A STUDY OF THE LIMITED MEMORY SR1 METHOD IN PRACTICE

by

XUEHUA LU

B. S., Department of Mathematics, Nanjing University, 1983

M. S., Institute of Applied Mathematics, Chinese Academy of Sciences, 1986

M. S., Department of Computer Science, University of Colorado at Boulder, 1992

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

1996

This thesis for the Doctor of Philosophy degree by

Xuehua Lu

has been approved for the

Department of

Computer Science

by

_____

Richard H. Byrd

_____

Robert B. Schnabel

Date _____

Lu, Xuehua (Ph. D., Computer Science)

A Study of the Limited Memory SR1 Method in Practice

Thesis directed by Professor Richard H. Byrd

Limited Memory BFGS method (L-BFGS) has been shown to be an effective way to solve large scale optimization problems. Generally, an m-step Limited Memory method is essentially a quasi-Newton method where only m of the updates to the $n \times n$ Hessian approximation are stored ($m \ll n$). In this dissertation, we present an implementation of the Limited Memory Symmetric Rank One method (L-SR1) with the assistance of the compact representation of SR1 updates. Our numerical experiments indicate that the performance of L-SR1 method is sensitive to the initial scaling parameter and with our special choice, it can perform comparably with the L-BFGS method. We also prove the global and Q-linear local convergence, which is actually true for any Hessian approximations as long as they are uniformly bounded.

DEDICATION

To my parents, my wife Qing and my daughter Shindy

ACKNOWLEDGEMENTS

I am indebted to many people who have dedicated one way or the other to this dissertation. In the first place I want to thank Professor Richard H. Byrd for introducing me to the research direction of Limited Memory methods. His guidance, encouragement and patience have greatly contributed to the advances in this study. Special thanks go to Professor Robert Schnabel and Professor Elizabeth Jessup of University of Colorado at Boulder, and to Professor Jorge Nocedal of Northwestern University for their important suggestions and continual supports. Last but not least, I would like to extend my appreciation to Dr. Yuanfu Xie, Mr. Chung-Shang Shao and Mr. Zhihong Zou for the concern and help provided while I am discussing with them about my research.

CONTENTS

TABLES

TABLE

CHAPTER 1

INTRODUCTION

1.1 Definition of Unconstrained Optimization Problems

Optimization problems are not created by scientists, but come from the real world. The simple example is to find the highest point of a throw with the initial speed given. More sophisticated examples and applications can be found in science, engineering, economics and other disciplines. Basically, any problem involving ascertaining the maximum or minimum could be catalogued to the field of optimization. For instance, the growing rate of a given vegetable, say celery, is a function of several factors such as soil, moisture, sunshine and temperature. All these factors should be appropriately controlled. Otherwise, the celery will not grow well or even die. For each such factor, there is a most appropriate level such that celery grows at its fastest rate. The way to find such an optimal setting based on a given objective function is the subject of **numerical optimization**, or simply **optimization**.

The systematic study of numerical optimization can only be traced back to 1940's, although Newton's method had long been known by that time. It was the invention of the computer that started the all-round study of numerical optimization, and indeed, of the whole numerical analysis. In 1959, Davidon first introduced the so-called variable metric methods, bringing researchers out of studying problems of special structures. There is no doubt that the 1970's were the paramount in the development of optimization. In

those years, Broyden, Fletcher, Goldfarb and Shanno discovered the BFGS method independently, and Powell proved its superlinear convergence without exact line search. Moreover, Dennis, Moré and Ortega established a rigorous numerical optimization theory and discovered several important tools for further study. Nowadays, people are concentrating on methods for supercomputing and for solving large-scale problems, which are considered to be the research direction of the future. Some others are still enjoying refining the optimization theory.

Mathematically, an optimization problem can be defined as

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{such that} \quad x \in \mathcal{S}. \end{aligned} \tag{1.1}$$

where $x \in \mathcal{R}^n$ and $\mathcal{S}$ is a subset of $\mathcal{R}^n$. If $\mathcal{S} = \mathcal{R}^n$, we say that the problem is **unconstrained**, otherwise it is **constrained**. In this dissertation, we will talk about unconstrained problems only. Actually, constrained problems can sometimes be tackled by being related to unconstrained problems.

There are currently plenty of methods to solve unconstrained optimization problems. They are all based on the same iteration idea: repeat

$$x_{k+1} = x_k + s_k, \tag{1.2}$$

until an acceptable x_k is found; only in the way to compute s_k do they differ. Typically, for quasi-Newton methods, $s_k = -\lambda_k B_k^{-1} g_k$ or $s_k = -(B_k + \nu_k)^{-1} g_k$, where $\lambda_k, \nu_k > 0$, $g_k = \nabla f(x_k)$ and B_k is an $n \times n$ matrix approximating the Hessian $\nabla^2 f(x_k)$. Basically, all small problems today can be well worked out by quasi-Newton methods, if what we are concerned about is just local minimizers. But for large problems (saying $n > 500$), even though we can finally get the answer, the computing time is frequently unbearable. A single

matrix-vector multiplication for s_k takes n^2 scalar multiplications. Also, to store the matrix B_k, we need n^2 memory locations. That means for a problem of size 5000, we need at least 25M memory locations. Not to mention the case when n is as big as millions.

There is no clear-cut answer as to which method is superior to others. Many issues have to be taken into account when we choose a method to solve a specific problem. This is especially true for large-scale problems, since they come in all sorts of forms. Among all the distinct classes of methods is the class of limited memory methods. Limited memory methods are appropriate for large problems in which the Hessian matrix cannot be computed at reasonable cost and there is no advantage we can take on the structure of the objective function.

In the next two sections, we will briefly review methods related to large scale unconstrained optimization in which the structure of the objective function is unknown.

1.2 Methods Requiring $O(n^2)$ Memory

Methods in this class require at least n^2 memory locations to store an $n \times n$ matrix. The typical ones are Newton's method and all kinds of standard quasi-Newton methods.

- **Newton's Method:**

Newton's method is the very first method to tackle optimization problems. Its s_k is solved from the linear system

$$G_k s_k = -g_k, \tag{1.3}$$

where $g_k = \nabla f(x_k)$ and $G_k = \nabla^2 f(x_k)$. Its local convergence rate is pretty

fast: Q-quadratic. For a problem of dimension less than 50, only $5 \sim 15$ iterations are required to achieve a 7-digit precision. This ideal property is preserved even when we approximate G_k using finite differences. The biggest disadvantage of Newton's method is the availability of Hessian matrix $\nabla^2 f(x)$. Even it is available, n^2 function evaluations at a single iteration are often too expensive. The other weakness of the Newton's method is that we have to take $O(n^3)$ solve an $n \times n$ linear system at each iteration. Unless G_k has a good sparsity structure, it is in practice impossible to get the exact solution within acceptable time.

- **Quasi-Newton Methods:**

  These methods solve a similar system to Newton's method:

$$B_k s_k = -g_k, \tag{1.4}$$

where B_k is a nonsingular symmetric (hopefully also positive definite) square matrix updated at each iteration by some formula of the form:

$$B_k = \mathbf{U}(B_{k-1}, y_{k-1}, s_{k-1}), \tag{1.5}$$

where $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. In order to achieve the Q-superlinear convergence, B_k is required to approximate $\nabla^2 f(x_k)$ at least in the direction of s_k. The well known methods, DFP, BFGS and SR1 are in this class. Among them, BFGS has been proved a fast and reliable method of practical applications. SR1 is an awkward method, even though its performance is in general a little better than BFGS. The problem is, the SR1 update is not guaranteed to be nonsingular, even when the iterate is close enough to the solution, not to mention the positive definiteness of B_k. However, SR1 method has its own unique wonder: it terminates on a quadratic function in at most $n+1$ iterations.

More work is needed on the SR1 method. As for DFP method, experiments have shown that it is slower than almost all the other existing Quasi-Newton methods.

Quasi-Newton methods are usually a little bit slower than Newton's method. For example, for a problem of dimension less than 50, BFGS method requires about $10 \sim 40$ iterations to achieve a 7-digit precision. However, quasi-Newton methods do not need the Hessian matrix $\nabla^2 f(x)$ for its computation. All it needs is the gradient $\nabla f(x)$. Also, the $n \times n$ linear system (1.4) can be solved with $O(n^2)$ operations. Compared to the Newton's method, it is a considerable saving.

1.3 Methods Requiring $O(n)$ Memory

Methods in this class require at most $O(n)$ memory locations to store some n-dimensional vectors. The typical ones are Truncated Newton method and all kinds of limited memory methods.

• Truncated Newton Method:

The Truncated Newton method originates from the conjugate gradient method. Hestenes and Steifel's original conjugate gradient method solves the linear system (1.3) within n (inner-)iterations over the following procedure. Initially, $p_0 = r_0 = -G_k s_k^{(0)} - g_0$. At j-th iteration,

$$\alpha_j = \|r_j\|^2 / p_j^T G_k p_j, \qquad r_{j+1} = r_j - \alpha_j G_k p_j,$$

$$\beta_j = \|r_{j+1}\|^2 / \|r_j\|^2, \qquad p_{j+1} = r_{j+1} + \beta_j p_j,$$

$$s_k^{(j+1)} = s_k^{(j)} + \alpha_j p_j.$$

Note that, although we have the Hessian matrix G_k in the formula, all we really need is the vector value of $G_k p_j$, which can be approximated by a finite

difference with one more gradient evaluation.

Nash (1984)'s Truncated Newton method solves the Newton system (1.3) approximately by the conjugate gradient method with a preconditioner. The termination of the inner iteration is based on the standard

$$j\left(1-\frac{\Delta f_k^{(pred)}(p_{j-1})}{\Delta f_k^{(pred)}(p_j)}\right)\leq 0.5,$$

where j is the counter for the inner iteration and $\Delta f_k^{(pred)}(p)$ is the predicted reduction of function value at x_k along direction p, $\Delta f_k^{(pred)}(p)=p^Tg_k+\frac{1}{2}p^TG_kp$. This test guarantees a linear rate of convergence and also detects the convergence of the inner loop.

The Truncated Newton method is considered a successful algorithm for solving large-scale unconstrained optimization problems. It has been implemented as a software package, TNPACK.

• **Limited Memory Quasi-Newton Methods:**

Since the early works by Buckley (1978a,b), Nazareth (1979) and Nocedal (1980), limited memory methods have received expanding consideration in the past two decades. It is mainly because they are considered as methods filling the gap between conjugate gradient and quasi-Newton methods. The former use only $O(n)$ memory locations, but converge rather slowly and require expensive line searches; on the other hand, quasi-Newton methods have the converse features: fast rate of convergence (theoretically superlinear), no need for exact line searches, but large memory requirement, namely $O(n^2)$ storage locations. The actual amount of storage required by limited memory methods is variable and can be determined by the availability of space.

