COLLEGE OF NATURAL AND COMPUTATIONAL SCIENCE
DEPARTMENT OF MATHEMATICS

TRUST REGION NEWTON WITH CONJUGATE GRADIENT METHOD

A Project Submitted in Partial Fulfillment of the Requirement for the Degree of Master of Science in Mathematics

Hewan Leul

Advisor: Berhanu Guta (PhD)

June, 2018

Addis Ababa, Ethiopia

Addis Ababa University
Department of Mathematics

The undersigned hereby certify that they have read and recommend to the department of mathematics for acceptance of this project entitled **"Trust-Region Newton with Conjugate Gradient Method"** which is done by **Hewan Leul** in partial fulfillment of the requirements for the Degree of Master of Science in Mathematics.

Advisor: Dr. Berhanu Guta

Signature: _____

Date _____

Examiner 1: Dr. Yibeltal Yitayew

Signature: _____

Date _____

Examiner 2: Dr. Tadesse Bekeshie

Signature: _____

Date_____

ADDIS ABABA UNIVERSITY
DEPARTMENT OF MATHEMATICS

Author: Hewan Leul

Title: Trust-Region Newton with Conjugate Gradient Method

Department: Mathematics

Degree: M.Sc.

Convocation: June

Year: 2018

Permission is herewith granted to Addis Ababa University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Hewan Leul:

Signature: _____

Date _____

Contents

Acknowledgements

First of all, I would like to express my immeasurable thanks and gratitude to Almighty GOD who supported and encouraged me throughout my study period. Many individuals participated and contributed a lot in one way or another to ensure the completion of this project. A special thanks goes to my divisor Dr. Berhanu Guta for dedicating his time, providing his generous and constructive comments, advice and moral support towards the successful completion of this project work. I would like to express my gratitude to my family specially my mom who supported me a lot to conduct the project with great courage and motive throughout the study period. I also wish to recognize the School of Graduate Study of Addis Ababa University (AAU) which gave me research grant. Lastly, I am deeply indebted to my instructors who shared me their experiences and gave me scholastic support.

Abstract

In this project we give trust-region algorithms for nonlinear optimization. Trust-region methods are robust and can be applied to ill-conditioned problems. A conjugate gradient trust-region algorithm is presented to demonstrate the trust region approaches. Convergence properties of trust-region with conjugate gradient are given. A trust-region method subproblems for solving unconstrained optimization is proposed. At every iteration, we use the conjugate gradient method or its variation to solve the subproblems approximately. We show that this method has the same convergence properties as the trust-region method based on the conjugate gradient method. Numerical results show that this method is as reliable and more efficient in respect of iterations and evaluations using MATLAB.

Keywords: line search, Newton method, steepest descent, conjugate gradient, trust-region

Introduction

We consider the optimization problem:

$$\begin{aligned} \min \quad & f(x) \\ s.t \quad & x \in \Re^n \end{aligned} \tag{1}$$

Where f is a real-valued twice continuously differentiable function.

For these kinds of problems to find zeros of gradient may not be easy. If f is not convex, there are three possibilities at a zero of its gradient: Maxima, Minima, or Saddle. We can solve this problem by using Newton's method. The advantages of the Newton's method are quadratically convergent from a good starting point if $\nabla^2 \mathrm{f}(x^*)$ is positive definite, simple and easy to implement. However, this method has not been globally convergent for many problems and/or may converge towards a maximum or saddle point of f; the system of linear equations to be solved in each iteration may be ill-conditioned or singular and requires second order derivatives of f. Thus, unconstrained optimization problems are essential in mathematical programming because they occur frequently in many real-world applications, and the methods for such problems are fundamental in the sense that these methods can be either directly applied or extended to optimization problems with constraints. Numerical methods for nonlinear optimization problems are iterative. At the k^{th} iteration, a current approximate solution x_k is available. A new point x_{k+1} is computed by certain techniques, and this process is repeated until a point can be accepted as a solution.

There are many effective algorithms designed for unconstrained optimization problems. Most of these algorithms can be classified into two categories: line search algorithms and trust-region algorithms.

The classical methods for optimization are line search algorithms. Such an algorithm obtains a search direction in each iteration, and searches along this direction to obtain a better point. The search direction is a descent direction, normally computed by solving a subproblem that approximates the original optimization problem near the current iterate. Therefore, unless a stationary point is reached, there always exist better points along the search direction.

Trust-region method is efficient for solving (1). It has mature framework, strong convergence properties and satisfactory numerical results. However, sometimes the trust-region step may be too conservative, especially when the objective function has large convex basins. The trust-region technique may require quit a few iterations to stretch the trust-region in order to contain the local minimizer. Thus, it is natural for us to consider modifying the standard trust-region method to give a new algorithm which can maintain the convergence properties of the trust-region method and needs less computational cost.

By considering quadratic function trust-region method we construct subproblem and find an approximate solution for it. A subproblem without trust-region constraint was introduced into the trust-region framework in order to use the unit stepsize Newton step. This unconstrained subproblem normally gives a longer step, and consequently it is likely that the overall algorithm will reduce the computational cost. Based on the information in the previous iterations, either the trust-region step or the Newton step is used. Moreover, the idea of combining trust-region and line search techniques are also used in that algorithm. The algorithm inherits the convergence result of trust-region method and gives better performance by making good use of the Newton step. However, as the exact minimizer of the trust-region subproblem is needed and the Cholesky factorization of the Hessian matrix is used to decide whether the model function is convex, that algorithm is always not fit for large problems.

Therefore, in this project we propose a trust-region algorithm with subproblems using conjugate gradient method and its variation to solve the subproblems. Our method can be regarded as a modification to the standard trust region method for unconstrained optimization in such a way that the Newton step can be taken as often as possible. A slightly modified conjugate gradient method is used to compute the Newton step. The global convergence and local super-linear convergence results of the algorithm are also given in the project.

The project is organized as follows. In Chapter One, we present some background materials. In Chapter Two, we discuss about conjugate gradient method. In Chapter Three, we give the framework of the method, describe our algorithm and the convergence properties and the numerical results are provided in Chapter Four. Some conclusions are given as the last part.

Chapter 1

Preliminaries

1.1 Basic optimality conditions

Definition 1.1.1. *Let f(x) be objective function and $S \subseteq \Re^n$ be its feasible set, x^* is called optimal solution for a minimization problem,*

$$\begin{aligned} \min \quad & f(x) \\ s.t \quad & x \in S, \end{aligned}$$

if $x^* \in S$ and $f(x^*) \leq f(x)$, $\quad \forall\ x \in S$.

Definition 1.1.2. ***Optimal value***

If $x^* \in S \subseteq \Re^n$ is an optimal solution, then $f(x^*)$ is called the optimal (objective) value.

Definition 1.1.3. ***Global minimizer***

A point x^* is a global minimizer if $f(x^*) \leq f(x)$, $\forall\ x \in \Re^n$.

Definition 1.1.4. ***Local minimizer***

A point x^* is called a local minimizer if there exists $\epsilon < 0$ such that $f(x^*) \leq f(x)$, $\forall\ x \in \Re^n$ satisfying $\|x^* - x\| \leq \epsilon$.

Definition 1.1.5. ***Gradient***

Let f: $\Re^n \to \Re$ be differentiable with respect to each component of its argument. The gradient of f is the vector

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ . \\ . \\ . \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix}$$

Definition 1.1.6. *Hessian*

Let f: $\Re^n \to \Re$ *be twice differentiable with respect to each component of its argument. The Hessian of f is the matrix*

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(x)... & \frac{\partial f}{\partial x_n \partial x_1}(x) \\ . & . \\ . & . \\ . & . \\ \frac{\partial^2 f}{\partial x_1 \partial x_n}(x)... & \frac{\partial^2 f}{\partial x_n \partial x_n}(x) \end{bmatrix}$$

Note that the Hessian ∇^2f is a symmetric matrix. The gradient and Hessian are used in the next Theorem, which is well known from calculus. It tells about approximations to the value of f when you change the argument from x to a neighbouring point x+h.

Theorem 1.1. ***First and second order Taylor expansions***

If f: $\Re^n \to \Re$ *has continuous partial derivatives of second order, then*

$$f(x+h) = f(x) + h^T \nabla f(x) + o(||h||)^2.$$

If f has continuous partial derivatives of third order, then

$$f(x+h) = f(x) + h^T \nabla f(x) + \tfrac{1}{2} h^T \nabla^2 f(x) h + o(||h||^3),$$

$$\nabla f(x+h) = \nabla f(x) + \nabla^2 f(x) h + \eta, \; ||\eta|| = o(||h||^2).$$

Definition 1.1.7. *Let A be any* $n \times n$ *symmetric matrix. The matrix A is said to be positive definite if all of its eigenvalues are strictly positive. This is denoted by* $A > 0$, $x^T A x > 0$, $\forall\ x \in \Re^n\ x \neq 0$.

Remark: If A^T = A, then the matrix A is said to be symmetric matrix.

Definition 1.1.8. *Two vectors d_i and d_j are said to be orthogonal provided their dot product is zero: d_i . d_j = 0; $\forall i \neq j$.*

Definition 1.1.9. *A functional f: $\Re^n \rightarrow \Re$ is said to be convex if for any x_1, $x_2 \in \Re^n$ and any $\lambda \in$ [0,1],*

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2).$$

Theorem 1.2. ***First order necessary condition***

If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*)$=0.

Proof. Suppose that ∇ f(x^*) $\neq$ 0. Define the vector d = ∇f(x^*) and note that $d^T \nabla f(x^*) < 0$. Because ∇f is continuous near x^*, there is a scalar T > 0 such that

$$d^T \nabla f(x^* + td) < 0, \quad \forall t \in [0, T].$$

For any $\bar{t} \in$ (0,T], we have by Taylors theorem that

$$f(x^* + \bar{t}d) = f(x^*) + \bar{t} d^T \nabla f(x^* + td), \qquad for\ some\ \ t \in (0, \bar{t}).$$

Therefore, f($x^* + \bar{t}$d) < f(x^*) for all $\bar{t} \in$ (0,T]. We have found a direction leading away from x^* along which f decreases, so x^* is not a local minimizer, and we have a contradiction. □

The local minimizers are not the only points with ∇f(x) = 0. Such points have a special name we call it stationary point.

Definition 1.1.10. ***Stationary point***

If $\nabla f(x) = 0$, then x is said to be a stationary point for f.

For the next result, let us recall that a matrix A is positive definite if d^TAd > 0, $\forall$ d $\neq$ 0, and positive semi-definite if d^TAd $\geq$ 0, $\forall$ d.

Theorem 1.3. ***Second order necessary conditions***.

Suppose f: $\Re^n \rightarrow \Re$ is twice continuously differentiable. If x^* is a local minimizer of f on $\Re^n$, then $\nabla f(x^*) = 0$ and its Hessian matrix at x^*, $\nabla^2 f(x^*)$ is positive semi-denite.

