UPDATING CONJUGATE DIRECTIONS BY THE BFGS FORMULA WITH VARIABLE STORAGE

by

Ann Lee

B.Sc., Shanghai University of Science and Technology, 1982
M.Sc., Shanghai University of Technology, 1987

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

Dr. A. Buckley, Supervisor (Department of Computer Science)

[redacted]

Dr. D. D. Olesky, Departmental Member (Department of Computer Science)

[redacted]

Dr. D. Hewgill, Outside Member (Department of Mathematics)

[redacted]

Dr. W. S. Lu, External Examiner (Department of Electrical Engineering)

Supervisor: Dr. Albert Buckley

Abstract

In solving unconstrained nonlinear optimization problems, the quasi-Newton (QN) methods and the conjugate gradient (CG) methods are widely used. These two methods can be regarded as representatives of two quite different approaches: the QN methods usually have very good convergence properties, but require $O(n^2)$ memory space; on the contrary, the CG methods require only $3n$ or $4n$ locations of memory space, but usually have a slower convergence rate than the QN methods. Some algorithms have been developed trying to achieve some kind of trade-off between the QN and CG methods in terms of memory requirements and convergence speed. These include the Variable Storage Conjugate Gradient method (VSCG) which runs the QN algorithm for a number of iterations (depending on the available memory space), then switches to the preconditioned CG method until termination.

Powell proposed a method to use a matrix Z instead of the quasi-Newton matrix H, where Z can be updated in a similar way to updating H by the BFGS formula. The advantage of using Z is that ZZ^T remains positive definite despite accumulated roundoff errors during the iterations, which is important to keep good accuracy for those problems that are very sensitive to roundoff errors. The disadvantage of Powell's method is that Z is not symmetric, so this method requires twice as much storage as the QN method.

In this thesis, we develop a so-called VS-ZZ^T algorithm by applying the strategy of the VSCG method to Powell's method, so that we can largely maintain the good properties

of Powell's method, but with a variable storage requirement. Instead of storing the whole matrix Z, the VS-ZZ^T algorithm consists of two parts: the QN-part runs a modified Powell's updating formula based on the information saved at each iteration; the CG-part runs the preconditioned CG algorithm, with a fixed preconditioner $H_m = Z_m Z_m^T$ obtained from the QN-part. Numerical experiments have been made to verify that our algorithm does keep the main advantages of Powell's method, but the storage requirement can be adjusted by the user depending on the available memory space.

Examiners:

[redacted]

Dr. A. Buckley, Supervisor (Department of Computer Science)

[redacted]

Dr. D. D. Olesky, Departmental Member (Department of Computer Science)

[redacted]

Dr. D. Hewgill, Outside Member (Department of Mathematics)

[redacted]

Dr. W. S. Lu, External Examiner (Department of Electrical Engineering)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Basic Problem of Nonlinear Optimization

This thesis deals with the problem of minimizing an unconstrained nonlinear function $f(x)$ with n variables $x = (x^{(1)}, x^{(2)}, \cdots, x^{(n)})$, assuming that $f(x)$ is smooth and n is large (i.e. $n > 1000$). Usually, iterative methods are used for obtaining a minimizer of $f(x)$, that is, starting from an initial point x_0, $f(x)$ is minimized along lines of specified directions $d_1, d_2, \cdots$, until some termination criterion is satisfied. In solving this kind of problem, we are mainly concerned with the convergence properties and the storage requirements of an algorithm.

Among many different approaches, the quasi-Newton (QN) methods and the conjugate gradient (CG) methods are widely employed. These two methods can be regarded as representatives of two quite different approaches with respect to their storage requirement and convergence speed. In general, the quasi-Newton methods have very good convergence

properties, but they need $O(n^2)$ memory space to store a quasi-Newton matrix H for calculating the search direction $d_{i+1} = -H_i g_i$. For convenience, sometimes we drop the subscripts and write, for example, $d = -Hg$.

Here H_i denotes the quasi-Newton matrix H at x_i and it is a positive definite matrix approximating the Hessian of $f(x)$, while g_i denotes the gradient of $f(x)$ at x_i. At each iteration, H_i is updated based on H_{i-1}, the step $s_i = x_i - x_{i-1}$, and the gradient difference $y_i = g_i - g_{i-1}$. In this thesis, x_i is used to denote the i^{th} iteration point ($i = 0, 1, 2, \cdots$), with $x_i = x_{i-1} + \lambda_i d_i$, where λ_i is a scalar.

On the contrary, the conjugate gradient methods need only a memory storage of 3 or 4 n-dimensional vectors to get the search direction $d_{i+1} = -g_i + \beta_i d_i$, where β_i is a scalar. However, the CG methods need more iterations than the QN methods to obtain an equally good approximation to a minimizer of $f(x)$. When n is large, the difference in the memory requirements of these two methods is quite obvious. One important variation of the CG method is called the preconditioned CG method. It uses a preconditioning matrix H in the calculation of the direction $d_{i+1} = -Hg_i + \beta_i d_i$. A well-chosen preconditioner will greatly speed up the convergence rate of the CG method.

Some algorithms have been developed, based on the above two methods, to achieve some kind of trade-off between the storage requirement and convergence speed. Notable among these algorithms are those which use a variable amount of storage for allowing users to make the best use of the memory space available on a machine. Two successful examples are Nocedal's Product Form Variable Storage method [12], and Buckley and LeNir's Variable Storage Conjugate Gradient method (VSCG) [4].

The main idea of Nocedal's method and the VSCG method is as follows: both methods use a user-specified number m to denote the update limitation, where the value of m depends on the available storage and is much smaller than n. In practice, the typical value of m is chosen around 3 to 5. Instead of storing the whole quasi-Newton matrix, both methods conduct the update based on some information stored in each iteration. During the first m iterations, both methods use a type of updating scheme which is equivalent to the quasi-Newton method, where Nocedal's method uses the product form of the BFGS formula (Broyden[2], Fletcher[6], Goldfarb[9] and Shanno[14]) and the VSCG method uses the sum form of the BFGS formula. After m iterations, another type of updating scheme, based on the same formula but with no additional memory requirement, is used until the algorithm terminates.

In Nocedal's method, after m iterations, the quasi-Newton matrix is constructed based on the information of the last m iterations, and the oldest information is replaced by the newest information each time. On the other hand, in the VSCG method, after m iterations, the preconditioned CG method is used, with the quasi-Newton matrix H_m as a preconditioner, where H_m results from the first m iterations.

In this way, we are able to adjust the memory requirement depending on the available memory space to get better performance than with the CG method. If there is enough memory, we can make m large enough such that these two methods are in fact equivalent to the QN method.

The storage requirement for Nocedal's method is $m \times (2n + 1)$, and for the VSCG method it is $m \times (2n + 2)$. The key point here is that the more storage is allocated, the

more information can be saved. These two methods can be thought of as trying to reach a compromise between the CG and the QN methods and both methods have attained a rather good compromise.

It is because of the advantages of variable storage methods that we were inspired to apply the strategy of the VSCG method to Powell's factored form of BFGS updating method for achieving better efficiency in the memory usage.

1.2 Powell's Method

As was mentioned above, some nonlinear optimization algorithms, such as the quasi-Newton methods, use the BFGS formula to update a positive definite matrix H for approximating the Hessian of $f(x)$. H is then used to calculate a direction by computing $d = -Hg$. Powell [13] proposed a method which uses a square matrix Z to replace H such that

$$ZZ^T = H. \tag{1.1}$$

Powell updates Z in a way similar to updating H by the BFGS formula. Z satisfies the condition

$$Z^T H^{-1} Z = I, \tag{1.2}$$

which means that the columns of Z are normalized, mutually conjugate directions with respect to H^{-1}.

Obviously, if Z is available, $d = -ZZ^T g$ can also be computed in $O(n^2)$ operations and the proposed procedure is more stable than direct use of the BFGS formula, because ZZ^T

will remain positive definite despite accumulated roundoff errors during the iterations. In particular, Powell's algorithm can maintain good accuracy even in the presence of computer rounding errors when H is nearly singular. In addition, an extension to this procedure provides an automatic rescaling of the columns of Z. It avoids some inefficiencies due to a poor choice of the initial second derivative approximation H_0, so that good efficiency can be obtained for a much wider range of initial choices Z_0.

The shortcomings of this method are:

- the unsymmetric matrix Z needs n^2 storage locations, which is about twice as much as most quasi-Newton algorithms for unconstrained nonlinear optimization;
- compared with the quasi-Newton method, this algorithm needs more computations for updating Z.

1.3 Thesis Work

In this thesis, we devise an algorithm by applying the main ideas of the VSCG method to Powell's method in order to reduce its memory requirement of $O(n^2)$. The main idea of our algorithm is to avoid storing the whole matrix Z. This is feasible for the following reasons: the initial matrix Z_0 is usually an identity matrix and we don't have to save it explicitly for updating; also, we have devised an updating scheme to reconstruct the columns $z^{(j)}$ of $Z = (z^{(1)}, z^{(2)}, ..., z^{(n)})$ in a manner equivalent to Powell's method based on some necessary information saved at each previous iteration. The search direction d

can then be calculated by a summation

$$d = -ZZ^T g = -\begin{pmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(n)} \end{pmatrix} \begin{pmatrix} z^{(1)^T} \\ z^{(2)^T} \\ \vdots \\ z^{(n)^T} \end{pmatrix} g = -\sum_{j=1}^{n} z^{(j)} z^{(j)^T} g. \qquad (1.3)$$

As long as the gradient g is available and the columns of Z can be calculated column by column, we have no difficulty to obtain d.

Similarly to the VSCG method, in the first $m+1$ iterations, the algorithm updates the columns of Z in a way equivalent to Powell's method; after $m+1$ iterations, the algorithm uses the preconditioned CG method with the preconditioner $H_m = Z_m Z_m^T$. The storage requirement for our algorithm is as follows. During the first m iterations, we need $5n+3$ memory locations to keep the necessary information for each iteration; after m iterations, another $5n+3$ locations are used to store some intermediate update information, so the total storage requirement for updating Z is $(m+1) \times (5n+3)$. If m is much smaller than n, we can reduce the memory requirement by a large amount compared to the $O(n^2)$ locations required by Powell's method.

The advantages of our method are:

- the algorithm uses a variable amount of storage;
- although the whole matrix Z is not stored, the algorithm still preserves conjugate directions at each update and maintains the accuracy of Powell's method;
- by applying Powell's automatic rescaling strategy in our procedure, the algorithm can overcome the problem caused by roundoff errors in the iterations.

We call our algorithm the VS-ZZ^T method. We present the detailed mathematical derivation and give encouraging numerical results to show that our procedure has desirable properties.

As mentioned earlier, the initial matrix used in our algorithm is an identity matrix, which is a popular choice in practice. But we also devise a transformation scheme for allowing the possibility of using an arbitrary matrix as the initial matrix. We intended to be able to reproduce Powell's results through this approach for the purpose of comparison.

Note that, to be consistent in our later discussion, we use subscripts to denote the iteration and superscripts to denote the column or element, i.e. $z_i^{(j)}$ stands for the j^{th} column of the matrix Z at the i^{th} iteration and $s_i^{(j)}$ stands for the j^{th} element of the vector s at the i^{th} iteration.

Chapter 2

Basic Nonlinear Optimization Methods

This chapter reviews some preliminary knowledge about solving nonlinear optimization problems. We mainly discuss two basic methods: the quasi-Newton method and the conjugate gradient method, as well as their variations.

2.1 Preliminary Knowledge

The most straightforward way to solve the problem of minimizing a nonlinear function $f(x)$, with $x = (x^{(1)}, x^{(2)}, \cdots, x^{(n)})$, is generally by iteration. From a given point x_0, we set $x_i = x_{i-1} + \lambda_i d_i$ $(i = 1, 2, \cdots)$, where λ_i denotes a properly chosen scalar which reduces $f(x_{i-1} + \lambda_i d_i)$. If we choose λ_i in such a way that $f(x_{i-1} + \lambda_i d_i)$ is the minimum along the direction d_i, then we call it an "exact line search" (ELS); otherwise it is an inexact

line search. For an exact line search, the slope $\frac{\partial f}{\partial \lambda}$ at λ_i must be zero. That is

$$\frac{\partial f}{\partial \lambda}\bigg|_{\lambda=\lambda_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x^{(j)}} \frac{\partial x^{(j)}}{\partial \lambda}\bigg|_{\lambda=\lambda_i} = \nabla f_i^T d_i = g_i^T d_i = 0. \tag{2.1}$$

Equation (2.1) shows that the gradient of f at $x_i = x_{i-1} + \lambda_i d_i$ is orthogonal to the search direction d_i for an ELS.

If one of the points $x_1, x_2, \cdots$ is exactly equal to the minimum point of $f(x)$ within a finite number of steps, i.e. $x_i = x^*$, for some i, we call this occurrence "finite termination".

The most commonly used model in nonlinear optimization is the quadratic function

$$f(x) = \frac{1}{2} x^T A x + b^T x + c, \tag{2.2}$$

where A is a symmetric and positive definite matrix, b is a fixed n-dimensional vector and c is a scalar. This is because the quadratic function is one of the simplest smooth functions with a well determined minimum. Also, a general smooth function $f(x)$ can be approximated by a quadratic function in a region about the minimum point x^*, which agrees with $f(x)$ to a certain accuracy given by the Taylor series.

In practice, the line search is usually done by cubic or quadratic interpolation. Thus if $f(x)$ is quadratic, we can achieve an exact line search along any given direction. Moreover, if the search directions d_i are conjugate, that is, if they satisfy the condition

$$d_i^T A d_j = 0, \qquad i \neq j, \tag{2.3}$$

then we can find the minimizer x^* of $f(x)$ in at most n steps [7].

2.2 The Quasi-Newton Method

In the Newton method, the search direction is given by

$$d_i = -G_{i-1}^{-1} g_{i-1}, \qquad i = 1, 2, \cdots, \tag{2.4}$$

where G_{i-1} is the Hessian of $f(x)$ at x_{i-1}. By analogy to (2.4), the search direction d_i of the quasi-Newton method is given by

$$d_i = -H_{i-1} g_{i-1}, \qquad i = 1, 2, \cdots, \tag{2.5}$$

where H_{i-1} is a positive definite matrix which is updated from iteration to iteration.

Compared to the Newton method, the quasi-Newton method possesses many important features. Mainly, it avoids having to solve a set of linear algebraic equations at each iteration and it does not need explicit calculation of the Hessian matrix as the Newton method does. Also, the positive definite matrix H_{i-1} implies the descent property $d_i^T g_{i-1} < 0$, while the Hessian of $f(x)$ may be indefinite.

Given the initial point x_0 and an initial positive definite matrix H_0, from $i = 1$, the **Quasi-Newton Algorithm** is as follows:

Algorithm 2.1

(1) set $d_i = -H_{i-1} g_{i-1}$;

(2) perform a line search along d_i and set $x_i = x_{i-1} + \lambda_i d_i$;

(3) update H_{i-1} to form H_i (the method is discussed in the following);

(4) let $i = i + 1$ and go back to step (1).

The initial H_0 can be any positive definite matrix, but usually we choose $H_0 = I$.

In the quasi-Newton method, the approximation H_i to the inverse Hessian matrix is obtained by updating H_{i-1} by a rank-two matrix. Since a stationary point x^* of $f(x)$ is a local minimum if the Hessian matrix at x^* is positive definite, it is desirable for the approximating matrix H_i to be positive definite. Furthermore, if the Hessian matrix is positive definite, the approximating quadratic function has a unique local minimum, and the search direction d_i computed from (2.5) is a descent direction. Thus, it is usually required that the update formulae possess the property of giving hereditary positive definiteness, i.e. if H_{i-1} is positive definite, H_i must be positive definite too. So, choosing a proper updating method is very important for the quasi-Newton method. If the matrices become indefinite, many properties of this algorithm cease to be true.

Considerable research has been done in order to determine a better choice for the updating formula. However, much of the work is of purely theoretical interest. It is now generally accepted that the most effective update formula is the BFGS formula, which was independently suggested by Broyden[2], Fletcher[6], Goldfarb[9], and Shanno[14] in 1970. It must be mentioned here that the BFGS formula still may not guarantee positive definiteness at every iteration, although in theory it does. This is entirely due to roundoff error. However, compared to other updating formulae, the BFGS formula is still the most commonly used method in QN updating.

The sum form of the BFGS update formula computes

$$H_i = H_{i-1} - \frac{(s_i y_i^T H_{i-1} + H_{i-1} y_i s_i^T)}{s_i^T y_i} + (1 + \frac{y_i^T H_{i-1} y_i}{s_i^T y_i}) \frac{s_i s_i^T}{s_i^T y_i}. \tag{2.6}$$

An easy way to guarantee the positive definiteness of H_i is by ensuring $s_i^T y_i > 0$. From now on we assume that $s_i^T y_i > 0$ for all i.

Another important property for the QN method is the "quasi-Newton condition" which states

$$H_{i+1}y_i = s_i. \tag{2.7}$$

The quasi-Newton condition forces the Hessian approximation to produce a specified curvature along the search direction at a particular iteration. However, subsequent modification may destroy this property, and hence it is desirable that the curvature information from previous iterations should be retained, i.e. for $j \leq i$

$$H_{i+1}y_j = s_j. \tag{2.8}$$

If this holds for n linearly independent vectors $\{s_j\}$, the QN method will terminate in a finite number of iterations when $f(x)$ is quadratic.

