conditions) *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. If x^* satisfies $g(x^*) = 0$ and $H(x^*)$ is positive definite, i.e., $s^T H(x^*)s > 0$ for all $s \neq 0$, then x^* is a strict local minimizer of f.*

The idea of a descent direction will be important for finding minimizers. In particular, they represent directions for which lower values of the objective function are guaranteed to exist. Such directions are critical to line search methods.

Definition 1.4.5 (descent direction) *We say that p is a descent direction at x for the objective function f of problem (1.1) if $g(x)^T p < 0$.*

Chapter 2

Popular Convexification Schemes

As mentioned in Section 1.2, non-convexity has a substantial impact in optimization. In particular, it plays a critical part in the overall effectiveness of optimization algorithms, and its influence is, in some ways, more direct for line-search methods. The sections in this chapter describe various popular linear-search convexification procedures. Before discussing them, however, we describe the basic iteration scheme as well as standard conditions used by line-search methods, for completeness.

Line search methods are typically *descent methods* in the sense that the generated iterates $\{x_k\} \subset \mathbb{R}^n$ satisfy $f(x_{k+1}) < f(x_k)$. To ensure this property, line search methods update the estimate of solution by the formula

$$x_{k+1} = x_k + \alpha_k p_k, \tag{2.1}$$

where p_k is a descent direction (i.e., $g(x_k)^T p_k < 0$) and $\alpha_k > 0$ is a step length chosen to ensure $f(x_{k+1}) < f(x_k)$. Since p_k is a descent direction, one may use Taylor's Theorem to show that $f(x_k + \alpha p_k) < f(x_k)$ for all sufficiently small $\alpha > 0$.

Although many minor algorithmic variants exist, the two key differences between line-search methods are in calculating the descent direction p_k and the step length α_k. Loosely speaking, most (if not all) choices of p_k can be described as being *approximate* solutions to the optimization problem

$$\underset{p \in \mathbb{R}^n}{\text{minimize}} \;\; f_k + g_k^T p + \tfrac{1}{2} p^T B_k p \tag{2.2}$$

for some choice of symmetric matrix B_k that is positive definite when restricted to some subspace of $\mathbb{R}^n$. In Sections 2.1–2.6 we consider the most popular choices for choosing the matrix B_k in addition to describing the exact meaning of what we mean by an *approximate* solution to (2.2).

Once a descent direction p_k is calculated, we must compute the step length α_k, either exactly or inexactly. By exactly, we mean that α is computed as the exact minimizer of $h(\alpha) := f(x_k + \alpha p_k)$, which is expensive and inefficient to compute. Thus, the search steps are normally determined inexactly by identifying an α_k that satisfies certain sufficient decrease conditions on f that are satisfied by many values of α, not just by minimizing ones. The two most popular choices are the *backtracking Armijo* line-search method[9] and a line-search based on satisfying the *(strong) Wolfe*

conditions.[10] Although the precise nature in which $\alpha_k > 0$ is chosen is important, it is not the focus of this thesis; we focus on the effects that the different convexification procedures have on the efficiency and robustness of the algorithm. Still, for completeness we briefly describe these two procedures.

Definition 2.0.6 (Armijo condition) *Given a constant $\eta \in (0,1)$ and a descent direction p_k at x_k, we say that the step length $\alpha_k > 0$ satisfies the Armijo condition if and only if*

$$f(x_k + \alpha_k p_k) \leq f(x_k) + \eta \alpha_k g_k^T p_k.$$

A backtracking Armijo line-search method finds a valid step length α_k by starting with some guess (typically the value 1), and gradually shrinking its size until the Armijo condition is satisfied. This procedure is described in Algorithm 1.

Algorithm 1 A backtracking Armijo line-search.

1: **input** x_k and descent direction p_k.
2: Choose $\alpha_{int} > 0$, $\eta \in (0,1)$, and $\tau \in (0,1)$.
3: Set $\alpha^{(0)} \leftarrow \alpha_{int}$ and $l \leftarrow 0$.
4: **while** $f(x_k + \alpha^{(l)} p_k) > f(x_k) + \eta \alpha^{(l)} g_k^T p_k$ **do**
5: Set $\alpha^{(l+1)} \leftarrow \tau \alpha^{(l)}$.
6: Set $l \leftarrow l + 1$.
7: **return** $\alpha_k \leftarrow \alpha^{(l)}$.

In contrast to the backtracking Armijo line search, a line search based on satisfying the Wolfe conditions seeks better approximations to an exact line-search by combining the Armijo condition with other conditions that hold near a local minimizer. It is worth commenting that the Wolfe conditions are especially popular for the BFGS

quasi-Newton updating method since they ensure that the updates will be successful.

Definition 2.0.7 (weak Wolfe conditions) *Given constants* $0 < c_1 < c_2 < 1$ *and a descent direction* p_k *at* x_k*, we say that the step length* $\alpha_k > 0$ *satisfies the weak Wolfe conditions if and only if*

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k g(x_k)^T p_k \quad \textit{and}$$
$$g(x_k + \alpha_k p_k)^T p_k \geq c_2 g(x_k)^T p_k. \tag{2.3}$$

The strong Wolfe conditions impose, not surprisingly, stronger conditions than those given by the weak Wolfe conditions.

Definition 2.0.8 (strong Wolfe condition) *Given constants* $0 < c_1 < c_2 < 1$ *and a descent direction* p_k *at* x_k*, we say that the step length* $\alpha_k > 0$ *satisfies the strong Wolfe conditions if and only if*

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k g(x_k)^T p_k \quad \textit{and} \tag{2.4}$$
$$\left| g(x_k + \alpha_k p_k)^T p_k \right| \leq c_2 \left| g(x_k)^T p_k \right|. \tag{2.5}$$

One can see that in comparison to the weak Wolfe conditions, the strong Wolfe conditions ensures that the directional derivative at the updated point is smaller than at the initial point in absolute value (see (2.5)), not just greater than a fraction of the directional derivative at the initial point (see (2.3)).

2.1 Steepest Descent

The *steepest descent* direction is given by $p_k = -g_k$. It is simple in this case to see that $g_k^T p_k = -g_k^T g_k = -\|g_k\|_2 < 0$ (provided $g_k \neq 0$) so that the steepest descent direction is, in fact, a descent direction. It is less commonly known that the steepest descent direction is the solution to (2.2) with the choice $B_k = I$. Thus, the steepest descent direction is the trial step that minimizers a model of $f(x_k + s)$ given by (2.2) defined using the simple Hessian approximation $B_k = I$. It should, therefore, not be a surprise that the steepest descent direction is often a poor direction to use in the sense that many iterations are often required before near optimality is reached. Nonetheless, the simplicity of the steepest descent direction is a great advantage that should not be overlooked, nor should the fact that it often achieves fast decrease in the objective function initially. The main weakness, i.e., slow convergence, is (typically) most evident during later iterations because it is difficult to obtain the *final* termination tolerance that is often requested by the user.

We may conclude from the previous paragraph that the convexification used by the steepest descent method is to use the matrix $B_k = I$ rather than the exact second derivative matrix $H(x_k)$. This is clearly a very simple convexification procedure.

2.2 Shifted Spectra

To obtain, in general, a better search directions p_k, we should search for better matrices B_k (as opposed to $B_k = I$ associated with the steepest descent direction) that, in some sense, more accurately approximates the curvature of the objective function. Since we know that second-derivate information is more accurately obtained from H_k, we seek a procedure that chooses B_k by using the matrix H_k, as opposed to completely discarding it as is done with the steepest descent direction.

One such strategy is what we call the *shifted spectra* method (see Algorithm 2). The goal of this strategy is to shift the spectrum of H by adding to it a diagonal matrix with large enough positive elements. In this strategy, if the smallest eigenvalue of H_k is greater than the ϵ that is input, no modification will take place so that $B_k = H_k$. In this case, the solution to (2.2) satisfies $H_k p_k = -g_k$, which is the Newton iteration for finding a zero of $g(x)$ from the base point x_k.

Algorithm 2 Shifted spectra method

1: **Input** H and $\varepsilon > 0$.
2: Compute the spectral decomposition $H = V\Lambda V^T$.
3: Find the minimum eigenvalue λ_{min} among the diagonal elements of Λ.
4: **if** $\lambda_{min} \geq \varepsilon$ **then**
5: Set $\bar{\Lambda} \leftarrow \Lambda$.
6: **else**
7: Set $\bar{\Lambda} \leftarrow \Lambda + (\varepsilon - \lambda_{min})I$.
8: **return** $B = V\bar{\Lambda}V^T \succ 0$

Algorithm 2 results in a matrix $B_k \succ 0$. This ensures that the direction p_k that solves (2.2) is a descent direction. It is readily seen that $B_k \succ 0$ implies that $B_k^{-1} \succ 0$

since if (v, λ) is an eigenpair of B_k (λ must be strictly positive) that

$$B_k v = \lambda v \quad \text{if and only if} \quad \frac{1}{\lambda} v = B_k^{-1} v$$

from which it follows that B_k^{-1} is also positive definite. Thus, we know that p_k is a descent direction anytime $g_k \neq 0$ (i.e., anytime x_k does not satisfy the first-order necessary optimality condition) since

$$g_k^T p_k = -g_k^T B_k^{-1} g_k < 0.$$

We mention this since line-search methods require p_k to be a descent direction.