Proof. We know from Theorem 1.2 that $\nabla f(x^*) = 0$. For contradiction, assume that $\nabla^2 f(x^*)$ is not positive semi-denite. Then we can choose a vector d such that $d^T \nabla^2 f(x^*) d < 0$, and because $\nabla^2 f$ is continuous near x^*, there is a scalar $T > 0$ such that $d^T \nabla^2 f(x^* + td)d < 0$, $\forall\ t \in [0,T]$.
By doing a Taylor series expansion around x^*, we have $\forall\ \bar{t} \in (0,T]$ and some $t \in (0,\bar{t})$ that

$$f(x^* + \bar{t}d) = f(x^*) + \bar{t}d^T \nabla f(x^*) + \frac{1}{2}(\bar{t})^2 d^T \nabla^2 f(x^* + td)d < f(x^*).$$

As in Theorem 1.2, we have found a direction from x^* along which f is decreasing, and so, x^* is not a local minimizer.

□

We now describe sufficient conditions, which are conditions on the derivatives of f at the point x that guarantee x^* is a local minimizer.

Theorem 1.4. ***Second order sufficient condition***
If $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, then x^* is a local minimizer of f.

Proof. Because the Hessian is continuous and positive definite at x^*, we can choose a radius $r > 0$ so that $\nabla^2 f(x)$ remains positive denite for all x in the open ball

$$D = \{x \mid ||x - x^*|| < r\}.$$

Taking any nonzero vector d with $||d|| < r$, we have $x^* + d \in D$ and so

$$\begin{aligned} f(x^* + d) &= f(x^*) + d^T \nabla f(x^*) + \frac{1}{2} d^T \nabla^2 f(x) d \\ &= f(x^*) + \frac{1}{2} d^T \nabla^2 f(x) d, \end{aligned}$$

where $x = x^* + td$ for some $t \in (0,1)$. Since $x \in D$, we have $d^T \nabla^2 f(x)d > 0$ and therefore $f(x^* + td) > f(x^*)$, giving the result.

□

1.2 Convergence and rate of convergence

Other important issues for iterative methods are **convergence** and **rate (speed)** of convergence.

I. **Convergence**: Ensure that the sequence of iterates $\{x_k\}$ converges to a desired solution x^* (x^* is optimal solution, atleast locally).

- Some conditions may be needed to guarantee convergence.

- There are various ways to check convergence.

The following Theorem is used frequently to check convergence of iterative procedures.

Theorem 1.5. *If there is some $r \in$ (0,1) and some $k_0 \geq 0$ such that*

$$||x_{k+1} - x^*|| \leq r||x_k - x^*|| \qquad \forall k \geq k_0,$$

then $x_k \to x^*$.

- An iterative method is said to be:

1. **Local convergent**: if the sequence of iterates generated by the method converges to a solution when the initial approximation is already close enough to the solution.
2. **Global convergent**: if the sequence of iterates generated by the method i.e, for any initial approximation.

II. **Rate of convergence**: Measures how fast the sequence does converge. Common rates : linear, super-linear and quadratic.

Let $\{x_k\}$ be a sequence in $\Re^n$ that convergence to x^*. The rate of convergence is said to be

a. **linear** if there is a constant r $\in$ (0,1) and $k_0 \geq 0$ such that

$$\frac{||x_{k+1} - x^*||}{||x_k - x^*||} \leq r, \qquad \forall k \geq k_0$$

b. **super-linear** if

$$\lim_{k \longrightarrow \infty} \frac{||x_{k+1} - x^*||}{||x_k - x^*||} = 0$$

c. **quadratic** if there is a constant M > 0 and $k_0 \geq 0$ such that

$$\frac{||x_{k+1} - x^*||}{||x_k - x^*||^2} \leq M, \qquad \forall k \geq k_0$$

Note: Quadratic rate is the fastest; and super-linear is faster than linear.

1.3 Descent methods

In general, only exceptional cases allow the explicit calculation of (local) solutions of the minimization problem (1)

In practice, iterative methods are applied for computing approximate (local) minimizers. After a convergence analysis, these methods are normally represented in algorithmic form and implemented on a computer. For this reason, we now consider descent methods for finding solutions of problem (1), in which f: $\Re^n \to \Re$ is a continuously differentiable function. The fundamental idea of this method is

i. At a point x $\in \Re^n$, one chooses a direction $d_k \in \Re^n$ in which the function value decreases **(descent method)**.

ii. Starting at x, one proceeds along this direction d_k as long as the function value of f reduces sufficiently **(step size strategy)**.

These steps will be formalized.

Definition 1.3.1. *Consider a function f: $\Re^n \to \Re$, twice continuously differentiable and a point x $\in$ dom(f). A nonzero vector d $\in \Re^n$ is called a* ***descent direction*** *of f at x if there exists $\delta > 0$ such that*

$$f(x + \alpha d) < f(x), \qquad \forall \alpha \in (0, \delta).$$

If a descent direction d is found at iterate x, then

$$x_{k+1} = x + \alpha d, \qquad for \;\; some \;\; suitable \;\; \alpha \geq 0.$$

If so, d is called **step direction** and α is called **step length** at x.
The following Theorem helps us to check whether a vector d is a descent direction of f at a given point x.

1. **Properties of descent direction**

   The following theorem helps us to check whether a vector d_k is a descent direction of f at a given point x_k.

   Theorem 1.6. *Let f: $\Re^n \to \Re$ be a differentiable function at $x_0 \in \Re^n$ and d be a nonzero vector in $\Re^n$.*

a. If $\nabla f(x)^T \mathrm{d} < 0$, then d is a descent direction of f at x_0
   (The angle between d and -∇f(x) must be acute angle).

b. If $\nabla \mathrm{f}(x_0) \neq 0$, then d = -$\nabla \mathrm{f}(x_0)$ is a descent direction of f at x_0
   (which is the steepest descent direction of f at x_0).

c. If x_0 is a local (global) minimizer of f, then there is no descent direction of f at x_0.

1.4 Line search method

Line search, also called one-dimensional search, refers to an optimization procedure for univariable functions. It is the base of multivariable optimization. As stated before, in multivariable optimization algorithms, for given x_k, the iterative scheme is

$$x_{k+1} = x_k + \alpha_k d_k. \tag{1.1}$$

Most line search algorithms require d_k to a descent direction one for which $d_k^T \nabla f_k < 0$ because this property guarantees that the function f can be reduced along this direction. Moreover, the search direction often has the form

$$d_k = -B_k^{-1} \nabla f_k, \tag{1.2}$$

Where B_k is a symmetric and nonsingular matrix. In steepest descent method, B_k is simply the identity matrix I, while in Newton's method, B_k is the exact Hessian $\nabla^2 \mathrm{f}(x_k)$. When d_k is defined by (1.2) and B_k is positive definite, we have

$$d_k^T \nabla f_k = \text{-} \nabla f_k^T B_k^{-1} \nabla f_k < 0,$$

and therefore d_k is a descent direction. We would like to choose α_k to give a substantial reduction of f, but at the same time we do not want to spend much time making the choice. The ideal choice would be the global minimizer of the univariate function $\phi(.)$ defined by

$$\phi(\alpha) = f(x_k + \alpha d_k), \quad \alpha > 0,$$

Hence, the problem that departs from x_k and finds a step size in the direction d_k such that

$$\phi(\alpha_k) < \phi(0)$$

is just line search about α. If we find α_k such that the objective function in the direction d_k is minimized, i.e,

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

or

$$\phi(\alpha_k) = \min_{\alpha>0} \phi(\alpha),$$

such a line search is called **exact line search** or **optimal line search**, and α_k is called **optimal step size**. If we choose α_k such that the objective function has acceptable descent amount, i.e., such that the descent f(x_k) - f(x_k + $\alpha_k d_k$) > 0 is acceptable by users, such a line search is called **inexact line search**, or **approximate line search**, or **acceptable line search**.
In practical computation exact optimal step size cannot generally be found, and it is expensive to find almost exact step size; therefore, the inexact line search with less computation load is highly popular. The framework of line search is as follows. First, determine or give an initial search interval which contains the minimizer; then employ some section techniques or interpolations to reduce the interval iteratively until the length of the interval is less than some given tolerance.

However, commonly the exact line search is expensive. Especially, when an iterate is far from the solution of the problem, it is not effective to solve exactly a one-dimension subproblem. Furthermore, in practice, for many optimization methods, for example Newton method and quasi-Newton method, their convergence rate does not depend on the exact line search.
Therefore, as long as there is an acceptable steplength rule which ensures that the objective function has sufficient descent, the exact line search can be avoided, and the computing efforts will be decreased greatly. The following are desired conditions that α_k is required to satisfy:

Wolfe Conditions:

1. f(x_k + $\alpha_k d_k$) $\leq$ f(x_k) + $c_1 \alpha_k \nabla f_k^T d_k$ for some $0 < c_1 < 1$.

2. $\nabla f(x_k + \alpha_k d_k)^T d_k \geq c_2 \nabla f_k^T d_k$ for some $c_1 < c_2 < 1$.

Remark: If α_k is optimal (exact) solution, then it satisfies Wolfe conditions.

Theorem 1.7. *Suppose that f is smooth, bounded below on $\Re^n$ and $\nabla f(x)$ is Lipschitz continuous. If the iteration x_k, k = 1,2,... is generated by a scheme,*

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

where d_k is a descent direction at x_k for each k, and α_k satisfies the Wolfe conditions, then

$$\sum_{k=1}^{\infty} \cos^2 \theta_k ||\nabla f(x_k)||^2 < \infty$$

where θ_k is the angle between -$\nabla f(x_k)$ and d_k at each k.

1.5 Steepest descent method

The steepest descent method is one of the simplest and the most fundamental minimization methods for unconstrained optimization. Since it uses the negative gradient as its descent direction, it is also called the gradient method. Suppose that f(x) is continuously differentiable near x_k, and the gradient

$$g_k = \mathrm{g}(x_k) = \nabla \mathrm{f}(x_k) \neq 0.$$

From the Taylor expansion

$$\mathrm{f(x)} = \mathrm{f}(x_k) + (x - x_k)^T g_k + \mathrm{O}(||x - x_k||),$$

we know that, if we write $x - x_k = \alpha d_k$, then the direction d_k satisfying $d_k^T g_k < 0$ is called a descent direction that is such that f(x) < f(x_k). Fixing α, it follows that the smaller the value $d_k^T g_k$ is (i.e, the larger the value $|d_k^T g^k|$), the faster the function value decreases. By the Cauchy-Schwartz inequality

$$|d_k^T g_k| \leq ||d_k|| ||g_k||,$$

we have that the value $d_k^T g_k$ is the smallest if and only if d_k = -g_k. Therefore, the steepest descent direction is given by

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

The iterative scheme of the steepest descent method is

$$x_{k+1} = x_k + \alpha_k d_k, \quad k = 0, 1, ..., n$$

where α_k is chosen

$$\alpha_k = \arg\min_\alpha \, f(x_k + \alpha d_k).$$

The steepest descent method is of importance in the area of optimization from the theoretical point of view. The importance of its convergence theory is not only in itself but also in other optimization methods.