There are actually two distinct approaches to limited memory methods. The first was introduced by Buckley (1978a,b), and Buckley and LeNir

(1983). It has two stages, the BFGS stage and the conjugate gradient stage. In the BFGS stage, they update the BFGS matrix just for few steps, in which the BFGS matrices are not formed, but are represented by a set of vectors. In the conjugate gradient stage, they use the matrix from the first stage as a preconditioner for the conjugate gradient method.

Then, Nocedal (1980) realized the potential of the single use of the first stage and developed an elegant two-loop algorithm of the limited memory BFGS (L-BFGS) method. This approach takes a pure quasi-Newton point of view. It is identical to the standard BFGS method in all its implementation except that the BFGS matrix is not explicitly present. The studies by Gilbert and Lemaréchal (1989), and Nash and Nocedal (1989) indicate that L-BFGS outperforms Buckley's limited memory method and can be as promising as Truncated Newton method. Liu and Nocedal (1989) proved its R-linear convergence. The algorithm performs well for a very small m between 2 and 5, and big n ranging from 100 to 10,000. It needs just $2mn + 2n$ memory locations for the whole algorithm and $4mn + 2n$ multiplications per iteration.

In general, it has been observed that the Hessian approximation B_k defined by (1.5) is actually a matrix made of k pairs of information: (y_{k-1}, s_{k-1}), (y_{k-2}, s_{k-2}), ..., (y_0, s_0). If we apply (1.5) m times, and write B_{k-m} as $B_{k,0}$, then we can get a "partial" Hessian matrix

$$B_k = \underbrace{\mathbf{U}(\mathbf{U}(\cdots \mathbf{U}(}_{m} B_{k,0}, y_{k-m}, s_{k-m}), \cdots), y_{k-1}, s_{k-1}).$$

Instead of explicitly storing the matrix B_k, we store in memory the "initial matrix" $B_{k,0}$ and the m most recent vector pairs $\{y_{k-i}, s_{k-i}\}_{i=1}^{m}$. After computing the new iterate, the oldest pair is deleted from the set $\{y_{k-i}, s_{k-i}\}_{i=1}^{m}$, and is replaced by the newest one.

If we choose $B_{k,0}$ to be $B_{k,0} = \gamma_k I$ where γ_k is a positive number, then B_k can be stored implicitly with $2mn+1$ locations of memory. Compared to the original quasi-Newton method, limited memory methods save memory significantly, if $m \ll n$.

1.4 Motivation and Overview

The work by Conn, Gould and Toint (1991), Khalfan, Byrd and Schnabel (1990), and Byrd, Khalfan and Schnabel (1993) reveals that the Symmetric Rank One (SR1) method is competitive with BFGS method in both theory and practice. In some sense, SR1 is even better. Now that the L-BFGS is efficient in practice, a natural question is then: How about limited memory SR1 method (L-SR1)? Can it be implemented with no more than $2mn + 2n$ memory locations and $4mn + 2n$ multiplications per iteration?

Recent work by Byrd, Nocedal and Schnabel (1992) made the answer positive. They derived the compact representations for the SR1 update and pointed out the possibility of L-SR1 implementation using the compact form. We also note that Sally (1992) had an interesting test of L-SR1 method. He explicitly formed the L-SR1 matrix $B_k^{(\mathrm{LSR1})}$ in memory and used a modified Cholesky decomposition to make it safely positive definite. Then the line search strategy was used to safeguard the convergence. His result shows that the L-BFGS method clearly dominates. It is worth mentioning that he chose the initial scaling parameter γ_k to be $\|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$.

In this dissertation, we discuss the implementation of L-SR1 method. Different from the L-BFGS method which is line search based, our L-SR1 method is trust-region based. That is, at each iteration, we solve a trust region subproblem. Because of the special structure of L-SR1 matrix, the subproblem

can be solved very efficiently with a little bit more memory than the L-BFGS method.

In Chapter 1, we briefly summarize some fundamental issues in unconstrained optimization which are related to algorithms for large-scale problems. Then in Chapter 2, we prepare basic knowledge for limited memory method, especially L-BFGS and L-SR1. After that, we discuss the L-SR1 method in Chapter 3. Because of the importance of the initial scaling parameter γ_k, we devote the whole Chapter 4 to its analysis and computation. In Chapter 5, we analyze the memory and time expenses of our L-SR1 algorithm. In the same chapter, we also prove the global and Q-linear local convergence of the L-SR1 algorithm. These convergence results are actually true for general Hessian approximations as long as they are uniformly bounded. In Chapter 6, we do some numerical comparisons between the L-SR1 method and the L-BFGS method. Finally, we have a short summary in Chapter 7.

Except for explicitly specified, the norm $\|\cdot\|$ we use throughout the thesis means the Euclidean norm.

CHAPTER 2

BACKGROUND

Suppose we are given the unconstrained optimization problem

$$\begin{aligned} &\text{minimize} \quad f(x), \\ &x \in \mathcal{R}^n \end{aligned} \tag{2.1}$$

and we hope to solve it by iterative method. At the k-th iteration, we compute an acceptable step s_k and then advance the iterate from x_k to the next point $x_{k+1} = x_k + s_k$. Thus, the ascertainment of (2.1) can be reduced to find an acceptable s_k, which means

$$f(x_k + s_k) \leq f(x_k) + \alpha g_k^T s_k, \tag{2.2}$$

where $g_k = \nabla f(x_k)$ and α is a constant in $(0, 1)$ and usually we take $\alpha = 10^{-4}$. Following is the general optimization procedure. The focus of this thesis will be mainly in Step 3, how to compute s_k.[1]

Algorithm 2.1 OPT_FRAME (IN: f, x_0; OUT: x_*)

1. $k := 0$;
2. DO WHILE $\nabla f(x_k)$ is not close enough to zero
3. Compute s_k;
4. $x_{k+1} := x_k + s_k$;
5. $k := k + 1$;
6. END DO;
7. $x_* := x_k$;
8. return. { success }

□

[1]Sometimes, we also call s_k a step. A reader should not be confused with their meanings.

There are two widely used strategies for computing s_k, line search and trust region methods, which will be briefly addressed in the next section. These two strategies use a Hessian approximation matrix B_k to adjust the steepest descent direction $-g_k$. When B_k is the exact genuine Hessian, it is called Newton's method. When B_k is constructed with first order information, it is called quasi-Newton method. Usually, B_k is stored in memory with an $n \times n$ array.

We are interested in quasi-Newton methods only. Quasi-Newton Hessian[2] B_k is usually updated at every iteration by some formula of the form

$$B_{k+1} = \mathbf{U}(B_k, y_k, s_k), \tag{2.3}$$

where $y_k = g_{k+1} - g_k$ is the displacement of gradients. The two most widely used quasi-Newton methods are the well-known BFGS and the SR1, whose updating formulae are

$$\mathbf{U}^{BFGS}(B, y, s) = B - \frac{Bss^TB}{s^TBs} + \frac{yy^T}{y^Ts} \tag{2.4}$$

and

$$\mathbf{U}^{SR1}(B, y, s) = B + \frac{(y - Bs)(y - Bs)^T}{(y - Bs)^Ts}. \tag{2.5}$$

respectively.

For BFGS update (2.4), it is guaranteed that, if B_k is positive definite and $y_k^Ts_k > 0$, then the new matrix $B_{k+1} = \mathbf{U}^{BFGS}(B_k, y_k, s_k)$ is also positive definite. But for SR1 update (2.5), we don't have such a desired result even when the objective function $f(x)$ is strictly convex.

[2]Sometimes, we also call a quasi-Newton Hessian matrix B_k Hessian matrix. In situations where we need to differentiate between a real Hessian and a quasi-Hessian, an adjective like "real", "true" or "genuine" is added for the real Hessian $\nabla^2 f(x)$.

If the Hessian matrix B_k is positive definite, then $-B_k^{-1}g_k$ is a descent direction. We can use either line search or trust region method to compute s_k. But if B_k is indefinite, $-B_k^{-1}g_k$ is not necessarily a descent direction any more. The trust region strategy is then recommended.

2.1 Global Strategies

As we just mentioned, line search and trust region methods are two widely used strategies for computing s_k. Both are proved in practice very effective, and in theory to possess global and fast local convergence under appropriate conditions.

2.1.1 Line Search Method Let d_k be a descent direction of $f(x)$ at x_k, or $\nabla g_k^T d_k < 0$. Typical descent directions can be $-G_k^{-1}g_k$, $-B_k^{-1}g_k$ or just $-g_k$, if $G_k = \nabla^2 f(x_k)$ and B_k are positive definite. Find a steplength $\lambda_k \in (0, 1]$ such that

$$f(x_k + \lambda_k d_k) \leq f(x_k) + \alpha\lambda_k \nabla f(x_k)^T d_k, \tag{2.6}$$

$$\nabla f(x_k + \lambda_k d_k)^T d_k \geq \beta \nabla f(x_k)^T d_k, \tag{2.7}$$

where the two constants α and β satisfy $\alpha \in (0, \frac{1}{2})$ and $\beta \in (\alpha, 1)$. Then let $s_k = \lambda_k d_k$. We call (2.6) the α-condition, (2.7) the β-condition and (2.6), (2.7) together the **Wolfe** condition. The α-condition ensures that the function is reduced sufficiently, and the β-condition prevents the steps from being too small. It is easy to show that if d_k is a descent direction, $f(x)$ is continuously differentiable and $\{f(x_k + \lambda d_k) : \lambda > 0\}$ is bounded below, then there is a steplength $\lambda_k > 0$ which satisfies the Wolfe condition. Following is a sample algorithm which finds a $\lambda_k > 0$ such that both (2.6) and (2.7) are true.

Algorithm 2.2 LINE_SEARCH (IN: f, x_k, d_k; OUT: λ_k)

$\lambda := 1$, $\lambda_- := 1$, $\lambda_l := 0$, $\lambda_u := 0$;
DO WHILE either (2.6) or (2.7) is not satisfied
    IF (2.6) is satisfied THEN
        IF $\lambda_u = 0$ THEN
            $\lambda_+ := 2 * \lambda$;
        ELSE
            $\lambda_+ := 0.5 * (\lambda + \lambda_u)$;
        END IF;
    ELSE
        $\lambda_u := \lambda$;
        IF $\lambda \geq \lambda_-$ THEN $\lambda_l := \lambda_-$;
        backtrack λ_+ from (λ_l, λ_u);
    END IF;
    $\lambda_- := \lambda$, $\lambda := \lambda_+$;
END DO;
$\lambda_k := \lambda$;
return.