We finish this section with a few comments. First, as described above, when H_k is sufficiently positive definite, Algorithm 2 will produce $B_k = H_k$ and the resulting step p_k will be the Newton step. Thus we would expect *local* fast convergence of a line search method based on the shifted spectra convexification procedure, at least near a minimizer x^* that satisfies second-order sufficient conditions. Globally, however, there is no guarantee about the size of the modification that is produced by Algorithm 2. Intuitively, however, one might suspect that this procedure would often lead to a Hessian matrix approximation B_k that is more accurate and informative than the simple choice of $B_k = I$ used by the steepest descent direction.

2.3 Modified Newton

The *modified Newton* procedure for obtaining the matrix B_k is more refined than simply shifting the spectrum of H_k as is done in the previous section by the shifted spectra scheme. The idea behind modified Newton is to change only the eigenvalues that are not sufficiently positive, which is reasonable since the matrix B_k is positive definite if and only if the eigenvalues of B_k are all strictly positive.

The eigenvalues of H_k are obtained, and ultimately modified, by performing a spectral decomposition of H_k. Several strategies for modifying the eigenvalues are known, and here we present two of them; see Algorithms 3 and 4. We also mention that a related strategy based on using a symmetric block factorization is also possible and is more relevant when the number of variables n becomes larger than is practical for computing the spectral decomposition in Algorithms 3 and 4.

Algorithm 3 A modified Newton method: variant 1

Input a symmetric matrix H.
Choose $\beta > 1$ as the desired bound on the condition number of B.
Compute the spectral decomposition $H = V\Lambda V^T$.
if $H = 0$ **then**
    Set $\varepsilon \leftarrow 1$.
else
    Set $\varepsilon \leftarrow \|H\|_2/\beta > 0$.
Compute

$$\bar{\Lambda} = \operatorname{diag}(\bar{\lambda}_1, \bar{\lambda}_2, \ldots, \bar{\lambda}_n) \;\text{ with }\; \bar{\lambda}_j = \begin{cases} \lambda_j & \text{if } \lambda_j \geq \varepsilon, \\ \varepsilon & \text{otherwise.} \end{cases}$$

return $B = V\bar{\Lambda}V^T \succ 0$, which satisfies $\operatorname{cond}(B) \leq \beta$.

In Algorithm 3, the eigenvalues of H that are smaller than ε are replaced by ε. The

resulting positive-definite matrix B is then the sum of H and a positive semidefinite matrix E, where the ith eigenvalue of E is the maximum of zero and $\varepsilon - \lambda_i$, where λ_i is the ith eigenvalue of H.

In contrast to Algorithm 3, Algorithm 4 only modifies eigenvalues with absolute value less than ε. This strategy may result in better approximations B to H (in comparison with Algorithm 3) since large negative eigenvalues are preserved in magnitude, but not sign.

Algorithm 4 A modified Newton method: variant 2

Input a symmetric matrix H.
Choose $\beta > 1$ as the desired bound on the condition number of B.
Compute the spectral decomposition $H = V\Lambda V^T$.
if $H = 0$ **then**
    Set $\varepsilon \leftarrow 1$.
else
    Set $\varepsilon \leftarrow \|H\|_2/\beta > 0$.
Compute

$$\bar{\Lambda} = \operatorname{diag}(\bar{\lambda}_1, \bar{\lambda}_2, \ldots, \bar{\lambda}_n) \text{ with } \bar{\lambda}_j = \begin{cases} \lambda_j & \text{if } \lambda_j \geq \varepsilon, \\ -\lambda_j & \text{if } \lambda_j \leq -\varepsilon, \\ \varepsilon & \text{otherwise.} \end{cases}$$

return $B = V\bar{\Lambda}V^T \succ 0$, which satisfies $\operatorname{cond}(B) \leq \beta$.

Modified Newton methods typically produce iterates with a fast local rate of convergence, and are efficient on small to medium size problems. However, computation of the required spectral decomposition is generally very expensive and makes these methods not suitable for large scale problem.

2.4 Quasi-Newton

Quasi-Newton and other related methods have been well studied.[11–15] Here, we focus on quasi-Newton methods that are most applicable to line search methods and, in particular, aim to have a positive-definite matrix B_k during each iteration. (This contrasts, e.g., the *SR1* quasi-Newton update[16] that only aims for symmetry). Unlike the modified Newton paradigm in which the Hessian matrix H_k is modified to obtain B_k, quasi-Newton methods maintain a positive-definite matrix by updating (in a simple way) the previous approximation B_{k-1} in a manner that forces B_k to have reliable curvature information along certain directions.

2.4.1 BFGS

In this thesis, we focus on the most popular quasi-Newton updating scheme known as the BFGS update,[17–20] which is also generally accepted as the best update for line search based algorithms. We comment, however, that an entire class of updates (the Broyden class[21]) has been studied. The *BFGS* algorithm for computing the sequence of matrices $\{B_k\}$ is given by Algorithm 5.

Algorithm 5 The BFGS update.

1: **Input** vectors x_{k-1}, x_k, g_{k-1}, and g_k, and symmetric matrix $B_{k-1} \succ 0$.
2: Set $s_{k-1} \leftarrow x_k - x_{k-1}$ and $y_{k-1} \leftarrow g_k - g_{k-1}$.
3: Set $B_k \leftarrow B_{k-1} - \frac{B_{k-1}s_{k-1}(B_{k-1}s_{k-1})^T}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1}y_{k-1}^T}{y_{k-1}^T s_{k-1}}$.
4: **return** B_k

Rather than computing B_k directly from H_k, the BFGS update obtains B_k by updating the previous approximation B_{k-1} with a rank-2 matrix. In fact, the BFGS update may be derived by seeking a rank-2 update to B_{k-1} that is symmetric and satisfies the *secant equation* $B_k s_{k-1} = y_{k-1}$. Of course, the matrix B_k should also be positive definite so that the subsequent search direction p_k computed via $B_k p_k = -g_k$ is a descent direction for f at x_k. With this in mind, we mention that the step length α_k is usually chosen to satisfy the *Wolfe* conditions since it may be shown that in that case the updated matrix B_k will be positive definite.[22]

Since the BFGS update does not require the Hessian matrix, it is an attractive option when second derivative matrices are either unavailable or very expensive to compute. Since the BFGS updated matrix contains some curvature information, a line search method based on BFGS usually converges faster than the steepest descent method, which implicitly uses the identity matrix to approximate H_k.

2.4.2 LBFGS

To use the approximation B_k resulting from the BFGS update requires either the explicit formulation of the matrix, which is likely to be dense, or the ability to compute the action of B_k on any given vector, which may be computed as a sequence of products with rank-two vectors. The bottom line is that this approach is generally too expensive for large scale problems. *Limited Memory BFGS (LBFGS)* [23] is a truncated version of BFGS: the formulation of B_k only requires the most recent rank-

2 update vectors, and is typically only represented implicitly by storing these pairs.

In Algorithm 6, we present the "two loop recursion" for LBFGS as presented in [16, Algorithm 9.1]. We note that this form directly approximates the inverse of B_k. To be more precise, the output of the algorithm is the vector $p_k = -W_k g_k$, where W_k is the LBFGS update to the *inverse* of H_k. We note that the secant equation for the approximation of the inverse of the Hessian is $W_k y_{k-1} = s_{k-1}$. This scheme is quite effective for unconstrained optimization. (We also note that it has been extended to bound-constrained problems[16] since it directly outputs the search direction p_k.)

Algorithm 6 The LBFGS two-loop recursion.

Input the gradient g_k and memory length $m \geq 1$.
Set $q \leftarrow g_k$.
for i=$k-1$, $k-2$, ..., $k-m$ **do**
    Set $\rho_i \leftarrow 1/(y_i^T s_i)$.
    Set $\alpha_i \leftarrow \rho_i s_i^T q$.
    Set $q \leftarrow q - \alpha_i y_i$.
Set $W_k^{(0)} \leftarrow y_{k-1}^T s_{k-1}/(y_{k-1}^T y_{k-1})$.
Set $z \leftarrow W_k^{(0)} q$.
for i= $k-m$, $k-m+1$,..., $k-1$ **do**
    Set $\chi \leftarrow \rho_i y_i^T z$.
    Set $z \leftarrow z + s_i(\alpha_i - \chi)$.
return the search direction $p_k \leftarrow -z$, i.e., $p_k = -W_k g_k$.