In [7], Fletcher stated several important properties of the QN algorithm when using the BFGS formula:

I. for a quadratic function with ELS

(a) Algorithm 2.1 terminates in at most n iterations, with $H_n = A^{-1}$;

(b) previous quasi-Newton conditions $H_{i+1}y_j = s_j$ are preserved for $j \leq i$;

(c) it generates conjugate directions and conjugate gradients when $H_0 = I$.

II. for a general function

(a) it preserves positive definite matrices H_i, so the descent property holds;

(b) it requires only $3n^2 + O(n)$ multiplications per iteration;

(c) it has a superlinear order of convergence;

(d) global convergence is obtained for a strictly convex function with ELS.

Much research has shown that the quasi-Newton method is very successful for minimizing nonlinear functions. The main disadvantage of this method is that it requires $O(n^2)$ storage for the updating matrix H_i. For large problems, the storage requirement is very large and it may not be possible to retain the matrices in the high speed storage of a computer.

2.3 The Conjugate Gradient Method

Another commonly used method in nonlinear optimization is the conjugate gradient method. This method forms a class of algorithms which generate the search directions without storing any matrix H_i, in contrast to the quasi-Newton method described above. it is essential in the situation where the quasi-Newton method is not suitable because the relevant matrix is too large.

The basic **Conjugate Gradient Algorithm** can be described as follows:

Algorithm 2.2

(1) select an initial point x_0 and define $d_1 = -g_0$;

(2) find $x_i = x_{i-1} + \lambda_i d_i$ by a line search;

(3) determine the next direction $d_{i+1} = -g_i + \beta_i d_i$;

(4) let $i = i + 1$ and go back to (2).

Here β_i is a scalar. One choice is

$$\beta_i = \frac{g_i^T g_i}{g_{i-1}^T g_{i-1}}, \tag{2.9}$$

which is known as the Fletcher-Reeves formula. Another variation is the Polak-Ribière

formula which defines

$$\beta_i = \frac{(g_i - g_{i-1})^T g_i}{g_{i-1}^T g_{i-1}} = \frac{y_i^T g_i}{g_{i-1}^T g_{i-1}}. \tag{2.10}$$

For a quadratic function and exact line search, (2.9) and (2.10) are equivalent, but for general functions or inexact line searches, each choice leads to a distinct algorithm.

The following are some properties of the CG algorithm for quadratic functions with exact line searches:

(1) finite termination occurs in at most n steps;

(2) the algorithm generates conjugate directions;

(3) orthogonality of gradients is obtained, i.e.

$$g_i^T g_j = 0 \qquad i \neq j; \tag{2.11}$$

(4) descent search directions are obtained, with

$$d_{i+1}^T g_i = -g_i^T g_i < 0 \qquad i = 0, 1, \cdots. \tag{2.12}$$

The most important property of the conjugate gradient method is that it does not require any matrix operations to form d_i. In fact, Algorithm 2.2 requires only 3 vectors of storage to implement using formula (2.9), and requires 4 vectors of storage using formula (2.10). This enables the conjugate gradient method to be applied to some very large problems.

However, compared to the quasi-Newton method, the conjugate gradient method needs more iterations to achieve an equally good approximation to the minimum of $f(x)$. Also, it is less robust. Therefore, it would not be preferred in many circumstances.

2.4 The Preconditioned CG Method

An important variation for accelerating the CG method is the use of preconditioning. The idea originated in partial differential equation research [8], but is applicable more generally now in other areas based on the CG method. Theoretically speaking, to minimize a quadratic function of (2.2) with exact arithmetic, the number of iterations required is at most equal to the number of distinct eigenvalues of A. If the original function can be transformed to an equivalent function in which the transformed matrix $\hat{A}$ has many equal eigenvalues, the rate of convergence could be significantly improved.

Define a transformation of variables $y = H^{-\frac{1}{2}}x$, where H is a positive definite matrix. Apply the CG Algorithm in the transformed coordinates to obtain a sequence of points and directions. Then transforming back, we can get these same points and directions in the original coordinates. This leads to the **Preconditioned CG Algorithm**:

Algorithm 2.3

(1) select an initial point x_0 and define $d_1 = -Hg_0$;

(2) find the point $x_i = x_{i-1} + \lambda_i d_i$ by a line search;

(3) let $\hat{\beta}_i = \dfrac{g_i^T H(g_i - g_{i-1})}{g_{i-1}^T H g_{i-1}}$;

(4) determine the next direction $d_{i+1} = -Hg_i + \hat{\beta}_i d_i$;

(5) let $i = i + 1$ and go back step (2).

Note, in the preconditioned CG algorithm, that the initial d_1 is not the steepest descent direction. When $H = I$, it reduces to the normal CG algorithm. The properties of the preconditioned CG method are largely the same as the normal CG method.

2.5 The VSCG Method

2.5.1 The Motivation of the VSCG Method

To motivate the VSCG method, we first recall the advantages and disadvantages of the QN and the CG methods. In particular, the QN method has a faster convergence rate than the CG method, but the CG algorithms require much less storage. The VSCG method tries to reach a compromise between these two methods by using a variable amount of storage, with the performance of the algorithm depending on the availability of memory space.

Before we describe the VSCG algorithm, it is necessary to give an explanation about the connection between the BFGS quasi-Newton update formula and the preconditioned conjugate gradient method.

First, we rewrite the BFGS update formula (2.6) as

$$H^* = H + U(H,i) \equiv H - \frac{s_i y_i^T H + H y_i s_i^T}{s_i^T y_i} + (1 + \frac{y_i^T H y_i}{s_i^T y_i})\frac{s_i s_i^T}{s_i^T y_i}. \qquad (2.13)$$

This can be most easily read as "H^* is H plus an Update of H at the i^{th} step". The search direction for the QN method at the $(i+1)^{th}$ step is then

$$d_{i+1} = -H_i g_i = -[H_{i-1} - \frac{s_i y_i^T H_{i-1} + H_{i-1} y_i s_i^T}{s_i^T y_i} + (1 + \frac{y_i^T H_{i-1} y_i}{s_i^T y_i})\frac{s_i s_i^T}{s_i^T y_i}]g_i. \qquad (2.14)$$

If f is quadratic and an exact line search is performed, by equation (2.1) we have

$$s_i^T g_i = 0. \qquad (2.15)$$

Thus several terms in (2.14) containing $s_i^T g_i$ will disappear and d_{i+1} can now be written

as

$$d_{i+1} = -H_{i-1}g_i + \frac{y_i^T H_{i-1} g_i}{s_i^T y_i} s_i = -H_{i-1}g_i + \frac{y_i^T H_{i-1} g_i}{d_i^T y_i} d_i = -H_{i-1}g_i + \hat{\beta}_i d_i. \qquad (2.16)$$

Two points must be mentioned here:

(1) For a quadratic function with exact line search, equation (2.16) has the same form as the preconditioned CG Algorithm 2.3, since

$$y_i^T d_i = (g_i^T - g_{i-1}^T) d_i = -g_{i-1}^T d_i = -g_{i-1}^T (-H_{i-1} g_{i-1}) = g_{i-1}^T H_{i-1} g_{i-1}.$$

(2) The basic difference in these two methods is that the quasi-Newton matrix H in the preconditioned CG method doesn't change, but in the QN method, it changes at every step.

Thus, the preconditioned conjugate gradient algorithm can be regarded as the quasi-Newton algorithm with a constant matrix H.

2.5.2 The VSCG Algorithm

Based on the similarity between the QN method and the preconditioned CG method, the VSCG method was devised, without storing the matrix H. It uses a limited amount of memory space to update H in the same way as the QN method for the first m steps; then it switches to the preconditioned CG method using the fixed matrix H_m. The expectation is that the more memory space is available, the closer H_m is to the inverse Hessian, and better convergence can be obtained.

Starting from the initial point x_0, and going along the descent direction such that $s_i^T y_i > 0$, the **VSCG Algorithm** consists of the following two parts:

Algorithm 2.4

QN-part: Choose a positive definite matrix H_0. Iterate for $i = 1, 2, \cdots, m$:

$$d_i = -H_{i-1} g_{i-1};$$

$$x_i = x_{i-1} + \lambda_i d_i.$$

$$H_i = H_{i-1} + U(H_{i-1}, i);$$

CG-part: From the point x_m reached by the QN-part, use H_m as preconditioner. Iterate for $i = m+1, m+2, \cdots$:

$$d_i = -H_{i-1} g_{i-1};$$

$$x_i = x_{i-1} + \lambda_i d_i.$$

$$H_i = H_m + U(H_m, i);$$

Thus the VSCG algorithm simply combines the QN method and the preconditioned CG method. The QN-part can be viewed as constructing an appropriate preconditioner H_m and the CG-part is the implementation of a preconditioned CG algorithm.

The algorithm has some important properties:

- For a quadratic function and an exact line search, all directions are descent directions, as each H_i is positive definite, provided that $s_i^T y_i > 0$.
- By changing the order of equation (2.14), the computation of the direction $d = -Hg$ can be expressed in a vector form and hence the total storage required for constructing H_m is $m \times (2n + 2)$.

The following is the detailed explanation of the second property, which is necessary to understand our work in Chapter 4.

According to (2.14), at each iteration the search direction is calculated as $d_{i+1} = -H_i g_i$, where

$$\begin{aligned} H_i g_i &= H_{i-1} g_i - \left[\frac{u_i^T g_i}{\eta_i} - (1 + \frac{\nu_i}{\eta_i}) \frac{s_i^T g_i}{\eta_i} \right] s_i - \frac{s_i^T g_i}{\eta_i} u_i \\ &= H_{i-1} g_i - \sigma_i s_i - \mu_i u_i, \end{aligned} \tag{2.17}$$

in which $\nu_i = y_i^T H_{i-1} y_i$, $\eta_i = s_i^T y_i$ and $u_i = H_{i-1} y_i$, for $i = 1, 2, \cdots, m$ in the QN-part. From $i = m+1, \cdots$ in the CG-part, to calculate ν_i, u_i and $H_{i-1} g_i$ in (2.17), we use H_m to replace H_{i-1}. With regard to equation (2.17), we only need to calculate the product of H_{i-1} and a vector v, where $v = g_i$ or $v = y_i$.

Let H_0 be an identity matrix. We can construct H_{i-1} from H_0 by a recursive procedure:

$$\begin{aligned} H_1 &= H_0 + U(H_0, 1) \\ H_2 &= H_1 + U(H_1, 2) = H_0 + U(H_0, 1) + U(H_1, 2) \\ &\vdots \\ H_{i-1} &= H_0 + \sum_{j=1}^{i-1} U(H_{j-1}, j). \end{aligned} \tag{2.18}$$

Using (2.18), we obtain $H_{i-1} v$ by

$$H_{i-1} v = H_0 v - \sum_{j=1}^{i-1} \Big([\frac{u_j^T v}{\eta_j} - (1 + \frac{\nu_j}{\eta_j}) \frac{s_j^T v}{\eta_j}] s_j + \frac{s_j^T v}{\eta_j} u_j \Big) = H_0 v - \sum_{j=1}^{i-1} (\sigma_j s_j + \mu_j u_j). \tag{2.19}$$

To avoid unnecessary repetition in constructing $H_{i-1} v$, we store two vectors s_j, u_j and two scalars ν_j, η_j at each iteration.

Then, according to the following order, we calculate $H_i g_i$ by:

(1) $H_{i-1} g_i$;

(2) $u_i = H_{i-1} y_i$, $\nu_i = y_i^T u_i$ and $\eta_i = s_i^T y_i$;

(3) $u_i^T g_i$, $s_i^T g_i$, $\sigma_i s_i$ and $\mu_i u_i$.

It needs $(i-1) \times (4n)$ operations for calculating $H_{i-1} v$, ignoring $H_0 v$. So, the total operation count is $(8m+6) \times n$ for $i \geq m$. Usually $m \ll n$, so this is only a moderate multiple of n. The variables saved in each iteration require $2n+2$ locations, so the total storage needed for $H_1, H_2, \cdots, H_m$ is then $m \times (2n+2)$ locations.

The VSCG method is an efficient method, which can use whatever storage is available. In practice, the common situation we meet is that the available memory is sometimes larger than the $4n$ required by the CG algorithm, but smaller than the $\frac{1}{2}n^2$ required by the QN algorithm. The VSCG algorithm provides a more flexible choice for the user to use the storage efficiently. Also, many tests have verified that for small values of m, the numerical results are better than the normal CG method. Additional space aiding performance is what is expected.

Chapter 3

Powell's Method

In this chapter, we will describe Powell's method [13] of "updating conjugate directions by the BFGS formula". This method, together with the VSCG method described in Chapter 2, is the main source of our thesis work. In addition to the basic algorithm, we also describe an automatic rescaling technique in detail. Some numerical examples are used to justify our interest.

3.1 The Main Idea of Powell's Method

In addition to the "sum form" of BFGS formula (2.6), there is another so called "product form" of the BFGS formula given by Brodlie, Gourlay and Greenstadt [1], which can be expressed as

$$H^* = (I - sp^T)H(I - ps^T), \tag{3.1}$$

where p is the vector

$$p = \frac{H^{-1}s}{\sqrt{(s^T y)(s^T H^{-1} s)}} + \frac{y}{s^T y}. \tag{3.2}$$

The step in (3.2) can be written as

$$s = \lambda d = -\lambda H g, \tag{3.3}$$

or

$$H^{-1}s = -\lambda g. \tag{3.4}$$

Substituting (3.4) into (3.2), we are able to calculate the vector p without knowing H^{-1} explicitly.

Based on (3.1), Powell [13] recommended using a factored updating method which uses a square matrix Z instead of H, where Z satisfies

$$ZZ^T = H. \tag{3.5}$$

One advantage of using Z is that ZZ^T will remain positive definite despite accumulated roundoff errors. Moreover, we can rewrite the equation (3.5) in the form of (1.2), so that the columns of Z still satisfy the conjugacy condition (2.3), that is, they are normalized and mutually conjugate with respect to H^{-1}.

Using Z in place of H in (3.1), the updating formula for Z^* can be expressed as

$$Z^* = (I - sp^T)Z, \tag{3.6}$$

where Z^* satisfies $Z^* Z^{*T} = H^*$. The search direction is then calculated by $d = -ZZ^T g$, rather than using H directly.

In the next section, we show a new algorithm Powell used to compute Z^* based on a special case that s is a multiple of the first column of Z. However, to deal with the general case that s is not a multiple of $z^{(1)}$, Powell used $\bar{Z}$ to replace Z in (3.6), with

$$\bar{Z} = Z\Omega, \tag{3.7}$$

when Ω is a lower Hessenberg orthogonal matrix. The aim of this replacement is to make the step s a multiple of the first column of new matrix $\bar{Z}$. Since

$$\bar{Z}\bar{Z}^T = Z\Omega(Z\Omega)^T = Z\Omega\Omega^T Z^T = ZZ^T,$$

the condition (3.5) is still satisfied. In that case, (3.6) is modified by substituting $\bar{Z}$ in place of Z, that is,

$$Z^* = (I - sp^T)\bar{Z}. \tag{3.8}$$

Powell showed that the use of Z instead of H is important to maintain good accuracy when computer roundoff errors have significant effects. For example, if H has $n-1$ eigenvalues of magnitude one and one very big eigenvalue whose eigenvector is $v = (1, 1, \cdots, 1)^T$, then H would become a nearly singular matrix whose elements are a large multiple of one. However, if we use ZZ^T to replace H, the large elements may be confined to the first column of Z, and Z is a nonsingular matrix, so the algorithm may still keep good accuracy.

Consider the following example. Suppose $\lambda_1 = 10^6$ is the largest eigenvalue of H with the eigenvector $v_1^T = (1, 1, \cdots, 1)$, and all other eigenvalues are negligible compared to λ_1. Then H can be approximated by

$$H = \sum_{i=1}^{n} \lambda_i v_i v_i^T \approx \lambda_1 v_1 v_1^T$$

$$= 10^6 \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} (1\ 1 \cdots 1) = 10^6 \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}. \tag{3.9}$$

So, H is nearly a singular matrix. But if we use Z instead of H, we can get a nonsingular matrix Z, with

$$Z = \begin{pmatrix} 10^3 & * & \dots & * \\ 10^3 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 10^3 & * & \dots & * \end{pmatrix},$$

where $*$ denotes a small element. Then

$$ZZ^T = \begin{pmatrix} 10^3 & * & \dots & * \\ 10^3 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 10^3 & * & \dots & * \end{pmatrix} \begin{pmatrix} 10^3 & 10^3 & \dots & 10^3 \\ * & * & \dots & * \\ \vdots & \vdots & & \vdots \\ * & * & \dots & * \end{pmatrix} \approx 10^6 \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}.$$

Thus the nonsingular matrix Z is used in place of the nearly singular matrix H.