□

Note that the β-condition implies that

$$y_k^T s_k \geq -(1 - \beta_k) g_k^T s_k > 0.$$

So, the BFGS method with line search (2.6)-(2.7) retains the positive definiteness of its Hessian matrix. This is an important feature, because the line search direction d_k is usually computed by

$$d_k = -B_k^{-1} g_k.$$

When B_k is positive definite, the search direction is then automatically descent. Another important property for line search (2.6)-(2.7) is the admissibility of steplength one when x_k is close enough to the solution x_*, proved by Dennis and Moré (1974). This allows the q-superlinear convergence, which is desirable by all regular quasi-Newton methods. When we implement line search method, we always try $\lambda_k = 1$ first.

In practice, we usually ignore the β-condition, and backtrack λ until α-condition is satisfied. When a step which violates $y_k^T s_k > 0$ is detected, the update simply skip it. However, for Limited Memory methods, all information pairs $\{y_i, s_i\}$ are precious. None of them should be skipped imprudently. VA15, the Harwell subroutine for Limited Memory BFGS method, uses both α- and β-conditions.

2.1.2 Trust Region Method In view of the good performance of the steepest descent direction $-g_k$ when x_k is remote to solution and the quasi-Newton direction $-B_k^{-1}g_k$ when x_k is close to solution, it would be desirable to locate the new iterate in a direction like $-(B_k + \nu I)^{-1}g_k$, where the scalar ν is chosen so that the matrix $B_k + \nu I$ is positive definite. Hopefully, we have $\nu = 0$ when it is close to solution and $\nu > 0$ when $-B_k^{-1}g_k$ predicts an unexpected direction.

Precisely, at each iteration, we solve the quadratic trust region subproblem

$$\begin{aligned} &\text{minimize} \quad f(x_k) + \nabla f(x_k)^T s + \tfrac{1}{2} s^T B_k s \\ &\text{such that} \quad \|s\| \leq \delta_k \end{aligned} \tag{2.8}$$

where δ_k is the trust region radius. If we define the $\mathcal{R}^1 \to \mathcal{R}^n$ function

$$s_k(\nu) = -(B_k + \nu I)^{-1} \nabla f(x_k), \tag{2.9}$$

then when B_k is positive definite and $\|s_k(0)\| \leq \delta_k$, the Newton step $s_k(0)$ solves (2.8). Otherwise, some $\nu = \nu_k \geq \bar{\nu}_k$ such that

$$\|s_k(\nu)\| = \delta_k \tag{2.10}$$

can solve (2.8), where

$$\bar{\nu}_k = -\min\{0, \text{ all eigenvalues of } B_k\}.$$

We see that, for trust region method, the Hessian B_k does not necessarily have to be positive definite.

While the one-dimensional equation problem (2.10) is crucial to the success of trust region method, the update of δ_k also plays a significant role. We can define the ratio

$$\frac{\Delta f_k}{\Delta f_k^{(pred)}}$$

as the accuracy to which the quadratic model $f(x_k) + \nabla f(x_k)^T s_k + \frac{1}{2} s_k^T B_k s_k$ approximates $f(x_k + s_k)$, where

$$\begin{aligned} \Delta f_k^{(pred)} &= -\nabla f(x_k)^T s_k - \frac{1}{2} s_k^T B_k s_k, \\ \Delta f_k &= f(x_k) - f(x_k + s_k). \end{aligned}$$

Then we decrease the trust region radius when we have to and increase it adaptively when the quadratic model maintains a certain degree of agreement with the original function. We always try to make δ_k as large as possible (internal doubling). But when x_k is close to the solution, ideally the constraint of (2.8) should be inactive. Before proceeding to any specific method for solving (2.10), we would like to give a prototype algorithm for updating δ_k first.

Algorithm 2.3 TRUST_REGION_UPDATE_1 (IN: f, x_k, B_k, δ_k;
OUT: s_k, δ_{k+1})

1. compute s_k based on δ_k;
2. IF $f(x_k + s_k) > f(x_k) + \alpha g_k^T s_k$ THEN
3. DO
4. reduce δ_k;
5. compute s_k based on δ_k;
6. UNTIL $f(x_k + s_k) \leq f(x_k) + \alpha g_k^T s_k$;
7. ELSE IF s_k is not Newton step AND $\Delta f_k \geq \mu * \Delta f_k^{(pred)}$ THEN
8. DO
9. $\delta_- := \delta_k$, $s_- := s_k$;

10. $\delta_k := 2 * \delta_k$;
11. compute s_k based on δ_k;
12. UNTIL s_k is Newton step OR $\Delta f_k < \mu * \Delta f_k^{(pred)}$
13. IF $f(x_k + s_k) > f(x_k + s_-)$ THEN $s_k := s_-$;
14. END IF;
15. $\delta_{k+1} := \delta_k$;
16. return.

□

We will use Algorithm 2.3 for the L-SR1 algorithm. However, there is another trust region updating technique which evaluates the objective function once per iteration. For completion, we list it here as Algorithm 2.4, where μ and ξ are constants, $0 < \xi < \mu < 1$.

Algorithm 2.4 TRUST_REGION_UPDATE_2 (IN: f, x_k, B_k, δ_k; OUT: s_k, δ_{k+1})

1. compute s_k based on δ_k;
2. IF $\Delta f_k < \mu * \Delta f_k^{(pred)}$ THEN
3. $\delta_{k+1} := \|s_k\|/4$;
4. ELSE IF s_k is Newton step AND $\Delta f_k > (1 - \mu) * \Delta f_k^{(pred)}$ THEN
5. $\delta_{k+1} := 2\delta_k$;
6. ELSE
7. $\delta_{k+1} := \delta_k$;
8. END IF;
9. IF $\Delta f_k < \xi * \Delta f_k^{(pred)}$ THEN $s_k = 0$;
10. return.

□

Now we turn to ways to compute s_k based on the current trust region radius δ_k (Step 1, 5 and 11 of Algorithm 2.3). Since no finite method exists for solving (2.10), we have to consider computational alternatives. There are several successful strategies. Interested reader can refer to the book by Dennis

and Schnabel (1983). The most straightforward method is to eigendecompose B_k first and then solve a one-dimensional rootfinding problem. Basically, if we have the eigendecomposition of matrix B_k,

$$B_k = K_k \begin{bmatrix} \omega_1 & 0 & \cdots & 0 \\ 0 & \omega_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \omega_n \end{bmatrix} K_k^T \tag{2.11}$$

where K_k is an orthonormal matrix and all ω_i's are eigenvalues of B_k, then equation (2.10) can be reduced to

$$\phi_k(\nu) - \delta_k^2 = 0 \tag{2.12}$$

where

$$\phi_k(\nu) = \|s_k(\nu)\|^2 = \sum_{i=1}^{n} \frac{\bar{g}_i^2}{(\omega_i + \nu)^2}, \tag{2.13}$$

and

$$\begin{bmatrix} \bar{g}_1 \\ \bar{g}_2 \\ \vdots \\ \bar{g}_n \end{bmatrix} \equiv K_k^T g_k.$$

We can see that (2.12) can be easily solved by applying Newton's iterative method.

This method has an obvious drawback. The eigendecomposition (2.11) can be very expensive. When n is large ($n > 100$), it can make the algorithm unbearably slow. However, for specially structured B_k, its eigendecomposition may have some special way out. The discovery of compact forms of quasi-Newton matrices by Byrd, Nocedal and Schnabel (1992) makes it possible for all quasi-Newton methods to have a cheap eigendecomposition of their

Hessian matrices.

Finally, we give the algorithm to compute the step s_k.

Algorithm 2.5 TRUST_REGION_STEP (IN: B_k, g_k, δ_k; OUT: $s_k, newton$)

1. $[K, \Omega] :=$ eigendecomposition (B_k);
2. DEFINE function $\phi_k(\nu)$ by (2.13);
3. IF all $\omega_i > 0$ AND $\phi_k(0) \leq \delta_k^2$ THEN
4. $\quad \nu := 0$;
5. $\quad newton := TRUE$;
6. ELSE
7. $\quad$ find $\nu_k > \max\{0, -\omega_i\}$ from (2.12) by Newton's method;
8. $\quad newton := FALSE$;
9. END IF;
10. compute s_k by (2.9);
11. return.

□

2.2 Limited Memory BFGS Method

The BFGS update was discovered independently by Broyden, Fletcher, Goldfarb, and Shanno in 1970. Given $B_k^{(\mathrm{BFGS})}$, $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$, the new BFGS update for the next iteration is

$$B_{k+1}^{(\mathrm{BFGS})} = B_k^{(\mathrm{BFGS})} - \frac{B_k^{(\mathrm{BFGS})} s_k s_k^T B_k^{(\mathrm{BFGS})}}{s_k^T B_k^{(\mathrm{BFGS})} s_k} + \frac{y_k y_k^T}{y_k^T s_k}.$$

If $B_k^{(\mathrm{BFGS})}$ is symmetric positive definite and $y_k^T s_k > 0$, then $B_{k+1}^{(\mathrm{BFGS})}$ is also symmetric positive definite. Powell (1976) proved that, if $f(x)$ is strictly convex, then for any positive definite starting matrix B_0 and any starting point x_0, the BFGS method gives $\liminf \|g_k\| = 0$. The Dennis-Moré (1974) theory then guarantees the Q-superlinear convergence.

The BFGS method is fast and robust, and is currently being used to solve a myriad of optimization problems. A natural idea is then try to get a

limited memory method based on it. When studying L-BFGS, Nocedal (1980) considered the inverse BFGS formula of the form

$$H_{j+1}^{(\mathrm{BFGS})} = V_j^T H_j^{(\mathrm{BFGS})} V_j + \rho_j s_j s_j^T$$

and applied it m times recursively, where $\rho_j = 1/y_j^T s_j$ is a scalar and

$$V_j = I - \rho_j y_j s_j^T$$

is an $n \times n$ matrix. Formally, the inverse L-BFGS Hessian is

$$\begin{aligned} H_k^{(\mathrm{LBFGS})} = \quad & (V_{k-1}^T \cdots V_{k-m}^T) H_k^{(0)} (V_{k-m} \cdots V_{k-1}) \\ + \quad & \rho_{k-m} (V_{k-1}^T \cdots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\ + \quad & \rho_{k-m+1} (V_{k-1}^T \cdots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) \\ + \quad & \cdots \\ + \quad & \rho_{k-1} s_{k-1} s_{k-1}^T. \end{aligned}$$

It looks bizarre. But the computation of line search direction $d_k = -H_k^{(\mathrm{LBFGS})} g_k$ can be very efficient. Following is the famous two-loop recursion pseudo-code for the case when $H_k^{(0)} = \gamma_k I$. The line search direction is finally stored in vector d_k.