Although the LBFGS method may not approximate the Hessian (more accurately its inverse) as well as BFGS, it is far less expensive to obtain when the *memory* length $m > 0$ is small (typically m is chosen to be between 3 and 10). This is the primary reason why methods based on LBFGS are common in many large-scale applications in machine learning.[24]

2.5 Linear Conjugate Gradient Method

The linear *conjugate gradient (CG)* method is a popular choice for approximately solving large-scale and sparse linear systems of equations. It was first introduced by Magnus Hestenes and Eduard Stiefel[25] to solve linear systems of equations defined by a positive-definite matrix, and later generalized by Fletcher and Reeves[26] to handle more general nonlinear problems, with much related work following.[27–31] In this thesis only the *linear CG* method is relevant, and is the topic of this section.

The linear CG method is used to approximately solve the linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, $x \in \mathbb{R}^n$, and $b \in \mathbb{R}^n$. This problem is equivalent to finding the minimizer of $q(x) := \frac{1}{2}x^TAx - b^Tx$ since $\nabla q(x) = Ax - b$ and $\nabla^2 q(x) \equiv A \succ 0$. If we define the residual of the system to be $r := \nabla f(x) = Ax - b$, then when r is approaching zero, Ax is approaching b.

The linear CG algorithm for solving $Ax = b$ is given by Algorithm 7. In short, linear CG iteratively computes estimates of a solution to $Ax = b$ by minimizing q over a sequence of expanding subspaces (Krylov subspaces). These subspaces are determined through the computation of A-conjugate (sometimes called A-orthogonal) directions $\{s_k\}$, which satisfy $s_i^TAs_j = 0$ for all $i \neq j$. The dominant cost during each iteration of linear CG is a single matrix-vector product, which makes this a particularly attractive option for large-scale sparse problems.

Algorithm 7 The CG method.

Input a symmetric matrix $A \succ 0$ and vectors x_0 and b.
Set $r_0 \leftarrow Ax_0 - b$, $s_0 \leftarrow -r_0$, and $k \leftarrow 0$.
while $\|r_k\| \geq 10^{-8} \max(1, \|r_0\|_2)$ **do**
    Set $\alpha_k \leftarrow (r_k^T r_k)/(s_k^T A s_k)$.
    Set $x_{k+1} \leftarrow x_k + \alpha_k s_k$.
    Set $r_{k+1} \leftarrow r_k + \alpha_k A s_k$.
    Set $\beta_{k+1} \leftarrow (r_{k+1}^T r_{k+1})/(r_k^T r_k)$.
    Set $s_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} s_k$.
    Set $k \leftarrow k + 1$.
return x_k

It remains to show how the linear CG method, which is defined for solving positive-definite linear systems of equations, can be used to solve unconstrained nonlinear optimization problems. This can be motivated by problem (2.2) for which we assume (temporarily) that $H_k \equiv B_k$. With this choice, if the solution to (2.2) exists, then it also solves the linear system of equations $H_k p = -g_k$. It is then tempting to apply the linear CG method to this system of equations, but one must be careful because H_k is not necessarily positive definite. In particular, it is possible that Algorithm 7 might not succeed because some of the denominators may be zero, for example. Even if it does not "crash", there is no reason to suspect that the output is of any real use.

Algorithm 8 is a modification of the linear CG method that addresses the concerns discussed in the previous paragraph. First, the comparison of $Ax = b$ to the system $Hp = -g$ motivates the replacements $A \leftarrow H$, $b \leftarrow -g$, and $x \leftarrow p$. Based on this comparison, it makes sense that Algorithm 8 is called *Newton CG* (N-CG).

Algorithm 8 The Newton CG method

Input a symmetric matrix H and vector g.
Set $p_0 \leftarrow 0$, $r_0 \leftarrow g$, $s_0 \leftarrow -g$ and $k \leftarrow 0$.
while $\|r_k\| \geq 10^{-8} \max(1, \|r_0\|_2)$ **do**
    if $s_k^T H s_k > 0$ **then**
        Set $\alpha_k \leftarrow (r_k^T r_k)/(s_k^T H s_k)$.
    else
        if $k = 0$ **then**
            return $p_k \leftarrow -g$
        else
            return p_k
    Set $p_{k+1} \leftarrow p_k + \alpha_k s_k$.
    Set $r_{k+1} \leftarrow r_k + \alpha_k H s_k$.
    Set $\beta_{k+1} \leftarrow (r_{k+1}^T r_{k+1})/(r_k^T r_k)$.
    Set $s_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} s_k$.
    Set $k \leftarrow k + 1$.
return p_k

In Algorithm 8, if negative curvature is never encountered, i.e., $s_k^T H s_k > 0$ for all directions s_k, then it reduces to exactly the linear CG method. However, if negative curvature is detected, termination ensues with either the steepest descent step (if $k = 0$) or the previously computed CG iterate (if $k \geq 1$). Since there is no guarantee on how many CG iterations may be performed before negative curvature is encountered, it is a real possibility that early termination in Algorithm 8 will result in an poor search direction p_k. We provide a new approach in Chapter 3 based on linear CG that appears to be more reliable and only slightly more expensive.

2.6 Tensor Model Based Methods

Tensor methods for unconstrained optimization were first introduced by Schnabel and Chow[32] for small to moderate-sized problems. Their method was based on minimizing an approximation to the fourth-order tensor model subject to a trust-region constraint. (Bouaricha[33] later introduced a cheaper way to minimize the tensor model that made the method suitable for large scale and sparse problems.) A second approach was taken by Biglari and Ebadian[34] by using a fourth-order tensor model to formulate a modified LBFGS update based on using higher-order derivatives to formulate the secant equation. We now briefly describe these two approaches.

2.6.1 Tensor Model Minimization Methods

A less well-known, but interesting, method for unconstrained optimization may be derived from fourth-order tensor model of the object function f given by

$$f(x_k + p) \approx f(x_k) + g_k \cdot p + \frac{1}{2} H_k \cdot p^2 + \frac{1}{6} T_k \cdot p^3 + \frac{1}{24} V_k \cdot p^4, \tag{2.6}$$

where $g_k \cdot p$ denotes $g_k^T p$, $H_k \cdot p^2$ denotes $p^T H_k p$, $T_k \cdot p^3$ denotes $T_k \cdot ppp$, and $V_k \cdot p^4$ denotes $V_k \cdot pppp$. The third and fourth order tensors T_k and V_k are approximations to $\nabla^3 f(x_k)$ and $\nabla^4 f(x_k)$, and their actions are defined as follows.

Definition 2.6.1 (Third-order tensor term) *Let $T \in \mathbb{R}^{n \times n \times n}$, then given the*

three vectors $u, v, w \in \mathbb{R}^n$, we define

$$T \cdot uvw = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} T(i,j,k)u(i)v(j)w(k).$$

Definition 2.6.2 (Fourth-order tensor term) *Let $V \in \mathbb{R}^{n \times n \times n \times n}$, then given the four vectors $r, u, v, w \in \mathbb{R}^n$, we define*

$$V \cdot ruvw = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} V(i,j,k,l)r(i)u(j)v(k)w(l).$$

The choice of $V_k = \nabla^3 f(x_k)$ and $T_k = \nabla^4 f(x_k)$ in (2.6) leads to models that are expensive and difficult to optimize. Schnabel and Chow chose T_k and V_k to be simple low-rank symmetric approximations to $\nabla^3 f(x_k)$ and $\nabla^4 f(x_k)$ that are formed from previous function/gradient values. They then computed the search direction by finding an approximate minimizer of the model subject to a trust region constraint. Bouaricha extended their method to large sparse problems by only using the most recent function/gradient values to formulate similar simple low-rank approximations. They then proceed to find the search direction by using a Cholesky decomposition instead of a QR factorization, which better utilizes the sparsity of the Hessian. Algorithm 9 gives the basic idea of this general tensor model minimization framework.

Algorithm 9 A tensor method framework.

Input $x_0 \in \mathbb{R}^n$, $\delta_0 > 0$, and an integer $n_p \geq 1$.
Choose stopping tolerance $\varepsilon > 0$.
Set $k \leftarrow 0$.
loop
    Compute f_k, g_k, and H_k.
    if $\|g_k\| < \varepsilon$ **then**
        return x_k
    Select n_p past points to use in the tensor model.
    Compute the tensor matrices T_k and V_k.
    Find p_{tensor} as a solution to a trust-region problem with radius δ_k.
    Find p_k as the solution to (2.2) for some $B_k \succ 0$.
    Find $\alpha_k > 0$ such that the Armijo condition (see Definition 2.0.6) is satisfied.
    if $f(x_k + p_{tensor}) \leq f(x_k + \alpha_k p_k)$ **then**
        Set $x_{k+1} \leftarrow x_k + p_{tensor}$ and $\delta_{k+1} \leftarrow 2\delta_k$.
    else
        Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and $\delta_{k+1} \leftarrow \frac{1}{2}\delta_k$.
    Set $k \leftarrow k + 1$.