3.2 The Basic Algorithm

In this section, we describe a method to compute Z^* given by Powell. Let $z^{(j)}, \bar{z}^{(j)}$ and $z^{(j)^*}$ be the j^{th} column of Z, $\bar{Z}$ and Z^*, respectively, for $j = 1, 2, \cdots, n$. Let c_1, c_2 and c be constants and assume for the moment that the step s is a multiple of $z^{(1)}$, that is, $s = c_1 z^{(1)}$. From equation (3.6) we get

$$(z^{(1)^*}\ z^{(2)^*}\ \cdots\ z^{(n)^*}) = (I - c_1 z^{(1)} p^T)(z^{(1)}\ z^{(2)}\ \cdots\ z^{(n)}). \tag{3.10}$$

The first column of Z^* can be expressed as

$$z^{(1)^*} = (I - c_1 z^{(1)} p^T) z^{(1)} = (z^{(1)} - c_1 z^{(1)} p^T z^{(1)}) = (1 - c_1 p^T z^{(1)}) z^{(1)} = c_2 z^{(1)} = \frac{c_2}{c_1} s = cs. \tag{3.11}$$

Using equations(1.2) and (3.11) and the quasi-Newton equation $s = Hy$ from (2.8), we have

$$z^{(1)*T} H^{*-1} z^{(1)*} = cs^T H^{*-1} cs = c^2 s^T H^{*-1} s = 1,$$

so

$$c = \frac{1}{\sqrt{s^T H^{*-1} s}} = \frac{1}{\sqrt{s^T y}}. \tag{3.12}$$

Thus we obtain the first column of Z^*,

$$z^{(1)*} = \frac{s}{\sqrt{s^T y}}. \tag{3.13}$$

Note that $z^{(1)*}$ is independent of p in this expression.

Moreover, the conditions $Z^T H^{-1} Z = I$ and[1] $s \| z^{(1)}$ imply $(H^{-1}s)^T z^{(j)} = 0$, $j = 2, 3, \cdots, n$. Thus, we have

$$z^{(j)*} = (I - sp^T) z^{(j)} = z^{(j)} - s\Big(\frac{(H^{-1}s)^T}{\sqrt{(s^T y)(s^T H^{-1} s)}} + \frac{y^T}{s^T y}\Big) z^{(j)} = z^{(j)} - \frac{y^T z^{(j)}}{s^T y} s. \tag{3.14}$$

So, the update formula for Z^* can be expressed as follows:

$$z^{(j)*} = \begin{cases} \dfrac{s}{\sqrt{s^T y}}, & j = 1, \\ z^{(j)} - \dfrac{y^T z^{(j)}}{s^T y} s, & j = 2, 3, \cdots, n. \end{cases} \tag{3.15}$$

Two advantages of this updating method are:

(1) H^{-1} is not present;

(2) if $\|z^{(1)*}\| \ll \|z^{(1)}\|$, the reduction is achieved by a suitable multiplication rather than by cancellation.

---

[1] here, $\|$ means that s is a multiple of $z^{(1)}$

However, when s is not a multiple of $z^{(1)}$, the update formula (3.15) does not follow. Powell then suggested using an orthogonal matrix Ω to make a transformation as in equation (3.7) so that the first column of $\bar{Z} = Z\Omega$ becomes a multiple of s. If this condition is satisfied, then we can apply the same procedure to derive an update formula for $\bar{Z}$ similar to (3.15):

$$z^{(j)^*} = \begin{cases} \dfrac{s}{\sqrt{s^T y}}, & j = 1, \\ \bar{z}^{(j)} - \dfrac{y^T \bar{z}^{(j)}}{s^T y} s, & j = 2, 3, \cdots, n, \end{cases} \tag{3.16}$$

where $z^{(j)}$ in (3.15) is replaced by $\bar{z}^{(j)}$.

In order to make $\bar{z}^{(1)}$ a multiple of s, that is

$$\bar{z}^{(1)} = \bar{Z} e_1 = Z\Omega e_1 = \alpha s,$$

where α is a scalar, the following two equations must hold:

$$\Omega e_1 = \alpha \hat{s}, \tag{3.17}$$

$$s = Z\hat{s}. \tag{3.18}$$

Here $\hat{s} = -Z^T g$ is an auxiliary vector and α is chosen to normalize $\hat{s}$, i.e. $\alpha = \pm\frac{1}{||\hat{s}||}$. Then $z^{(1)}$ becomes the multiple of s as desired,

$$\bar{z}^{(1)} = \bar{Z} e_1 = Z\Omega e_1 = \pm Z\frac{\hat{s}}{||\hat{s}||} = \pm\frac{s}{||\hat{s}||}. \tag{3.19}$$

The only thing unknown here is Ω. There are several ways to choose an orthogonal matrix Ω to satisfy $\Omega e_1 || \hat{s}$, and Powell recommended a lower Hessenberg matrix

$$\Omega = \Omega_{n-1}\Omega_{n-2}\cdots\Omega_1, \tag{3.20}$$

where Ω_k is a rotation matrix, $k = n-1, \cdots, 1$. Rewrite the condition (3.17) as $\Omega^T \hat{s} || e_1$, or $\Omega_1^T \cdots \Omega_{n-1}^T \hat{s} || e_1$. We define Ω_k^T in such a way that when left-multiplied into vector $\Omega_{k+1}^T \cdots \Omega_{n-1}^T \hat{s}$, it can make the $(k+1)^{th}$ component of the vector zero, and the k^{th} component positive, while all other components are untouched. Thus from $k = n-1$ to $k = 1$, each Ω_k^T has the effect of making one component of $\Omega_{k+1}^T \cdots \Omega_{n-1}^T \hat{s}$ zero, and this component will not be altered in the later processing. Finally, the vector $\Omega^T \hat{s}$ has only one non-zero component at the top; thus it is a multiple of e_1. In case the last r components of $\hat{s}$ are already zero, we can simply set $\Omega_k = I$ and $\bar{z}^{(k+1)} = z^{(k+1)}$ for $k = n-1, \cdots, n-r$. The following describes how Ω_k^T is calculated.

Suppose vector $v = \Omega_k^T \cdots \Omega_{n-1}^T \hat{s}$, and its last $n-k$ components have already become zero, or let $v = \hat{s}$ for $k = n$. The task of Ω_{k-1}^T is to make the k^{th} component of v zero. A simple rotation matrix is sufficient to do this:

$$\Omega_{k-1}^T = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & 0 & \\ & & 1 & & & & \\ & & & a^{(k)} & b^{(k)} & & \\ & & & -b^{(k)} & a^{(k)} & & \\ & & & & & 1 & \\ & 0 & & & & & \ddots \\ & & & & & & & 1 \end{pmatrix}, \tag{3.21}$$

where

$$a^{(k)} = \frac{v^{(k-1)^2}}{\sqrt{v^{(k-1)^2} + v^{(k)^2}}} \quad \text{and} \quad b^{(k)} = \frac{v^{(k)^2}}{\sqrt{v^{(k-1)^2} + v^{(k)^2}}}$$

are in the k^{th} and $(k-1)^{th}$ column and row. Thus, if $\hat{s}$ is given, theoretically, the computation of $\bar{Z} = Z\Omega$ should consist of two steps: the first step is to calculate Ω which satisfies $\Omega^T \hat{s} || e_1$; the second step is to calculate $\bar{Z} = Z\Omega$. This seems not very efficient. Goldfarb [10] gave a good algorithm to directly calculate the last $n-1$ columns of $\bar{Z}$. The

idea is to iteratively calculate the columns of $\bar{Z}$ directly from Z and $\hat{s}$, instead of explicitly computing Ω. Here is how it works.

Since $\bar{Z} = Z\Omega = Z\Omega_{n-1}\Omega_{n-2}\cdots\Omega_1$, the product of the first two matrices can be written as

$$\begin{aligned} Z\Omega_{n-1} &= \begin{pmatrix} z^{(1)} & \cdots & z^{(n-2)} & z^{(n-1)} & z^{(n)} \end{pmatrix} \begin{pmatrix} 1 & & & & \\ & \ddots & & 0 & \\ & & 1 & & \\ & 0 & & a^{(n)} & -b^{(n)} \\ & & & b^{(n)} & a^{(n)} \end{pmatrix} \\ &= \begin{pmatrix} z^{(1)} & \cdots & z^{(n-2)} & a^{(n)}z^{(n-1)} + b^{(n)}z^{(n)} & -b^{(n)}z^{(n-1)} + a^{(n)}z^{(n)} \end{pmatrix}, \end{aligned} \tag{3.22}$$

where

$$a^{(n)} = \frac{\hat{s}^{(n-1)}}{\sqrt{\hat{s}^{(n)2} + \hat{s}^{(n-1)2}}} \quad \text{and} \quad b^{(n)} = \frac{\hat{s}^{(n)}}{\sqrt{\hat{s}^{(n)2} + \hat{s}^{(n-1)2}}}.$$

Defining two variables $h = \hat{s}^{(n)}z^{(n)}$ and $\phi = \hat{s}^{(n)2}$, then

$$\begin{aligned} \bar{z}^{(n)} = -b^{(n)}z^{(n-1)} + a^{(n)}z^{(n)} &= \frac{1}{\sqrt{\hat{s}^{(n-1)2} + \hat{s}^{(n)2}}}\left(-\hat{s}^{(n)}z^{(n-1)} + \hat{s}^{(n-1)}z^{(n)}\right) \\ &= \pm\sqrt{\frac{\hat{s}^{(n)2}}{\hat{s}^{(n-1)2} + \hat{s}^{(n)2}}}\left(-z^{(n-1)} + \frac{\hat{s}^{(n-1)}\hat{s}^{(n)}z^{(n)}}{\hat{s}^{(n)2}}\right) \\ &= \pm\sqrt{\frac{\phi}{\hat{s}^{(n-1)2} + \phi}}\left(-z^{(n-1)} + \frac{\hat{s}^{(n-1)}}{\phi}h\right), \end{aligned}$$

$$a^{(n)}z^{(n-1)} + b^{(n)}z^{(n)} = \frac{1}{\sqrt{\hat{s}^{(n-1)2} + \hat{s}^{(n)2}}}\left(\hat{s}^{(n-1)}z^{(n-1)} + \hat{s}^{(n)}z^{(n)}\right). \tag{3.23}$$

Note that after $\bar{z}^{(n)}$ is done, it will not be altered in the later processing, but $a^{(n)}z^{(n-1)} + b^{(n)}z^{(n)}$ is only intermediate. Adding $\hat{s}^{(n-1)}z^{(n-1)}$ to h and $\hat{s}^{(n-1)2}$ to ϕ, immediately from (3.23) we have

$$a^{(n)}z^{(n-1)} + b^{(n)}z^{(n)} = \phi^{-1/2}h.$$

So,

$$Z\Omega_{n-1} = \left(z^{(1)} \ \cdots \ z^{(n-2)} \ \phi^{-1/2}h \ \bar{z}^{(n)}\right). \tag{3.24}$$

Note that the vector $\hat{s}$ is computed accordingly

$$\hat{s} = (\hat{s}^{(1)} \ \cdots \ \hat{s}^{(n-2)} \ \phi^{1/2} \ 0)^T.$$

In the next iteration, Ω_{n-2} is multiplied into $Z\Omega_{n-1}$:

$$\begin{aligned} Z\Omega_{n-1}\Omega_{n-2} &= \left(z^{(1)} \ \cdots \ z^{(n-2)} \ \phi^{-1/2}h \ \bar{z}^{(n)}\right) \begin{pmatrix} 1 & & & & \\ & \ddots & & 0 & \\ & & 1 & & \\ & & & a^{(n-1)} & -b^{(n-1)} \\ & 0 & & b^{(n-1)} & a^{(n-1)} \\ & & & & & 1 \end{pmatrix} \\ &= \Big(z^{(1)} \ \cdots \ z^{(n-3)} \ a^{(n-1)}z^{(n-2)} + b^{(n-1)}\phi^{-1/2}h \\ &\quad -b^{(n-1)}z^{(n-2)} + a^{(n-1)}\phi^{-1/2}h \ \bar{z}^{(n)}\Big), \end{aligned} \tag{3.25}$$

where

$$a^{(n-1)} = \frac{\hat{s}^{(n-2)}}{\sqrt{\hat{s}^{(n-2)^2} + \phi}} \quad \text{and} \quad b^{(n-1)} = \frac{\phi^{1/2}}{\sqrt{\hat{s}^{(n-2)^2} + \phi}}.$$

In the same way, we get

$$Z\Omega_{n-1}\Omega_{n-2} = \left(z^{(1)} \ \cdots \ z^{(n-3)} \ \phi^{-1/2}h \ \bar{z}^{(n-1)} \ \bar{z}^{(n)}\right),$$

where $h = \sum_{j=n-2}^{n} \hat{s}^{(j)}z^{(j)}$ and $\phi = \sum_{j=n-2}^{n} \hat{s}^{(j)^2}$.

Repeating the above procedure iteratively by multiplying $\Omega_k, k = n-1, \cdots, 1$ into Z, we get the transformed vectors $\bar{z}^{(n)}, \bar{z}^{(n-1)}, \ldots, \bar{z}^{(1)}$. Each time, the variables h and ϕ are updated by adding $\hat{s}^{(k)}z^{(k)}$ and $\hat{s}^{(k)^2}$, respectively.

This gives us Goldfarb's **Transformation Algorithm** for calculating the last $n-1$ columns of $\bar{Z}$ used in the formula (3.16).

Algorithm 3.1

step 0: Set k to be the greatest integer in $(1,\cdots,n)$, such that $\hat{s}^{(k)} \neq 0$.

Set an auxiliary vector $h = \hat{s}^{(k)} z^{(k)}$ and a scalar $\phi = \hat{s}^{(k)^2}$.

If $k < n$, set $\bar{z}^{(j)} = z^{(j)}$ for $j = k+1, k+2, \cdots, n$;

step 1: Terminate if $k = 1$;

step 2: Calculate the vector $\bar{z}^{(k)} = \sqrt{\frac{\phi}{\hat{s}^{(k-1)^2} + \phi}} \left(-z^{(k-1)} + \frac{\hat{s}^{(k-1)}}{\phi} h \right)$.

Then add $\hat{s}^{(k-1)} z^{(k-1)}$ to h and add $\hat{s}^{(k-1)^2}$ to ϕ;

step 3: Let $k = k-1$ and go back to step 1.

Since $\hat{s} = -Z^T g$, h is in fact the required search direction, i.e.

$$d = -ZZ^T g = Z\hat{s} = h.$$

Powell used Algorithm 3.1 to obtain the search direction d before conducting the update procedure. The calculation of both $\bar{Z} = Z\Omega$ and the search direction $-ZZ^T g$ can be done with $4n^2 + O(n)$ multiplications. Another $2n^2 + O(n)$ multiplications are needed for updating equation (3.16). Thus the total number of operations is roughly $6n^2$.

3.3 Automatic Rescaling and Numerical Results

When studying the convergence speed of the updating algorithm described in the last section, Powell found that the relative values between the norm of the initial matrix Z_0 and the norm of the Hessian matrix A have great impact on the algorithm termination.

From the last section we know that, at each iteration, the following condition must hold,

$$Z_i Z_i^T = H_i, \qquad i = 0, 1, 2, \cdots. \tag{3.26}$$

In theory, for a quadratic function with an exact line search, finite termination should be realized in at most n steps, that is,

$$Z_n Z_n^T = H_n = A^{-1}. \tag{3.27}$$

However in practice, due to roundoff error, sometimes finite termination may not occur for a given Z_0. Consider the case that the Hessian matrix A is the product of a constant θ and a positive definite matrix B whose elements are of order of unity, that is

$$A = \theta B = \theta(O(1)). \tag{3.28}$$

With this, A^{-1} is proportional to the product of θ^{-1} and some fixed matrix B^{-1}, that is

$$A^{-1} = \frac{1}{\theta} B^{-1}. \tag{3.29}$$

As

$$||A^{-1}|| = ||\frac{1}{\theta} B^{-1}|| = |\frac{1}{\theta}|\, ||B^{-1}||, \tag{3.30}$$

where $||B^{-1}||$ is a constant, clearly, $||A^{-1}|| \sim \frac{1}{\theta}$. Because both A and B are fixed during our computations, and because the following discussion could be modified to take account of $||B^{-1}||$, for convenience, we will assume that $||B^{-1}|| \approx 1$, which is the case with the example we have chosen.

For a given Z_0 with either $||Z_0|| < ||A^{-1/2}||$ or $||Z_0|| > ||A^{-1/2}||$, during the iterations, $||Z_i||$ should increase or decrease to approach $||A^{-1/2}||$, for $i = 1, 2, \cdots$. From (3.27), at

termination, we should have

$$||Z_n|| \cong ||A^{-1/2}|| \sim \theta^{-1/2}. \tag{3.31}$$

In his experiments, Powell observed that, when $||Z_0|| > \theta^{-1/2}$, good accuracy was achieved and the algorithm terminated in n iterations. This is because of scaling of $z^{(1)}$, according to equation (3.16) . However, when $||Z_0|| < \theta^{-1/2}$, more than n steps were often needed to reach (3.31). The number of iterations needed to achieve the same accuracy increased with the decrease of θ. The following example given by Powell illustrates this problem.

Assume a quadratic function $f(x) = \frac{1}{2}x^T Ax, x \in R^4$, with $x^* = 0$, where

$$A = \theta \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 4 \end{pmatrix}, \qquad Z_0 = \begin{pmatrix} 1 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 2 \end{pmatrix}, \quad \text{and} \quad x_0 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T.$$

Powell evaluated the accuracy of his algorithm by measuring the value

$$\eta = \max_{1 \le i \le n, 1 \le j \le n} \left| \; (Z^{*T} A Z^* - I)_{ij} \; \right|, \tag{3.32}$$

which should be close to zero at termination.