Algorithm 2.6 L-BFGS_DIRECTION

(IN: $H_k^{(0)}, g_k, \rho_{k-i}|_{i=2}^m, \{y_{k-i}, s_{k-i}\}_{i=1}^m$; OUT: d_k)

$\rho_{k-1} = 1/y_{k-1}^T s_{k-1}$
$q = g_k$
DO FOR $i = 1, 2, \ldots, m$
    $\alpha_{k-i} = \rho_{k-i} s_{k-i}^T q$
    $q = q - \alpha_{k-i} y_{k-i}$
END DO;
$d_k = H_k^{(0)} q$

DO $i = m, m-1, \ldots, 1$
$\quad \beta = \rho_{k-i} y_{k-i}^T d$
$\quad d_k = d_k + (\alpha_{k-i} - \beta) s_{k-i}$
END DO;
return.

□

Note that, we don't have to calculate all $\rho_{k-i}, i = 1, 2, \ldots, m$, every time when Algorithm 2.6 is called. There are always $(m-1)$ of them can be carried from iteration to iteration. So, if $H_k^{(0)}$ is diagonal, a total of $4mn + 2n$ multiplications and $2mn + 2n$ memory locations are needed for step matrix computation.

The L-BFGS method is considered a successful new tool in solving large-scale unconstrained optimization problems. Liu and Nocedal (1989) showed its R-linear convergence and reported an obvious win of L-BFGS over Buckley and LeNir's (1983) Variable Storage Conjugate Gradient method. Nash and Nocedal (1991)'s work revealed a close tie with Truncated Newton's method. L-BFGS has been collected into Harwell Subroutine Library as VA15.

L-BFGS method can be also implemented through the compact form of BFGS formula. Its computation cost and memory requirement can be the same as those of the two-loop recursion. One can refer to Byrd, Nocedal and Schnabel (1992) for details.

2.3 Standard SR1 Method

The Symmetric Rank One update (2.5) was first introduced by Davidon (1959) and then re-discovered by several other researchers about ten years

later. It is the only symmetric rank one method which satisfies the quasi-Newton condition $B_{k+1}s_k = y_k$. Although its formula is simpler, it does share some nice properties with other quasi-Newton methods. The first property below is even unique to SR1 update.

Properties: Suppose the objective function f is a strictly convex quadratic function, $f(x) = \frac{1}{2}x^T Gx$, where G is a symmetric positive definite matrix, and the Hessian matrices are defined by

$$B_{k+1}^{(\mathrm{SR1})} = B_k^{(\mathrm{SR1})} + \frac{(y_k - B_k^{(\mathrm{SR1})}s_k)(y_k - B_k^{(\mathrm{SR1})}s_k)^T}{(y_k - B_k^{(\mathrm{SR1})}s_k)^T s_k}. \tag{2.14}$$

(1) If we take quasi-Newton step $s_k = -[B_k^{(\mathrm{SR1})}]^{-1}g_k$ at each iteration and all the steps are linearly independent, then the SR1 method terminates within $n+1$ iterations with $B_{n+1}^{(\mathrm{SR1})} = G$.

(2) SR1 method with exact line searches terminates within $n_1 \leq n$ iterations and the following is true for $j = 1, 2, \ldots, n_1$

$$B_{j+1}^{(\mathrm{SR1})}s_i = y_i, \quad i = 1, 2, \ldots, j$$

$$s_i^T G s_j = 0, \quad i = 1, 2, \ldots, j-1.$$

Also, if $n_1 = n$, then $B_{n+1}^{(\mathrm{SR1})} = G$.

(3) SR1 method is scaling invariant. That is, for a general objective function f, if we make a nonsingular transformation $\tilde{x} = Ax$ and apply the same method to the new optimization problem $\min_{\tilde{x}\in\mathcal{R}^n} f(A^{-1}\tilde{x})$ with initial $\tilde{x}_0 = Ax_0$ and $\tilde{B}_0^{(\mathrm{SR1})} = A^{-T}B_0^{(\mathrm{SR1})}A^{-1}$, then $\tilde{x}_k = Ax_k$ for all $k \geq 1$.

SR1 method had been taken for granted as a method of "fatal flaw" since its discovery, even though careful numerical computation suggests a good

perspective. Nevertheless, it is understandable. SR1 update does not necessarily produce a positive definite matrix $B_{k+1}^{(\mathrm{SR1})}$ when $B_k^{(\mathrm{SR1})}$ is positive definite and $y_k^T s_k > 0$, and in fact, the denominator $\left(y_k - B_k^{(\mathrm{SR1})} s_k\right)^T s_K$ can be suddenly exactly zero when we apply the method to a strictly convex quadratic problem.

Recently, several new theoretical and experimental studies have reversed the traditional prejudice. Instead of being considered inherently weak, it is now regarded by many researchers as a serious challenger to BFGS method, especially in applications where the positive definiteness of BFGS matrix is hard to retain.

Conn, Gould and Toint (1991) showed that, if the sequence of steps $\{s_k\}$ is uniformly linearly independent (u.l.i.), and

$$\left(y_k - B_k^{(\mathrm{SR1})} s_k\right)^T s_k \neq 0 \tag{2.15}$$

and

$$\cos\theta_k \equiv \frac{\left|\left(y_k - B_k^{(\mathrm{SR1})} s_k\right)^T s_k\right|}{\left\|y_k - B_k^{(\mathrm{SR1})} s_k\right\| \|s_k\|} > \rho \tag{2.16}$$

for some constant $\rho \in (0,1)$, then the Hessian approximations $\left\{B_k^{(\mathrm{SR1})}\right\}$ generated by SR1 formula converge globally to the true Hessian $\nabla_x^2 f(x_*)$. It is really a unique result because for all other superlinearly convergent quasi-Newton methods, one can expect at most

$$\left\|\left[B_k^{(\mathrm{SR1})} - \nabla_x^2 f(x_k)\right] s_k\right\| = o(\|s_k\|).$$

Khalfan, Byrd and Schnabel (1990) proved that, SR1 method is actually $(n+1)$-step Q-superlinearly and $(2n)$-step Q-quadratically convergent even with the u.l.i. condition replaced by the boundedness of $\left\|B_k^{(\mathrm{SR1})}\right\|$.

Numerical experiments indicate that SR1 method appears to be substantially competitive with other quasi-Newton methods, like BFGS, especially when trust region method is used to solve subproblems. In some cases, SR1 method shows slight superiority over BFGS method. One of the possible reasons is that when an iterate is remote to a local optimizer, there is no reason to assume any positive definiteness of the true Hessian at that point. Therefore, any approximation of positive definiteness (e.g. BFGS) does not reflect the fact and might prevent the iterate from a longer step.

Since we haven't known much about the behavior of denominator $\left(y_k - B_k^{(\mathrm{SR1})} s_k\right)^T s_k$ yet, it is suggested that the safeguard check

$$\left|\left(y_k - B_k^{(\mathrm{SR1})} s_k\right)^T s_k\right| > \rho \left\|y_k - B_k^{(\mathrm{SR1})} s_k\right\| \|s_k\| \tag{2.17}$$

be enforced at each iteration. Whenever a violation is detected, the update is skipped, or $B_{k+1}^{(\mathrm{SR1})} = B_k^{(\mathrm{SR1})}$. Note that, condition (2.17) includes both (2.15) and (2.16). One could refer to Khalfan, Byrd and Schnabel (1990) for more implementation skills.

It is worth mentioning here that SR1 method is self-dual, that is, the inverse SR1 matrix $H_k^{(\mathrm{SR1})} = [B_k^{(\mathrm{SR1})}]^{-1}$ can be computed recursively by a similar formula

$$H_{k+1}^{(\mathrm{SR1})} = H_k^{(\mathrm{SR1})} + \frac{\left(s_k - H_k^{(\mathrm{SR1})} y_k\right)\left(s_k - H_k^{(\mathrm{SR1})} y_k\right)^T}{\left(s_k - H_k^{(\mathrm{SR1})} y_k\right)^T y_k}. \tag{2.18}$$

2.4 Limited Memory SR1 Method

Generally, let B_k be a quasi-Newton matrix defined by (2.3). Note that this is a recursively defined matrix. The raw data are information pairs $\{y_i, s_i\}_{i=0}^{k-1}\}$. To get a formula of B_k with respect to the raw data, we have to

recursively apply (2.3) k times. Now we recursively apply (2.3) just a certain number of times, saying m. Then we get a formula for B_k depending only on B_{k-m} and the m most recent vector pairs $\{y_i, s_i\}_{i=k-m}^{k-1}$:

$$B_k = \underbrace{\mathbf{U}(\mathbf{U}(\cdots \mathbf{U}(}_{m} B_{k-m}, y_{k-m}, s_{k-m}), \cdots), y_{k-1}, s_{k-1}). \tag{2.19}$$

Instead of storing the matrices B_k, we store in memory the "initial matrix" B_{k-m} and the m most recent vector pairs $\{y_i, s_i\}_{i=k-m}^{k-1}$. After computing the new iterate, the oldest pair $\{y_{k-m}, s_{k-m}\}$ is deleted from the set and is replaced by the newest one $\{y_k, s_k\}$. B_{k-m} in (2.19) is usaually rewritten as $B_{k,0}$.

If we choose $B_{k,0}$ to be $B_{k,0} = \gamma_k I$ where γ_k is a positive number, then B_k can be stored implicitly with $2mn + 1$ locations of memory. Compared to the original n^2, limited memory methods save memory significantly, if $m \ll n$.

If we apply the SR1 update (2.14) m times recursively to itself, we get the so-called L-SR1 formula. According to Byrd, Nocedal and Schnabel (1992), the L-SR1 formula can be written in a compact form:

$$B_k^{(\mathrm{LSR1})} = \gamma_k I + Q_k M_k^{-1} Q_k^T, \tag{2.20}$$

where

$$\begin{aligned} Q_k &= Y_k - \gamma_k S_k, \\ M_k &= W_k - \gamma_k S_k^T S_k, \\ S_k &= [s_{k-m} \ \cdots \ s_{k-2} \ s_{k-1}], \\ Y_k &= [y_{k-m} \ \cdots \ y_{k-2} \ y_{k-1}], \end{aligned} \tag{2.21}$$

and

$$[W_k]_{ij} = \begin{cases} y_{i+k-m-1}^T s_{j+k-m-1} & \text{if } i \le j, \\ y_{j+k-m-1}^T s_{i+k-m-1} & \text{if } i > j. \end{cases} \tag{2.22}$$

However, in that paper, the authors did not give any implementation details on step computation. They did point out that condition (2.15) is equivalent to the nonsingularity of the principal minors of M_k, which could be tested when computing a triangular factorization of M_k without pivoting. But they failed to find out the counterpart of condition (2.16).

Matrix W_k is interesting. It is symmetric and its upper triangular is identical to that of $Y_k^T S_k$. We now know that, an SR1 update is valid only if all its previous denominators are nonzero, which is equivalent to the nonsingularity of all principal minors of M_k. But even if some principal minors of M_k are singular, the whole matrix M_k may still be invertible. That means, even if some previous denominators are zero, the current SR1 matrix can still be well defined through the compact form (2.20). This reveals that SR1 update is kind of "self-adjustable" and explains why the SR1 matrix update should be rarely skipped.