Algorithm 9 requires the computation of the step p_{tensor} based on the tensor model (2.6) and the step p_k based on the quadratic model (2.2). After a line search along p_k is permed, the update is then performed based on which one achieved a greater function decrease. In this manner, it is not difficult to see that global convergence results may be established, but at the cost of additional computation.

2.6.2 Tensor Model Based Modified LBFGS

Biglari and Ebadian[34] modified the secant equation used in the LBFGS framework by employing the fourth order tensor model (2.6). Roughly speaking, they eliminate the fourth order tensor V_k by combining two equations: the equation that results from

substituting $p = -s_{k-1}$ into (2.6) and the equation obtained by taking the gradient of both sides of (2.6) and then substituting $p = -s_{k-1}$. After this manipulation and defining the auxiliary quantity

$$\phi_{k-1} := 4(f_{k-1} - f_k) + 2(g_k + g_{k-1})^T s_{k-1},$$

they arrived at the identity

$$\begin{aligned} s_{k-1}^T H_k s_{k-1} &= \phi_{k-1} + y_{k-1}^T s_{k-1} + \frac{1}{6} T_k s_{k-1}^3 + \mathcal{O}(\|s_{k-1}\|^5) \\ &= \phi_{k-1} + y_{k-1}^T s_{k-1} + \mathcal{O}(\|s_{k-1}\|^3) \\ &= \widetilde{y}_{k-1}^T s_{k-1} + \mathcal{O}(\|s_{k-1}\|^3), \end{aligned} \tag{2.7}$$

where

$$\widetilde{y}_{k-1} = \left(1 + \frac{\phi_{k-1}}{y_{k-1}^T s_{k-1}}\right) y_{k-1}. \tag{2.8}$$

(For more details of this derivation, see [34, Section 2].) We can use (2.7) to motivate the use of the *modified* secant equation $W_k \tilde{y}_{k-1} = s_{k-1}$, where W_k (see Algorithm 6) approximates the inverse of the Hessian. Specifically, if we multiply both sides by W_k^{-1} and then by s_{k-1}^T, we arrive at $s_{k-1}^T \widetilde{y}_{k-1} = s_{k-1}^T W_k^{-1} s_{k-1}$. This can then be compared with (2.7) to motivate the secant equation, if we keep in mind that W_k^{-1} may be viewed as an approximation to H_k. Therefore, to implement their method one only needs to base the LBFGS update on the modified secant equation $W_k \widetilde{y}_{k-1} = s_{k-1}$ instead of

the traditional secant equation $W_k y_{k-1} = s_{k-1}$; this amounts to substituting $\widetilde{y}_{k-1}$ for y_{k-1} in the two loop recursion formula associated with LBFGS given by Algorithm 6.

The authors claim to use higher order information from the tensor model (2.6), but it appears from the derivation that, in fact, they choose to ignore the third order term. Thus, it is unclear whether this modified secant equation leads to an LBFGS update that is any better than the traditional LBFGS update.

Chapter 3

A New Convexification Procedure

In the previous chapter, we reviewed several commonly used line-search convexification procedures, and in the process pointed out a shortcoming of N-CG (see Section 2.5). In this chapter, we introduce a modification of N-CG that we call MN-CG, which is more robust and requires very little extra computation.

The N-CG Algorithm 8 obtains the search direction by applying CG to the linear system $Hp = -g$, where we use $H = H_k$ and $g = g_k$ since they are fixed throughout the chapter. Also, recall that CG is only guaranteed to succeed if $H \succ 0$. To avoid the failure that can arise from negative eigenvalues in H, N-CG terminates CG if a direction of negative curvature is encountered. Although early termination does ensure that the direction returned, i.e., the previous CG iterate, is a descent direction, it is very possible that it is not a very good one. We provide a new method called MN-CG (Modified Newton CG) that is designed to improve this situation.

3.1 The Derivation of MN-CG

Inspired by N-CG and modified Newton methods, we modify H when negative curvature within the CG iteration is encountered. At a high level, our method computes a sequence of matrices $\{B^{(j)}\}$ from H with $B^{(0)} = H$. In particular, $B^{(j+1)}$ is obtained from $B^{(j)}$ by a rank-one modification along the first direction of negative curvature encountered when CG is applied to the system $B^{(j)}p = -g$. It is, therefore, convenient to define these directions as the set $\{s^{(j)}\}$ that depends on a set of symmetric (not necessarily positive-definite) matrices $\{B^{(j)}\}$ with $B^{(0)} = H$.

Definition 3.1.1 (The sequence $\{s^{(j)}\}$) *Define $s^{(j)} \in \mathbb{R}^n$ to be the first direction of non-positive curvature found in the CG Algorithm 7 when applied to $B^{(j)}p = -g$.*

Since $s^{(j)}$ is a direction of negative curvature for $B^{(j)}$, we choose to define

$$B^{(j+1)} = B^{(j)} + \theta^{(j)} s^{(j)} s^{(j)^T} \tag{3.1}$$

for some positive number $\theta^{(j)}$ chosen to ensure ${s^{(j)}}^T B^{(j+1)} s^{(j)} > 0$. To have a chance of being practical, we choose to limit the number of times that CG must be restarted, i.e., how large j may become. We accomplish this by simply including a hard upper bound j_{max} on the number of times for which the update (3.1) may be performed. If the maximum allowed updates j_{max} is reached, we obtain the search direction p_k from algorithm N-CG with input matrix $B^{(j_{max})}$. There is no guarantee that this will give a good search direction, but there is reason to believe that it may be more

reliable than simply terminating as soon as any negative curvature is encountered while solving the original system $Hp = -g$, as is done by N-CG, i.e., Algorithm 8.

Next, we discuss how to efficiently calculate with $B^{(j)}$ as needed. To this end, suppose that we have $B^{(j)}$ and call Algorithm 7 to solve the linear system $B^{(j)}p = -g$. Also, note that from (3.1) that

$$B^{(j)} = H + \sum_{l=0}^{j-1} \theta^{(l)} s^{(l)} {s^{(l)}}^T \tag{3.2}$$

so that, in particular, we have $B^{(0)} = H$. It is then easy to see from (3.2) that matrix-vector multiplications (as needed by the CG algorithm for solving $B^{(j)}p = -g$) may be cheaply computed since for any vector v we have

$$B^{(j)}v = Hv + \sum_{l=0}^{j-1} \theta^{(l)} (v^T s^{(l)}) s^{(l)}, \tag{3.3}$$

which requires a matrix-vector product with H, j inner products with v, and $j + 1$ vector additions. From this, it is also easy to see that

$$v^T B^{(j)} v = v^T H v + \sum_{l=0}^{j-1} \theta^{(l)} \left(v^T s^{(l)}\right)^2, \tag{3.4}$$

which only requires one additional inner product calculation. Note, that this implies, since $s^{(j)}$ is by definition a negative curvature direction for $B^{(j)}$, that

$$s^{(j)^T} B^{(j)} s^{(j)} = s^{(j)^T} H s^{(j)} + \sum_{l=0}^{j-1} \theta^{(l)} \left(s^{(j)^T} s^{(l)} \right)^2 < 0,$$

which implies, since all terms in the summation are positive, that

$$s^{(j)^T} H s^{(j)} < 0. \tag{3.5}$$

We now discuss how to choose the positive sequence $\{\theta^{(j)}\}$ in (3.1) appropriately. In particular, we will simply show how to compute $\theta^{(j)}$ under the assumption that $\{\theta^{(l)}\}_{l=0}^{j-1}$ have already been computed. Motivated by (3.2), we choose $\theta^{(j)}$ such that

$$s^{(j)^T} B^{(j+1)} s^{(j)} = s^{(j)^T} B^{(j)} s^{(j)} + \theta^{(j)} (s^{(j)^T} s^{(j)})^2 = \gamma^{(j)} > 0, \tag{3.6}$$

which means that

$$\theta^{(j)} = \frac{\gamma^{(j)} - s^{(j)^T} B^{(j)} s^{(j)}}{\|s^{(j)}\|^4} = \frac{\gamma^{(j)} - s^{(j)^T} H s^{(j)} - \sum_{l=0}^{j-1} \theta^{(l)} \left(s^{(l)^T} s^{(j)} \right)^2}{\|s^{(j)}\|^4}. \tag{3.7}$$