In the tests, good accuracy was observed for all $\theta \geq 1$, for which $||Z_0|| > ||A^{-1/2}||$. Even for an example with $\theta = 10^{20}$, with which the ordinary QN method had failed to terminate, Powell found that, after 4 iterations, good accuracy was achieved, with $||x|| = 3 \times 10^{-7}$ and $\eta = 4.2 \times 10^{-6}$. The reason for achieving good accuracy by Powell's method is that the norms of the columns of Z^* are made very small only by the first part of equation (3.16).

θ	Iterations	η
1	4	1.7×10^{-7}
0.1	5	3.6×10^{-7}
0.01	6	2.6×10^{-7}
10^{-4}	7	5.2×10^{-7}
10^{-6}	7	1.1×10^{-7}
10^{-8}	10	4.8×10^{-7}

Table 3.1: The relationship between θ and the number of iterations

But when $\theta < 1$, for which $||Z_0|| < ||A^{-1/2}||$,the accuracy observed deteriorated seriously. Powell pointed out that this was because an error of magnitude ε in x can induce an error of magnitude ε/θ in $x + \lambda d$ [13]; thus the error accumulated from iteration to iteration and became a considerable factor. Table 3.1 shows that the number of iterations required to achieve the same accuracy of $||x|| \leq 10^{-6}$ increases as θ decreases.

Therefore, Powell suggested that the norm of the initial Z_0 should be larger than $||A^{-1/2}|| \sim |\theta^{-1/2}|$, and recommended the following "rescaling" extension to magnify the columns of Z_i automatically at each iteration if necessary.

At each iteration, the **Rescaling Algorithm** includes the following two steps:

Algorithm 3.2

(1) **Set** σ

$$\sigma = ||z_i^{(1)}||, \qquad i = 1,$$

$$\sigma = \min[\sigma, ||z_i^{(1)}||], \qquad i = 2, 3, \cdots.$$

(2) **Scale**

calculate $z_i^{(j)}$ by equation (3.16), for $j = 2, 3, \cdots, n$

if $||z_i^{(j)}|| < \sigma$, rescale by setting $z_i^{(j)} = z_i^{(j)} \dfrac{\sigma}{||z_i^{(j)}||}$

Because this extension only alters the Euclidean lengths of the columns of Z, it preserves all the conjugacy properties for the quadratic function with ELS, and the normalized search directions remain the same. Therefore it does not affect the theoretical finite termination of the algorithm.

When the extended algorithm is applied to the above problems with $\theta < 1$, only 5 iterations are needed to reach termination, the value of η remains small, and there is no deterioration in efficiency as θ is decreased.

Powell also made some other tests. One test showed that the rescaling technique is useful to deal even with the case where the initial Z_0 is singular as the rescaling technique provides a way of using rounding errors to correct the singularity. Let

$$A = \theta \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 4 \end{pmatrix}, \qquad Z_0 = \begin{pmatrix} 0 & 1 & 4 & 9 \\ 1 & 0 & 1 & 4 \\ 4 & 1 & 0 & 1 \\ 9 & 4 & 1 & 0 \end{pmatrix}, \quad \text{and} \quad x_0 = e_1.$$

Table 3.2 shows the results of calculations with scaling and without scaling. We can see from the table that the algorithm terminates much faster with rescaling than without rescaling.

Another test shows that the rescaling technique is also useful to deal with the case where the initial Z_0 is ill-conditioned, but not singular, which is more likely in practice.

	Without Scaling		With Scaling	
θ	Iterations	η	Iterations	η
1	8	5.5×10^{-7}	5	3.2×10^{-7}
0.1	11	2.5×10^{-7}	5	2.7×10^{-7}
0.01	31	8.3×10^{-7}	6	4.8×10^{-7}
0.001	18	3.9×10^{-7}	5	2.4×10^{-7}
10^{-10}	58	4.4×10^{-7}	5	4.5×10^{-7}

Table 3.2: The results with a singular Z_0

Let Z_0 be a Hilbert matrix, so

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 4 \end{pmatrix}, \qquad (Z_0)_{ij} = 1/(i+j+\mu), \quad \text{and} \quad x_0 = e_1,$$

where μ is a parameter used to alter the condition number of Z_0.

The results shown in Table 3.3 again demonstrate that the rescaling technique is useful in this case.

These numerical results verified that, by using a rescaling technique, good accuracy could be obtained for a much wider range of initial choice of Z_0. It makes the procedure more stable than direct use of the product form of the BFGS formula. But the two major shortcomings for this method are that, first, the symmetry is lost when using ZZ^T to replace H, so it requires $O(n^2)$ elements of storage, which is about twice as much as most quasi-Newton algorithms; and second, compared with the quasi-Newton method,

	Without Scaling		With Scaling	
μ	Iterations	η	Iterations	η
0	8	1.2×10^{-7}	4	6.9×10^{-7}
1	7	7.2×10^{-7}	4	1.1×10^{-7}
2	9	6.6×10^{-7}	5	2.2×10^{-7}
5	11	5.1×10^{-7}	5	5.0×10^{-7}
10	46	1.2×10^{-7}	5	4.8×10^{-7}
10^6	-	-	5	3.6×10^{-7}

Table 3.3: The results with Z_0 being a Hilbert matrix

Powell's method needs more computations to postmultiply by an orthogonal matrix Ω and to calculate $\hat{s} = -Z^T g$ for updating Z^* at each iteration.

Chapter 4

The VS-ZZ^T Method

This chapter describes the major work of our thesis. We have developed an algorithm which combines Powell's method and the VSCG method, in order to maintain the properties of Powell's method, but with the memory requirement adjustable according to the available memory space. We will give the details of our algorithm as well as the update scheme.

4.1 The Objective

In the last chapter, we mentioned that Powell's method has the advantages of maintaining good accuracy in the presence of serious roundoff errors, and preserving normalized and mutually conjugate directions. But just like many other algorithms, it achieves these good features at the expense of memory space.

The objective of our thesis is to use a variable storage strategy to overcome the weakness of Powell's method for requiring $O(n^2)$ memory space. At the same time, we hope our

algorithm still preserves conjugate directions at each iteration and maintains the accuracy properties of Powell's method.

The critical point of our method lies in updating Z and computing the search direction d at each iteration without storing the whole matrix Z.

From the basic update formula (3.16) of Powell's method, we know that the update of $z^{(j)}$ can be done column by column, and it does not include any matrix operations. Furthermore, since the commonly used initial matrix Z_0 is an identity matrix I, we don't have to store Z_0 explicitly. Thus, if we know s and y, as well as some other necessary information for updating and transformation, we should be able to update Z column by column, without storing the whole matrix.

Also, from (1.3), the computation of the search direction d can be expressed as

$$d = -ZZ^T g = -\sum_{j=1}^{n} z^{(j)} z^{(j)^T} g \equiv \sum_{j=1}^{n} \hat{s}^{(j)} z^{(j)}, \tag{4.1}$$

where the scalar $\hat{s}^{(j)} = -z^{(j)^T} g$. With this formula, we can obtain d without knowing the whole matrix, so long as we can get $z^{(j)}$ column by column. Once a column has been used, it can be discarded to leave the space for the next column.

However, in order to update the columns of Z and compute the search direction d at the next iteration, we still need to reconstruct the current Z column by column from the very first iteration. This is done by an update scheme called "Column-by-column and Up-row Update Scheme", with a set of minimal information saved at each iteration, consisting of vectors s, y, $\hat{s}$, h as well as $||z_i^{(j)}||$ for $j = 1, \cdots, n$ and scalars ϕ, k and η.

In order to achieve our goal of using variable memory storage, we apply the VSCG strategy to our algorithm: in the first $m + 1$ iterations, we use Powell's factored BFGS

quasi-Newton formula to update Z; from the $(m+2)^{th}$ iteration, we switch to the preconditioned conjugate gradient method using the constructed preconditioner $H_m = Z_m Z_m^T$. Usually, m is provided by the user and depends on the available memory space. In practice, the value of m is usually chosen around 3 to 5. With $m \ll n$, the memory requirement for our algorithm would be much less than the n^2 required by Powell's method.

Since the initial matrix Z_0 used in our algorithm is always the identity matrix, and we apply Powell's rescaling technique in the first m iterations, our algorithm can avoid the problem caused by accumulated roundoff errors. In the next chapter, we will give some examples to show this advantage.

4.2 The Transformation and Update Algorithm

In Powell's method, the update of Z^* from Z needs two separate steps: Transformation and Updating. That is,

$$Z \xrightarrow{T} \bar{Z} \xrightarrow{U} Z^*.$$

In our new algorithm, we combine these two steps into one formula which is described as follows.

According to Goldfarb's **Transformation Algorithm 3.1**, we can iteratively compute

$$\bar{z}^{(j)} = \sqrt{\frac{\phi}{\hat{s}^{(j-1)^2} + \phi}}\Big(-z^{(j-1)} + \frac{\hat{s}^{(j-1)}}{\phi}h\Big), \qquad j = n, \cdots, 2. \tag{4.2}$$

Substituting $\bar{z}^{(j)}$ into Powell's factored form of the update equation (3.16) and defining a new variable $U = (I - \frac{sy^T}{s^T y})$, for $j = n, \cdots, 2$ we compute:

$$
\begin{aligned}
z^{(j)^*} &= \bar{z}^{(j)} - \frac{y^T \bar{z}^{(j)}}{s^T y} s = (I - \frac{sy^T}{s^T y})\bar{z}^{(j)} \\
&\equiv U\bar{z}^{(j)} = U\sqrt{\frac{\phi}{\hat{s}^{(j-1)^2} + \phi}}\Big(-z^{(j-1)} + \frac{\hat{s}^{(j-1)}}{\phi} h\Big) \\
&= \sqrt{\frac{\phi}{\hat{s}^{(j-1)^2} + \phi}}\Big(-Uz^{(j-1)} + \frac{\hat{s}^{(j-1)}}{\phi} Uh\Big). \qquad (4.3)
\end{aligned}
$$

This equation allows us to update the columns of Z directly, instead of using two separate steps. In addition, the multiplication of the matrix U and a vector v does not require any matrix operations, because

$$Uv = (I - \frac{sy^T}{s^T y})v = v - \frac{y^T v}{s^T y} s \equiv v - as, \qquad (4.4)$$

where a is a scalar.

Thus with Goldfarb's **Transformation Algorithm 3.1** and Powell's updating formula, we devised our new transformation and updating algorithm to calculate the last $(n-1)$ columns of Z in one step. Comparing (4.3) with Algorithm 3.1, we see that the only difference is that U appears in (4.3). In the following explanation, we show that U in (4.3) doesn't interfere with the update of Z.

From (3.21) to (3.25), we know that h in (4.3) can be expressed as

$$h = \sum_{i=j}^{k} \hat{s}^{(i)} z^{(i)},$$

where j denotes the j^{th} column to be transformed and k is the greatest integer in $(1, \cdots, n)$ such that $\hat{s}^{(k)} \neq 0$. Let $h' = Uh$. Since U is a linear operator, h' can be expressed as

$$h' \equiv Uh = U\sum_{i=j}^{k} \hat{s}^{(i)} z^{(i)} = \sum_{i=j}^{k} \hat{s}^{(i)} U z^{(i)}. \qquad (4.5)$$

If we use h' to replace Uh in (4.3) and each time we update h' by adding $\hat{s}^{(i)}Uz^{(i)}$ to it, the result is the same as multiplying U into (4.2) to obtain $z^{(j)*}$. So, we use the following equation in our algorithm to do the transformation and update in one step,

$$z^{(j)*} = \sqrt{\frac{\phi}{\hat{s}^{(j-1)^2} + \phi}}\left(-Uz^{(j-1)} + \frac{\hat{s}^{(j-1)}}{\phi}h'\right). \tag{4.6}$$

Note that the product $Uz^{(j-1)}$ in (4.6) is also used to update h' directly.

The following is our modified **Transformation and Update Algorithm**:

Algorithm 4.1

step 0: Set k to be the greatest integer in $(1, \cdots, n)$ such that $\hat{s}^{(k)} \neq 0$.

Set an auxiliary vector $h' = \hat{s}^{(k)}Uz^{(k)}$ and a scalar $\phi = \hat{s}^{(k)^2}$.

If $k < n$, set $z^{(j)} = Uz^{(j)}$ for $j = k+1, k+2, \cdots, n$;

step 1: Terminate if $k = 1$;

step 2: Calculate the vector $z^{(k)} = \sqrt{\frac{\phi}{\hat{s}^{(k-1)^2} + \phi}}\left(-Uz^{(k-1)} + \frac{\hat{s}^{(k-1)}}{\phi}h'\right)$.

Then add $\hat{s}^{(k-1)}Uz^{(k-1)}$ to h' and add $\hat{s}^{(k-1)^2}$ to ϕ;

step 3: Let $k = k - 1$ and go back to step 1.

4.3 The Column-by-column and Up-row Updating Scheme

Since we don't save the matrix Z in our algorithm, when we conduct the transformation and the update, we have to do it column by column. In **Algorithm 4.1**, when one column of a matrix Z_l is updated and used, it is no longer needed, so $z_l^{(j-1)}$ may be overwritten by $z_{l+1}^{(j)}$. However, in the next iteration, we still need to reproduce each column of Z_l, because the update of Z_{l+1} depends on $Z_l, \cdots, Z_0$. So, at each iteration, we have to reproduce the

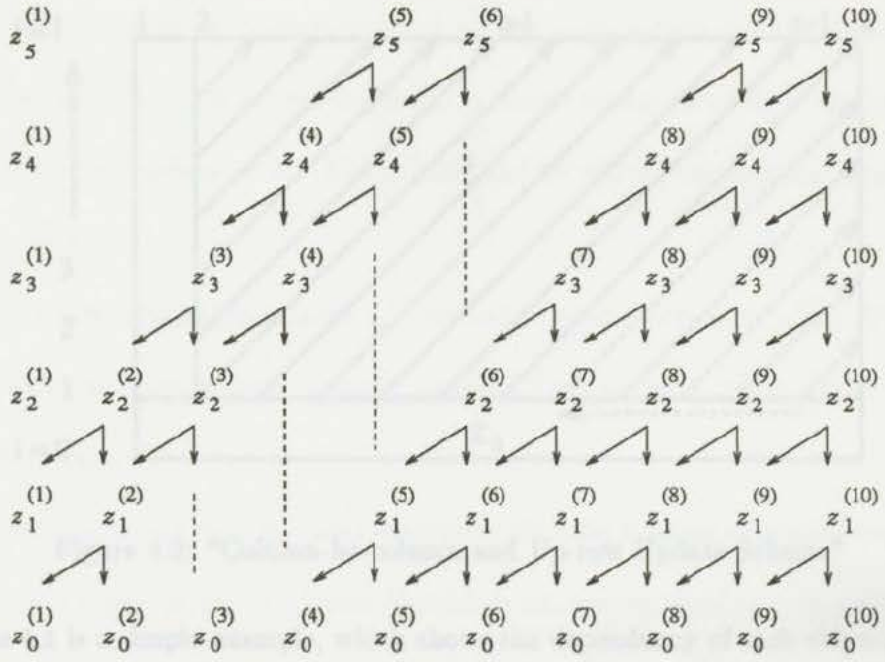

Figure 4.1: A simple example of the updating procedure

intermediate columns of each Z_i from the very beginning.

In order to avoid unnecessary repetition in reproducing $Z_1, \cdots, Z_l$, we need to save some basic information at each iteration. In the algorithm described below, for each iteration $i = 1, \cdots, l$, we need to save vectors s_i, y_i and their dot product $\eta_i = s_i^T y_i$ for updating; we need to save $\hat{s}_i$ and k_i for the transformation, where $k_i < n$ is the greatest integer at each iteration such that $\hat{s}_i^{(k_i)} \neq 0$; and we need to save the norm of each $z_i^{(j)}, j = 1, \cdots, n$ for scaling. In addition, we need two intermediate variables to record h_i' and ϕ_i for the transformation. In total, 5 vectors and 3 scalars are needed at each iteration for reproduction of the matrix Z_l.

There could be a lot of repeated computation in the reconstruction of Z_j; therefore, it is very important to use a proper scheme to make it efficient.