The inverse formula of $B_k^{(\mathrm{LSR1})}$ can also be obtained by applying the inverse SR1 update (2.18) m times recursively. We will use it to choose an appropriate γ_k to make $B_k^{(\mathrm{LSR1})}$ positive definite in the next chapter.

$$H_k^{(\mathrm{LSR1})} = \frac{1}{\gamma_k} I + \left(S_k - \frac{1}{\gamma_k} Y_k \right) \left(\tilde{W}_k - \frac{1}{\gamma_k} Y_k^T Y_k \right)^{-1} \left(S_k - \frac{1}{\gamma_k} Y_k \right)^T, \quad (2.23)$$

where

$$[\tilde{W}_k]_{ij} = \begin{cases} y_{j+k-m-1}^T s_{i+k-m-1} & \text{if } i \le j, \\ y_{i+k-m-1}^T s_{j+k-m-1} & \text{if } i > j. \end{cases} \quad (2.24)$$

Note that $\tilde{W}_k$ is a symmetric matrix whose lower triangular is identical to that of $Y_k^T S_k$.

CHAPTER 3

DESCRIPTION OF THE L-SR1 METHOD

In the beginning of Chapter 2, we gave a program frame for general optimization algorithm, Algorithm 2.1, where Step 3 will be carried out by Algorithm 2.3 for our L-SR1 method. We also mentioned there that we would like to use Algorithm 2.5 for Step 1, 5 and 11 of Algorithm 2.3. In this chapter, we will give the frame of L-SR1 algorithm first, and then describe some of Algorithm 2.3 and 2.5 in details, especially the way to eigendecompose Hessian[1] B_k (Step 1 of Algorithm 2.5) and the way to compute s_k (Step 11 of Algorithm 2.5). We will leave the computation of initial γ_k to the next chapter.

3.1 The L-SR1 Algorithm

The L-SR1 algorithm is basically a trust region method (Algorithm 2.3) with a different way to compute the step s_k. Following is the algorithm, where α and μ are two constants. We take the value $\alpha = 10^{-4}$ and $\mu = \frac{1}{2}$.

Algorithm 3.1 L-SR1 (IN: f, x_0; OUT: x_*)

1. $k := 0$;
2. DO WHILE g_k is not close enough to zero
3. compute initial γ_k;
4. finiteness assurance and index adjustment;
5. compute s_k based on δ_k;
6. IF $f(x_k + s_k) > f(x_k) + \alpha g_k^T s_k$ THEN
7. $\delta_{orig} := \delta_k$;
8. DO

[1] From this Chapter on, we will use B_k exclusively for $B_k^{(\text{L-SR1})}$ unless explicitly specified.

9. reduce δ_k;
10. compute s_k based on δ_k;
11. UNTIL $f(x_k + s_k) \leq f(x_k) + \alpha g_k^T s_k$;
12. $\delta_{k+1} :=$ max $(\delta_k, 0.05\delta_{orig})$;
13. ELSE IF $\|s_k\| = \delta_k$ AND $\Delta f_k \geq \mu * \Delta f_k^{(pred)}$ THEN
14. DO
15. $\delta_- := \delta_k$, $s_- := s_k$;
16. $\delta_k := 2\delta_k$;
17. compute s_k based on δ_k;
18. UNTIL $\|s_k\| < \delta_k$ OR $\Delta f_k < \mu * \Delta f_k^{(pred)}$
19. $\delta_{k+1} := \delta_-$;
20 IF $f(x_k + s_k) > f(x_k + s_-)$ THEN $s_k := s_-$;
21. END IF;
22. $x_{k+1} := x_k + s_k, g_{k+1} := \nabla f(x_{k+1})$;
23. update Y_k and S_k; { and others }
24. $k := k + 1$;
25. END DO;
26. $x_* := x_k$;
27. return.

□

The computation of initial γ_k (Step 3) is very important for our study. We will devote all of Chapter 4 to this issue. For now, we can assume it is a positive number. The finiteness assurance and index adjustment (Step 4) deals with a cosine value condition similar to (2.16) and index adjustment if the condition fails. This will be covered in Section 3.2. Since the L-SR1 matrix has special structure, its eigendecomposition and step assembly can be much cheaper than its original version (Algorithm 2.5). We will talk about step computation (Step 5, 10 and 17) in Section 3.3. The last section of this chapter will discuss the trust region radius reduction when the current step is not acceptable.

3.2 Finiteness Assurance and Index Adjustment

Since the standard SR1 method performs well only if both (2.15) and (2.16) hold, a natural question is then: How to test (2.15) and (2.16) with the compact form (2.20)? For standard SR1 method, we have $B_k^{(\mathrm{SR1})}$ in memory and therefore the test of (2.15) and (2.16) is just some simple matrix/vector operations. But for L-SR1 method, we don't have any recursively-defined intermediate matrices $B_{k,j}^{(\mathrm{LSR1})}$ in memory. More importantly, we don't want to have any expensive extra cost on computation. In this section, we will talk about how to test the cosine value condition efficiently and how to adjust the Hessian when some defects are detected.

3.2.1 Finiteness Assurance The recursively-defined version of $B_k^{(\mathrm{LSR1})}$ can be written as

$$B_{k,j+1}^{(\mathrm{LSR1})} = B_{k,j}^{(\mathrm{LSR1})} + \frac{1}{\eta_{k,j}} r_{k,j} r_{k,j}^T, \tag{3.1}$$

where

$$r_{k,j} = y_{k-m+j} - B_{k,j}^{(\mathrm{LSR1})} s_{k-m+j}, \tag{3.2}$$

and

$$\eta_{k,j} = r_{k,j}^T s_{k-m+j}. \tag{3.3}$$

With $B_{k,0}^{(\mathrm{LSR1})} = \gamma_k I$ given, we will arrive at the identity $B_k^{(\mathrm{LSR1})} = B_{k,m}^{(\mathrm{LSR1})}$. See Byrd, Nocedal and Schnabel (1992) for details.

If we define

$$\cos\theta_{k,j} = \frac{\eta_{k,j}}{\|r_{k,j}\|\|s_{k-m+j}\|}, \tag{3.4}$$

then the counterparts of (2.15) and (2.16) can then be expressed officially by

$$\eta_{k,j} \neq 0 \quad \text{and} \quad |\cos\theta_{k,j}| > \rho, \tag{3.5}$$

or

$$|\eta_{k,j}| > \rho \|r_{k,j}\| \|s_{k-m+j}\|, \tag{3.6}$$

for all $j = 0, 1, \ldots, m-1$. We call (3.6) the **finiteness assurance** of the L-SR1 method. Since $B_{k,j}^{(\mathrm{LSR1})}$ is not explicitly stored in memory, the finiteness assurance can not be easily executed as in the implementation of standard SR1 method.

Before stating the theorem that leads to the algorithm for computing both $\eta_{k,j}$ and $|\cos\theta_{k,j}|$, we introduce a special notation. Let E be a matrix. We use $[E]_{i_1:i_2,j_1:j_2}$ to denote the $(i_2 - i_1 + 1) \times (j_2 - j_1 + 1)$ submatrix of E that is the intersection block of rows i_1 through i_2 with columns j_1 through j_2. We use $[E]_{i,j_1:j_2}$ to denote the $(j_2 - j_1 + 1)$-dimensional row vector which consists of j_1-th through j_2-th elements on row i. We use $[E]_{i_1:i_2,j}$ to denote the $(i_2 - i_1 + 1)$-dimensional column vector which consists of i_1-th through i_2-th elements on column j. The notation $[E]_{subscript}^{superscript}$ always represents for $([E]_{subscript})^{superscript}$.

Theorem 3.2 Given $m \times m$ symmetric square matrix M_k defined by (2.21). If all its leading submatrices are nonsingular, then there is a unique $m \times m$ unit lower triangular matrix L_k and a unique $m \times m$ diagonal matrix H_k such that

$$M_k = L_k^{-1} A_k L_k^{-T}. \tag{3.7}$$

Moreover,

$$A_k = \text{ diag } (\eta_{k,0}, \eta_{k,1}, \ldots, \eta_{k,m-1}). \tag{3.8}$$

Proof:

We prove the uniqueness first. Suppose M_k has another decomposition of the form (3.7),

$$M_k = \tilde{L}_k^{-1} \tilde{A}_k \tilde{L}_k^{-T},$$

where $\tilde{L}_k$ is a unit lower triangular matrix and $\tilde{H}_k$ is a diagonal matrix. Then

$$\tilde{L}_k L_k^{-1} A_k = \tilde{A}_k \tilde{L}_k^{-T} L_k^T. \tag{3.9}$$

The left hand side of (3.9) is a lower triangular matrix while the right hand side an upper triangular one. So both sides of (3.9) are actually a diagonal matrix. That means $\tilde{L}_k L_k^{-1}$ is a diagonal matrix with all diagonal entries being one. Thus $L_k = \tilde{L}_k$ and furthermore $A_k = \tilde{A}_k$.

To prove the existence of the decomposition, we make the following notations for the reason of simplicity.

$$\begin{aligned} M_{k,j} &= [M_k]_{1:j+1,1:j+1}, \\ Q_{k,j} &= [Q_k]_{1:n,1:j+1}, \\ u_{k,j} &= Q_{k,j-1}^T s_{k-m+j}, \\ \alpha_{k,j} &= [Q_k]_{1:n,j+1}^T s_{k-m+j}. \end{aligned}$$

where, for a certain j, $M_{k,j}$ is a $(j+1) \times (j+1)$ square matrices; $Q_{k,j}$ is an $n \times (j+1)$ matrix; $u_{k,j}$ is j-vectors; and $\alpha_{k,j}$ is a scalar. The index j can run from 0 through $m-1$.

For $j = 0, 1, \ldots, m-1$, let $H_{k,j}$ be the $(j+1) \times (j+1)$ diagonal matrix defined by

$$A_{k,j} = \text{ diag } (\eta_{k,0}, \eta_{k,1}, \ldots, \eta_{k,j}),$$

and $L_{k,j}$ the $(j+1) \times (j+1)$ unit lower triangular matrix defined recursively

by

$$L_{k,j} = \begin{bmatrix} L_{k,j-1} & 0 \\ l_{k,j}^T & 1 \end{bmatrix}$$

where $L_{k,-1}$ and $l_{k,0}$ can be considered empty (Thus, $L_{k,0} = [1]$) and

$$l_{k,j} = -[M_k]_{1:j,1:j}^{-1} Q_{k,j-1}^T s_{k-m+j}. \tag{3.10}$$

What we will prove is actually a more general result:

$$M_{k,j} = L_{k,j}^{-1} A_{k,j} L_{k,j}^{-T} \tag{3.11}$$

for $j = 0, 1, \ldots, m-1$. The existence of the decomposition (3.7) is simply the special case of (3.11) when $j = m-1$. We prove (3.11) by induction.