It follows from (3.6) that a sufficient condition for $s^{(j)^T} B^{(j+1)} s^{(j)} = \gamma^{(j)}$ to be positive is to choose $\gamma^{(j)}$ at least as large as some fraction of $-s^{(j)^T} H s^{(j)}$. There are many such choices and we consider several. In an attempt to ensure sufficient

directions of negative curvature, we first consider the following three choices of $\gamma^{(j)}$:

$$\gamma_1^{(j)} = \varepsilon\|s^{(j)}\| - {s^{(j)}}^T H s^{(j)} \text{ for some choice of } \varepsilon > 0, \tag{3.8}$$

$$\gamma_2^{(j)} = -{s^{(j)}}^T H s^{(j)}, \text{ and} \tag{3.9}$$

$$\gamma_3^{(j)} = (1-a){s^{(j)}}^T H s^{(j)} + \sum_{l=0}^{j-1} \theta^{(l)} \left({s^{(l)}}^T s^{(j)}\right)^2 \text{ for some choice of } a > 2, \tag{3.10}$$

which in view of (3.7) lead to the updates

$$\theta_1^{(j)} = \frac{\varepsilon\|s^{(j)}\|_2 - 2{s^{(j)}}^T H s^{(j)} - \sum_{l=0}^{j-1} \theta^{(l)} \left({s^{(l)}}^T s^{(j)}\right)^2}{\|{s^{(j)}}^4\|}, \tag{3.11}$$

$$\theta_2^{(j)} = \frac{-2{s^{(j)}}^T H s^{(j)} - \sum_{l=0}^{j-1} \theta^{(l)} \left({s^{(l)}}^T s^{(j)}\right)^2}{\|{s^{(j)}}^4\|}, \text{ and} \tag{3.12}$$

$$\theta_3^{(j)} = \frac{-a s_k^T H s_k}{\|s^{(j)}\|^4}. \tag{3.13}$$

Finally, we consider the choice of

$$\theta_4^{(j)} = \frac{-2{s^{(j)}}^T H s^{(j)} - j \min_{0\le l\le j-1}\{\theta^{(l)}\} \min_{0\le l\le j-1}\{{s^{(l)}}^T s^{(j)}\}^2}{\|s^{(j)}\|^4}, \tag{3.14}$$

which is a valid choice since $\theta_4^{(j)} \ge \theta_2^{(j)}$. In the next section we formally state our complete algorithm.

3.2 The Formal Statement of MN-CG

Our new algorithm, called MN-CG, is stated as Algorithm 10. The algorithm is fairly simple. During each iteration, we apply linear CG to the system of equations $B^{(j)}p = -g$. Either the CG method finds an approximate solution p_k that satisfies its own termination conditions, or it encounters a direction $s^{(j)}$ of negative curvature for $B^{(j)}$. In the latter case, we modify $B^{(j)}$ as described by (3.1) to obtain $B^{(j)}$. The integer j_{max} is used to limit the number of times that this process may be attempted. We note that anytime CG is called in line 4 or 9, Algorithm 7 requires matrix vector products with matrices of the form $B^{(j)}$, which can be efficiently computed from (3.3) provided j_{max} is not chosen too large and H is sparse.

Algorithm 10 Our modified Newton CG algorithm (MN-CG).

1: **Input** matrix H, positive integer j_{max}, and vector g.
2: Set $j \leftarrow 0$ and $B^{(0)} \leftarrow H$.
3: **while** $j < j_{max}$ **do**
4: Apply CG (Algorithm 7) to the linear system $B^{(j)}p = -g$.
5: **if** Algorithm 7 encounters a direction of negative curvature, call it $s^{(j)}$ **then**
6: Compute $B^{(j+1)}$ from (3.1) with $\theta^{(j)}$ defined by any of (3.11)–(3.14).
7: **else**
8: **return** the vector p_k that is returned by Algorithm 7.
9: **return** the p_k that is returned by N-CG Algorithm 8 with input $H = B^{(j_{max})}$.

Chapter 4

Numerical Experiments

In this chapter, we report the result of numerical experiments performed to evaluate the eight popular convexification schemes in Chapter 2 and the four variants of our new scheme described in Chapter 3. The test problems consist of the 67 unconstrained optimization problems from the CUTEst[35] collection with less than or equal to 500 variables. Our implementation is written in MATLAB 2013a, and the tests run on a PC with a 2.4GHz CPU, 2GB of RAM, running the Ubuntu operating system.

In Section 4.1, we present the numerical experiments for only the most commonly used convexification procedures. In Section 4.2, we give the numerical results for the four variants of our newly proposed algorithm. Finally, in Section 4.3, we compare the best performer among the commonly used convexification methods to the best performer among our newly proposed variants, to arrive at the best overall scheme for our specifically chosen selection of test problems.

For every variant tested, the iterations were terminated if the condition

$$\|g_k\|_\infty \leq 10^{-6} \max(1, \|g_0\|_\infty) \tag{4.1}$$

was satisfied, since it implied that x_k was an approximate first-order solution to (1.1). Since some test problems were unbounded below, we also terminated if the condition

$$f(x_k) \leq -10^{14} \tag{4.2}$$

were satisfied, which indicated that the problem was unbounded below. In our experiments, a problem was deemed to have been successfully solved if either (4.1) or (4.2) was verified to hold before the maximum allowed number of iterations of 1000 was reached or (in a few cases) a function/gradient/Hessian evaluation resulted in a "NaN" value, i.e., not a number.

We had to choose other control parameters in addition to the maximum iteration limit of 1000. In particular, in the backtracking line-search procedure (Algorithm 1), we chose $\tau = 0.5$ and $\eta = 10^{-3}$, and for the strong Wolfe conditions (2.4) and (2.5) we chose $c_1 = 10^{-4}$ and $c_2 = 0.9$. We also comment that we used [16, Algorithm 3.5] for finding step lengths satisfying the strong Wolfe conditions, when needed. Finally, we mention that other control parameters are still needed on a method-by-method basis, and will be presented as needed.

4.1 Popular Convexification Schemes

The convexification schemes from Chapter 2 that we tested are listed below. Each method is also accompanied by an acronym that will be used throughout.

- SD: The steepest decent method (Section 2.1).
- MOD-1: modified Newton 1 method (Algorithm 3).
- MOD-2: modified Newton 2 method (Algorithm 4).
- SS: shifted spectra method (Algorithm 2).
- N-CG: Newton CG method (Algorithm 8).
- BFGS: BFGS method (Algorithm 5).
- LBFGS: limited memory BFGS method with memory $m = 4$ (Algorithm 6).
- M-LBFGS: tensor based BFGS method with memory $m = 4$ (Section 2.6.2).

We note that for LBFGS and M-LBFGS a memory of $m = 4$ was used since anything greater was computationally more expensive and indistinguishable otherwise. Also, we used a value of $\beta = 10^8$ as the bound on the condition number for the modified matrices computed for MOD-1 and MOD-2 as needed by Algorithms 3 and 4.

We now clarify exactly the structure of the algorithms that we tested. Every algorithm used the iterate updating formula (2.1). Moreover, they all used the backtracking Armijo linesearch (Algorithm 1), except for the BFGS method, which was

based on satisfying the strong Wolfe conditions (2.4) and (2.5). Thus, it only remains to discuss how the search direction p_k was obtained for each algorithm. For SD this amounted to $p_k = -g_k$. For MOD-1, MOD-2, SS, and BFGS, the search direction p_k was computed as the unique solution to $B_k p = -g_k$, where the symmetric matrix $B_k \succ 0$ was obtained from Algorithms 3, 4, 2, and 5, respectively. The search direction for N-CG was obtained directly from Algorithm 8. The search direction for LBFGS and M-LBGS were obtained from the two-loop recursion Algorithm 6. (For M-LBFGS we replaced in Algorithm 6 the quantity y_{k-1} by $\widetilde{y}_{k-1}$ as motivated by (2.8).)

Our first set of results are shown in Table 4.1. For each method we report the following: the total number of problems successfully solved (#Succ), i.e., termination occurred because either (4.1) or (4.2) was verified; the total number of problems for which the maximum number of allowed iterations was reached (#Maxit); the total number of problems for which a NaN was encountered when evaluating the objective function or its derivatives (#NaN) ; and the average number of iterations (μ(iter)) and function evaluations (μ(feval)) with the average only including those problems that were successfully solved. Based on these results, it appears that the MOD-2 and N-CG methods provide the best balance between robustness (i.e., solving the most problems) and efficiency (i.e., requiring the fewest iterations and function evaluations). Also, it appears that the higher-order tensor information used in M-LBFGS has a positive effect, when compared to is parent algorithm LBFGS, especially for the number of required function evaluations.

Table 4.1: Results for popular convexification methods on the CUTEst problems.

Schemes	SD	MOD-1	MOD-2	SS	N-CG	BFGS	LBFGS	M-LBFGS
#Succ	32	63	63	61	63	59	61	62
#Maxit	34	3	4	5	4	7	6	3
#NaN	1	1	0	1	0	1	0	2
μ(iter)	143	42	37	35	51	40	64	56
μ(feval)	1140	246	81	186	133	117	401	149

Next, we present the results from the numerical tests in the form of performance profiles, as introduced by Dolan and Moré.[36] Consider a performance profile that measures performance in terms of the number of iterations until a solution is found. If the graph associated with an algorithm passes through the point $(\alpha, 0.\beta)$, then it means that on $\beta\%$ of the test problems, the number of iterations required by the algorithm was less than α times the number of iterations required by the algorithm that required the fewest. This means that an algorithm with a higher value on the vertical axis may be considered as more efficient, whereas an algorithm on top at the far right may be considered as more reliable or robust. We note that for every profile, a problem was considered to be successfully solved if termination occurred because either (4.1) or (4.2) was verified to hold.