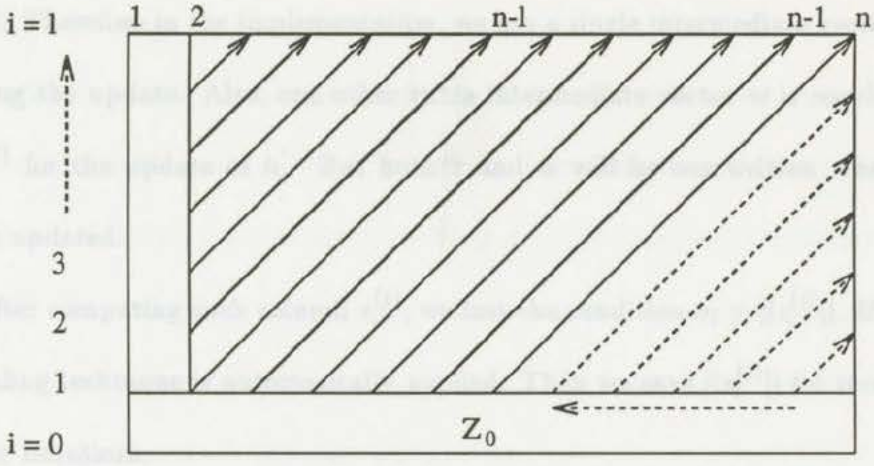

Figure 4.2: "Column-by-column and Up-row Update Scheme"

Figure 4.1 is a simple example, which shows the dependency of each column $z_i^{(j)}$ with respect to the columns of the previous matrix Z_{i-1}. For simplicity, Figure 4.1 only shows the relationship between $z_i^{(j)}$ and its two previous columns $z_{i-1}^{(j)}$ and $z_{i-1}^{(j-1)}$. But in fact, the column $z_i^{(j)}$ $j = 2, 3, \cdots, n$, depends directly on $z_{i-1}^{(j-1)}$ and indirectly on $z_{i-1}^{(j)}, \cdots, z_{i-1}^{(k_i)}$. However, these indirect dependencies are "contained" in h_i' according to (4.5).

Figure 4.2 shows the "Column-by-column and Up-row Update Scheme", which we devised for our algorithm. The dashed arrow lines in the figure represent the order of the columns to be calculated for computing $z_l^{(n)}$, which begins from the right most column.

Some notes should be given here with respect to this update scheme:

(1) The initial matrix Z_0 is an identity matrix, so we don't need to save it explicitly.

(2) The first column $z_i^{(1)}, i = 1, 2, \cdots, l$ is calculated directly from s_i and y_i according to equation (3.16), instead of depending on the previous iteration.

(3) Once $z_{i-1}^{(j)}$ has been used for updating $z_i^{(j)}$, it is no longer needed and can be

discarded. Therefore in the implementation, we use a single intermediate vector z to hold $z_{i-1}^{(j)}$ during the update. Also, one other extra intermediate vector w is required to hold $U_{i-1}z_{i-1}^{(j-1)}$ for the update of h_i'. But both z and w will be overwritten when the next column is updated.

(4) After computing each column $z_l^{(j)}$, we test the condition $\sigma_l > ||z_l^{(j)}||$. If true, Powell's rescaling technique is automatically applied. Then we save $||z_l^{(j)}||$ for reconstructing Z_l at later iterations.

(5) From Figure 4.2, we can easily determine that the number of columns to be reconstructed at iteration l is $l(n-1)$. From Algorithm 4.1, we can show that $6n$ multiplications are required to calculate the transformation and update for each column $z_i^{(j)}, j = 2, \cdots, n$. So, the total number of multiplications required is $6ln(n-1) + O(n)$ for computing Z_l.

(6) The search direction d_{l+1} is calculated by (4.1). After each $z_l^{(j)}$ is formed, we add $\hat{s}_{l+1}^{(j)} z_l^{(j)}$ to d_{l+1}, where the scalar $\hat{s}_{l+1}^{(j)} = -z_l^{(j)^T} g_l$. Thus, when all $z_l^{(j)}, j = 1, \cdots, n$ have been calculated, d_{l+1} is done at the same time.

4.4 The VS-ZZ^T Algorithm

Using Algorithm 4.1 and the "Column-by-column and Up-row Updating Scheme", we are able to use Powell's method without storing the whole matrix Z. What we need to save are only 5 vectors and 3 scalars at each iteration. With this information, we can reproduce the columns of $Z_l, l = 1, 2, \cdots$, when necessary. But if we have to iterate many times, say more than $\frac{n}{5}$, in the computation, we actually use the same amount of memory space as Powell's, and we have a larger computational burden. In such a case, this idea would not

be attractive.

Our strategy to solve this problem is to combine Powell's method and the VSCG method to use variable memory. That is, the algorithm runs in two parts, a QN-part and a preconditioned CG-part. In the first $m+1$ iterations, it runs Powell's column by column factored QN update method; from the $(m+2)^{th}$ iteration, it runs the preconditioned CG algorithm with a fixed preconditioner $H_m = Z_m Z_m^T$. However, if the memory is sufficient, we can let $m \geq n$ to run the QN-part all the time.

Note here, in the preconditioned CG-part from $i = m+2$, we use the sum form of BFGS formula (2.14) to calculate the search direction d_{i+1}, but with a fixed preconditioner, i.e.

$$d_{i+1} = -H_i g_i = -[H_m - \frac{s_i y_i^T H_m + H_m y_i s_i^T}{s_i^T y_i} + (1 + \frac{y_i^T H_m y_i}{s_i^T y_i})\frac{s_i s_i^T}{s_i^T y_i}]g_i, \qquad (4.7)$$

where $H_m = Z_m Z_m^T$.

For $i \leq m$, $5n+3$ storage locations are required at each iteration for the information of transformation, update and scaling, and for $i > m$ another $5n+3$ locations are used to record some intermediate information, so the total storage requirement of the **VS-ZZ^T Algorithm** is $(m+1)(5n+3)+2n$, where m is given by the user; the $2n$ is for auxiliary vectors w and z. In practice, $m \ll n$, so it is only a moderate multiple of n, much smaller than the n^2 originally required by Powell's method.

Note here, compared to Algorithm 2.4, the following Algorithm 4.2 takes one more step in the QN-part. But in fact the last $5n+3$ memory locations used in the QN-part are only for some intermediate updating information, which is equivalent to the VSCG algorithm that uses the last $2n+2$ locations for intermediate information in the CG-part. We do this because the VS-ZZ^T algorithm shifts its updating formula from factored form

to sum form, while the VSCG algorithm uses the sum form all the time.

In the following **VS-ZZ^T Algorithm**, we use $\hat{U}(z_{i-1}^{(j)}, i)$ to denote updating $z_{i-1}^{(j)}$ using the Transformation and Update Algorithm 4.1 and "Column-by-column and Up-row Updating Scheme". In the CG-part, two intermediate vectors hg and hy are used for representing $H_m g_i$ and $H_m y_i$ in (4.7) and two scalars $t1$ and $t2$ are used for $z_m^{(j)^T} g_i$ and $z_m^{(j)^T} y_i$.

Algorithm 4.2

Choosing an initial Z_0 and starting from the initial point x_0, the initial descent search direction is $d_1 = -Z_0 Z_0^T g_0$. Then:

QN-part: From the point x_1, iterate with $i = 1, 2, \cdots, m+1$:

save $s_i, y_i, \hat{s}_i, \eta_i$ and k_i for updating and the transformation;

set $d_{i+1} = 0$;

iterate for $j = n, n-1, \cdots, 1$ to get d_{i+1}:

$z_i^{(j)} = \hat{U}(z_{i-1}^{(j)}, i)$;

if $i \leq m$, save $||z_i^{(j)}||$ and do rescaling, if necessary;

$\hat{s}_{i+1} = -z_i^{(j)^T} g_i$;

$d_{i+1} = d_{i+1} + \hat{s}_{i+1}^{(j)} z_i^{(j)}$;

$x_{i+1} = x_i + \lambda_i d_{i+1}$;

CG-part: From the point x_{m+2} reached by the QN-part, use $H_m = Z_m Z_m^T$,

as preconditioner and iterate for $i = m+2, m+3, \cdots$:

replace s_{m+1}, y_{m+1} and η_{m+1} by s_i, y_i and η_i;

set $hg = 0; \quad hy = 0;$

iterate for $j = n, n-1, \cdots, 1$ to get $H_m g_i$ and $H_m y_i$:

$z_m^{(j)} = \hat{U}(z_{m-1}^{(j)}, m);$

$t1 = z_m^{(j)^T} g_i; \quad hg = hg + t1 \times z_m^{(j)};$

$t2 = z_m^{(j)^T} y_i; \quad hy = hy + t2 \times z_m^{(j)};$

$d_{i+1} = -H_i g_i$ using equation (4.7);

$x_{i+1} = x_i + \lambda_i d_{i+1}.$

As we can see, in the QN-part, we use Powell's factored form of BFGS formula, and in the CG-part, we use the sum form of BFGS formula. In the CG-part, the computation of vectors $H_m g_i$ and $H_m y_i$ can be done by using the same procedure which is used to construct $z_i^{(j)}, j = n, n-1, \cdots, 1$ and calculate the sum of $z_i^{(j)} z_i^{(j)^T} v$, where $v = g_i$ for the QN-part and $v = g_i$ or $v = y_i$ for the CG-part but with a fixed $z_m^{(j)}$. In some sense, the VS-ZZ^T Algorithm is equivalent to the VSCG method, that is, the QN-part can be viewed as constructing an appropriate preconditioner $H_m = Z_m Z_m^T$, and the CG-part is the implementation of a preconditioned conjugate gradient algorithm which we described in Chapter 2. So, for the quadratic function with exact line search, the VS-ZZ^T Algorithm can keep descent directions by ensuring that each $H_i = Z_i Z_i^T$ is positive definite, provided that $s_i^T y_i > 0$, and finite termination is expected in n steps, which follows from a theorem given in Buckley [3].

In our algorithm implementation, we also studied the possibility of using a factored

form of BFGS formula to update each column of Z_i in the CG-part. Although in theory the justification is still incomplete, the results we offer show it does indeed work. We put these results in our Appendix B. An interested reader can access it for comparison.

Chapter 5

Numerical Experiments and Conclusions

In the previous chapter, we discussed the main idea of our VS-ZZ^T algorithm and its details. In this chapter, we present the results of our numerical experiments to verify that we have met our main objective as desired, that is, with variable memory space, we can obtain equivalent results to Powell's method. At the end of this chapter, we give some concluding remarks on our thesis work.

5.1 Purpose of Experiments and Methodology

As we mentioned in the previous chapter, the VS-ZZ^T algorithm is developed based on Powell's algorithm. The main difference between the two is that the VS-ZZ^T algorithm allows us to use variable storage space. Therefore, in our experiments, we want to in-

vestigate how different memory allocations affect the termination and the accuracy of the VS-ZZ^T algorithm. The parameter m in the following experiments specifies how much memory space is used in the computation, as we have explained in Section 4.4. For $m = 0$, we take Z_0 as the preconditioner to calculate the search direction d in the CG-part. For $m > 0$, $(m+1)(5n+3)+2n$ memory locations are used in the computation. The larger the value of m, the more iterations are run in the QN-part, and the more memory space that is used. For $m = -1$, we use the normal quasi-Newton method for comparison.

Another important thing to observe is how the relative value θ between $||Z_0||$ and $||A^{-1/2}||$ affects the convergence rate and the accuracy. As in (3.28), we let A be the product of θ and a matrix of order 1. According to the explanation of Powell's method in Chapter 3, the roundoff error in x is inversely proportional to θ, and the termination is also related to θ, even if Z_0 is well conditioned. Thus by adjusting θ, we can change the relative values of $||Z_0||$ and $||A^{-1/2}||$ to investigate how this affects the algorithm termination. We then conduct tests, with and without scaling, to study how Powell's rescaling technique works in our algorithm.

To obtain the results of Powell's algorithm for comparison, we run our program by letting m be large enough such that the algorithm terminates while still in the QN-part. Since this part of the algorithm is in fact the same as Powell's algorithm, we should get the same results as directly running Powell's algorithm. In the following experiments, we get Powell's results in this way and we compare the results of these two methods by their iteration steps, as well as the accuracy of $||x||$, when the algorithm terminated. According to Algorithm 4.2, by adjusting m, if a group of tests terminate still in the QN-part with k

steps, those results from $m \geq k-1$ are all the same as Powell's, as we use $m+1$ steps in the QN-part. For simplicity, we only show the results with $m \leq k$, because the algorithm should still terminate in k steps even if we increase m.

In all experiments, the objective function to minimize has the following form:

$$f(x) = \frac{1}{2} x^T A x, \qquad x \in \Re^n.$$

All the experiments are conducted using an Exact Line Search (ELS). The termination criterion is chosen as $||x|| \leq ACC$, as Powell did. We use $ACC = 10^{-10}$ and double precision in the computations. In the following numerical results, $(*)$ denotes that the algorithm generates a non-descent direction and terminates prematurely and $(**)$ denotes the function evaluations exceeded the limitation.

5.1.1 Experiments with $Z_0 = I$

Our first experiment investigates cases with the identity matrix as the initial matrix Z_0. This is the default initial matrix of the VS-ZZ^T algorithm, so that we do not have to save it explicitly. Furthermore, in practice, an identity matrix is often used as the initial matrix in quasi-Newton algorithms.

	m =	step=	‖g‖ =	‖x‖ =	f =
QN method	-1	11	.85E-24	1.086D-25	3.025D-50
	10	10	.24E-15	3.020D-17	3.517D-33
	9	10	.24E-15	3.020D-17	3.517D-33
	8	10	.24E-15	3.020D-17	3.517D-33
with or	7	10	.24E-15	3.010D-17	3.512D-33
without	6	10	.24E-15	3.323D-17	3.636D-33
scaling	5	10	.23E-15	5.203D-17	3.530D-33
	4	10	.24E-15	3.724D-17	3.847D-33
	3	10	.28E-15	3.290D-17	4.157D-33
	2	10	.19E-15	1.532D-16	5.005D-33
	1	10	.10E-15	1.603D-16	4.994D-33
	0	10	.55E-11	3.560D-13	9.702D-25

Table 5.1: $Z_0 = I, \theta = 1$

For comparison with Powell's method, we choose the following function as our example:

$$A = \theta \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 6 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 7 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 10 \end{pmatrix},$$

$$Z_0 = I, \quad \text{and} \quad x_0 = e_1.$$

Table 5.1 shows that, with $\theta = 1$ and $m = 0, \cdots, 10$, tests terminate in 10 steps as expected. The normal QN method ($m = -1$) takes 11 steps to reach the minimum, which is acceptable. In this case, because Z_0 is an identity matrix and $\theta = 1$, scaling has no effect on the results. Notice in Table 5.1, from $m \geq 9$ the algorithm terminates while still

	m =	step=	‖g‖ =	‖x‖ =	f =
QN method	-1	11	.36E-26	3.598D-25	4.429D-52
with scaling	11	11	.30E-24	2.020D-23	2.967D-48
	10	11	.30E-24	2.020D-23	2.967D-48
	9	11	.49E-25	3.290D-24	7.861D-50
	8	11	.19E-18	6.866D-17	5.317D-36
	7	11	.19E-17	4.104D-16	3.339D-34
	6	11	.90E-17	8.454D-15	2.198D-32
	5	11	.33E-16	2.517D-14	1.858D-31
	4	11	.79E-16	1.898D-13	5.113D-30
	3	11	.25E-15	2.874D-13	1.429D-29
	2	11	.15E-15	2.444D-13	1.076D-29
	1	10	.27E-13	1.980D-12	2.344D-26
	0	22	.13E-12	6.629D-11	1.874D-24
without scaling	10	16	.56E-15	1.457D-13	3.361D-29
	9	17	.34E-15	8.984D-14	1.246D-29
	8	17	.35E-13	1.828D-11	1.548D-25
	7	19	.12E-14	1.174D-12	4.084D-28
	6	20	.25E-13	6.296D-12	4.720D-26
	5	23	.84E-14	9.297D-12	1.767D-26
	4	23	.86E-14	1.277D-11	2.320D-26
	3	31	.86E-13	9.235D-11	1.680D-24
	2	39	.32E-12	6.963D-11	9.467D-24
	1	27	.40E-13	6.905D-11	6.565D-25
	0	22	.13E-12	6.629D-11	1.874D-24

Table 5.2: $Z_0 = I, \theta = 10^{-3}$

	m =	step=	||g|| =	||x|| =	f =
QN method	-1	11	.17E-35	2.779D-25	1.364D-61
	11	11	.74E-33	5.029D-23	1.820D-56
	10	11	.74E-33	5.029D-23	1.820D-56
	9	11	.12E-33	8.331D-24	4.980D-58
	8	11	.46E-27	1.489D-16	3.101D-44
with	7	11	.81E-27	2.780D-16	6.943D-44
scaling	6	11	.53E-26	3.598D-15	4.787D-42
	5	11	.19E-25	4.783D-15	3.721D-41
	4	11	.34E-25	5.648D-14	5.590D-40
	3	11	.35E-24	6.041D-13	5.238D-38
	2	11	.18E-24	1.686D-13	6.127D-39
	1	10	.35E-22	2.428D-12	3.980D-35
	0*	4	-	8.791D-01	-

Table 5.3: $Z_0 = I, \theta = 10^{-12}$

in the QN-part, so we can take these results as Powell's.

Now we let $\theta = 0.001$ and conduct the computation with and without scaling. The advantage of applying scaling is shown in Table 5.2, in which the algorithm terminates properly in about 11 steps when scaling is applied. The case $m = 0$ takes 22 steps to reach the minimum; this is because the algorithm uses $H_0 = Z_0 Z_0^T$ as the preconditioner in the CG-part and there is no scaling in effect. If we update Z_i without scaling, more steps are required to satisfy the the same termination criterion, where $m = 2$ has the highest iteration count of 39.

Now we decrease θ again to 10^{-12}. In this situation the difference between $||Z_0||$ and $||A^{-1/2}||$ become very large and the algorithm can't work at all without scaling. However, except for $m = 0$, by using the rescaling technique we can still get satisfactory results compared with Powell's which are shown in Table 5.3 from $m \geq 10$.