Algorithm 1.1: Steepest descent method

1. Input $x_1 \in \Re^n$ and initialize the tolerance $\epsilon > 0$.
   Set k = 1.

2. While $||g_k|| > \epsilon$;

3. Let d_k = -g_k,

4. Find α_k that minimizes $\phi(\alpha)$ = f(x_k+αd_k),$\alpha \geq 0$.

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

   end(while)

   Output x^* = x_{k+1} and f(x^*) = $_{k+1}$,

   Otherwise, set k = k + 1 and repeat from Step 2.

Properties of steepest descent method

1. It is convergent (if the function has a minima) starting from any x_0.

2. However, the convergence is slow (linear rate) due to its zig-zagging nature : $\nabla f(x_{k+1}) \perp d_k$ at every consecutive iterate points.

3. Thus, it is rather faster with inexact line search (α_k satisfying Wolfe Conditions).

1.6 Newton's method

The basic idea of Newton′s method for unconstrained optimization is to iteratively use the quadratic approximation q_k to the objective function f at the current iterate x_k and to minimize the approximation q_k.

Let f: $\Re^n \to \Re$ be twice continuously differentiable, $x_k \in \Re^n$, and the Hessian $\nabla^2 f(x_k)$ positive definite. We model f at the current point x_k by the quadratic approximation q_k,

$$f(x_k + d) \approx q_k(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2 f(x_k) d, \tag{1.3}$$

where d = x - x_k. Minimizing q_k(d) yields

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k) \tag{1.4}$$

which is Newton′s formula. Set

$$B_k = \nabla^2 f(x_k), \quad g_k = \nabla f(x_k). \tag{1.5}$$

Then we write (1.4) as

$$x_{k+1} = x_k - B_k^{-1} g_k. \tag{1.6}$$

Where d_k = x_{k+1} - x_k = -$B_k^{-1} g_k$ is a Newton′s direction. Clearly, the Newton′s direction is a descent direction because it satisfies $g_k^T d_k$ = -$g_k^T B_k^{-1} g_k < 0$ if B_k is positive definite. The first and the second derivatives of f will be denoted by

$$g(x) = \nabla f(x), \quad B(x) = \nabla^2 f(x) \tag{1.7}$$

for convenience if they exist.

The corresponding algorithm is stated as follows.

Algorithm 1.2: Newton′s method

1. Choose a suitable initial guess $x_1 \in \Re^n$ k = 1;

2. If $||g_k|| \leq \epsilon$, stop ($x^* = x_k$ is an approximate solution)
   else

3. Solve $B_k d_k$ = -g_k;

4. Set $x_{k+1} = x_k + d_k$;

5. k := k + 1, and repeat from Step 2
   end(if)

Properties of Newton's method

1. If f is convex, then Newton's method is very fast (quadratic) and can find an approximate of x^* starting from any initial point.

2. If f is not convex, then the method can be successful only when the initial point is close to x^*.
   i.e, Newton's method has no global convergence property.

Definition 1.6.1. *A non-zero vector d satisfying $d^T B(x)d < 0$, where B(x) has a negative eigenvalue is called direction of negative curvature.*

Definition 1.6.2. *Let d$\in \Re^n$, A be symmetric real matrix, then the Krylove subspace generated by A and d is given by*

$$\kappa(A, d, k) = span\{d, Ad, A^2d, ..., A^{k-1}d\}.$$

Chapter 2

Conjugate-direction method

In the preceding chapter, we have discussed the steepest descent method and the Newton′s method. In this chapter, we introduce the conjugate gradient method which is one between the steepest descent method and the Newton′s method. The conjugate gradient method deflects the direction of the steepest descent method by adding to it a positive multiple of the direction used in the last step. This method only requires the first order derivatives but overcomes the steepest descent methods shortcoming of slow convergence. At the same time, the method does not need to save and compute the second order derivatives which are needed by Newton′s method. In particular, since it does not require the Hessian matrix or its approximation, it is widely used to solve large scale optimization problems.

In this chapter, we will discuss the properties, the algorithm and numerical experiments, and the convergence of the conjugate gradient method. To begin with, we first introduce the concept of conjugate direction method.

2.1 Conjugate direction methods

One of the main properties of the conjugate gradient method is that its directions are conjugate. Now, we first introduce conjugate directions and conjugate direction methods.
The class of conjugate direction methods can be viewed as being intermediate between the method of steepest descent and Newton's method. The conjugate direction methods have the following properties:

1. Solve quadratics of n variables in n steps;

2. The usual implementation, the conjugate gradient algorithm, requires no Hessian matrix evaluations;

3. No matrix inversion and no storage of an n x n matrix is required.

The conjugate direction methods typically perform better than the method of steepest descent, but not as well as Newton's method. As we saw from the method of steepest descent and Newton's method, the crucial factor in the efficiency of an iterative search method is the direction of search at each iteration. For a quadratic function of n variables
f(x) = $\frac{1}{2}x^T$Ax - x^Tb, x $\in \Re^n$, A = $A^T > 0$ (A is symmetric matrix), the best direction of search, as we shall see, is in the so-called conjugate direction. Basically, two directions d_1 and d_2 in $\Re^n$ are said to be **conjugate** if d_1^TA$d_2 = 0$. In general, we have the following definition.

Definition 2.1.1. *Let A be a real symmetric $n \times n$ matrix. The directions d_0, d_1, d_2 , . . . ,d_m are **conjugate** if, $\forall\ i \neq j$, we have $d_i^T Ad_j = 0$.*

If A = I_n, where I_n is the n x n identity matrix, then d_i^TA$d_j = 0\ \forall$ i $\neq$ j can be expressed as

$$d_i^T Ad_j = d_i^T I_n d_j = d_i^T d_j = 0, \quad for\ i \neq j.$$

This is the well known condition for orthogonality between vectors d_i and d_j.
If d_j for j = 0, 1, . . . , k are eigenvectors of A then

$$Ad_j = \lambda_j d_j,$$

where the λ_j are the eigenvalues of A. Hence, we have

$$d_i^T Ad_j = \lambda_j d_i^T d_j = 0, \quad for\ i \neq j$$

since d_i and d_j for i $\neq$ j are orthogonal. In effect, the set of eigenvectors d_j should belong to distinct eigenvalues with respect to A.

Theorem 2.1. ***Linear independence of conjugate directions*** *If nonzero vectors $d_0, d_1, ..., d_k$ form a conjugate set with respect to a positive definite matrix A, then they are linearly independent.*

Proof. Consider the system

$$\sum_{j=0}^{k} \alpha_j d_j = 0$$

On premultiplying by $d_i^T A$, where $0 \leq i \leq k$, and then using definition 2.1.1, we obtain

$$\sum_{j=0}^{k} \alpha_j d_i^T A d_j = \alpha_i d_i^T A d_i = 0.$$

Since A is positive definite, we have $d_i^T A d_i > 0$. Therefore, the above system has a solution if and only if $_j = 0$ for j = 0, 1, . . . , k, that is, vectors d_i are linearly independent. □

Now we present the conjugate direction for minimizing the quadratic function of n variables:

$$\min \quad f(x) = \frac{1}{2} x^T A x + b^T x + c, \tag{2.1}$$

where A is symmetric positive definite. The unique solution to this problem is also the unique solution to the equation

$$Ax + b = 0. \tag{2.2}$$

Suppose that d_0, . . . , d_{n-1} are n non-zero orthogonal vectors. Therefore they form an orthogonal basis for $\Re^n$.
Let x^* be the unique solution to (2.1) or (2.2). We can write

$$x^* = x_0 + \alpha_0 d_0 + ... + \alpha_{n-1} d_{n-1}, \tag{2.3}$$

for some real numbers α_0, . . . , α_{n-1}. Plugging the above into (2.2) yields

$$A(x^* - x_0) + b = A(\alpha_0 d_0 + ... + \alpha_{n-1} d_{n-1}) + b = 0$$

and

$$d_i^T A(\alpha_0 d_0 + ... + \alpha_{n-1} d_{n-1}) + d_i^T b = 0.$$

Due to the orthogonality of the $d_i s$, we can solve for these coefficients

$$\alpha_i = -\frac{d_i^T A x^* - A x_0}{d_i^T A d_i} = -\frac{d_i^T b - A x_0}{d_i^T A d_i}. \tag{2.4}$$

Thus we obtain the explicit formula

$$x^* = -\sum_{i=0}^{n-1} \frac{d_i^T b}{d_i^T A d_i} d_i. \tag{2.5}$$

There are two basic ideas embedded in (2.5). The first is the idea of selecting an orthogonal set of $d_i's$ by taking an appropriate scalar product, all terms on the right side of (2.3), except the i^{th}, vanish. This could, of course, have been accomplished by making the d_is orthogonal in the ordinary sense instead of making them orthogonal. The second basic observation, however, is that by using orthogonality, the resulting equation for α_i can be expressed in terms of the known vector b rather than the unknown vector x^*; hence, the coefficients can be evaluated without knowing x^*.

The expansion for x^* can be considered to be the result of an iterative process of n steps where at the i^{th} step $\alpha_i d_i$ is added. Viewing the procedure this way and allowing for an arbitrary initial point for the iteration, we obtain the basic conjugate direction method.

Let A be a symmetric and positive definite matrix, then we define ξ_k as the subspace of $\Re^n$ spanned by a set of orthogonal vectors $\{d_i\}_{i=0}^{k-1}$, or for short,

$$\xi_k = \text{span}\{d_0, d_1, ..., d_{k-1}\}.$$

Theorem 2.2. ***Expanding subspace***

Let d_0 , d_1,...,d_{n-1} be a set of nonzero orthogonal vectors in $\Re^n$ for any $x_0 \in \Re^n$, consider the sequence x_k generated by the rule

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

where, writing $g_k = Ax_k + b$,

$$\alpha_k = -\frac{g_k^T d_k}{d_k^T A d_k},$$

and

$$g_k = Ax_k + b.$$

The following statements hold:

i. the sequence x_k converges to the unique solution x^* of $Ax + b = 0$ after n steps. In other words, $x_n = x^*$ minimizes the function $f(x) = \frac{1}{2}x^T Ax + b^T x$.

ii. x_{k+1} minimizes the same function f(x) on the line $x = x_k + \alpha d_k$, $-\infty < \alpha < \infty$ as well as on the linear variety $x_0 + \xi_{k+1}$.

Proof. To prove (i), we make use of the linear independence of the $d_j's$. Notice that

$$x^* \text{ - } x_0 = \alpha_0 d_0 + ... + \alpha_{n-1} d_{n-1}$$

for some $\alpha_0, \ldots, \alpha_{n-1}$. We multiply both sides of the equation by A and take the inner product with d_k, yielding

$$\alpha_k = \frac{d_k^T A(x^* - x_o)}{d_k^T A d_k}. \tag{2.6}$$

Now we use induction to show that α_k defined in (2.6) equals

$$\alpha_k = -\frac{g_k^T d_k}{d_k^T A d_k}.$$

Suppose this is true for $\alpha_0, \ldots, \alpha_{k-1}$. Following the iterative steps from x_0 up to x_k, we have

$$x_k \text{ - } x_0 = \alpha_0 d_0 + ... + \alpha_{k-1} d_{k-1}.$$

By the orthogonality of the $d_j's$, it follows that

$$d_k^T \mathrm{A}(x_k \text{ - } x_0) = 0$$

Substituting the above into (2.6) we obtain that

$$\begin{aligned}
\alpha_k &= \frac{d_k^T A(x^* - x_k + x_k - x_0)}{d_k^T A d_k} \\
&= \frac{d_k^T A(x^* - x_k)}{d_k^T A d_k} \\
&= \frac{d_k^T (Ax^* - Ax_k)}{d_k^T A d_k} \\
&= \frac{d_k^T (-b - Ax_k)}{d_k^T A d_k} \\
&= -\frac{g_k^T d_k}{d_k^T A d_k}
\end{aligned}$$

$$\alpha_k = \frac{g_k^T g_k}{d_k^T A d_k}$$

since $d_k = \text{-}g_k$.