The correctness for $j = 0$ is trivial. Now suppose (3.11) is also true for $j-1$, i.e.,

$$M_{k,j-1} = L_{k,j-1}^{-1} A_{k,j-1} L_{k,j-1}^{-T}.$$

We can verify that

$$M_{k,j} = \begin{bmatrix} M_{k,j-1} & u_{k,j} \\ u_{k,j}^T & \alpha_{k,j} \end{bmatrix},$$

and

$$\begin{aligned} L_{k,j} M_{k,j} L_{k,j}^T &= \begin{bmatrix} L_{k,j-1} & 0 \\ l_{k,j}^T & 1 \end{bmatrix} \begin{bmatrix} M_{k,j-1} & u_{k,j} \\ u_{k,j}^T & \alpha_{k,j} \end{bmatrix} \begin{bmatrix} L_{k,j-1}^T & l_{k,j} \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} L_{k,j-1} M_{k,j-1} L_{k,j-1}^T & 0 \\ 0 & \alpha_{k,j} + u_{k,j}^T l_{k,j} \end{bmatrix}. \end{aligned}$$

But the identity

$$B_{k,j}^{(\mathrm{LSR1})} = \gamma_k I + Q_{k,j-1} M_{k,j-1}^{-1} Q_{k,j-1} \tag{3.12}$$

together with (3.2) and (3.3) implies that

$$\eta_{k,j} = \alpha_{k,j} + u_{k,j}^T l_{k,j}.$$

Therefore,

$$M_{k,j} = L_{k,j}^{-1} A_{k,j} L_{k,j}^{-T}.$$

□

We may call (3.7) the **inverse LDL^T decomposition** of M_k, since it can be implemented block by block forwards, a similar pattern as in doing LDL^T decomposition. We show the algorithm skeleton in Algorithm 3.3. The cost of the inverse LDL^T decomposition of an $m \times m$ matrix is $\frac{1}{6}m^3$ **flops**, the same as LDL^T decomposition.

Algorithm 3.3 INVERSE_LDL^T (IN: M; OUT: L, A)

FOR $j = 0$ TO $m - 1$ DO
  $L_{j+1,1:j}^T := -L_{1:j,1:j}^T A_{1:j,1:j}^{-1} L_{1:j,1:j} M_{1:j,j+1}$;
  $A_{j+1,j+1} := L_{j+1,1:j} M_{1:j,j+1} + M_{j+1,j+1}$;
END FOR;
return.

□

The inverse LDL^T decomposition can be carried out to the end if and only if all the leading submatrices of M_k are nonsingular. Since we can not make any assumption other than symmetry on M_k, numerical stability can be a problem. However, this decomposition is purely for finiteness assurance. Few digits accuracy is actually good enough for us. Also, after each step of Algorithm 3.3, we will check condition (3.6), and then reshuffle the order of information pairs $\{p_{k-i}, s_{k-i}\}_{i=1}^m$ so that no zero will show up on the diagonal of A_k. We will discuss this in detail in the next subsection.

Theorem 3.4 Suppose the matrix M_k has inverse LDL^T decomposition (3.7), where L_k is a unit lower triangular matrix and A_k is a diagonal matrix. Then the cosine values defined by (3.4) can be computed by

$$\cos\theta_{k,j} = \frac{[A_k]_{j+1,j+1}}{\|s_{k-m+j}\|\sqrt{[L_k]_{j+1,1:m}[Q_k^TQ_k]\,[L_k]_{j+1,1:m}^T}}. \tag{3.13}$$

Proof:

By (3.2) and (3.12),

$$\begin{aligned} r_{k,j} &= y_{k-m+j} - B_{k,j}^{(\mathrm{LSR1})}s_{k-m+j} \\ &= [Q_k]_{1:n,j+1} + Q_{k,j-1}l_{k,j} \\ &= Q_{k,j}\begin{bmatrix} l_{k,j} \\ 1 \end{bmatrix}, \end{aligned}$$

where the symbols $Q_{k,j}$ and $l_{k,j}$ are defined in the proof of the previous theorem. Then

$$\|r_{k,j}\| = \|Q_k[L_k]_{j+1,1:m}^T\|.$$

So,

$$\begin{aligned} \cos\theta_{k,j} &= \frac{\eta_{k,j}}{\|r_{k,j}\|\|s_{k-m+j}\|} \\ &= \frac{[A_k]_{j+1,j+1}}{\|s_{k-m+j}\|\sqrt{[L_k]_{j+1,1:m}[Q_k^TQ_k]\,[L_k]_{j+1,1:m}^T}}. \end{aligned}$$

□

In reality, we prefer to using a scaled version of M_k and Q_k for the finiteness assurance. Let

$$D_k = \mathrm{diag}(\|s_{k-m}\|, \cdots, \|s_{k-2}\|, \|s_{k-1}\|). \tag{3.14}$$

Define the scaled version of Q_k and M_k by

$$\begin{aligned} \bar{Q}_k &= Q_kD_k^{-1} \\ \bar{M}_k &= D_k^{-1}M_kD_k^{-1}. \end{aligned}$$

Then the L-SR1 matrix can be rewritten as

$$B_k = \gamma_k I + \bar{Q}_k \bar{M}_k^{-1} \bar{Q}_k^T.$$

Suppose we have the inverse LDL^T decompositions to matrices M_k and $\bar{M}_k$,

$$\begin{aligned} M_k &= L_k^{-1} A_k L_k^{-T}, \\ \bar{M}_k &= \bar{L}_k^{-1} \bar{A}_k \bar{L}_k^{-T}. \end{aligned}$$

Since

$$\begin{aligned} \bar{M}_k &= D_k^{-1} M_k D_k^{-1} \\ &= D_k^{-1} L_k^{-1} A_k L_k^{-T} D_k^{-1} \\ &= (D_k^{-1} L_k D_k)^{-1} (D_k^{-1} A_k D_k^{-1}) (D_k^{-1} L_k D_k)^{-T}, \end{aligned}$$

by the uniqueness of inverse LDL^T decomposition, we must have that

$$\begin{aligned} L_k &= D_k \bar{L}_k D_k^{-1} \\ A_k &= D_k \bar{A}_k D_k. \end{aligned}$$

Thus,

$$\begin{aligned} \cos\theta_{k,j} &= \frac{[D_k \bar{A}_k D_k]_{j+1,j+1}}{\|s_{k-m+j}\| \sqrt{[D_k \bar{L}_k D_k^{-1}]_{j+1,1:m} [Q_k^T Q_k] [D_k \bar{L}_k D_k^{-1}]_{j+1,1:m}^T}} \\ &= \frac{[\bar{A}_k]_{j+1,j+1}}{\sqrt{[\bar{L}_k]_{j+1,1:m} [D_k^{-1} Q_k^T Q_k D_k^{-1}] [\bar{L}_k]_{j+1,1:m}^T}}, \end{aligned}$$

and the cosine value condition (3.6) can be finally reduced to

$$[\bar{A}_k]_{j+1,j+1}^2 > \rho^2 \cdot [\bar{L}_k]_{j+1,1:m} [D_k^{-1} Q_k^T Q_k D_k^{-1}] [\bar{L}_k]_{j+1,1:m}^T, \tag{3.15}$$

which is the condition we will use in real computation.

3.2.2 Index Adjustment We discuss how to efficiently incorporate the process of inverse LDL^T decomposition and finiteness assurance together to fix possible defects of the L-SR1 matrix B_k.

To start, let's take a look at the mechanism how B_k is stored and updated in memory. Since we don't have the explicit expression of B_k in memory, B_k is actually represented by some other quantities. Logically, if we have Y_k, S_k and γ_k in memory, the whole matrix B_k and therefore all its related computations can be retrieved through the $(2mn + 1)$ memory locations.

First, we store the $n \times m$ array Y_k in $\mathcal{L}_Y$ and S_k in $\mathcal{L}_S$. The order is maintained through an index table I_k. Thus, the actual slots for y_{k-j} and s_{k-j} will be the $I_k(m-j+1)$-th columns of $\mathcal{L}_Y$ and $\mathcal{L}_S$, respectively. When B_k is updated, we replace the $I_k(1)$-th column of $\mathcal{L}_Y$ by the new y_k and the same column of $\mathcal{L}_S$ by s_k. Then let $I_k(j) = I_k(j+1)$ for $j = 1, 2, \ldots, m-1$ and $I_k(m)$ point to the position where the previous $I_k(1)$ points to.

Second, as we pointed out in the previous subsection, the inverse LDL^T decomposition is carried out in a block-by-block-forward pattern. If we examine the procedure carefully, we can find out that the j-th step touches information pairs $\{y_{k-i}, s_{k-i}\}_{i=m-j}^{m}$ only. At the same time, the j-th finiteness assurance touches only the same information pairs. The rest, $\{y_{k-i}, s_{k-i}\}_{i=1}^{m-j-1}$, remains intact. So, we don't have to start finiteness assurance after the finish of LDL^T decomposition. And also, if at some moment j the finiteness assurance fails, we don't have to drop the current information pair (y_{k-m+j}, s_{k-m+j}). But instead, we can keep them by temporarily swapping them with some other pair $\{y_{k-m+c}, s_{k-m+c}\}$, $c < j$, which can make the finiteness assurance pass. If no such pair exists, we ignore all the remaining information and reduce m

to the real size. We believe this is a better strategy than simply dropping (y_{k-m+j}, s_{k-m+j}). The swapping can be easily carried out by using the index table I_k.

So, our m is actually not a constant, it can vary from iteration to iteration. A better representation can be two symbols m_k and m_{fix} with m_k being the real size and m_{fix} being the maximum size. But using the single symbol m does not seem to bother our presentation and analysis. We will keep the way as it is.

Following is the algorithm for both finiteness assurance and adjustment of Hessian B_k. For simplicity, the index technique is not shown on matrix/vector operations.

Algorithm 3.5 FINITENESS (IN: $D_k, M_k, Q_k^T Q_k$; OUT: I_k)

$j := candidate := 0$;
$\bar{M}_k := D_k^{-1} M_k D_k^{-1}$;
DO WHILE $j < m$
  $[\bar{L}_k]^T_{j+1,1:j} := -[\bar{L}_k]^T_{1:j,1:j}[\bar{A}_k]^{-1}_{1:j,1:j}[\bar{L}_k]_{1:j,1:j}[\bar{M}_k]_{1:j,j+1}$;
  $[\bar{A}_k]_{j+1,j+1} := [\bar{L}_k]_{j+1,1:j}[\bar{M}_k]_{1:j,j+1} + [\bar{M}_k]_{j+1,j+1}$;
  IF (3.15) is true THEN
    $j := candidate := j + 1$;
  ELSE
    $candidate := candidate + 1$;
    IF $candidate < m$ THEN
      swap $I_k(j+1)$ with $I_k(candidate + 1)$;
    ELSE
      $m := j$;
      break DO;
    END IF;
  END IF;
END DO;
return.