For the same algorithms as in Table 4.1, the performance profile that measures the number of iterations is given by Figure 4.1, whereas the performance profile that measures the number of function evaluations may be found in Figure 4.2.

Figure 4.1: Performance profile for iterations on the CUTEst problems.

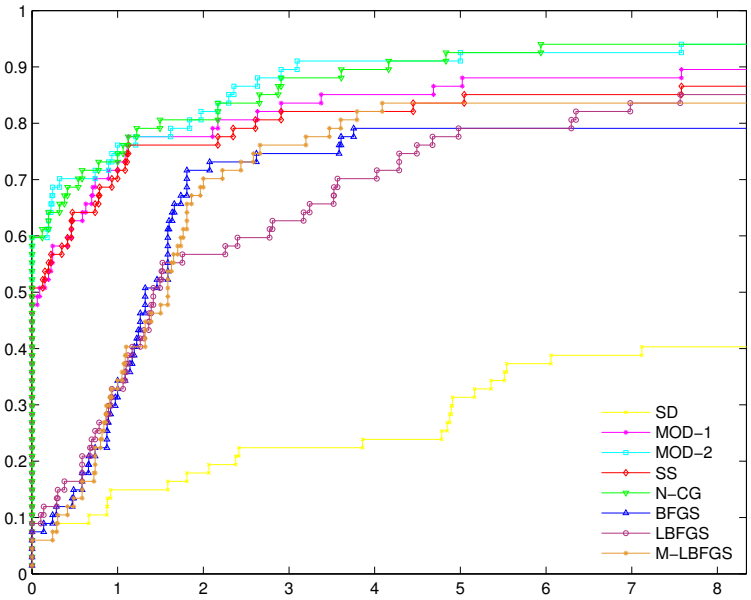
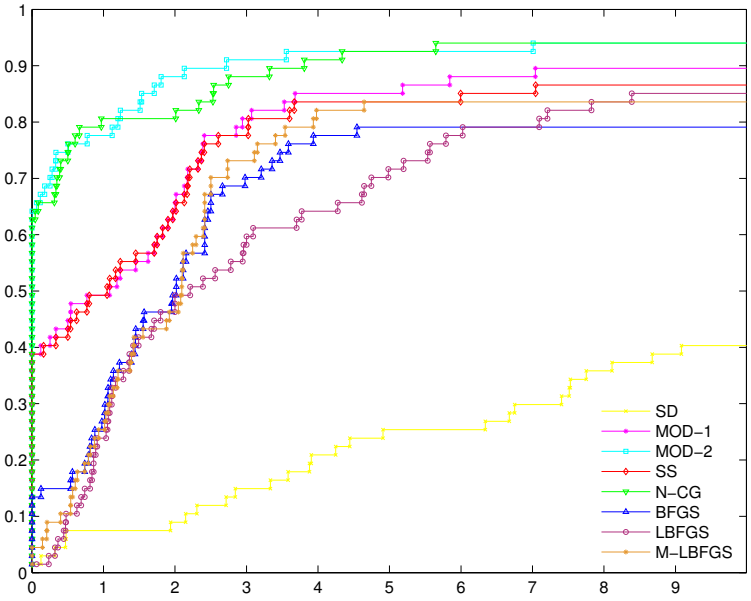


Figure 4.2: Performance profile for function evaluations on the CUTEst problems.



We see from Table 4.1 and Figures 4.1 and 4.2 that algorithms MOD-2 and N-CG performed the best. With respect to the number of problems successfully solved, Table 4.1 shows that MOD-1, MOD-2, and N-CG are tied as the best, followed by M-LBFGS. Although N-CG often requires more iterations and function evaluations than MOD-2, it does not need to factorize the Hessian matrix while MOD-2 does (so do MOD-1 and SS), which is very computationally expensive. We also observe that M-LBFGS is generally more efficient than its parent algorithm LBFGS with respect to the number of required average iterations and average function evaluations, and the number of problems successfully solved. The BFGS algorithm requires fewer iterations and function evaluations (but solver fewer problems) when compared to M-LBFGS and LBFGS, which is not particularly surprising since BFGS keeps full memory in computing the matrix B_k. Unfortunately, this also means that the computations for BFGS are more expensive than both LBFGS and M-LBFGS; in particular, this means that it is not practical for large-scale problems. Finally, algorithm SD failed on roughly half of the test problems, which is not surprising since is uses the very simple search direction $p_k = -g_k$. One must keep in mind, however, that its computation is the cheapest among all of the methods and, in practice, often produces iterates that satisfy a relaxed stopping condition (e.g., condition (4.1) with 10^{-6} replaced by 10^{-2}) in a modest number of iterations. Among all of these methods, we recommend the use of MOD-2 for problems that are small enough to allow for spectral factorizations to be computed, but otherwise recommend N-CG.

4.2 Various Instances of MN-CG

The four instances of MN-CG that were introduced in Chapter 3 are listed below. (In addition, we include the N-CG method as a baseline for comparison.) Each method is also accompanied by an acronym that will be used throughout.

- N-CG: the Newton CG method (Algorithm 8).
- MN-CG-1: the MN-CG method with $\theta^{(j)}$ defined by (3.11).
- MN-CG-2: the MN-CG method with $\theta^{(j)}$ defined by (3.12).
- MN-CG-3: the MN-CG method with $\theta^{(j)}$ defined by (3.13).
- MN-CG-4: the MN-CG method with $\theta^{(j)}$ defined by (3.14).

Every algorithm used the iterate updating formula (2.1) with step length α_k calculated from the back-tracking Armijo linesearch (Algorithm 1). The search direction p_k for MN-CG-1, MN-CG-2, MN-CG-3, and MN-CG-4 was computed by Algorithm 10 with their respective choices of $\theta^{(j)}$, while N-CG calculated p_k by Algorithm 8. (Note that, in fact, Algorithm 8 is equivalent to Algorithm 10 with the choice $j_{max} = 0$.)

Instance MN-CG-1 required a choice for the parameter ε in the update (3.11) to $\theta^{(j)}$. We tested multiple values and settled on $\varepsilon = 0.1$, which seemed to perform the best in our experiments, and is therefore the value used to generate the results in this section. In a similar vein, instance MN-CG-3 also required a choice for the parameter

a used in the update (3.13). Based on preliminary testing, the value $a = 10$ performed the best and is the value used in the results that we present.

The outcomes on the CUTEst collection of problems may be found in Table 4.2. The meaning of every row is the same as that for Table 4.1 in Section 4.1.

Table 4.2: Results for various instances of MN-CG on the CUTEst problems.

Schemes	N-CG	MN-CG-1	MN-CG-2	MN-CG-3	MN-CG-4
#Succ	63	61	65	64	66
#Maxit	4	6	2	3	1
#NaN	0	0	0	0	0
μ(iter)	51	24	34	38	48
μ(feval)	134	59	86	84	116

For the algorithms in Table 4.2, a performance profile that measures the number of iterations is given by Figure 4.3, whereas a performance profile that measures the number of function evaluations may be found in Figure 4.4. (See Section 4.1 for an explanation on how to interpret such performance profiles.)

Figure 4.3: Performance profile for iterations on the CUTEst problems.