To prove (ii), we show that x_{k+1} minimizes f over the linear variety $x_0 + \xi_{k+1}$, which contains the line x = $x_k + \alpha d_k$. Since the quadratic function f is strictly convex, a local minimum is also a global one. Thus, the conclusion will hold if it can be shown that the gradient g_{k+1} is orthogonal to ξ_{k+1}, that is, if the gradient is orthogonal to d_0, d_1, . . . , d_k. We prove this by induction. The hypothesis is true for k = 0 since ξ_0 is empty. Assume that $g_k \perp \xi_k$. We have

$$\begin{aligned} g_{k+1} &= Ax_{k+1} + b \\ &= A(x_k + \alpha_k d_k) + b \\ &= Ax_k + b + \alpha_k A d_k \\ &= g_k + \alpha_k A d_k. \end{aligned}$$

This implies

$$g_{k+1} = g_k + \alpha_k A d_k, \tag{2.7}$$

and hence by definition of α_k

$$\begin{aligned} d_k^T g_{k+1} &= d_k^T g_k + \alpha_k d_k^T A d_k \\ &= d_k^T g_k - \frac{g_k^T d_k}{d_k^T A d_k} d_k^T A d_k \\ &= d_k^T g_k - g_k^T d_k \\ &= 0. \end{aligned}$$

It also holds that

$$d_i^T g_{k+1} = d_i^T g_k + \alpha_k d_i^T A d_k, \qquad for \;\; i < k.$$

The first term on the right hand side of the above equation vanishes due to the induction hypothesis while the second term vanishes by the orthogonality of the d_is.
Thus, $g_{k+1} \perp \xi_{k+1}$. □

Corollary 2.1. *In the method of conjugate directions the gradients g_k, k = 0, 1, . . . , n, satisfy*

$$g_k^T \; d_i \; = \; 0, \qquad for \quad i < k.$$

This corollary tells us that the conjugate gradient algorithm really is a generalization of steepest descent. Each step of adding $\alpha_k d_k$ to the previous estimate is the same as doing a line minimization along the direction of d_k. Furthermore, the offset $\alpha_k d_k$ does not undo

previous progress, that is, the minimization is in fact a minimization over $x_0 + \xi_{k+1}$.

Thus, the $\xi_k's$ form a sequence of subspace with $\xi_k \subseteq \xi_{k+1}$. Because x_k minimizes f over $x_0 + \xi_k$, it is clear that x_n must be the overall minimum of f.

2.2 Conjugate gradient method

The conjugate gradient method is the conjugate direction method that is obtained by selecting the successive direction vectors as a conjugate version of the successive gradients obtained as the method progresses. Thus, the directions are not specified beforehand, but rather are determined sequentially at each step of the iteration. At step α_k one evaluates the current negative gradient vector and adds to it a linear combination of the previous direction vectors to obtain a new conjugate direction vector along which to move.

There are three primary advantages to this method of direction selection. First, unless the solution is attained in less than n steps, the gradient is always nonzero and linearly independent of all previous direction vectors. Indeed, the gradient g_k is orthogonal to the subspace ξ_k generated by d_0, d_1, ... ,d_{k-1}. If the solution is reached before n steps are taken, the gradient vanishes and the process terminates it, being unnecessary, in this case, to find additional directions.

Second, a more important advantage of the conjugate gradient method is the especially simple formula that is used to determine the new direction vector. This simplicity makes the method only slightly more complicated than steepest descent.

Third, because the directions are based on the gradients, the process makes good uniform progress toward the solution at every step. This is in contrast to the situation for arbitrary sequences of conjugate directions in which progress may be slight until the final few steps. Although for the pure quadratic problem uniform progress is of no great importance, it is important for generalizations to nonquadratic problems.

In this method, directions are generated sequentially, one per iteration. For iteration k+1, a new point x_{k+1} is generated by using the previous direction d_k. Then, a new direction d_{k+1} is generated by adding a vector $\beta_k d_k$ to -g_{k+1}, the negative of the gradient at the new point.

The conjugate gradient method is based on the following theorem. This is essentially the same as Theorem 2.2 except that the method of generating conjugate directions is now defined.

Theorem 2.3. ***Convergence of conjugate gradient method***

a. If A is a positive definite matrix, then for any initial point x_0 and an initial direction

$$d_0 = -g_0 = -(b + Ax_0),$$

the sequence generated by the recursive relation

$$x_{k+1} = x_k + \alpha_k d_k \tag{2.8}$$

where

$$\alpha_k = \frac{g_k^T g_k}{d_k^T A d_k} \tag{2.9}$$

$$g_k = Ax_k + b \tag{2.10}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \tag{2.11}$$

$$\beta_k = \frac{g_{k+1}^T A d_k}{g_k^T A d_k} \tag{2.12}$$

converges to the unique solution x^* of the problem given in Eq. (2.2).

b. The gradient g_k is orthogonal to $\{g_0, g_1, ..., g_{k-1}\}$, i.e,

$$g_k^T g_i = 0, \qquad for \quad 0 \leq i < k.$$

Proof. (a). The proof of convergence is the same as in Theorem 2.2. What remains to be proved is that directions d_0,d_1, . . . , d_{n-1} form a conjugate set, that is,

$$d_k^T \mathrm{A} d_i = 0, \qquad \text{for} \quad 0 \leq \mathrm{i} < \mathrm{k} \quad \text{and} \quad 1 \leq \mathrm{k} \leq \mathrm{n}.$$

The proof is by induction. We assume that

$$d_k^T A d_i = 0, \qquad for \;\; 0 \leq i < k \tag{2.13}$$

and show that

$$d_{k+1}^T A d_i = 0, \qquad for \;\; 0 \leq i < k+1. \tag{2.14}$$

Let S(v_0, v_1, . . . , v_k) be the subspace spanned by vectors v_0, v_1, . . . ,v_k from Eq. (2.7),i.e, $g_{k+1} = g_k + \alpha_k \mathrm{A} d_k$
and hence for k = 0, we have

$$g_1 = g_0 + \alpha_0 \mathrm{A}d_0 = g_0 \text{ - } \alpha_0 \mathrm{A}g_0$$

since $d_0 = \text{-}g_0$. In addition, Eq. (2.11) yields

$$d_1 = \text{-}g_1 + \beta_0 d_0 = \text{-}(1 + \beta_0)g_0 + \alpha_0 \mathrm{A}g_0;$$

that is, g_1 and d_1 are linear combinations of g_0 and $\mathrm{A}g_0$, and so

$$\mathrm{S}(g_0, g_1) = \mathrm{S}(d_0, d_1) = \mathrm{S}(g_0, \mathrm{A}g_0).$$

Similarly, for k = 2, we get

$$\begin{aligned} g_2 &= g_0 - [\alpha_0 + \alpha_1(1 + \beta_0)]Ag_0 + \alpha_0\alpha_1 A^2 g_0 \\ d_2 &= -[1 + (1 + \beta_0)\beta_1]g_0 + [\alpha_0 + \alpha_1(1 + \beta_0) + \alpha_0\beta_1]Ag_0 - \alpha_0\alpha_1 A^2 g_0, \end{aligned}$$

and hence

$$\begin{aligned} S(g_0, g_1, g_2) &= S(g_0, Ag_0, A^2 g_0) \\ S(d_0, d_1, d_2) &= S(g_0, Ag_0, A^2 g_0). \end{aligned}$$

By continuing the induction, we can show that

$$S(g_0, g_1, ..., g_k) = S(g_0, Ag_0, ..., A^k g_0) \tag{2.15}$$

$$S(d_0, d_1, ..., d_k) = S(g_0, Ag_0, ..., A^k g_0). \tag{2.16}$$

Now from Eq. (2.11)

$$d_{k+1}^T A d_i = -g_{k+1}^T d_i + \beta_k d_k^T d_i. \tag{2.17}$$

For i = k, the use of Eq. (2.11) gives

$$d_{k+1}^T A d_k = -g_{k+1}^T d_k + \beta_k d_k^T d_k = 0. \tag{2.18}$$

For i < k, Eq. (2.16) shows that

$$\mathrm{A}d_i \in \mathrm{S}(d_0, d_1, \ldots, d_k),$$

and thus Ad_i can be represented by the linear combination

$$Ad_i = \sum_{i=0}^{k} a_i d_i \tag{2.19}$$

where a_i for i = 0, 1, . . . , k are constants. Now from Eqs. (2.17) and (2.19)

$$d_{k+1}^T A d_i = -\sum_{i=0}^{k} a_i g_{k+1}^T d_i + \beta_k d_k^T A d_i = 0, \qquad for \;\; i < k$$

$$d_{k+1}^T A d_i = 0, \qquad for \;\; i < k. \tag{2.20}$$

The first term is zero from the orthogonality property whereas the second term is zero from the assumption in Eq. (2.13). By combining Eqs. (2.18) and (2.20), we have

$$d_{k+1}^T A d_i = 0, \qquad for \;\; 0 \leq i < k+1. \tag{2.21}$$

For k = 0, Eq. (2.21) gives

$$d_1^T \mathrm{A} d_i = 0, \qquad \text{for} \quad 0 \leq \mathrm{i} < 1,$$

and therefore, from Eqs. (2.13) and (2.21), we have

$$\begin{aligned} d_2^T \mathrm{A} d_i &= 0, \qquad \text{for} \quad 0 \leq \mathrm{i} < 2 \\ d_3^T \mathrm{A} d_i &= 0, \qquad \text{for} \quad 0 \leq \mathrm{i} < 3 \\ &\quad . \qquad\qquad . \\ &\quad . \qquad\qquad . \\ &\quad . \qquad\qquad . \\ d_k^T \mathrm{A} d_i &= 0, \qquad \text{for} \quad 0 \leq \mathrm{i} < \mathrm{k}. \end{aligned}$$