□

Obviously, the matrix M_k after finiteness assurance and index adjustment is invertible.

3.3 Computation of Step s_k

As we see from Section 2.1.2 that, the eigendecomposition of Hessian B_k is required to solve the trust region subproblem, which usually costs tremendous computation. However, if we take advantage of the special structure of L-SR1 matrix, the cost can be lowered significantly.

Let D_k be the diagonal scaling matrix defined by (3.14). Since $D_k^{-1}Q_k^TQ_kD_k^{-1}$ is symmetric semi-positive definite, it has singular-value decomposition of the form

$$D_k^{-1}Q_k^TQ_kD_k^{-1} = U_k\Sigma_k^2U_k^T, \tag{3.16}$$

where Σ_k is an $m \times m$ diagonal matrix and U_k an orthonormal matrix.

Then the matrix $\Sigma_kU_k^TD_kM_k^{-1}D_kU_k\Sigma_k$ is also symmetric. Let t be its rank. It can be eigendecomposed into the form

$$\Sigma_kU_k^TD_kM_k^{-1}D_kU_k\Sigma_k = V_k\Lambda_kV_k^T, \tag{3.17}$$

where Λ_k is a $t \times t$ nonsingular diagonal matrix and V_k an $m \times t$ orthonormal matrix.

We claim that the diagonal entries of matrix $\gamma_kI + \Lambda_k$ are B_k's eigenvalues and the columns of the matrix

$$P_k = Q_kM_k^{-1}D_kU_k\Sigma_kV_k\Lambda_k^{-1}$$

are the corresponding eigenvectors. All the other eigenvalues of B_k are γ_k. To prove the claim, let Z_k be an $n \times (n-t)$ orthonormal matrix spanning the null space of Q_k, i.e. $Q_k^TZ_k = 0$ and $Z_k^TZ_k = I$.

Theorem 3.6 The $n \times n$ matrix K_k defined by

$$K_k = [P_k \;\; Z_k]$$

is orthonormal and the Hessian B_k defined by (2.20) can be eigendecomposed as

$$B_k = K_k \begin{bmatrix} \gamma_k I + \Lambda_k & 0 \\ 0 & \gamma_k I \end{bmatrix} K_k^T.$$

Proof:

The matrix K_k is obviously orthonormal if we notice that $P_k^T P_k = I$, $Z_k^T Z_k = I$ and $P_k^T Z_k = 0$. In addition,

$$\begin{aligned} Q_k^T K_k &= [Q_k^T P_k \;\; 0] \\ &= [Q_k^T Q_k M_k^{-1} D_k U_k \Sigma_k V_k \Lambda_k^{-1} \;\; 0] \\ &= [D_k U_k \Sigma_k V_k \;\; 0]. \end{aligned}$$

Therefore

$$\begin{aligned} K_k^T B_k K_k &= K_k^T (\gamma_k I + Q_k M_k^{-1} Q_k^T) K_k \\ &= \gamma_k I + K_k^T Q_k M_k^{-1} Q_k^T K_k \\ &= \begin{bmatrix} \gamma_k I + \Lambda_k & 0 \\ 0 & \gamma_k I \end{bmatrix}. \end{aligned}$$

□

We will show later that the presence of Z_k is actually not necessary for our computation, since all we need to know about Z_k is the square matrix $Z_k Z_k^T$, which can be expressed by $Z_k Z_k^T = I - P_k P_k^T$.

Let

$$\Omega_k = \gamma_k I + \Lambda_k.$$

Then

$$\begin{aligned}
\phi_k(\nu) &= \|s_k(\nu)\|^2 \\
&= \left\| \begin{bmatrix} \Omega_k + \nu I & 0 \\ 0 & (\gamma_k + \nu) I \end{bmatrix}^{-1} K_k^T g_k \right\|^2 \\
&= \left\| \begin{array}{c} (\Omega_k + \nu I)^{-1} P_k^T g_k \\ (\gamma_k + \nu)^{-1} Z_k^T g_k \end{array} \right\|^2 \\
&= \left\| (\Omega_k + \nu I)^{-1} P_k^T g_k \right\|^2 + \frac{\|Z_k^T g_k\|^2}{(\gamma_k + \nu)^2}.
\end{aligned}$$

But

$$\|Z_k^T g_k\|^2 = g_k^T (I - P_k P_k^T) g_k.$$

So we have

$$\phi_k(\nu) = \left\| (\Omega_k + \nu I)^{-1} b_k \right\|^2 + \frac{\|g_k\|^2 - \|b_k\|^2}{(\gamma_k + \nu)^2}, \tag{3.18}$$

where

$$b_k = \Lambda_k^{-1} V_k^T \Sigma_k U_k^T D_k M_k^{-1} Q_k^T g_k. \tag{3.19}$$

If we use the Newton's method to solve the one-dimensional equation $\phi_k(\nu) - \delta_k^2 = 0$, the derivative of $\phi_k(\nu)$ with respect to ν can be calculated by

$$\frac{d\phi_k(\nu)}{d\nu} = -2 b_k^T (\Omega_k + \nu I)^{-3} b_k - 2 \left(\|g_k\|^2 - \|b_k\|^2 \right) / (\gamma_k + \nu)^3. \tag{3.20}$$

After getting ν_k, the computation of $s_k(\nu_k)$ can then be

$$\begin{aligned}
s_k(\nu) &= (B_k + \nu_k I)^{-1} g_k \\
&= -K_k \begin{bmatrix} \Omega_k + \nu I & 0 \\ 0 & (\gamma_k + \nu) I \end{bmatrix}^{-1} K_k^T g_k \\
&= -\left[P_k \left[\Omega_k + \nu I \right]^{-1} P_k^T + (\gamma_k + \nu)^{-1} Z_k Z_k^T \right] g_k \\
&= -\left[(\gamma_k + \nu)^{-1} I + P_k \left(\left[\Omega_k + \nu I \right]^{-1} - \left[\gamma_k + \nu \right]^{-1} I \right) P_k^T \right] g_k.
\end{aligned}$$

So, the actual formula for computing s_k is

$$s_k = s_k(\nu_k) = -\frac{1}{\gamma_k + \nu_k}(g_k - Q_k h_k), \tag{3.21}$$

where

$$h_k = M_k^{-1} D_k U_k \Sigma_k V_k (\Omega_k + \nu_k I)^{-1} b_k. \tag{3.22}$$

We can see that, since $m \ll n$, the main cost for computing s_k is the matrix/vector multiplication $Q_k^T g_k$ in (3.19) and $Q_k h_k$ in (3.21). All the other operations are minor. We will discuss the cost in detail in Chapter 5.

Algorithm 3.7 L-SR1_STEP (IN: $D_k, M_k, Q_k, \Lambda_k, \Sigma_k, U_k, V_k, \gamma_k, g_k, b_k$;

OUT: s_k)

1. DEFINE $\phi_k(\nu) = \left\| [\Lambda_k + (\nu + \gamma_k) I]^{-1} b_k \right\|^2 + \dfrac{\|g_k\|^2 - \|b_k\|^2}{(\gamma_k + \nu)^2}$;
2. IF $\Lambda_k + \gamma_k I$ positive definite AND $\phi_k(0) \leq \delta_k^2$ THEN
3. $\quad \nu_k := 0$;
4. ELSE
5. $\quad \nu_k^{(0)} := 1.1 * \max\,(0, -\text{min\_eigenvalue}\ (\Lambda_k + \gamma_k I))$;
6. $\quad$ solve $\phi_k(\nu) = \delta_k^2$ with initial $\nu_k^{(0)}$ to get ν_k;
7. END IF;
8. $h_k := M_k^{-1} D_k U_k \Sigma_k V_k [\Lambda_k + (\nu_k + \gamma_k) I]^{-1} b_k$;
9. $s_k := -\dfrac{1}{\gamma_k + \nu_k}(g_k - Q_k h_k)$;
10. return.

□

There is in fact another way to compute the step s_k. We describe it in Appendix B. However, we believe the method we present here is numerically more stable.

3.4 Updating Trust Region Radius

Given the trust region radius δ_k, we can get a solution s_k to the quadratic subproblem (2.8), no matter what method is used to solve the subproblem. However, the step s_k such obtained is not necessarily a good step for the original problem (2.1). So, after getting s_k, we have an acceptance test (2.2) to see if the objective function value is decreased significantly at $x_k + s_k$. If s_k fails the test, it means $f(x)$ vibrates a lot at x_k, or B_k is a bad approximation to the true Hessian $\nabla^2 f(x_k)$. A natural way to proceed is to keep backtracking δ_k until (2.2) is satisfied. The reduction factor can be determined by the quadratic model $q_k(\delta)$ which approximates $f\left(x_k + \frac{\delta}{\|s_k\|}s_k\right)$. With $q_k(0) = f(x_k)$, $q_k(\|s_k\|) = f(x_k + s_k)$ and $q_k'(0) = g_k^T s_k/\|s_k\|$ are given, $q_k(\delta)$ is uniquely determined and its minimizer is

$$\delta_k^{(pred)} = \frac{-g_k^T s_k}{2(f(x_k + s_k) - f(x_k) - g_k^T s_k)}\|s_k\|.$$

If $\delta_k^{(pred)} \in [\frac{1}{10}\delta_k, \frac{1}{2}\delta_k]$, we accept $\delta_k^{(pred)}$ as new δ_k. Otherwise, we set new δ_k to the closer endpoint of this interval. Here is the algorithm in detail.

Algorithm 3.8 TR_RADIUS_REDCTN (IN: $g_k, s_k, f(x_k), f(x_k + s_k)$; INOUT: δ_k)

1. IF $\|s_k\| = \delta_k$ THEN $\delta := \|s_k\|$;
2. IF $f(x_k + s_k) = f(x_k) + g_k^T s_k$ THEN
3. $\quad \delta := 0.5\delta_k$;
4. ELSE
5. $\quad \delta := -0.5\|s_k\| g_k^T s_k/[f(x_k + s_k) - f(x_k) - g_k^T s_k]$;
6. $\quad$ IF $\delta \leq 0.1\delta_k$ THEN
7. $\quad\quad \delta := 0.1\delta_k$;
8. $\quad$ ELSE IF $\delta \geq 0.5\delta_k$ THEN
9. $\quad\quad \delta := 0.5\delta_k$;
10. $\quad$ ELSE
11. $\quad\quad \delta := \delta_k$;

12. END IF;
13. END IF;
14. $\delta_k := \delta$;
15. return.

□

Now, suppose we finally find an acccptable step s_k with the trust region radius δ'_k. If δ_k is the original trust region radius in the beginning of the current iteration, we update

$$\delta_{k+1} = \max\left\{\delta'_k, \frac{1}{20}\delta_k\right\}.$$

for the next iteration. The term $\delta_k/20$ is supposed to avoid the influence of possibly too many contractions of trust region over the next iteration. It is necessary because sometimes B_k is so bad that (2.2) holds only when $\|s_k\|$ is sufficiently small while B_{k+1} might be pretty good.