From the above results, as well as other experiments we have conducted, the following conclusions can be made for the cases with $Z_0 = I$:

- With scaling, our algorithm can terminate normally when $m > 0$, even if θ is very small.
- However, for $m = 0$, as it uses the initial matrix $H_0 = Z_0^T Z_0$ as the preconditioner in the CG-part and there is no scaling in effect, the effect of accumulated errors is great, especially in the case of $||Z_0|| \ll ||A^{-1/2}||$.
- For those tests without scaling, more steps are needed to satisfy the same termination criterion than that with scaling.
- By using a small value of m ($m \neq 0$), we can still achieve satisfactory results. Increasing m (using a larger memory space) has little impact on the results. That is what we expected: using relatively small memory space, we can get equivalent results to Powell's method.

5.1.2 Experiments with $Z_0 \neq I$

To make a direct comparison between Powell's results and the VS-ZZ^T results, we tested some examples with an ill-conditioned initial matrix Z_0. This was carried out by using an option in our program to store the whole matrix Z_0 for updating.

The following three tests are given by Powell, as shown in Chapter 3. Although in practice the identity matrix is popularly accepted as the initial matrix and these examples are created artificially, only for the purpose of experimenting, we intend to use these

examples for comparison.

Example (1)

$$A = \theta \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 4 \end{pmatrix}, \qquad Z_0 = \begin{pmatrix} 0 & 1 & 4 & 9 \\ 1 & 0 & 1 & 4 \\ 4 & 1 & 0 & 1 \\ 9 & 4 & 1 & 0 \end{pmatrix}, \quad \text{and} \quad x_0 = e_1.$$

Note, in Table 5.4 we only show the results for $k \geq m \geq 3$, because for $m < 3$ the algorithm generates a non-descent direction and terminates prematurely, where k denotes the steps required by Powell's algorithm. One explanation is that the initial matrix Z_0 is singular and too few updating steps are not enough to let Z_m become a nonsingular matrix. Also, after m steps, there is no scaling in effect, and the preconditioned CG algorithm cannot generate descent directions with such a singular preconditioner $H_m = Z_m Z_m^T$. In the tests, we found that the algorithm does not work well if we choose no scaling option. However, with scaling, the results are not affected very much by decreasing θ.

Our Example (2), shown by Table 5.5, presents the results with the same matrix A and x_0, but using a Hilbert matrix as Z_0, i.e.

$$(Z_0)_{ij} = 1/(i + j + \mu),$$

where μ is used to adjust the value of condition number of Z_0. The results are similar to the first example; that is, the algorithm doesn't work well if m is very small ($m \leq 2$ in this case) or if no scaling is applied. But for the tests with scaling, the results are rather consistent even if the value of μ is changed.

	$\theta =$	m =	step=	$\|\|g\|\| =$	$\|\|x\|\| =$	f =
with scaling	1	7	7	.31E-10	1.720D-11	2.024D-22
		6	7	.31E-10	1.720D-11	2.024D-22
		5	7	.31E-10	1.720D-11	2.024D-22
		4	7	.74E-11	1.065D-11	2.780D-23
		3*	6	-	1.017D-04	-
	10^{-1}	6	6	.39E-11	3.479D-11	3.736D-23
		5	6	.39E-11	3.479D-11	3.736D-23
		4	6	.39E-11	3.479D-11	3.736D-23
		3*	6	-	3.823D-10	-
	10^{-2}	5	5	.94E-14	7.120D-13	2.422D-27
		4	5	.94E-14	7.120D-13	2.422D-27
		3	5	.94E-14	7.120D-13	2.422D-27
	10^{-3}	6	6	.60E-17	5.231D-15	8.998D-33
		5	6	.60E-17	5.231D-15	8.998D-33
		4	6	.60E-17	5.231D-15	8.998D-33
		3	5	.28E-15	4.504D-13	4.018D-29
	10^{-4}	7	7	.12E-15	4.023D-13	2.208D-29
		6	7	.12E-15	4.023D-13	2.208D-29
		5	7	.12E-15	4.023D-13	2.208D-29
		4	6	.64E-14	2.071D-11	5.858D-26
		3*	6	-	3.629D-07	-
	10^{-6}	7	7	.82E-20	3.157D-15	1.137D-35
		6	7	.82E-20	3.157D-15	1.137D-35
		5	7	.82E-20	3.157D-15	1.137D-35
		4	6	.59E-18	6.274D-13	1.233D-31
		3*	6	-	2.610D-10	-

Table 5.4: $Z_0 \neq I$, Example (1)

	$\mu =$	m =	step=	$\|\|g\|\| =$	$\|\|x\|\| =$	f =
with scaling	0	4	4	.59E-12	1.168D-13	3.121D-26
		3	4	.59E-12	1.168D-13	3.121D-26
		2*	5	-	4.584D-07	-
		1*	5	-	5.688D-02	-
	1	4	4	.22E-11	4.033D-13	4.192D-25
		3	4	.22E-11	4.033D-13	4.192D-25
		2*	5	-	8.011D-06	-
		1**	25	-	2.742D-05	-
	2	4	4	.89E-11	1.674D-12	7.061D-24
		3	4	.89E-11	1.674D-12	7.061D-24
		2*	5	-	4.768D-04	-
		1**	25	-	1.663D-08	-
	5	4	4	.12E-09	2.179D-11	1.236D-21
		3	4	.12E-09	2.179D-11	1.236D-21
		2*	5	-	1.078D-04	-
		1*	4	-	1.797D-01	-
	10	5	5	.16E-20	9.927D-22	6.356D-43
		4	5	.16E-20	9.927D-22	6.356D-43
		3	5	.45E-25	8.573D-26	1.336D-51
		2*	5	-	4.085D-02	-
		1*	4	-	1.626D-01	-

Table 5.5: $Z_0 \neq I$, Example (2)

	θ =	m =	step=	$\|\|g\|\|$ =	$\|\|x\|\|$ =	f =
with scaling	10^{-3}	4	4	.10E-14	2.040D-13	9.648D-29
		3	4	.10E-14	2.040D-13	9.648D-29
		2	4	.10E-14	2.040D-13	9.648D-29
		1	4	.10E-14	2.693D-13	9.661D-29
		0**	25	-	3.610D-04	-
	10^{-6}	4	4	.95E-18	2.283D-13	8.974D-32
		3	4	.95E-18	2.283D-13	8.974D-32
		2	4	.94E-18	2.546D-13	8.788D-32
		1	4	.97E-18	2.265D-13	9.183D-32
		0*	3	-	3.400D-01	-
	10^{-12}	4	4	.51E-24	1.528D-13	2.576D-38
		3	4	.51E-24	1.528D-13	2.576D-38
		2	4	.51E-24	1.528D-13	2.576D-38
		1	4	.47E-24	1.769D-13	2.323D-38
		0*	3	-	4.029D-01	-
without scaling	10^{-3}	4	6	.20E-16	1.681D-14	1.313D-31
		3	7	.13E-12	6.239D-11	3.829D-24
		2**	25	-	1.267D-05	-
		1**	25	-	1.456D-04	-
		0**	25	-	3.610D-04	-
	10^{-6}	4	6	.76E-18	1.505D-12	4.169D-31
		3*	8	-	1.620D-03	-
		2*	14	-	5.638D-05	-
		1*	5	-	3.026D-01	-
		0*	3	-	3.400D-01	-

Table 5.6: $Z_0 \neq I$, Example (3)

Example (3) is with the same matrix A, but Z_0 and x_0 are as chosen

$$Z_0 = \begin{pmatrix} 1 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad x_0 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T.$$

The results in Table 5.6 show that with scaling and $m > 0$, the algorithm works very well for different θ. However, the tests without scaling need more steps to satisfy the same termination criterion, but these results are much better than those of the previous two

examples.

Some points can be made based on these three tests.

- By using the VS-ZZ^T algorithm, we can get equivalent results to those of Powell's for $Z_0 \neq I$. Through the tests we can see that scaling can overcome the effect of accumulation of errors caused by poor initial scaling.
- When Z_0 is a singular or ill-conditioned matrix, too few update steps (that is, if m is too small) affect the performance of our algorithm. Because in these cases, matrix Z_m may still be singular or ill-conditioned, the CG method cannot generate the descent search direction for terminating properly.

5.1.3 Transforming $f(x)$

In the previous section, we compared our results with those of Powell's by conducting the same tests with the whole matrix Z_0 saved, where $Z_0 \neq I$. However, this is inconsistent with our algorithm because our algorithm is designed not to save the whole matrix explicitly.

It is instructive to consider the effect of a transformation of variables, so that the transformed function can be minimized by using an identity matrix as the initial matrix.

Considering a change of variable $\hat{x} = Tx$ and defining $\hat{f}(\hat{x}) = f(T^{-1}\hat{x})$, by using the chain rule we have

$$\nabla^2_{\hat{x}} \hat{f}(\hat{x}) = T^{-T} \nabla^2_x f(x) T^{-1}. \tag{5.1}$$

Because $\nabla_x f(x) = Ax$ and $\nabla_x^2 f(x) = A$, we get

$$\hat{A} = \nabla_{\hat{x}}^2 f(\hat{x}) = T^{-T} A T^{-1}. \tag{5.2}$$

In Chapters 2 and 3 we have indicated that the matrix H_i in a quasi-Newton algorithm is designed to approximate the inverse Hessian matrix A^{-1}. That is, starting from an initial matrix H_0, with each iteration, H_i is getting closer to A^{-1}. If we make a transformation as described above, the transformed $\hat{H}_0$ and $\hat{H}_i$ should have the form

$$\hat{H}_0^{-1} = T^{-T} H_0^{-1} T^{-1} \quad \text{and} \quad \hat{H}_i^{-1} = T^{-T} H_i^{-1} T^{-1}. \tag{5.3}$$

Since the purpose of the transformation is to make a problem with a nonidentity initial matrix applicable in our algorithm, that is to make $\hat{H}_0 = I$, the transformation matrix T^{-1} must satisfy the following condition

$$\hat{H}_0^{-1} = T^{-T} H_0^{-1} T^{-1} = T^{-T} Z_0^{-T} Z_0^{-1} T^{-1} = I. \tag{5.4}$$

This equation holds if

$$T = Z_0^{-1}. \tag{5.5}$$

Thus, (5.2) becomes

$$\hat{A} = Z_0^T A Z_0. \tag{5.6}$$

This equation tells us that minimizing the transformed function $\hat{f}(\hat{x})$ with $\hat{A}$ using initial $\hat{H}_0 = I$, is equivalent to using $H_0 = Z_0 Z_0^T$ to minimize the original $f(x)$. So, with this transformation method, we are able to use the VS-ZZ^T algorithm to indirectly solve problems with $H_0 \neq I$, but the limitation is that H_0 must be nonsingular.

In the first test, we choose the following function as a simple example of the transformation:

$$A = \theta \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 4 \end{pmatrix}, \quad Z_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}, \quad \text{and} \quad x_0 = e_1.$$

Making the following transformation of variables, we compute:

$$\hat{A} = Z_0^T A Z_0 = \theta \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 8 & 6 & 8 \\ 3 & 6 & 27 & 12 \\ 4 & 8 & 12 & 64 \end{pmatrix}, \quad \hat{Z}_0 = I, \quad \text{and} \quad \hat{x}_0 = Z_0^{-1} x_0 = e_1.$$

With $\theta = 1$ and $\theta = 0.1$, Table 5.7 shows that the results are equivalent with and without transformation. However, for $\theta = 0.01$, the algorithm fails to terminate for the transformed problem, unless we change the termination criterion to a larger value. To investigate the reason, we found that the ratio of the greatest to the least eigenvalue of the Hessian matrix is enlarged by the transformation. For the above example we found that

the eigenvalues of A are 1.3922, 0.2961, 2.5077 and 5.8039;

the eigenvalues of $\hat{A}$ are 0.3403, 6.0330, 24.2689 and 69.3578.

The change of eigenvalues also affects the condition number of the matrix, because the condition number can be defined as

$$\text{cond(A)} = |\frac{\lambda_{\max}}{\lambda_{\min}}|. \tag{5.7}$$

In this example, the condition numbers of A and $\hat{A}$ are 19.6018 and 203.7971 respectively. This might affect the termination of the algorithm and attainable accuracy.

In order to obtain general results by using an arbitrary initial Z_0, we tested Example (3) of the previous subsection by using a transformation, as our second example of trans-

	θ =	m =	step=	‖g‖ =	‖x‖ =	f =
with transformation	1	-1	5	.36E-24	1.332D-26	1.336D-51
		4	4	.35E-14	1.231D-15	6.332D-31
		3	4	.35E-14	1.231D-15	6.332D-31
		2	4	.35E-14	1.231D-15	6.332D-31
		1	4	.48E-13	3.610D-15	4.916D-29
		0	4	.14E-11	2.054D-14	1.336D-26
	0.1	-1	5	.35E-25	9.139D-27	1.399D-52
		4	4	.27E-12	7.029D-12	8.518D-25
		3	4	.27E-12	7.029D-12	8.518D-25
		2	4	.27E-12	7.029D-12	8.518D-25
		1	4	.39E-13	1.020D-12	1.790D-26
		0	4	.42E-12	6.546D-12	7.476D-25
	0.01	-1	5	.44E-25	1.685D-25	2.441D-51
		4*	5	-	5.408D-10	-
		3*	5	-	5.408D-10	-
		2*	5	-	5.265D-10	-
		1*	5	-	1.970D-09	-
		0*	5	-	1.169D-08	-
without transformation	1	4	4	.25E-14	7.234D-15	7.907D-30
		3	4	.25E-14	7.234D-15	7.907D-30
		2	4	.25E-14	7.234D-15	7.907D-30
		1	4	.46E-14	6.280D-15	8.582D-30
		0	4	.28E-12	5.353D-14	7.004D-27
	0.1	4	4	.48E-13	1.530D-12	3.518D-26
		3	4	.48E-13	1.530D-12	3.518D-26
		2	4	.48E-13	1.530D-12	3.518D-26
		1	4	.15E-13	4.872D-13	3.549D-27
		0	4	.39E-12	8.463D-12	1.142D-24
	0.01	4	4	.11E-13	4.901D-13	2.247D-27
		3	4	.11E-13	4.901D-13	2.247D-27
		2	4	.11E-13	4.901D-13	2.247D-27
		1	4	.43E-14	1.154D-13	1.966D-28
		0	5	.66E-11	2.093D-09	6.549D-21

Table 5.7: An transformation example

	θ =	m =	step=	||g|| =	||x|| =	f =
with transformation	1	-1	6	.55E-19	1.128D-19	2.456D-39
		4	4	.88E-12	2.996D-12	9.054D-25
		3	4	.88E-12	2.996D-12	9.054D-25
		2	4	.88E-12	2.996D-12	9.054D-25
		1	4	.95E-12	2.022D-12	8.040D-25
		0	4	.61E-09	8.266D-12	1.147D-21
	0.1	-1	6	.47E-22	4.449D-22	3.792D-45
		4	5	.82E-22	1.273D-21	3.728D-44
		3	5	.10E-21	1.271D-21	3.727D-44
		2	10	.94E-12	9.171D-11	4.115D-23
		1*	5	-	4.531D-09	-
		0	11	.71E-12	5.472D-11	1.583D-23
without transformation	1	4	4	.61E-13	2.159D-14	3.817D-28
		3	4	.61E-13	2.159D-14	3.817D-28
		2	4	.61E-13	2.159D-14	3.819D-28
		1	4	.59E-13	1.240D-14	3.258D-28
		0	5	.97E-14	2.794D-14	1.201D-28
	0.1	4	4	.59E-13	1.237D-13	3.217D-27
		3	4	.59E-13	1.237D-13	3.217D-27
		2	4	.59E-13	1.237D-13	3.217D-27
		1	4	.57E-13	2.209D-13	3.456D-27
		0	5	.76E-11	9.375D-11	2.549D-22

Table 5.8: The transformation for Example (2) with $n = 4$

formation tests. The results are shown in Table 5.8. We also enlarged n to 10 to see the results for a larger dimensional problem. The results are shown in Tables 5.9 and 5.10. Again, from these tests with $\theta = 1$, we can get the equivalent results with or without transformation. However, for $\theta = 0.1$, the tests after transformation need more iterations to terminate than that without transformation. Also, from Table 5.8 to 5.10, we can see that the normal QN method needs more than n steps for termination. We compared the matrix condition numbers for both A and $\hat{A}$, which are shown in Table 5.11.