(b). From Eqs. (2.15) - (2.16), g_0, g_1, . . . , g_k span the same subspace as d_0, d_1, . . . , d_k, and consequently, they are linearly independent. We can write

$$g_i = \sum_{j=0}^{i} a_j d_j$$

where a_j for j = 0, 1, . . . , i are constants. Therefore, from Theorem 2.2

$$g_k^T g_i = \sum_{j=0}^{i} a_j g_k^T d_j, \qquad for \;\; 0 \leq i < k.$$

□

The expressions for α_k and β_k in the above theorem can be simplified somewhat. From Eq. (2.11)

$$\text{-}g_k^T d_k = g_k^T g_k \text{ - } \beta_{k-1} g_k^T d_{k-1}$$

where

$$g_k^T {}_{k1} = 0$$

according to Theorem 2.2. Hence,

$$\text{-}g_k^T d_k = g_k^T g_k,$$

and, therefore, the expression for α_k in Eq. (2.9) is modified as

$$\alpha_k = \frac{g_k^T g_k}{d_k^T A d_k}. \tag{2.22}$$

On the other hand, from Eq. (2.7)

$$\mathrm{A}d_k = \tfrac{1}{\alpha_k}(g_{k+1} \text{ - } g_k),$$

and so

$$g_{k+1}^T A d_k = \frac{1}{\alpha_k}(g_{k+1}^T g_{k+1} - g_{k+1}^T g_k). \tag{2.23}$$

Now from Eqs. (2.15) and (2.16)

$$g_k \in \mathrm{S}(d_0, d_1, \ldots, d_k)$$

or

$$g_k = \textstyle\sum_{i=0}^{k} a_i d_i,$$

and as a result,

$$g_{k+1}^T g_k = \sum_{i=0}^{k} a_i g_{k+1}^T d_i = 0 \tag{2.24}$$

by virtue of Theorem 2.2. Therefore, Eqs. (2.12) and (2.22) (2.24) yield

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}.$$

The above principles and theorems lead to the following algorithm:

Algorithm 2.1: Conjugate gradient

step(1) *set* $k = 0$; select the initial point $x_0 \in \Re^n$, $\epsilon > 0$;

step(2) $r_0 = Ax_0 - b, \qquad p_0 = -r_0,$

while $\|r_k\| \geq \epsilon$

step(3) $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$;

step(4) $x_{k+1} = x_k + \alpha_k p_k$;

step(5) $r_{k+1} = r_k + \alpha_k A p_k$;

step(6) $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$;

step(7) $p_{k+1} = r_{k+1} + \beta_k p_k$;

step(8) *set* $k = k + 1$; go to step 3

end(%while)

Chapter 3

Trust region Newton's with conjugate gradient method

3.1 Trust-region method

Some line search methods such as Newton's methods, steepest descent methods and quasi-Newton's methods and trust-region methods generate steps with the help of a quadratic model of the objective function, but they use this model in different ways. Line search methods use to generate a search direction, and then focus their efforts on finding a suitable step length α along this direction. Trust-region methods define a region around the current iterate within which they trust the model to be an adequate representation of the objective function, and then choose the step to be the approximate minimizer of the model in this region. In trust-region methods choose the direction and length of the step simultaneously. If a step is not acceptable, then reduce the size of the region and find a new minimizer. In general, the direction of the step changes whenever the size of the trust-region is altered.

The size of the trust-region is critical to the effectiveness of each step. If the region is too small, the algorithm misses an opportunity to take a substantial step that will move it much closer to the minimizer of the objective function. If too large, the minimizer of the model may be far from the minimizer of the objective function in the region, so we may have to reduce the size of the region and try again. In practical algorithms, we choose the size of the region according to the performance of the algorithm during previous iterations. If the model is consistently reliable, producing good steps and accurately predicting the behavior of the objective function along these steps, the size of the trust-region may be increased to allow longer, more ambitious steps to be taken. A failed step is an indication that our model is an inadequate representation of the objective function over the current trust-region. After taking this step, we reduce the size of the region and try again.

The trust-region approach on a function f of two variables in which the current point x_k and the minimizer x^* lie at opposite ends of a curved valley the quadratic model function

m_k, whose elliptical contours are shown as dashed lines, is constructed from function and derivative information at x_k and possibly also on information accumulated from previous iterations and steps. A line search method based on this model searches along the step to the minimizer of m_k, but this direction will yield at most a small reduction in f even if the optimal steplength is used. The trust-region method steps to the minimizer of m_k within the dotted circle, yielding a more significant reduction in f and better progress toward the solution.

In this chapter, we will assume that the model function m_k that is used at each iterate x_k is quadratic. Moreover, m_k is based on the Taylor series expansion of f around x_k, which is

$$f(x_k + p) = f_k + g_k^T p + \frac{1}{2} p^T \nabla^2 f(x_k + p) p, \tag{3.1}$$

where f_k = f (x_k) and $g_k = \nabla$f $(x_k\ )$, and t is some scalar in the interval (0,1). By using an approximation B_k to the Hessian in the second-order term, m_k is defined as follows:

$$\begin{aligned} \min \quad m_k(p) &= f_k + g_k^T p + \frac{1}{2} p^T B_k p \\ s.t \quad ||p|| &\leq \Delta_k, \end{aligned} \tag{3.2}$$

where $\Delta_k > 0$ is the trust-region radius. In most of our discussion we define $||.||$ to be the Euclidean norm, so that the solution p_k^* of (3.2) is the minimizer of m_k in the ball of radiusΔ_k. Thus, the trust-region approach requires us to solve a sequence of subproblems (3.2) in which the objective function and constraint (which can be written as $p^T p \leq \Delta_k^2$) are both quadratic. When B_k is positive definite and $||B_k^{-1} g_k|| \leq \Delta_k$, the solution of (3.2) is easy to identify: it is simply the unconstrained minimum $p_k^B = -B_k^{-1} g_k$ of the quadratic $m_k(p)$. In this case, we call p_k^B the full step. The solution of (3.2) is not so obvious in other cases, but it can usually be found without too much computational expense. In any case, we need only an approximate solution to obtain convergence and good practical behavior.

1. **Actual reduction and predicted reduction**

One of the key ingredients in a trust-region algorithm is the strategy for choosing the trust-region radius Δ_k at each iteration. We base this choice on the agreement between the model function m_k(p) and the objective function f(x_k + p) at previous iterations. Given a step p_k, we define the ratio as

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}. \tag{3.3}$$

The numerator is called the actual reduction, and the denominator is the predicted reduction (that is, the reduction in f predicted by the model function). Note that since the step

p_k is obtained by minimizing the model m_k over a region that includes p = 0, the predicted reduction will always be nonnegative. Hence, if ρ_k is negative, the new objective value $f(x_k + p_k)$ is greater than the current value $f(x_k)$, so the step must be rejected. On the other hand, if ρ_k is close to 1, there is good agreement between the model m_k and the function f over this step, so it is safe to expand the trust-region for the next iteration. If ρ_k is positive but signicantly smaller than 1, we do not alter the trust-region; however, but if it is close to zero or negative, we shrink the trust-region by reducing Δ_k at the next iteration.

The following procedure summarizes the process.

2. **Trust-region Algorithm**

Before implementing the trust-region algorithm, we should first determine some parameters. $\hat{\Delta}$ is the upper bound for the size of the trust-region. η_1, η_2 and η_3, t_1, t_2 are the threshold values for evaluating the goodness of the quadratic model for determining the trust-region's size in the next iteration. A typical set for these values are

Algorithm 3.1: Trust-region

Given Given initial guess x_0, $\hat{\Delta} > 0$, $\Delta_0 \in (0, \hat{\Delta})$ and $\eta \in [0,1/4)$:
for k = 0,1,2,...,n

1. Evaluate p_k (by approximately) solving (3.2);
   Determine ρ_k from (3.3);

2. if $\rho_k < \frac{1}{4}$
   $\Delta_{k+1} = \frac{1}{4}\Delta_k$
   else

3. if $\rho_k > \frac{3}{4}$ and $||p_k|| = \Delta_k$
   $\Delta_{k+1} = \min(2\Delta_k, \hat{\Delta})$
   else
   $\Delta_{k+1} = \Delta_k$;
   end(if)

4. if $\rho_k > \eta$
   $x_{k+1} = x_k + p_k$
   else
   $x_{k+1} = x_k$;
   end(for).

3.2 Method of solving the trust-region subproblem

In order to make the trust-region Algorithm (3.1) practical, we have to establish a way of obtaining p_k from the trust-region subproblem(3.2).

Trust-region subproblem can be solved exactly or approximately. Newton-like method is an exact solution. This method is meant to be applied to very small dimensional problems due to its computational complexity. The exact solution p_k to the trust-region subproblem (3.2) satisfies

$$(B_k + \lambda I)p_k = -g, \qquad for \;\; some \;\; \lambda \geq 0.$$

Remark: The vector p_k is a global solution of the trust-region subproblem.

$$\min_{p \in \Re^n} \;\; m(p) = f_k + g_k^T p + \frac{1}{2} p^T B_k p, \;\; s.t. \;\; ||p|| \leq \Delta,$$

If the problem dimension is not too large, the choice

$$B_k = \nabla^2 \mathrm{f}(x_k)$$

is reasonable and leads to the 2^{nd} order Taylor model

$$m_k(p) = f_k + g_k^T p + \frac{1}{2} p^T B_k p. \tag{3.4}$$

Methods in which B_k is the actual Hessian of f each iterate x_k are called trust-region Newton methods. If and only if p_k is feasible and there is a scalar $\lambda \geq 0$ such that the following conditions are satisfied :

$$\begin{aligned} &(B_k + \lambda I)p_k = -g, \qquad for \;\; some \;\; \lambda \geq 0, \\ &\lambda(\Delta_k - ||p_k||) = 0, \\ &(B_k + \lambda I) \;\; is \;\; positive \;\; semi - definite. \end{aligned}$$

There are some strategies for finding approximate solutions p_k of the subproblem (3.2), which achieve at least as much reduction in m_k as the reduction achieved by the so-called Cauchy point. This point is simply the minimizer of m_k along the steepest descent direction -g_k. subject to the trust-region bound. The first approximate strategy is the dogleg method, which is appropriate when the model Hessian B_k is positive definite. The second strategy, known as two-dimensional subspace minimization, can be applied when B_k is indefinite,

though it requires an estimate of the most negative eigenvalue of this matrix. A third strategy, described in this chapter, uses an approach based on the conjugate gradient method to minimize m_k, and it can therefore be applied when B_k is large and sparse. However, this project mainly focuses on the third strategy.

Note that: If B_k is positive definite, the unconstrained minimizer of

$$m_k(\mathrm{p}) = f_k + \nabla f_k^T \mathrm{p} + \tfrac{1}{2} p^T B_k \mathrm{p}$$

is $p_k^B = -B_k^{-1}\nabla f_k$, called Newton point. Thus, if B_k is positive definite and $||p_k^B|| \leq \Delta_k$, then $p_k = p_k^B$ is exact solution. Nevertheless, p_k^B may not be in trust-region or even may not be available; i.e, B_k may not be positive definite. Therefore, the common approach to solve the trust-region subproblem is to first find the so-called Cauchy point defined below and then look for improvement on it.

3.2.1 Cauchy point calculation

In line search methods, we may find an improving direction from the gradient information; that is by taking the steepest descent direction with regard to the maximum range we could make. We can solve the trust-region subproblem in an inexpensive way. This method also expresses the improving step explicitly by the following closed-form equations.