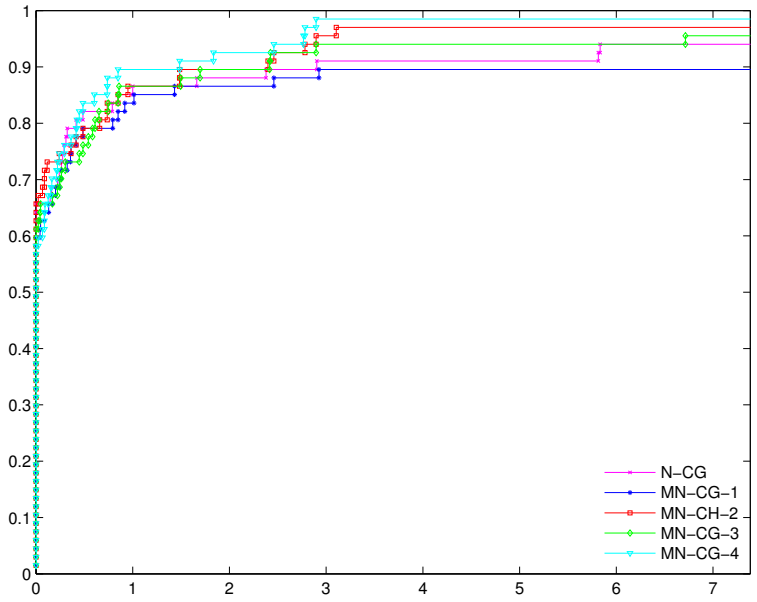
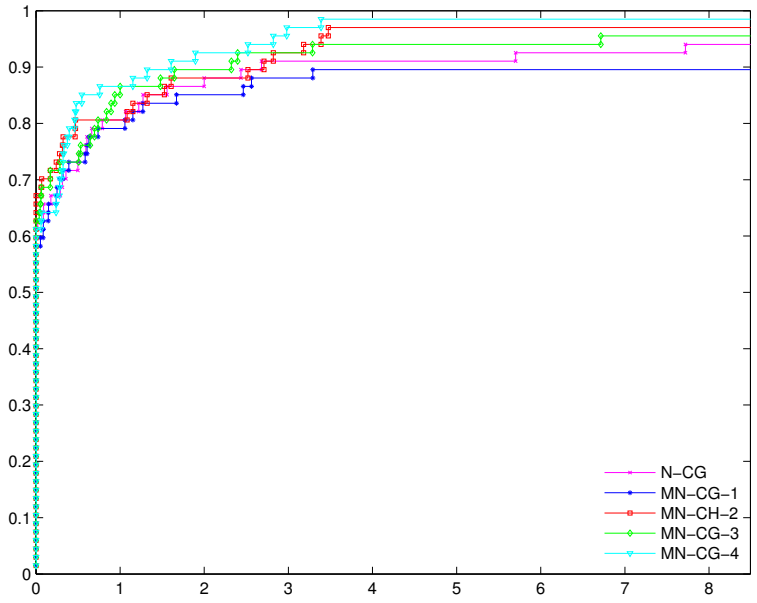


Figure 4.4: Performance profile for function evaluations on the CUTEst problems.



We now make some observations about Table 4.2 and the performance profiles in Figures 4.3 and 4.4. First, the four instances of the new method (i.e., MN-CG-1, MN-CG-2, MN-CG-3, and MN-CG-4) improved upon the baseline algorithm N-CG in terms of the average number of iterations and function evaluations. All of them, except MN-CG-1, also improved upon N-CG in the sense of successfully solving more problems. Second, MN-CG-4 solved the most problems (66/67), with the second best being MN-CG-2 (65/67). The extra problem solved by MN-CG-4 (which took many iterations) was included in the averages μ(iter) and μ(feval), and accounts for why they are lower for MN-CG-2 than for MN-CG4. In summary, we conclude that instances MN-CG-2 and MN-CG-4 perform essentially the same, and are superior to the other instances including the baseline method N-CG.

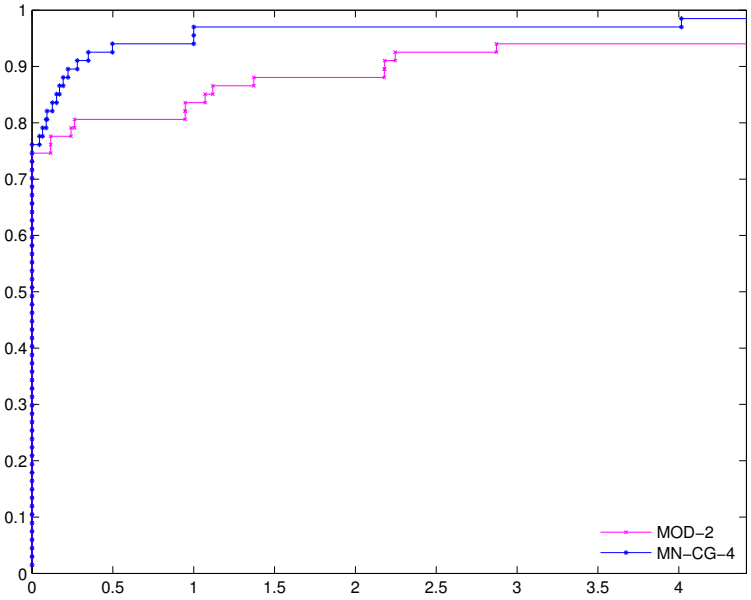
4.3 The Best Overall Methods

We now compare the best method from Section 4.1, namely MOD-2, with the best method from Section 4.2, namely MN-CG-4. Tables 4.1 and 4.2 show that MN-CG-4 solves more problems than MOD-2, but requires a greater average number iterations and function evaluations (computed over the problems that they solved). This latter fact is caused from the fact that the extra problem solved by MN-CG-4 required many iterations and function evaluations, which biased the averages.

Performance profiles that only include algorithms MOD-2 and MN-CG-4 are given

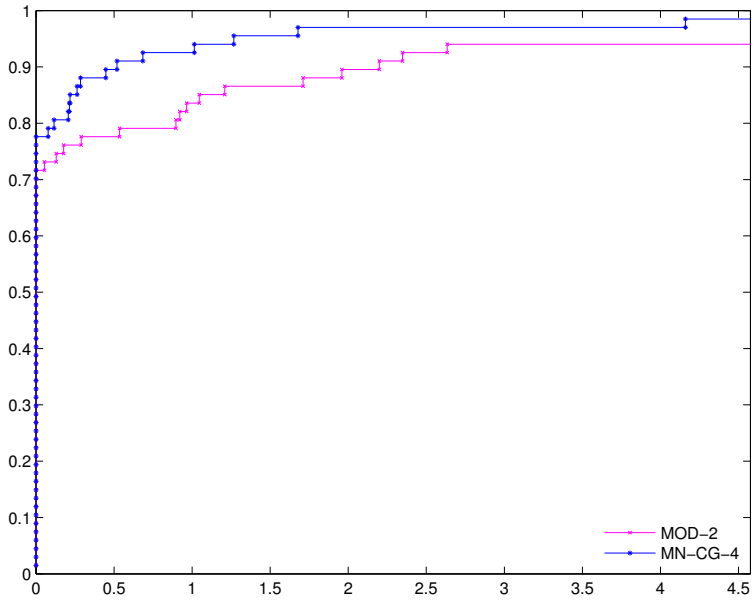
by Figures 4.5 and 4.6. These seem to indicate that MN-CG-4 is better than MOD-2 in terms of iteration and function evaluation counts. This may appear to contradict our claim in the previous paragraph, but one must recall that the performance profiles present a more overall picture of performance for a desired quantity of interest. This contrasts the *average* number of iterations needed for the problems solved, which can easily be biased. The performance profiles simply highlight this fact.

Figure 4.5: Performance profiles for iterations on the CUTEst problems.



It is interesting to see that NM-CG-4 performed better than MOD-2 in our numerical experiments. This is interesting because the implementation of NM-CG-4 only requires the computation of matrix-vector products, which makes it practical for small and large-scale sparse problems. In contrast, algorithm MOD-2 has to compute a spectral decomposition of the Hessian matrix during each iteration, which generally

Figure 4.6: Performance profile for function evaluations on the CUTEst problems.



makes it practical for only small and medium-scale problems. We are quite content with these results since it is great when a single method (e.g., NM-CG-4) performs the best on a large class of problems (e.g., small-, medium-, and large-scale problems).

Chapter 5

Conclusions

In this thesis we studied the effect that convexification methods had on the computed search directions used within line search methods for unconstrained optimization. In Chapter 2 we presented the most commonly used methods, one of which was based on the linear CG method and called N-CG (see Section 2.5). In Chapter 3 we presented a new algorithm based on linear CG that we called MN-CG since it was a modification of the N-CG algorithm. This new method allowed for sequential convexification of the Hessian matrix along certain carefully chosen directions. These directions were chosen as directions of negative curvature encountered during the CG method. The new method was applicable to large scale problems for which Hessian vector products were cheap to compute, such as when the Hessian was sparse. In Chapter 4 we performed numerical experiments that showed that our new method was slightly more expensive per iteration compared to N-CG, but was more reliable.

More generally, our experiments revealed that N-CG and MN-CG, which are both applicable to large-scale problems, were competitive with methods such as modified Newton (Section 2.3) and BFGS (Section 2.4.1), which are both only practical on small to medium scale problems. A variant of BFGS, called LBFGS (Section 2.4.2), which is a common choice for large-scale problems, was not as robust or efficient as N-CG or MN-CG on our collection of test problems.

The new method described in Chapter 3 may be extended easily to *equality* constrained optimization problems. For example, augmented Lagrangian methods solve general nonlinear equality constrained problems by solving a sequence of related nonlinear unconstrained optimization problems. Thus we may use our new convexification scheme to solve this sequence of unconstrained optimization problems.

It may also be possible to extend our ideas to optimization problems with *inequality* constraints. Again, using an augmented Lagrangian approach would mean that a sequence of nonlinear objective functions would be minimized subject to bounds on the optimization variables. Our new convexification scheme is not directly applicable because of the bound constraints. Nonetheless, it may be possible to adapt the ideas developed here by replacing the directions of negative curvature computed during CG with directions of negative curvature that arise while solving the bound-constrained quadratic problems commonly used with, for example, a projected gradient solver. We leave this as a direction of future research.