From the above results, as well as some other experiments we have conducted, com-

	θ =	m =	step=	$\|\|g\|\|$ =	$\|\|x\|\|$ =	f =
with transformation	1	-1	17	.27E-17	2.746D-18	1.011D-36
		10	10	.34E-13	1.489D-14	1.874D-28
		9	10	.34E-13	1.489D-14	1.874D-28
		8	10	.34E-13	1.489D-14	1.874D-28
		7	10	.44E-13	6.684D-14	6.356D-28
		6	10	.19E-12	6.423D-13	2.985D-26
		5	10	.40E-12	1.011D-12	8.529D-26
		4	10	.20E-12	4.150D-13	1.719D-26
		3	11	.29E-12	6.694D-13	4.144D-26
		2	11	.13E-12	1.270D-13	3.599D-27
		1	12	.14E-12	4.755D-13	1.576D-26
		0	13	.83E-09	8.224D-11	3.406D-20
without transformation	1	10	10	.29E-14	2.431D-16	2.939D-31
		9	10	.29E-14	2.431D-16	2.939D-31
		8	10	.29E-14	2.431D-16	2.939D-31
		7	10	.34E-14	1.802D-15	9.737D-31
		6	10	.31E-14	9.619D-16	4.761D-31
		5	10	.29E-14	6.892D-16	3.639D-31
		4	10	.32E-14	1.247D-15	6.446D-31
		3	10	.15E-11	4.540D-13	3.066D-25
		2	10	.76E-10	1.721D-11	5.251D-22
		1	11	.30E-10	6.202D-12	7.348D-23
		0	13	.60E-10	1.159D-11	2.847D-22

Table 5.9: The transformation for Example (2) with $n = 10$ and $\theta = 1$

	θ =	m =	step=	$\|\|g\|\|$ =	$\|\|x\|\|$ =	f =
with transformation	0.1	-1	17	.75E-18	1.259D-17	1.690D-36
		10	11	.59E-18	6.563D-20	6.693D-40
		9	11	.61E-20	6.573D-20	9.168D-41
		8	13	.94E-13	6.939D-13	2.270D-26
		7	14	.63E-12	4.611D-12	1.009D-24
		6	15	.73E-11	4.711D-11	1.343D-22
		5	16	.30E-12	4.787D-12	6.762D-25
		4	18	.74E-13	2.783D-13	7.859D-27
		3	19	.31E-11	4.984D-12	6.002D-24
		2	18	.14E-10	6.136D-11	1.411D-22
		1	20	.13E-11	1.916D-11	8.876D-24
		0	23	.89E-11	9.183D-11	3.828D-22
without transformation	0.1	10	10	.95E-12	3.432D-12	1.002D-24
		9	10	.95E-12	3.432D-12	1.002D-24
		8	10	.95E-12	3.435D-12	1.001D-24
		7	10	.95E-12	3.354D-12	9.901D-25
		6	10	.16E-11	5.163D-12	2.523D-24
		5	10	.27E-11	8.674D-12	7.239D-24
		4	10	.28E-11	2.869D-12	2.937D-24
		3	10	.86E-11	1.263D-11	3.327D-23
		2	10	.22E-10	2.175D-11	2.109D-22
		1	10	.25E-10	2.203D-11	2.105D-22
		0	13	.17E-10	3.034D-11	2.225D-22

Table 5.10: The transformation for Example (2) with $n = 10$ and $\theta = 0.1$

	$n = 4$		$n = 10$	
	A	$\hat{A}$	A	$\hat{A}$
the least eigenvalue	0.2961	0.0961	0.2340	0.0896
the largest eigenvalue	5.8039	160.6399	15.3100	2963.5135
condition number	19.6011	1671.5911	65.4274	33074.9275

Table 5.11: The condition numbers of A and $\hat{A}$

paring the tests with and without transformation, some conclusions can be made for the transformation method:

- In our experiments, especially for these cases with $\theta = 1$, the comparable results can be expected.
- The transformation itself could introduce roundoff errors, especially for those Z_0 that are not well conditioned. For some tests with ill-conditioned Z_0, the termination needs more steps to reach the same accuracy.
- Even for a well conditioned matrix Z_0, with transformation we still might need more iterations. In these cases, we found the condition numbers of the matrix $\hat{A}$ have been enlarged and scaling has less effect in these cases.

5.2 Concluding Remarks

According to the above described experiments, the following remarks can be made about our VS-ZZ^T algorithm:

(1) The algorithm works very well for some tests with initial Z_0 being an identity matrix. By using variable memory space, we can obtain the equivalent results to Powell's method. In particular, the rescaling technique can effectively overcome the accumulated errors in our computation. Even for small values of m, satisfactory results can still be achieved.

(2) For those problems with $||Z_0|| \ll ||A^{-1/2}||$ (that is, θ is very small in our case), very small m ($m = 0$ in our tests) might result in more iterations to reach the same termination

criterion. This is mainly because too few updates in the QN-part cannot make $||Z_m||$ close enough to $||A^{-1/2}||$, so that the preconditioned CG is not very efficient.

(3) Since we use the sum form of BFGS formula in the CG-part, the rescaling technique cannot be used in this part. As in (2), if m is too small, the rescaling is only applied to the beginning few steps, so this might not be enough to suppress the accumulated errors for those problems which are very sensitive to the effect of roundoff errors.

(4) The transformation method allows us to solve some problems with an arbitrary initial matrix. However, if the initial Z_0 is an ill-conditioned matrix, transformation itself would bring in roundoff errors, which deteriorates the performance. Even if Z_0 is a well conditioned matrix, compared to Powell's results, we might still need a few more iterations to reach the same criterion, especially when $||Z_0|| \ll ||A^{-1/2}||$.

* * *

Some future research can be done relating to the VS-ZZ^T algorithm. For example, one could study the possibility of using a restart technique to overcome the problem of using very small m, that is, find a proper restart procedure with corresponding $Z_{restart}$ for minimizing $f(x)$ efficiently. Another possibility is to confirm its applicability by using this algorithm to solve large dimensional problems, as our algorithm is designed for the purpose of solving these problems with large variables. Moreover, it could be also instructive to see the possibility of applying it to general nonlinear functions, by studying their termination properties, as well as the memory requirements.

Bibliography

[1] K. W. Brodlie, A. R. Gourlay, and J. Greenstadt. Rank-one and rank-two corrections to positive definite matrices expressed in product form. *Journal of the Institute of Mathematics and its Applications*, 11:73–82, 1973.

[2] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 2. The new algorithm. *Journal of the Institute of Mathematics and its Applications*, 6:222–231, 1970.

[3] A. Buckley. Extending the relationship between the conjugate gradient and BFGS algorithms. *Mathematical Programming*, 15:343–348, 1978.

[4] A. Buckley and A. LeNir. QN-like variable storage conjugate gradients. Technical Report MIS 82–2, Management Information Systems, College of Business and Public Administration, University of Arizona, Tucson, Arizona, March 1982. See [5].

[5] A. Buckley and A. LeNir. QN-like variable storage conjugate gradients. *Mathematical Programming*, 27:155–175, 1983. See [4].

[6] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*,

13(3):317–322, August 1970.

[7] R. Fletcher. Practical methods of optimization. 1990. John Wiley Sons, (second edition).

[8] I. Gladwell and R. Wait. A survey of numerical methods for partial differential equations. 1979. Clarendon Press Oxford.

[9] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, January 1970.

[10] Donald Goldfarb. Factorized variable metric methods for unconstrained optimization. Manuscript, City University of New York, New York, N. Y. 10031, U. S. A., November 1973.

[11] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. Rpt. No. 185, Instituto de Investigaciones en Matematicas Aplicadas y en Sistemas, Universidad Nacional Autonoma de Mexico, 1979. See [12].

[12] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, July 1980. See [11].

[13] M. J. D. Powell. Updating conjugate directions by the BFGS formula. Technical Report Rpt. DAMTP 1985/NA11 (Rev.), Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, November 1986.

[14] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, July 1970.

Appendix A

The Implementation of the VS-ZZ^T Algorithm

The VS-ZZ^T algorithm was written as a model integrated into the published Algorithm 630 "BBVSCG", due to Buckley and Lenir [4]. So the line search, termination and some other testing are done by the main part.

The VS-ZZ^T model mainly includes three procedures, (1)`BBPOWL`, (2) `BBFACT` and (3) `ZZOUSD`. The first procedure is used as a convenience to organize the memory locations for the information saved at each iteration and the second one is used to shorten the calling procedure to `ZZOUSD` by setting some initial flags for different testing.

The procedure `ZZOUSD` is the main part of our VS-ZZ^T algorithm. The name is given by the first letter of the major functions performed, i.e. Orthogonalizing, Updating, Scaling and Direction. It is based on the Transformation and Updating Algorithm 4.1, as well as the "Column-by-column and Up-row Updating Scheme", but we have to add two more

functions: scaling and calculating the sum for the direction d, which is given by Algorithm 3.2 and equation (4.1). Actually, the procedure is used to compute $\sum_j z_i^{(j)} z_i^{(j)^T} v$, where $v = g_i$ for the QN-part and $v = g_i$ or y_i for the CG-part, but with a fixed $z_m^{(j)}$.

ZZOUSD **Procedure**

The variable explanation of ZZOUSD Procedure:

k — a scalar for saving the largest index which $\hat{s}^{(k)} \neq 0$;

d — a vector for the search direction, i.e. $d = -\sum z^{(j)} z^{(j)^T} g$;

σ — a vector for the first column of Z, i.e. $\sigma = \dfrac{s}{\sqrt{s^T y}}$;

h — an intermediate vector for the sum of $\hat{s}^{(j)} z^{(j)}$, for $j = k, \cdots, 1$;

ϕ — an intermediate scalar for the sum of $\hat{s}^{(j)^2}$, for $j = k, \cdots, 1$;

w — an auxiliary vector for temporarily saving $U z^{(j-1)}$ at each step;

z — an auxiliary vector for temporarily saving $z^{(j)}$ at each step.

For simplicity, in the following procedure we use $U_t h_t$ to denote $h_t - (\dfrac{y_t^T h_t}{s_t^T y_t}) s_t$, and $U_t w$ to denote $w - (\dfrac{y_t^T w}{s_t^T y_t}) s_t$, according to our explanation in the section 4.2.

ZZOUSD Procedure:

Initialization: **Set:** kgot=false; $\hat{s}_1 = -g_0$; $d_{i+1} = 0$; $h_1 = z_0^{(n)}$;

Compute $z_i^{(n)}$

for $l = n-1, n-i, -1$

if $(l > 0)$ then $w = z_0^{(l)}$;

$j = \max(l, 0)$;

for $t = \max(1, -l+1), n-l$

Case 1. $(j+1) > k_t$

$z = U_t h_t$; $h_t = w$;

Case 2. $(j+1) = k_t$ and $j \neq 0$

$\phi_t = \hat{s}_t^{(j+1)}$; $h_t = \hat{s}_t^{(j+1)} U_t h_t$;

$w = U_t w$;

$$z = \sqrt{\frac{\phi_t}{\hat{s}_t^{(j)^2} + \phi_t}} \left(-w + \frac{\hat{s}_t^{(j)}}{\phi_t} h_t \right);$$

if $(\hat{s}_t^{(j+1)} < 0)$ then $z = -z$;

if $(j > 1)$ then
$h_t = h_t + \hat{s}_t^{(j)} w$;
$\phi_t = \phi_t + \hat{s}_t^{(j)^2}$;
end if

Case 3. $(j+1) < k_t$ and $j \neq 0$

$w = U_t w$;

$$z = \sqrt{\frac{\phi_t}{\hat{s}_t^{(j)^2} + \phi_t}} \left(-w + \frac{\hat{s}_t^{(j)}}{\phi_t} h_t \right);$$

if $(j > 1)$ then
$h_t = h_t + \hat{s}_t^{(j)} w$;
$\phi_t = \phi_t + \hat{s}_t^{(j)^2}$;
end if

Case 4. $j = 0$

$z = \sigma_t$;

if $(t = i)$ then save $||z||$;

Scaling call do-scaling procedure;

if $(j \neq n-1)$ then $w = z$;
$j = j + 1$;
end for (t **loop**)
$h_{n-l+1} = z$;
end for (l **loop**)

Calculate $\hat{s}_{i+1}^{(n)}$ $\hat{s}_{i+1}^{(n)} = -z^T g_i$;

Find k_{i+1} if (kgot = false) then
if (abs($\hat{s}_{i+1}^{(n)}$) > 0) then
$k_{i+1} = n$; kgot=ture;
end if
end if

Calculate d_{i+1} $d_{i+1} = d_{i+1} + \hat{s}_{i+1}^{(n)} z$;

Calculate $z_n, \cdots, z_1$ for $ll = n-1, 1, -1$
$l = ll - i$;
if $(l > 0)$ then $w = z_0^{(l)}$;
$j = \max(l, 0)$;
for $t = \max(1, -l+1), i$

Case 1. $(j+1) > k_t$

$z = U_t h_t$; $h_t = w$;

Case 2. $(j+1) = k_t$ and $j \neq 0$

$\phi_t = \hat{s}_t^{(j+1)}$; $h_t = \hat{s}_t^{(j+1)} U_t h_t$;
$w = U_t w$;

$z = \sqrt{\frac{\phi_t}{\hat{s}_t^{(j)^2} + \phi_t}}\left(-w + \frac{\hat{s}_t^{(j)}}{\phi_t} h_t\right)$;
if ($\hat{s}_t^{(j+1)} < 0$) then $z = -z$;
if ($j > 1$) then
$h_t = h_t + \hat{s}_t^{(j)} w$;
$\phi_t = \phi_t + \hat{s}_t^{(j)^2}$;
end if

Case 3. $(j+1) < k_t$ and $j \neq 0$
$w = U_t w$;
$z = \sqrt{\frac{\phi_t}{\hat{s}_t^{(j)2} + \phi_t}}\left(-w + \frac{\hat{s}_t^{(j)}}{\phi_t} h_t\right)$;
if ($j > 1$) then
$h_t = h_t + \hat{s}_t^{(j)}$w;
$\phi_t = \phi_t + \hat{s}_t^{(j)^2}$;
end if

Case 4. $j = 0$
$z = \sigma_t$;
if ($t = i$) then save $||z||$;

Scaling call do-scaling procedure;
if ($j \neq n-1$) then $w = z$;
$j = j + 1$;
end for (t **loop**)

Calculate $\hat{s}_{i+1}^{(ll)}$ $\hat{s}_{i+1}^{(ll)} = -z^T g_i$;
Find k_{i+1} if (kgot = false) then
if (abs($\hat{s}_{i+1}^{(n)}$) > 0) then
$k_{i+1} = ll$; kgot=true;
end if
end if

Calculate d_{i+1} $d_{i+1} = d_{i+1} + \hat{s}_{i+1}^{(ll)} z$;
end for (ll **loop**)

(**end of the algorithm**)

Appendix B

Another Updating Approach for the CG-part

In our algorithm implementation, we also studied the possibility of continuing Powell's updating method in the CG-part. That is, after m steps, we still use the factored form of BFGS update formula to do the update for Z_i, which is based on Z_m at each time. However, in the CG-part, we have to assume

$$\hat{s}_i = -Z_m^T g_{i-1}, \qquad i = m+1, \cdots. \tag{B.1}$$

This is because in the CG-part, the matrix Z_i is updated based on Z_m at each iteration, instead of from Z_{i-1} directly. According to the definition of Powell's method, which we described in Chapter 3, we have

$$Z_i = (I - s_i p_i^T)\bar{Z}_m, \tag{B.2}$$

where $\bar{Z}_m = Z_m \Omega_i$. From the derivation of Chapter 3, we know in order to let the first column of $\bar{Z}_m$ become the multiple of s_i, the following equation must be satisfied

$$Z_m \Omega_i e_1 = \alpha_i s_i = -\alpha_i Z_{i-1} Z_{i-1}^T g_{i-1}, \tag{B.3}$$

where α_i is a scalar. It is equivalent to

$$\hat{s}_i = c\Omega_i e_1 = c Z_m^{-1} Z_{i-1} Z_{i-1}^T g_{i-1}, \tag{B.4}$$

where c is a scalar. The computation of the matrix inverse is not efficient. This could be a very big problem, because it requires $\frac{n^2}{2}$ memory space which is contradictory to our method, and it also adds more computational burden on our algorithm. Moreover, since our algorithm does not save Z explicitly, it could be very difficult to design (if possible) a method to compute the inverse Z_m. Also, we can not guarantee that the results will be improved greatly, as inverse matrix operation itself could bring in more roundoff errors (especially for ill-conditioned case). So, in our implementation, we have to let $Z_{i-1} \approx Z_m$ and use (B.1) to approximate $\hat{s}_i$.

Although in theory, the justification for this assumption is still doubted and it has some influences on the effect of scaling for $i > m$, the algorithm does indeed work and we can chieve satisfactory results comparable to the results given in Chapter 5.

In the following Algorithm, we still use $\hat{U}(z_{i-1}^{(j)}, i)$ to stand for updating $z_{i-1}^{(j)}$ using the Transformation and Update Algorithm 4.1 and "Column-by-column and Up-row Updating Scheme". As in the CG-part $\hat{s}_{i+1}^{(j)} = -z_m^{(j)^T} g_i$ is only used for the transformation, we use a scalar $t = -z_i^{(j)^T} g_i$ for computing the sum of search direction d_{i+1}.