Cauchy point, denoted by P_k^c, is the minimizer of $m_k(\mathrm{p})$ in trust-region in the direction of $-\nabla f_k$. So, if P_k^s is the point on the boundary of the trust-region in the direction of $-\nabla f_k$, then $P_k^c = \tau_k P_k^s$, for some $\tau_k \in (0,1]$.

Where $\tau_k \in (0,1]$ is the minimizer of $\phi(\tau) = m_k(\tau P_k^s)$ s.t $\tau \in (0;1]$,
then

$$\phi'(\tau) = -\Delta_k ||g_k|| + \tau (\frac{\Delta_k}{||g_k||})^2 g_k^T B_k g_k.$$

If $g_k^T B_k g_k \leq 0$, then $\phi'(\tau) < 0$ for all τ implies that $\tau_k = 1$. That is,

$$p_k^c = -\frac{\Delta_k}{||g_k||} g_k, \qquad when \quad g_k^T B_k g_k \ \leq \ 0.$$

If $g_k^T B_k g_k > 0$, then $\phi'(\tau) = 0$ when $\tau = \frac{||g_k||^3}{(\Delta_k g_k^T B_k g_k)}$.

To obtain τ_k explicitly, we consider the cases of $g_k^T B_k g_k \leq 0$ and $g_k^T B_k g_k > 0$ separately.

For the former case, the function $m_k(\tau p_k^S)$ decreases monotonically with τ whenever $g_k \neq 0$, so τ_k is simply the largest value that satisfies the trust-region bound, namely,

$\tau_k = 1$. For the second case, $g_k^T B_k g_k > 0$, $m_k(\tau p_k^s)$ is a convex quadratic in τ, so τ_k is either the unconstrained minimizer of this quadratic,

$$\frac{||g_k||^3}{(\Delta_k g_k^T B_k g_k)},$$

or the boundary value 1, whichever comes first. In summary, we have

$$p_k^s = \frac{-\Delta_k}{||g_k||} g_k$$

$$p_k^c = -\tau_k \frac{\Delta_k}{||g_k||} g_k, \qquad 0 < \tau_k \leq 1 \tag{3.5}$$

$$\tau_k = \begin{cases} 1, & if \quad g_k^T B_k g_k \leq 0 \\ \min(\frac{||g_k||^3}{(\Delta_k g_k^T B_k g_k)}, 1), & otherwise. \end{cases} \tag{3.6}$$

The Cauchy step p_k^c is inexpensive to calculate for no matrix factorizations are required, and it is of crucial importance in deciding if an approximate solution of the trust-region subproblem is acceptable. Specifically, a trust-region method will be globally convergent if its steps p_k give a reduction in the model m_k that is at least some fixed positive multiple of the decrease attained by the Cauchy step.

Limitations and further improvements

Since the Cauchy point p_k^c provides sufficient reduction in the model function m_k to yield global convergence, and since the cost of calculating it is so small because we always take the Cauchy point as our step, we are simply implementing the steepest descent method with a particular choice of step length.

The Cauchy point does not depend on the matrix B_k, which is used only in the calculation of the step length. Rapid convergence can be expected only if B_k plays a role in determining the direction of the step as well as its length and if B_k contains valid curvature information about the function.

A number of trust-region algorithms compute the Cauchy point and then try to improve on it. The improvement strategy is often designed so that the full step $p_k^B = \text{-}B_k^{-1} g_k$ is chosen whenever B_k is positive definite and $||p_k^B|| \leq \Delta_k$. When B_k is the exact Hessian $\nabla^2 \text{f}(x_k)$ or a quasi-Newton approximation, this strategy can be expected to yield super-linear convergence.

There are three methods for finding approximate solutions to (3.2) that have the features just described. Throughout this section we will be focusing on the third method: the internal

workings of a single iteration. In this section, we denote the solution of (3.2) by $p^*(\Delta)$, to emphasize the dependence on Δ.

3.2.2 Conjugate gradient Steihaug's method

The methods for solving the trust-region subproblem described above require the solution of a linear system. When the problem is large, the operation may be quite costly. Steihaug-Toint proposed a technique based on conjugate gradient method and trust-region method, which solves the trust-region subproblem approximately. This method is usually called Steihaug-conjugate gradient method, and it is also called the Steihaug-Toint method.

The most widely used method for solving a trust-region subproblem is by using the idea of conjugate gradient method for minimizing a quadratic function since conjugate gradient guarantees convergence within a finite number of iterations for a quadratic programming. Steihaug conjugate gradient has a super-linear convergence rate and inexpensiveness to compute (No expensive Hessian evaluation).

Now we present conjugate gradient Steihaug algorithm to trust-region subproblem (3.2) but first we have to find the value of τ. It can be determined by the following procedure and define
$p_k = x_j + \tau d_j$ and $||p_k|| = \Delta_k$

$$\begin{aligned}
\Delta_k^2 &= ||p_k||^2 \\
\Delta_k^2 &=< p_k, p_k > \\
\Delta_k^2 &=< x_j + \tau d_j, x_j + \tau d_j > \\
\Delta_k^2 &= ||x_j||^2 + \tau x_j^T d_j + \tau d_j^T x_j + \tau^2 ||d_j||^2 \\
0 &= ||x_j||^2 + 2\tau x_j^T d_j + ||x_j||^2 - \Delta_k^2 \\
\textit{let}\ \ a = ||d_j||^2;\ \ b = 2x_j^T d_j;\ \ c &= ||x_j||^2 - \Delta_k^2 \\
0 &= a\tau^2 + b\tau + c \\
\tau &= \frac{-b + \sqrt{b^2 - 4ac}}{2a}. \qquad (*)
\end{aligned}$$

Algorithm 3.2: Conjugate gradient Steihaug

Input f(x); an initial guess x_0; $\epsilon_k > 0$,
g_k=$\nabla f_k = \nabla$f(x_k); $B_k = \nabla^2$f(x_k)

1. set $g_0 = \nabla f_0$,d_0 = -g_0 = -∇f_0 ,

2. If $||g_0|| < \epsilon_k$
   return $p_k = x_0 = 0$;
   for j = 0, 1, 2, ...

3. If $d_j^T B_k d_j \leq 0$
   From (*) find τ such that $p_k = x_j + \tau d_j$ minimizes $m_k(p_k)$
   and satisfies $||p_k|| = \Delta_k$;
   Set $\alpha_j = g_j^T g_j / d_j^T B_k d_j$;
   Set $x_{j+1} = x_j + \alpha_j d_j$;
   return p_k ;

4. If $||x_{j+1}|| \geq \Delta_k$
   From (*) find $\tau \geq 0$ such that $p_k = x_j + \tau d_j$ satisfies $||p_k|| = \Delta_k$;
   Set $g_{j+1} = g_j + \alpha_j B_k d_j$;
   return p_k ;

5. If $||g_{j+1}|| < \epsilon_k$
   Set $\beta_{j+1} = g_{j+1}^T g_{j+1} / g_j^T g_j$;
   Set d_{j+1} = -$g_{j+1} + \beta_{j+1} d_j$;
   $x_{j+1} = x_j + \alpha_k d_j$;
   return $p_k = x_{j+1}$;
   end (for).
   Out put $x_{k+1} = x_k + p_k$

The first **if** statement inside the loop stops the method if its current search direction d_j is a direction of non-positive curvature along B_k; whereas, the second **if** statement inside the loop causes termination if x_{j+1} violates the trust-region bound. In both cases, the method returns the step p_k obtained by intersecting the current search direction with the trust-region boundary.

The choice of the tolerance ϵ_k at each call to Algorithm 3.2 is important in keeping the overall cost of the trust-region Newton conjugate gradient method low. Near a well behaved

solution x^*, the trust-region bound becomes inactive, and the method reduces to the inexact Newton method. Rapid convergence can be obtained in these circumstances by choosing ϵ_k.

The essential differences between Algorithm 2.1 and the inner loop of Algorithm 3.2 are that the latter terminates when it violates the trust-region bound $||p|| \leq \Delta$ and encounters a direction of negative curvature in B_k.The initialization of x_0 to zero in Algorithm 3.2 is a crucial feature of the algorithm because it needs an initial point. Provided $||g_k|| \geq \epsilon_k$, Algorithm 3.2 terminates at a point p_k for which $m_k(p_k) \leq m_k(p_k^c)$, that is when the reduction in model function equals or exceeds that of the Cauchy point. To demonstrate this fact, we consider several cases. First, if $d_0^T B_k d_0$ $g_k^T B_k g_k \leq 0$, then the condition in the first **if** statement is satisfied, and the algorithm returns the Cauchy point
$p_k^c = -\frac{\Delta_k}{||g_k||}\ g_k$. Otherwise, Algorithm 3.2 defines x_1 as follows:

$$x_1 = \alpha_0 d_0 = \frac{g_0^T g_0}{d_0^T B_k} d_0 = -\frac{g_k^T g_k}{g_k^T B_k g_k} g_k.$$

If $||x_1|| < \Delta_k$, then x_1 is exactly the Cauchy point. Subsequent steps of Algorithm 3.2 ensure that the final p_k satisfies $m_k(p_k) \leq m_k(x_1)$. When $||x_1|| \geq \Delta_k$, on the other hand, the second **if** statement is activated, and Algorithm 3.2 terminates at the Cauchy point, proving our claim. This property is important for global convergence since each step is at least as good as the Cauchy point in reducing the model m_k, Algorithm 3.2 is globally convergent.

Another crucial property of the method is that each iterate x_j is larger in norm than its predecessor. This property is another consequence of the initialization $x_0 = 0$. Its main implication is that it is acceptable to stop iterating as soon as the trust-region boundary is reached because no further iterates giving a lower value of the model function m_k will lie inside the trust region. We state and prove this property formally in the following theorem, which makes use of the expanding subspace property of the conjugate gradient algorithm.