Bibliography

[1] M. L. Johnson and S. G. Frasier, "[16] nonlinear least-squares analysis," *Methods in enzymology*, vol. 117, pp. 301–342, 1985.

[2] N. I. Gould and P. L. Toint, "Filtrane, a fortran 95 filter-trust-region package for solving nonlinear least-squares and nonlinear feasibility problems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 1, p. 3, 2007.

[3] H. Waki, S. Kim, M. Kojima, and M. Muramatsu, "Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 218–242, 2006.

[4] C. L. Lawson and R. J. Hanson, *Solving least squares problems*. SIAM, 1974, vol. 161.

[5] S. R. Bohme, J. Zorabedian, and D. S. Egilman, "Maximizing profit and endangering health: corporate strategies to avoid litigation and regulation," *International journal of occupational and environmental health*, vol. 11, no. 4, pp. 338–348, 2005.

BIBLIOGRAPHY

[6] C.-J. Ho and H.-S. Lau, "Minimizing total cost in scheduling outpatient appointments," *Management science*, vol. 38, no. 12, pp. 1750–1764, 1992.

[7] A. F. Perold, "Large-scale portfolio optimization," *Management science*, vol. 30, no. 10, pp. 1143–1160, 1984.

[8] B. M. Beamon, "Supply chain design and analysis::: Models and methods," *International journal of production economics*, vol. 55, no. 3, pp. 281–294, 1998.

[9] L. Armijo, "Minimization of functions having lipschitz continuous first partial derivatives," *Pacific Journal of mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[10] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[11] Y.-x. Yuan, "A modified bfgs algorithm for unconstrained optimization," *IMA Journal of Numerical Analysis*, vol. 11, no. 3, pp. 325–332, 1991.

[12] J. Zhang, N. Deng, and L. Chen, "New quasi-newton equation and related methods for unconstrained optimization," *Journal of Optimization Theory and applications*, vol. 102, no. 1, pp. 147–167, 1999.

[13] J.-F. Gerbeau and M. Vidrascu, "A quasi-newton algorithm based on a reduced model for fluid-structure interaction problems in blood flows," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 37, no. 04, pp. 631–647, 2003.

BIBLIOGRAPHY

[14] H. Liu, J. Shao, H. Wang, and B. Chang, "An adaptive sizing bfgs method for unconstrained optimization," *Calcolo*, pp. 1–12, 2014.

[15] F. Modarres Khiyabani and W. June Leong, "Quasi-newton methods based on ordinary differential equation approach for unconstrained nonlinear optimization," *Applied Mathematics and Computation*, vol. 233, pp. 272–291, 2014.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer-Verlag, 1999.

[17] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms 2. the new algorithm," *IMA Journal of Applied Mathematics*, vol. 6, no. 3, pp. 222–231, 1970.

[18] R. Fletcher, "A new approach to variable metric algorithms," *The computer journal*, vol. 13, no. 3, pp. 317–322, 1970.

[19] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Mathematics of computation*, vol. 24, no. 109, pp. 23–26, 1970.

[20] D. F. Shanno, "Conditioning of quasi-newton methods for function minimization," *Mathematics of computation*, vol. 24, no. 111, pp. 647–656, 1970.

[21] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of computation*, pp. 577–593, 1965.

[22] t. . Q. Peter Blomgren.

[23] J. Nocedal, "Updating quasi-newton matrices with limited storage," *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980.

[24] G. Andrew and J. Gao, "Scalable training of l 1-regularized log-linear models," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 33–40.

[25] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*. National Bureau of Standards Washington, DC, 1952, vol. 49.

[26] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[27] J. W. Daniel, "The conjugate gradient method for linear and nonlinear operator equations," *SIAM Journal on Numerical Analysis*, vol. 4, no. 1, pp. 10–26, 1967.

[28] W. W. Hager and H. Zhang, "A new conjugate gradient method with guaranteed descent and an efficient line search," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 170–192, 2005.

[29] Y.-H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM Journal on Optimization*, vol. 10, no. 1, pp. 177–182, 1999.

[30] W. W. Hager and H. Zhang, "A survey of nonlinear conjugate gradient methods," *Pacific journal of Optimization*, vol. 2, no. 1, pp. 35–58, 2006.

BIBLIOGRAPHY

[31] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[32] R. B. Schnabel and T.-T. Chow, "Tensor methods for unconstrained optimization using second derivatives," *SIAM Journal on Optimization*, vol. 1, no. 3, pp. 293–315, 1991.

[33] A. Bouaricha, "Tensor methods for large, sparse unconstrained optimization," *SIAM Journal on Optimization*, vol. 7, no. 3, pp. 732–756, 1997.

[34] F. Biglari and A. Ebadian, "Limited memory bfgs method based on a high-order tensor model," *Computational Optimization and Applications*, pp. 1–10, 2014.

[35] N. I. Gould, D. Orban, and P. L. Toint, "Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization," *Computational Optimization and Applications*, pp. 1–13, 2014.

[36] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

Lingzhou Hong

08/19/1990 Hohhot, Inner Mongolia, China

500 West University Parkway, Apt.15P

Baltimore MD 21210

(410) 736-8392

lhong3@jhu.edu

EDUCATION

09/13 – 12/14 **Johns Hopkins University**, Baltimore, Maryland

MSE. Degree in Applied Mathematics & Statistics

09/09 – 07/13 **Central University of Finance and Economics**, Beijing, China

BA. Degree in Statistics

RESEARCH EXPERIENCE

Data Mining Research

09/14 –Present

- Hurricane Occurrence Frequency Prediction

  Johns Hopkins University

  Advisor: Prof. Seth D. Guikema

  Developed hurricane occurrence frequency predictive model with consideration of lead-time between variables and hurricane season.

02/13 – 05/13

- Heating Oil Futures Prices Prediction

  Central University of Finance and Economics

  Advisor: Prof. Jie Meng

  Developed Heating Oil Futures prices prediction model using Elastic Net and Hidden Markov methods, and identified key factors and their degree of influence on the futures prices.

04/12 – 06/12

- Analysis of Abnormal Housing Price in Hohhot

  Central University of Finance and Economics

  Advisor: Prof. Su Zhang

  Analyzed the long-term influence of macroeconomics factors on Hohhot's housing price. Analyzed the correlations of housing

prices in Hohhot and in major Chinese cities. Developed a neutral network model to predict short-term and long-term housing price, and identified the factors that caused the abnormality of Hohhot's housing price.

Optimization Research

06/14 – 12/14

- Convexification in Unconstrained Continuous Optimization

  Johns Hopkins University

  Advisor: Prof. Daniel P. Robinson

  Conducted a detailed investigation of the practical performance of the most commonly used convexification procedures, proposed a new convexification scheme, and demonstrated its improved performance.

Machine Learning Research

04/13 – 09/13

- Investigation of Taxi Waiting Time Based on GPS Data

  Central University of Finance and Economics

  Advisor: Prof. Jingyi Ma

  Divided collected taxi GPS data into several time-space categories with Fuzzy Clustering, and developed Poisson model to predict the average waiting time of taxi passengers.

Statistical Analysis Research

03/12 – 09/12 • Systemic Risk Analysis of Financial Sectors

Central University of Finance and Economics

Advisor: Prof. Hui Wang

Studied econometric connectedness among financial sectors in China, and observed increasing level of systemic risk and the asymmetry in the degree of connectedness.

06/12 – 07/12 • China's Banking System Stress Test

Central University of Finance of Economics

Advisor: Prof. Hui Wang, Prof. Jingyi Ma

Designed a simplified Elastic-Net-based China's banking system stress testing and scenario analysis framework. Analyzed the feedback effects among credit risk, market risk and liquidity risk. Conducted individual stress tests for a commercial bank using Balance Sheet Analysis method.

09/11 – 04/12 • Analysis of Public's Acceptance of Mobile TV

For China Broadcasting Co., Ltd.

Central University of Finance of Economics

Advisor: Prof. Su Zhang, Prof. Jie Meng

Analyzed public's acceptance of mobile TV through questionnaires. Estimated the market potential and the concentration of mobile TV among students. Provided marketing advice to the mobile TV service provider.

PUBLICATION

Lingzhou Hong, Jing Gao, Lingqing Hong (2012), Analysis of Abnormal Housing Price in Hohhot, *Northern Economy*, 13, 35-38.

HONORS & AWARDS

2013 Ace Manager Investment Banking Competition, top 10%, BNP

2013 Paribas, China

2013 Excellent Senior Thesis, Central University of Finance and

2012 Economics, China

2011 Outstanding Innovation Research, Central University of Finance and Economics, China

Mathematical Modeling Contest, Meritorious, COMAP, US

CUMCM Mathematical Modeling Contest, 2nd Prize, China