Algorithm B.1

Choosing an initial Z_0 and starting from the initial point x_0, the initial search direction goes along $d_1 = -Z_0 Z_0^T g_0$. Then:

QN-part: From the point x_1, iterate with $i = 1, 2, \cdots, m$:

save $s_i, y_i, \hat{s}_i, \eta_i$ and k_i for updating and the transformation;

$d_{i+1} = 0$;

iterate for $j = n, n-1, \cdots, 1$ to get d_{i+1}:

$z_i^{(j)} = \hat{U}(z_{i-1}^{(j)}, i)$;

save $||z_i^{(j)}||$ and do rescaling, if necessary;

$\hat{s}_{i+1}^{(j)} = -z_i^{(j)^T} g_i$;

$d_{i+1} = d_{i+1} + \hat{s}_{i+1}^{(j)} z_i^{(j)}$;

$x_{i+1} = x_i + \lambda_i d_{i+1}$;

CG-part: From the point x_{m+1}, iterate for $i = m+1, m+2, \cdots$:

if $i = m+1$, save $s_i, y_i, \hat{s}_i, \eta_i$ and k_i; otherwise, replace

$s_{m+1}, y_{m+1}, \hat{s}_{m+1}, \eta_{m+1}$ and k_{m+1} by $s_i, y_i, \hat{s}_i, \eta_i$ and k_i;

$d_{i+1} = 0$;

iterate for $j = n, n-1, \cdots, 1$ to get d_{i+1}:

$z_i^{(j)} = \hat{U}(z_m^{(j)}, i)$;

$\hat{s}_{i+1} = -z_m^{(j)^T} g_i$ (for transformaiton);

$t = -z_i^{(j)^T} g_i$;

$d_{i+1} = d_{i+1} + t z_i^{(j)}$;

$x_{i+1} = x_i + \lambda_i d_{i+1}$.

	m =	**step=**	**$\|\|g\|\|$ =**	**$\|\|x\|\|$ =**	**f =**
QN method	-1	11	.85E-24	1.086D-25	3.025D-50
with or without scaling	10	10	.24E-15	3.020D-17	3.517D-33
	9	10	.24E-15	3.020D-17	3.517D-33
	8	10	.24E-15	3.020D-17	3.517D-33
	7	10	.24E-15	3.014D-17	3.510D-33
	6	10	.24E-15	3.348D-17	3.500D-33
	5	10	.23E-15	3.472D-17	3.175D-33
	4	10	.26E-15	4.554D-17	3.960D-33
	3	10	.30E-15	3.760D-17	4.441D-33
	2	10	.18E-15	5.864D-17	2.258D-33
	1	10	.28E-15	2.928D-17	3.803D-33
	0	10	.48E-12	3.152D-14	7.604D-27

Table B.1: $Z_0 = I, \theta = 1$

For comparison, we present the results under the same tests as given in the Chapter 5.

	m =	step=	‖g‖ =	‖x‖ =	f =
QN method	-1	11	.36E-26	3.598D-25	4.429D-52
with scaling	11	11	.30E-24	2.020D-23	2.967D-48
	10	11	.30E-24	2.020D-23	2.967D-48
	9	11	.49E-25	3.290D-24	7.861D-50
	8	11	.19E-18	5.651D-17	4.177D-36
	7	11	.34E-17	1.767D-15	1.889D-33
	6	11	.90E-17	2.078D-15	8.144D-33
	5	11	.65E-16	6.974D-14	1.036D-30
	4	11	.78E-16	1.734D-13	4.323D-30
	3	11	.35E-15	5.977D-14	8.353D-30
	2	11	.66E-15	5.534D-13	6.127D-29
	1	11	.35E-15	4.023D-13	3.236D-29
	0	26	.26E-12	3.610D-11	3.324D-24
without scaling	10	16	.42E-15	1.078D-13	1.838D-29
	9	17	.31E-15	7.980D-14	1.031D-29
	8	17	.21E-13	1.113D-11	5.437D-26
	7	19	.28E-14	1.275D-12	8.898D-28
	6	20	.47E-13	7.365D-11	1.142D-24
	5	23	.34E-14	6.132D-12	6.821D-27
	4	23	.22E-12	6.673D-11	3.642D-24
	3	32	.31E-13	9.665D-11	1.167D-24
	2	28	.21E-12	9.701D-11	7.555D-24
	1	31	.10E-12	1.712D-11	7.321D-25
	0	26	.26E-12	3.610D-11	3.324D-24

Table B.2: $Z_0 = I, \theta = 10^{-3}$

	m =	**step=**	**||g|| =**	**||x|| =**	**f =**
QN method	-1	11	.17E-35	2.779D-25	1.364D-61
with scaling	11	11	.74E-33	5.029D-23	1.820D-56
	10	11	.74E-33	5.029D-23	1.820D-56
	9	11	.12E-33	8.331D-24	4.980D-58
	8	11	.16E-27	4.785D-17	2.996D-45
	7	11	.17E-26	8.410D-16	4.474D-43
	6	11	.28E-26	2.304D-15	1.672D-42
	5	11	.39E-25	3.700D-14	3.209D-40
	4	11	.60E-25	9.676D-14	1.506D-39
	3	11	.56E-24	6.787D-13	7.253D-38
	2	11	.10E-23	1.693D-12	3.937D-37
	1	10	.34E-22	2.632D-12	3.950D-35
	0*	10	-	8.813D-01	-

Table B.3: $Z_0 = I, \theta = 10^{-12}$

	θ =	m =	step=	$\|\|g\|\|$ =	$\|\|x\|\|$ =	f =
with scaling	1	7	7	.31E-10	1.720D-11	2.024D-22
		6	7	.31E-10	1.720D-11	2.024D-22
		5	7	.31E-10	1.720D-11	2.024D-22
		4	11	.26E-10	1.609D-11	1.441D-22
		3	6	.12E-10	2.558D-12	1.341D-23
	10^{-1}	6	6	.39E-11	3.479D-11	3.736D-23
		5	6	.39E-11	3.479D-11	3.736D-23
		4	6	.39E-11	3.479D-11	3.736D-23
		3	6	.36E-12	1.715D-12	1.979D-25
	10^{-2}	5	5	.94E-14	7.120D-13	2.422D-27
		4	5	.94E-14	7.120D-13	2.422D-27
		3	5	.94E-14	7.120D-13	2.422D-27
	10^{-3}	6	6	.60E-17	5.231D-15	8.998D-33
		5	6	.60E-17	5.231D-15	8.998D-33
		4	6	.60E-17	5.231D-15	8.998D-33
		3	5	.28E-15	4.504D-13	4.018D-29
	10^{-4}	7	7	.12E-15	4.023D-13	2.208D-29
		7	7	.12E-15	4.023D-13	2.208D-29
		7	7	.12E-15	4.023D-13	2.208D-29
		4	6	.64E-14	2.071D-11	5.858D-26
		3	6	.98E-15	2.524D-12	1.022D-27
	10^{-6}	7	7	.82E-20	3.157D-15	1.137D-35
		6	7	.82E-20	3.157D-15	1.137D-35
		5	7	.82E-20	3.157D-15	1.137D-35
		4	6	.59E-18	6.274D-13	1.233D-31
		3	6	.65E-19	1.508D-14	4.083D-34

Table B.4: $Z_0 \neq I$, Example (1)

	μ =	m =	step=	$\|\|g\|\|$ =	$\|\|x\|\|$ =	f =
with scaling	0	4	4	.59E-12	1.168D-13	3.121D-26
		3	4	.59E-12	1.168D-13	3.121D-26
		2	5	.15E-10	1.969D-11	9.131D-23
		1	13	.16E-10	2.102D-11	1.020D-22
	1	4	4	.22E-11	4.033D-13	4.192D-25
		3	4	.22E-11	4.033D-13	4.192D-25
		2	6	.60E-13	1.115D-14	3.201D-28
		1**	25	-	1.160D-01	-
	2	4	4	.89E-11	1.674D-12	7.061D-24
		3	4	.89E-11	1.674D-12	7.061D-24
		2	6	.14E-13	3.040D-15	1.894D-29
		1**	25	-	9.680D-02	-
	5	4	4	.12E-09	2.179D-11	1.236D-21
		3	4	.12E-09	2.179D-11	1.236D-21
		2	5	.13E-10	1.617D-11	6.423D-23
		1**	25	-	4.731D-03	-
without scaling	10	5	5	.16E-20	9.927D-22	6.356D-43
		4	5	.16E-20	9.927D-22	6.356D-43
		3	5	.45E-25	8.573D-26	1.336D-51
		2	6	.36E-11	6.466D-13	1.110D-24
		1	18	.93E-10	9.545D-11	2.824D-21

Table B.5: $Z_0 \neq I$, Example (2)

	$\theta =$	m =	step=	$\|\|g\|\| =$	$\|\|x\|\| =$	f =
with scaling	10^{-3}	4	4	.10E-14	2.040D-13	9.648D-29
		3	4	.10E-14	2.040D-13	9.648D-29
		2	4	.10E-14	2.040D-13	9.648D-29
		1	4	.10E-14	2.421D-13	9.984D-29
		0**	25	-	1.179D-03	-
	10^{-6}	4	4	.95E-18	2.283D-13	8.974D-32
		3	4	.95E-18	2.283D-13	8.974D-32
		2	4	.94E-18	2.546D-13	8.788D-32
		1	4	.92E-18	2.762D-13	8.661D-32
		0*	18	-	2.293D-01	-
	10^{-12}	4	4	.51E-24	1.528D-13	2.576D-38
		3	4	.51E-24	1.528D-13	2.576D-38
		2	4	.51E-24	1.528D-13	2.576D-38
		1	4	.49E-24	9.184D-14	2.168D-38
		0*	14	-	2.269D-01	-
without scaling	10^{-3}	4	6	.20E-16	1.681D-14	1.313D-31
		3	6	.20E-14	9.377D-13	8.830D-28
		2	23	.86E-13	5.163D-11	1.619D-24
		1**	25	-	1.458D-06	-
		0**	25	-	1.179D-03	-
	10^{-6}	4	6	.76E-18	1.505D-12	4.169D-31
		3*	21	-	1.968D-04	-
		2*	22	-	4.305D-03	-
		1*	22	-	2.355D-01	-
		0*	18	-	2.293D-01	-

Table B.6: $Z_0 \neq I$, Example (3)

	θ =	m =	step=	‖g‖ =	‖x‖ =	f =
with transformation	1	-1	5	.36E-24	1.332D-26	1.336D-51
		4	4	.35E-14	1.231D-15	6.332D-31
		3	4	.35E-14	1.231D-15	6.332D-31
		2	4	.35E-14	1.231D-15	6.332D-31
		1	4	.78E-14	7.279D-16	2.002D-30
		0	4	.17E-11	2.511D-14	2.121D-26
	0.1	-1	5	.35E-25	9.139D-27	1.399D-52
		4	4	.27E-12	7.029D-12	8.518D-25
		3	4	.27E-12	7.029D-12	8.518D-25
		2	4	.27E-12	7.029D-12	8.518D-25
		1	4	.27E-12	7.029D-12	8.518D-25
		0	4	.35E-12	4.296D-12	3.257D-25
	0.01	-1	5	.44E-25	1.685D-25	2.441D-51
		4*	5	-	5.408D-10	-
		3*	5	-	5.408D-10	-
		2	5	.25E-11	3.584D-11	3.776D-23
		1	5	.23E-11	3.805D-11	3.791D-23
		0	6	.14E-10	5.972D-11	2.134D-22
without transformation	1	4	4	.25E-14	7.234D-15	7.907D-30
		3	4	.25E-14	7.234D-15	7.907D-30
		2	4	.25E-14	7.234D-15	7.907D-30
		1	4	.10E-13	4.654D-15	2.003D-29
		0	4	.48E-12	9.433D-14	2.137D-26
	0.1	4	4	.48E-13	1.530D-12	3.518D-26
		3	4	.48E-13	1.530D-12	3.518D-26
		2	4	.48E-13	1.530D-12	3.518D-26
		1	4	.61E-14	1.786D-13	4.883D-28
		0	4	.32E-12	1.970D-12	1.441D-25
	0.01	4	4	.11E-13	4.901D-13	2.247D-27
		3	4	.11E-13	4.901D-13	2.247D-27
		2	4	.11E-13	4.901D-13	2.247D-27
		1	4	.40E-14	8.268D-14	1.503D-28
		0	6	.20E-11	5.346D-11	4.449D-23

Table B.7: An transformation example

	θ =	m =	step=	$\|\|g\|\|$ =	$\|\|x\|\|$ =	f =
with transformation	1	-1	6	.55E-19	1.128D-19	2.456D-39
		4	4	.88E-12	2.996D-12	9.054D-25
		3	4	.88E-12	2.996D-12	9.054D-25
		2	4	.88E-12	2.996D-12	9.054D-25
		1	4	.74E-12	1.788D-12	5.012D-25
		0	4	.53E-09	4.315D-12	8.844D-22
	0.1	-1	6	.47E-22	4.449D-22	3.792D-45
		4	5	.82E-22	1.273D-21	3.728D-44
		3	5	.10E-21	1.271D-21	3.727D-44
		2	8	.24E-12	3.698D-12	4.250D-25
		1	23	.70E-11	6.893D-11	5.546D-23
		0	11	.92E-11	3.940D-11	1.073D-23
without transformation	1	4	4	.61E-13	2.159D-14	3.817D-28
		3	4	.61E-13	2.159D-14	3.817D-28
		2	4	.61E-13	2.159D-14	3.819D-28
		1	4	.60E-13	1.227D-14	3.242D-28
		0	5	.74E-14	8.321D-15	2.298D-29
	0.1	4	4	.59E-13	1.237D-13	3.217D-27
		3	4	.59E-13	1.237D-13	3.217D-27
		2	4	.59E-13	1.237D-13	3.217D-27
		1	4	.59E-13	1.239D-13	3.309D-27
		0	5	.13E-10	9.405D-11	5.284D-22

Table B.8: The transformation for Example (2) with $n = 4$

	$\theta =$	**m =**	**step=**	**||g|| =**	**||x|| =**	**f =**
with transformation	1	-1	17	.27E-17	2.746D-18	1.011D-36
		10	10	.34E-13	1.489D-14	1.874D-28
		9	10	.34E-13	1.489D-14	1.874D-28
		8	10	.34E-13	1.489D-14	1.874D-28
		7	10	.39E-13	1.448D-14	1.715D-28
		6	10	.11E-12	3.930D-14	1.942D-27
		5	10	.12E-12	4.461D-14	2.414D-27
		4	10	.76E-13	1.047D-13	1.057D-27
		3	10	.55E-09	8.595D-11	1.927D-20
		2	10	.47E-09	4.630D-11	1.095D-20
		1	12	.55E-13	4.484D-14	7.734D-28
		0	14	.11E-10	3.639D-13	5.680D-26
without transformation	1	10	10	.29E-14	2.432D-16	2.939D-31
		9	10	.29E-14	2.431D-16	2.939D-31
		8	10	.29E-14	2.431D-16	2.939D-31
		7	10	.29E-14	2.511D-16	2.951D-31
		6	10	.30E-14	2.250D-16	3.043D-31
		5	10	.30E-14	2.280D-16	3.003D-31
		4	10	.30E-14	2.313D-16	3.036D-31
		3	10	.24E-11	7.277D-13	7.880D-25
		2	10	.66E-11	1.499D-12	3.985D-24
		1	11	.30E-10	6.063D-12	7.147D-23
		0	13	.53E-10	1.005D-11	2.164D-22

Table B.9: The transformation for Example (2) with $n = 10$ and $\theta = 1$

	$\theta =$	m =	step=	$\|\|g\|\| =$	$\|\|x\|\| =$	f =
with transformation	0.1	-1	17	.75E-18	1.259D-17	1.690D-36
		10	11	.59E-18	6.563D-20	6.693D-40
		9	11	.61E-20	6.573D-20	9.168D-41
		8	13	.31E-13	2.295D-13	2.462D-27
		7	14	.19E-12	1.352D-12	8.713D-26
		6	16	.53E-14	1.831D-14	4.691D-29
		5	17	.91E-14	3.036D-14	1.384D-28
		4	18	.28E-14	9.335D-15	1.307D-29
		3	19	.93E-11	9.468D-11	3.662D-22
		2	18	.10E-10	2.710D-11	5.200D-23
		1	20	.13E-10	9.698D-11	2.035D-22
		0	27	.10E-14	8.928D-16	4.440D-31
without transformation	0.1	10	10	.95E-12	3.432D-12	1.002D-24
		9	10	.95E-12	3.432D-12	1.002D-24
		8	10	.95E-12	3.435D-12	1.001D-24
		7	10	.96E-12	3.293D-12	9.832D-25
		6	10	.16E-11	4.960D-12	2.534D-24
		5	10	.27E-11	9.451D-12	7.408D-24
		4	10	.28E-11	3.658D-12	3.150D-24
		3	10	.12E-10	2.059D-11	1.010D-22
		2	10	.21E-10	2.359D-11	1.857D-22
		1	10	.35E-10	4.642D-11	6.560D-22
		0	13	.24E-10	3.966D-11	4.404D-22

Table B.10: The transformation for Example (2) with $n = 10$ and $\theta = 0.1$

VITA

Surname: Lee Given Names: Ann

Place of Birth: Shanghai, China Date of Birth: October 21, 1955

Educational Institutions Attended:

Shanghai University of Science and Technology	1978 to 1982
Shanghai University of Technology	1984 to 1987
University of Victoria	1990 to 1992

Degrees Awarded:

B.Sc.	Shanghai University of Science and Technology	1982
M.Sc.	Shanghai University of Technology	1987

PARTIAL COPYRIGHT LICENSE

Title of Thesis:

UPDATING CONJUGATE DIRECTIONS BY THE BFGS FORMULA WITH VARIABLE STORAGE

Author: [redacted]

Ann Lee

January 13, 1993