Theorem 3.1. *The sequence of vectors x_j generated by Algorithm 3.2 satisfies*

$$0 = ||x_0||_2 < ... < ||x_j||_2 < ||x_{j+1}||_2 < ... < ||p_k||_2 \leq \Delta_k.$$

Proof. We first show that the sequences of vectors generated by Algorithm 3.2 satisfy
$x_j^T = 0$ for j ≥ 0 and $x_j^T\ d_j > 0$ for j ≥ 1. Algorithm 3.2 computes x_{j+1} recursively in terms of x_j, but when all the terms of this recursion are written explicitly, we see that

$$x_j = x_0 + \sum_{i=0}^{j-1} \alpha_i d_i = \sum_{i=0}^{j-1} \alpha_i d_i.$$

Since $x_0 = 0$, multiplying by g_j and applying the expanding subspace property of conjugate gradients gives us

$$x_j^T g_j = \sum_{i=0}^{j-1} \alpha_i d_i g_j = 0. \tag{3.7}$$

An induction proof establishes the relation $x_j^T\ d_j > 0$. By applying the expanding subspace property again, we obtain

$$x_1^T d_1 = (\alpha_0 d_0)^T - g_1 + \beta_1 d_0 = \alpha_0 \beta_1 d_0^T d_0 > 0.$$

We now make the inductive hypothesis that $x_j^T\ d_j > 0$ and deduce that
$x_{j+1}^T\ d_{j+1} > 0$. From (3.7), we have $x_{j+1}^T\ g_{j+1} = 0$, and therefore

$$\begin{aligned} x_{j+1}^T d_{j+1} &= x_{j+1}^T (g_{j+1} + \beta_{j+1} d_j) \\ &= \beta_{j+1} x_{j+1}^T d_j \\ &= \beta_{j+1} (x_j + \alpha_j d_j)^T d_j \\ &= \beta_{j+1} x_j^T d_j + \alpha_j \beta_{j+1} d_j^T d_j. \end{aligned}$$

Because of the inductive hypothesis and positivity of β_{j+1} and α_j, the last expression is positive. We now prove the Theorem if Algorithm 3.2 terminates because
$d_j^T B_k d_j \leq 0$ or $||x_{j+1}||_2 \geq \Delta_k$, then the final point p_k is chosen to make $||p_k||_2 = \Delta_k$, which is the largest possible length. To cover all other possibilities in the algorithm, we must show that $||x_j||_2 < ||x_{j+1}||_2$ when $x_{j+1} = x_j + \alpha_j d_j$
and j ≥ 1. Observe that

$$||x_{j+1}||_2^2 = (x_j + \alpha_j d_j)^T (x_j + \alpha_j d_j) = ||x_j||_2^2 + 2\alpha_j x_j^T d_j + \alpha_j^2 ||d_j||_2^2.$$

It follows from this expression and our intermediate result that $||x_j||_2 < ||x_{j+1}||_2$, so our proof is complete.

□

From this theorem we see that Algorithm 3.2 sweeps out points x_j that move on some interpolating path from x_1 to the final solution p_k, a path in which every step increases its total distance from the start point. When $B_k = \nabla^2 f_k$ is positive definite, this path may be compared to the path of the dogleg method: Both methods start by minimizing m_k along the negative gradient direction g_k and subsequently progress toward p_k^B, until the trust-region boundary intervenes. One can show that, when $B_k = \nabla^2 f_k$ is positive definite, Algorithm 3.2 provides a decrease in the model $B_k p_k^B$ = -∇f_k that is at least half as good as the optimal decrease.

We can approximately solve the trust-region problem (3.2) by means of the conjugate gradient method with the termination tests proposed by Steihaug; see the conjugate gradient Steihaug algorithm. The step computation of this Newton conjugate gradient algorithm is obtained by setting $B_k = \nabla^2 f(x_k)$ at every iteration in Algorithm 3.2. This procedure amounts to applying the conjugate gradient method to the system

$$B_k p_k = \nabla f_k = g_k \qquad (**)$$

and stopping if (i) the size of the approximate solution exceeds the trust-region radius, (ii) the system (**) has been solved to a required accuracy, or (iii) if negative curvature is encountered. In the latter case we follow the direction of negative curvature to the boundary of the trust region $||p|| \leq \Delta_k$.

The control of the accuracy in the inner conjugate gradient iteration is important to keeping the cost of the Newton conjugate gradient method as low as possible. Near a well behaved solution x^*, the trust-region constraint becomes inactive, and the Newton conjugate gradient method reduces to the inexact Newton method. Which relate the choice of forcing sequence $\{\eta_k\}$ to the rate of convergence, become relevant during the later stages of the Newton conjugate gradient method.

The trust-region Newton conjugate gradient method has a number of attractive computational and theoretical properties. First, it is globally convergent. Its first step along the direction -g_k identifies the Cauchy point for the subproblem (3.2), and any subsequent conjugate gradient iterates only serve to improve the model value. Second, it requires no matrix factorizations, so we can exploit the sparsity structure of the Hessian B_k without worrying about fill-in during a direct factorization. Moreover, the conjugate gradient iteration the most computationally intensive part of the algorithm may be executed in parallel, since the key operation is a matrix vector product. When the Hessian matrix is positive definite, the Newton conjugate gradient method approximates the pure Newton step more and more closely as the solution x^* is approached, so rapid convergence is also possible.

Trust-region Newton conjugate gradient method accepts any direction of negative curvature, even when this direction gives an insignificant reduction in the model. Consider, for example, the case where the subproblem (3.2) is

$$m(p) = 10^{-3} p_1 + 10^{-4} p_1^2 - p_2$$
$$subject\ to\ \ ||p|| \leq 1,$$

where subscripts indicate elements of the vector p. The steepest descent direction at p = 0 is $(10^{-3}, 0)^T$, which is a direction of negative curvature for the model. Algorithm 3.1 would follow this direction to the boundary of the trust-region, yielding a reduction in model function m of about 10^{-3}. A step along e_2 also a direction of negative curvature would yield

a much greater reduction of 1. Although this method has limitation there are two advantages compared with the line search method, such as the lengths of the steps are controlled by the trust region and that directions of negative curvature are explored. Our computational experience shows that the latter is beneficial in practice, as it sometimes allows the iterates to move away from non minimizing stationary points.

3.3 Convergence rate of trust-region Newton's with conjugate gradient methods

In this section we present the convergence properties of the algorithm given in the previous section.

In our algorithm, if the unconstrained subproblem is chosen and the Hessian matrix is not positive definite, the trial step will be on the boundary of trust-region just the same as Steihaug conjugate gradient method. So the difference arises when the model function is convex and the trail step is large, which provides more decrease of the model function. Thus in fact the unconstrained model is used only when the model function is convex. The proof of the following Theorem is similar to that of Theorem 3.4 of Steihaug [11] and Powell [10].

We consider the unconstrained minimization problem

$$\min\{f(x): \quad x \in \Re^n\},$$

where f: $\Re^n \to \Re$ satisfies :

A.1. f is bounded below;

A.2. f is differentiable and the gradient is uniformly continuous. The framework of the algorithms is given below. For simplicity we assume that $C_k = \mathrm{I}$, where C is a symmetric and positive definite matrix and C_k is the weight of $||p||_c^2 = p^T \mathrm{Cp}$ Let $\bar{\Delta}$ be a fixed upper bound on the step lengths.

1. Given x_0, B_0, $\Delta_0 \leq \bar{\Delta}$ and ξ_0.
   Calculate f(x_0), g(x_0) and set k = 0.

2. Use the conjugate gradient algorithm to find approximate solution p_k with relative error ξ_k in $\frac{||g_{j+1}||}{||g||} \leq \xi$.

3. Calculate f($x_k + p_k$), g($x_k + p_k$) and (3.2)

4. Update x_{k+1} :

$$x_{k+1} = \begin{cases} x_k + p_k, & if \quad \rho_k > \alpha, \\ x_{k+1}, & otherwise. \end{cases}$$

Update Δ_{k+1} :

$$||p_k|| \leq \Delta_{k+1} \leq \min\{\gamma_3||p_k||, \bar{\Delta}\} \quad when \quad \rho_k \geq \alpha, \quad and \quad Compute \quad \gamma_i = d_i B d_i. \tag{3.8}$$

$$\gamma_4||p_k|| \leq \Delta_{k+1} \leq \gamma_5||p_k|| \quad when \quad \rho_k < \alpha. \tag{3.9}$$

Update B_{k+1} :

$$||B_{k+1}|| \leq \gamma_1 + \gamma_2 \sum_{i=0}^{k} ||p_i|| \tag{3.10}$$

Update ξ_{k+1}, k := k + 1, and continue with step 2.
The constants γ_1 and γ_2 depend on the update formula of B_{k+1}. Further,

$$\begin{aligned} 0 \leq \alpha_2 < \alpha < 1 \\ 0 < \gamma_4 \leq \gamma_5 < 1 \leq \gamma_3. \end{aligned} \tag{3.11}$$

If the matrix B_{k+1} is a secant update, then we have to decide whether or not to update when the step is rejected. If the step is rejected and the trust region parameter decreased, then the new direction in general is different from the old one, which is not the case in line searches. Hence, we can expect to get a better approximation by updating the matrix even if the step is rejected. However, we have that the new step can easily be computed if $B_{k+1} = B_k$ and $\rho_k \leq \alpha$. We note that this is the case when B_k only depends on x_k as in Newton's method. The first theorem is an important modification of the original convergence result of Powell [12]. In his convergence result Powell assumes that the direction P_k satisfies

$$-m_k(p_k) \geq c_1||g_k|| \min\{||p_k||, \frac{||g_k||}{||B_k||}\} \tag{3.12}$$

for some $c_1 > 0$, and he shows that

$$\min\{m_k(\tau g(x_k)) \quad | \quad ||\tau g(x_k)|| \leq ||p_k||\} \leq -\frac{1}{2}||g_k|| \min\{||p_k||, \frac{||g_k||}{||B_k||}\}.$$

But we have,

$$m_k(p_k) \leq \min\{m_k(\tau g(x_k))|||\tau g(x_k)|| \leq ||p_k||\}$$

with equality only if i = 0. Hence the direction p_k from Step 2 satisfies (3.12) with $c_1 = \frac{1}{2}$. Note that when Δ_k is reduced, $x_k + p_k$ is rejected and B_k is unchanged. We only include the part of the proof that differs from Powell's original proof. The algorithm includes the generalization of Thomas [1].

Theorem 3.2. *If $\xi_k \leq \xi < 1$, then*

$$\lim_{k\to\infty} \inf ||g(x_k)|| = 0.$$

Proof. The result is proved by obtaining a contradiction. Assume that $\{||g(x_k)||\}$ is bounded away from zero. First it is shown that

$$\sum_{k\geq 0} ||p_k|| < +\infty, \tag{3.13}$$

hence, $\{||B_k||\}$ is uniformly bounded. The next major step is to show that

$$\lim_{k\to\infty} \rho_k = 1. \tag{3.14}$$

Hence, from (3.9) and (3.11) we can conclude that for k sufficiently large,

$$\Delta_{k+l} \geq ||p_k||.$$

We will now show that for k sufficiently large, under the above assumption we do not use an inexact quasi-Newton step, i.e,

$$\frac{||r_k||}{||g(x_k)||} > \xi_k, \quad for\ k \geq k_0. \tag{3.15}$$

From (3.13) we have that $||p_k|| \to 0$. Since we have assumed that $\{g(x_k)\}$ moves away from zero, it follows that the fraction

$$\frac{||B_k p_k + g(x_k)||}{||g(x_k)||} > \xi_k, \quad for\ k \geq k_0$$

is arbitrarily close to 1, so for sufficiently large k we have that

$$\frac{||r_k||}{||g(x_k)||} > \xi, \qquad for\ k \geq k_0.$$

Thus we have the desired result (3.15).
For k_0 sufficiently large we are using termination (3) or (4), and we have

$$\Delta_{k+1} \geq \Delta_k = ||p_k||, \qquad for\ k \geq k_0.$$

Hence, $||p_{k+1}||\ ||p_k|| > 0$ for k $\geq k_0$. This contradicts (3.13) and shows that $\{||g(x_k)||\}$ is not bounded away from zero.

□

Theorem 3.3. *Let $\alpha_1 > \alpha_2 > 0$ and assume that $\{||B_k||\}$ is bounded.*
If $\xi_k \leq \xi < 1$, then

$$\lim g(x_k) = 0.$$

2. **Super-linear rate of convergence**

Theorem 3.4. *Suppose that f in (1) is twice continuously differentiable and bounded below and the norm of Hessian matrix is bounded, the iteration $\{x_k\}$ generated by Algorithm 3.2 satisfies $x_k \to x^*$ as $k \to \infty$ and the Hessian matrix $B(x^*)$ of f is positive definite. Let ξ_k be the relative error in the conjugate gradient method and Algorithm 3.1. If $\xi_k \to 0$ then $\{x_k\}$ converges super-linearly, i.e.,*

$$\lim_{k\to\infty} \frac{||x_{k+1} - x^*||}{||x_k - x^*||} = 0. \tag{3.16}$$

Proof. Because $x_k \to x^*$ and B(x^*) > 0, there exists k_1 such that $||B^{-1}(x_k)|| \leq 2||B^{-1}(x^*)||$ for all k $\geq k_1$. Therefore, the Newton step p_k^B = -$B^{-1}(x_k)$g(x_k) satisfies that

$$||p_k^B|| \leq 2||B^{-1}(x^*)||||g(x_k)|| \tag{3.17}$$

for all k $\geq k_1$. Following this, no matter p_k generated by our algorithm is a trust-region step or a Newton step, we have that

$$||p_k|| \leq 2||B^{-1}(x^*)||||g(x_k)|| \quad \forall\ k \geq k_1. \tag{3.18}$$

Our previous theorem implies that $||g(x_k)|| \to 0$. Inequality (3.18) shows that

$$\lim_{k\to\infty} \rho_k = 1. \tag{3.19}$$

Consequently, $x_{k+1} = x_k + p_k$ and $\Delta_{k+1} \geq \Delta_k$ for all sufficiently large k.
As a result, $||p_k|| < \Delta$ for all sufficiently large k, namely $||p_k||$, is an inexact Newton step for all large k, which indicates that

$$\frac{||g(x_k) + B_k p_k||}{||g(x_k)||} \leq \epsilon_k \tag{3.20}$$

for all sufficiently large k. Relation (3.20) shows that

$$\begin{aligned}
\lim_{k\to\infty} \frac{||g(x_{k+1})||}{||g(x_k)||} &= \lim_{k\to\infty} \frac{||g(x_k + p_k)||}{||g(x_k)||} \\
&= \lim_{k\to\infty} \frac{||g(x_k) + B_k p_k|| + o(||p_k||)}{||g(x_k)||} \\
&= \lim_{k\to\infty} \frac{||g(x_k) + B_k p_k||}{||g(x_k)||} \\
&\leq \lim_{k\to\infty} \epsilon_k = 0.
\end{aligned} \tag{3.21}$$

Now, (3.16) follows from the fact that $\mathrm{B}(x^*) > 0$ and $x_k \to x^*$. □

Chapter 4

Numerical implementation

In this chapter we report numerical results of our algorithm given in Chapter Three, and we also compare our algorithm with Newton and steepest descent algorithm. Test problems are the following,

i. $f(x, y, z) = e^{-x-y} + (x^4 + y)^2 + 2(y + z - 6)^2$

ii. $f(x, y) = x^2 + (y - 1)^2$

iii. $f(x, y, z) = (x - 2)^4 + (y - 5)^2 + 6(cos(z/2))$

We take $f(x, y, z) = e^{-x-y} + (x^4 + y)^2 + 2(y + z - 6)^2$, x_0=[100;5;0] and tolerance = 10^{-6}

Then using Newton's method, we set the following table

Table 4.1: Newton's method

k	x_1	x_2	x_3	$\|g\|$	obj
1	66.66667	0.00000	6.00000	120000.00000	100000027.000
2	44.44444	0.00000	6.00000	53333.33333	19753086.420
3	29.62963	0.00000	6.00000	23703.70370	3901844.231
16	0.49333	0.24012	5.75988	0.00102	0.597138
17	0.49333	0.24012	5.75988	0.00000	0.597138

We use the same function, initial guess and tolerance solve the given function by using Steepest descent method, we set the following table

Table 4.2: Steepest descent method

k	x_1	x_2	x_3	$\|\|g\|\|$	obj
1	-22.070313	4.999756	0.000061	36687973.373986	26158101.142163
2	-18.975185	8.089756	0.000061	96770.535594	183108.369398
3	-9.121508	14.604932	-0.000449	3036.115251	7283.940074
52	0.493327	0.240125	5.759875	0.000002	0.597138
53	0.493327	0.240124	5.759875	0.000001	0.597138

We use the same function, initial guess and tolerance to solve the given function by using conjugate gradient Steihaug method, we set the following table

Table 4.3: Conjugate gradient Steihaug method

k	x_1	x_2	x_3	$\|\|g\|\|$	obj
1	98.50000	5.00000	0.00000	4000000.00001	100000027.00000
2	95.50000	4.99999	0.00000	3822686.50001	94133682.06249
3	89.50000	4.99998	0.00001	3483935.50001	83178987.06247
20	0.49333	0.24012	5.75988	0.00238	0.59714
21	0.49333	0.24012	5.75988	0.00000	0.59714

• For Newton's method 17 iteration to obtain the approximate solution but the steepest descent method, we need 53 iteration and while the conjugate gradient steihaug method 21 iteration to obtain the approximate solution [0.4933,0.2401,5.7599] and the objective value is 0.5971. Thus, Newton's method is fast but steepest descent method is very slow.

• And we consider the function $f(x, y, z) = (x-2)^4 + (y-5)^2 + 6cos(z/2)$ and an initial guess $x_0 = (0, 3, \pi)$, tolerance = 10^{-6}

In Newton's method for this initial point is not defined which yields "NaN" in MATLAB program but in conjugate gradient Steihaug method to get an approximate solution [2.0063,5.0000,6.2832] and the minimum value -6.0000, we need 261 iterations.

Table 4.4: Conjugate Gradient Steihaug method

k	x_1	x_2	x_3	$\|g\|$	obj
1	1.48202	3.18525	3.28053	32.38827	20.00000
2	1.52603	3.47258	3.51745	4.73697	2.94881
3	1.56347	3.74111	3.77652	4.26607	1.26253
260	2.00629	5.00000	6.28319	0.00000	-6.00000
261	2.00628	5.00000	6.28319	0.00000	-6.00000

Conclusions

To find approximate solution as we have seen before, there are two iterative methods which are line search and trust-region methods. In relation to this, under line search method there are three categories: Newton, steepest and conjugate gradient method.

The conjugate gradient method is the conjugate direction method that is obtained by selecting the successive direction vectors as a conjugate version of the successive gradients obtained as the method progresses. Thus, the directions are not specified beforehand, rather they are determined sequentially at each step of the iteration. Accordingly, to compensate the drawback of Newton method we used conjugate gradient methods to get approximate solution of p_k, and after that we solved the trust-region subproblem. This algorithm (3.2) for unconstrained optimization is global convergence and has local super-linear convergence rate when the Hessian matrix of the objective function at the local minimizer is positive definite. The results showed that the algorithm (3.2) is more efficient than Newton$'$s method and steepest decent method in terms of the number of iteration.

From the numerical implementations, we get that the conjugate gradient trust-region method is better than Newton's method and steepest descent method.

Bibliography

[1] A. Antoniou, Wu-Sheng Lu. Practical Optimization: Algorithms and Engineering Applications, Newyork USA, 2007.

[2] C. T. Kelley, Iterative Methods For Optimization: In Applied Mathematics, Vol.2, USA, 1999.

[3] D. G. Lvenberger, Y. Ye. Linear and Nonlinear Programming. Springer Science+Business Media, LLC, third edition, New York USA, 2008.

[4] Fiacco, A.V. and G.P. McCormick. Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley and Sons, New York, 1968.

[5] K. P. CHONG, S.H. ZAK. An Introduction to Optimization. John Wiley and Sons Inc, Canada, Second edition, 2001.

[6] J. Jahn. Introduction to the Theory of Nonlinear Optimization. Springer-Verlag, Berlin, 1996.

[7] J. Nocedal and S. J. Wright. Numerical Optimization. Springer Series in Operations Research. Springer Verlag, New York, second edition, 2006.

[8] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. Nonlinear Programming, Theory and Algorithms. J. Wiley and Sons, New Jersey, third edition, 2006.

[9] W. SUN, Y. X. YUAN. Optimization Theory and Methods (Nonlinear Programming). Springer Science+Business Media, LLC, New York USA, 2006.

Appendices

MATLAB code for Newton's method

```
function Newton
x=input('Enter the column vector x');
  ob=ff(x);
  g=agra(x);
  j=hessu(x);
  k=1;
  tol=10^(-4);
  while norm(g)>tol
      ob=ff(x);
      g=agra(x);
      j=hessu(x);
      X=-inv(j)*g;
      x=x+X;
      k=k+1;
end
```

MATLAB code for steepest descent method

```
 function steep
x=input('Enter the column vector x');
  n=length(x);
  a= 1/2;
  b = 1/10;
  obj=func(x);
  g=grad(x);
  k=0;
  nf=1;
  tol=10^(-6);
while  norm(g) > tol
    d = -g;
    t = 1;
    newobj = func(x + t*d);
    nf = nf+1;
    while (newobj-obj)/t > a*g'*d
      t = t*b;
      newobj = func(x + t*d);
      nf = nf+1;
```

```
    end
    x = x + t*d;
    obj=newobj;
    g=grad(x);
    k = k + 1;
end
```

MATHLAB code for conjgate gradient Steihaug method

```
function p=CGS(g_k,B,delta_k,tol)
n=length(g_k);    %g_k=grad(x),B=hessian(x)
p_j=zeros(n,1);   %initial p_0
r_j=g_k;
d_j=-r_j;
delta_k=1;
k=1;

for j= 1:n
    D=d_j'*B*d_j;
    if D<=0            %d_j -negative curvature
        a=norm(d_j)^2;
        b=2*p_j'*d_j;
        c=norm(p_j)^2-(delta_k)^2;
        tau=(-b+sqrt(b^2-4*a*c))/(2*a)
        p=p_j+tau*d_j;
       % return
    end
    %do it con.t if D>0
    r_j2=r_j'*r_j;
    t_j=r_j2/D;
    p_j1=p_j+t_j*d_j;
    Np_j1=norm(p_j1);
    if Np_j1>=delta_k
        if Np_j1==delta_k
            p=p_j;
          %  return
        elseif Np_j1>delta_k
        a=norm(d_j)^2;
        b=2*p_j'*d_j;
        c=norm(p_j)^2-(delta_k)^2;
        tau=(-b+sqrt(b^2-4*a*c))/(2*a);
        p=p_j+tau*d_j;
```

```
        %return
        end
    end
    %do norm(p_j1)<delta_k
    p_j=p_j1;
    r_j=r_j+t_j*B*d_j; % residual at current iter
    Nr_j=norm(r_j);
    if Nr_j<tol
        p=p_j;
      %  return
    end
    beta=r_j'*r_j/r_j2;
    d_j=-r_j+beta*d_j;
    k=k+1;
end %for
p=p_j;
```