If an s_k passes test (2.2) at the first try, it might be because of a recovered B_k and a good quadratic model (2.8). In this case we should give it a chance to make a bigger stride. Formally, if

$$\left|\Delta f_k^{(pred)} - \Delta f_k\right| \leq \beta|\Delta f_k| \tag{3.23}$$

where

$$\begin{aligned} \Delta f_k^{(pred)} &= \frac{1}{2}(\nu_k\|s_k\|^2 - g_k^T s_k) \\ \Delta f_k &= f(x_k) - f(x_k + s_k), \end{aligned}$$

then we increase δ_k by a certain ratio until either (2.2) or (3.23) does not hold any more. We let δ_{k+1} be the last accepted δ_k for the next iteration. If at some moment the Newton step happens ($s_k = -B_k^{-1}g_k$), we stop expanding the trust region and let $\delta_{k+1} = \|s_k\|$.

CHAPTER 4

CHOOSING THE SCALING PARAMETER

As we know, for standard quasi-Newton methods, the initial Hessian approximation does not play a significant role in the performance. Usually, we choose $B_0 = I$ initially, and then after the first iteration, we use $\frac{y_0^T s_0}{s_0^T B_0 s_0} I$ as initial Hessian for updating, or

$$B_1 = \mathbf{U}\left(\frac{y_0^T s_0}{s_0^T B_0 s_0} I, y_0, s_0\right).$$

For limited memory methods, we can certainly do the same thing. But since we have lots of information in hand at the beginning of each iteration, we should be able to have a better initial guess for $B_k^{(0)}$. Usually, we let $B_k^{(0)}$ be the scalar matrix $\gamma_k I$ where γ_k is computed by either

$$\frac{\|y_{k-1}\|^2}{y_{k-1}^T s_{k-1}} \quad \text{or} \quad \frac{y_{k-1}^T s_{k-1}}{\|s_{k-1}\|^2}, \tag{4.1}$$

as long as $y_{k-1}^T s_{k-1} > 0$. Nocedal and Liu (1989) reported the importance of γ_k to L-BFGS method. They even tried a diagonal initial Hessian but it does not seem to have significant superiority over some scalar ones.

In this chapter, we will compare several different ways to compute the parameter γ_k for L-SR1 method. It looks like that our new method described in Section 4.3 is significantly superior to all others.

Traditionally, we call γ_k the scaling parameter and $B_k^{(0)}$ the scaling matrix. But we will see in Section 4.3 that, with the new method, γ_k is not simply a scaling parameter any more, it can change the Hessian's pattern.

4.1 Traditional Scaling Parameter

The traditional practice for choosing γ_k is to let it be one of the two listed in (4.1), provided $y_{k-1}^T s_{k-1} > 0$. Since $\|y_{k-1}\|^2/y_{k-1}^T s_{k-1} \geq y_{k-1}^T s_{k-1}/\|s_{k-1}\|^2$, we can consider $\|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$ as the "biggest" eigenvalue of the true Hessian $\nabla_x^2 f(x_{k-1})$ and $y_{k-1}^T s_{k-1}/\|s_{k-1}\|^2$ the "smallest". Many researchers observe that the choice of γ_k has significant effect for L-BFGS method and $\|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$ appears to be the best.

Actually, there is another reasonable choice of γ_k. It is the geometric mean of the two formers, $\|y_{k-1}\|/\|s_{k-1}\|$. Since

$$\frac{y_{k-1}^T s_{k-1}}{\|s_{k-1}\|^2} \leq \frac{\|y_{k-1}\|}{\|s_{k-1}\|} \leq \frac{\|y_{k-1}\|^2}{y_{k-1}^T s_{k-1}},$$

it looks like that $\|y_{k-1}\|/\|s_{k-1}\|$ can be a better choice. But our numerical test for L-SR1 shows that, the bigger the scaling parameter, the better its performance. Table 4.1 and 4.2 show the comparisons between the three choices. The test problems are selected from CUTE (We'll talk more about the test problems in Chapter 6). Their sizes for the experiment in Table 4.1 and 4.2 are all 100. The parameter m is set to 4. For the convenience of comparison, we list only those problems upon which all the methods run successfully.

Although the superiority of bigger scaling parameter over small ones is slight and even controversial for some test problems, this phenomenon persists in all of our experiments. While the real reason is unknown, we speculate that a bigger γ_k is easier to make Newton steps and therefore speeds up the convergence.

Another thing we can see from Table 4.1 and 4.2 is that the number,

$$\frac{fun}{itn},$$

Table 4.1: Traditional Scaling Parameters (M1/M2/M3/M4)

Pname	itn	fun
ARGLINA	1/1/1/2	3/3/3/4
ARGLINC	2/2/2/3	15/15/15/4
ARWHEAD	7/8/9/10	11/14/15/11
BDQRTIC	41/49/43/43	98/119/97/53
BROYDN7D	50/42/43/73	97/83/86/77
BRYBND	167/155/35/104	418/394/66/115
COSINE	7/7/7/13	9/9/9/36
CRAGGLVY	88/112/85/64	211/258/202/66
DQDRTIC	14/13/9/18	38/35/18/24
DQRTIC	46/43/45/38	86/69/78/41
ENGVAL1	15/13/21/14	31/22/40/15
FLETCBV2	336/303/230/165	892/768/587/175
FREUROTH	17/22/16/17	56/64/38/25
LIARWHD	18/17/18/20	31/28/31/23
MANCINO	10/10/10/13	18/26/26/17
MOREBV	234/287/320/351	573/688/807/365
NCB20B	96/96/101/98	250/223/224/106

M1: L-SR1 with $\gamma_k = y_{k-1}^T s_{k-1}/\|s_{k-1}\|^2$;
M2: L-SR1 with $\gamma_k = \|y_{k-1}\|/\|s_{k-1}\|$;
M3: L-SR1 with $\gamma_k = \|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$;
M4: L-BFGS with $\gamma_k = \|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$.

Table 4.2: Traditional Scaling Parameters (M1/M2/M3/M4), continue

Pname	itn	fun
NONCVXUN	289/247/380/177	749/571/954/186
NONDIA	10/10/10/17	20/20/20/19
PENALTY1	115/127/117/49	254/304/307/61
PENALTY2	15/16/16/16	22/37/28/17
POWELLSG	31/38/33/37	80/109/90/45
POWER	78/65/54/46	199/165/145/48
QUARTC	46/43/45/38	86/69/78/41
SCHMVETT	29/30/25/23	55/69/52/28
SENSORS	19/19/21/19	30/30/32/21
SINQUAD	82/62/63/80	308/179/170/109
SPARSQUR	21/27/23/21	28/43/33/22
SPMSRTLS	95/101/98/62	217/213/222/68
SROSENBR	12/18/13/14	17/35/24/17
TOINTGSS	11/8/8/21	29/19/24/30
TQUARTIC	16/14/13/16	30/30/26/24
TRIDIA	395/433/409/223	1009/1007/962/240
WOODS	103/95/96/108	279/232/258/133
Total	2516/2533/2419/2013	6249/5950/5767/2266

M1: L-SR1 with $\gamma_k = y_{k-1}^T s_{k-1}/\|s_{k-1}\|^2$;
M2: L-SR1 with $\gamma_k = \|y_{k-1}\|/\|s_{k-1}\|$;
M3: L-SR1 with $\gamma_k = \|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$;
M4: L-BFGS with $\gamma_k = \|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$.

the ratio of the number of function evaluations (fun) to the number of iterations (itn), looks irrational. In average, they are 2.48, 2.35 and 2.38, respectively for M1, M2 and M3.

The method M4 in Table 4.1 and 4.2 is the result for L-BFGS method. As we can see, both L-SR1 and L-BFGS methods have some wins and losses in the number of iterations with L-BFGS win in average. But talking about the number of function evaluations, L-BFGS wins definitely.

4.2 Osborne and Sun's Positive Scalar

It looks like that the losing of the L-SR1 with traditional scaling parameters is because of the lack of Newton steps[1]. One can confirm this by having a preview of Table 4.6, where the test problems are all strictly convex and the "0" in the "0/1" strings means indefinite L-SR1 matrix. It clearly shows that the L-SR1 matrices can be indefinite anywhere, no matter how close the iterate is close to the solution and how convex the objective function is.

Without Newton step, trust region method tends to change the radius δ_k frequently so that the solution to subproblem model predicts a better descent direction for the original problem. To have Newton steps, a positive definite Hessian is imperative. Now, let's have a look at the issue of positive definiteness of SR1 updates.

We know that the standard SR1 Hessian matrix is not necessarily positive definite. Although this creates some difficulty in theoretical analysis, it might explain why standard SR1 method is generally a little bit faster than

[1]By the term "Newton step", here we mean the step $s_k = -B_k^{-1} g_k$. Sometimes, we use another term "full step" for it.

standard BFGS method in unconstrained optimization.

However, a positive definite Hessian is highly desirable when the objective function becomes convex as the iterates approach the local minimizer. Osborne and Sun (1988) presented a scaled version of SR1 update (OSSR1) that if $y_k^T s_k > 0$ and $B_k^{(\mathrm{OSSR1})}$ is positive definite, then the new matrix $B_{k+1}^{(\mathrm{OSSR1})}$ updated by

$$B_{k+1}^{(\mathrm{OSSR1})} = \omega_k B_k^{(\mathrm{OSSR1})} + \frac{\left(y_k - \omega_k B_k^{(\mathrm{OSSR1})} s_k\right)\left(y_k - \omega_k B_k^{(\mathrm{OSSR1})} s_k\right)^T}{\left(y_k - \omega_k B_k^{(\mathrm{OSSR1})} s_k\right)^T s_k} \tag{4.2}$$

is positive definite if and only if the scaling parameter ω_k is chosen such that

$$0 < \omega_k < \frac{y_k^T s_k}{s_k^T B_k^{(\mathrm{OSSR1})} s_k} \quad \text{or} \quad \omega_k > \frac{y_k^T \left[B_k^{(\mathrm{OSSR1})}\right]^{-1} y_k}{y_k^T s_k}.$$

They also presented a way to compute ω_k by following an analysis by Davidon (1975). However their numerical experiment showed that a value of ω_k different from one should be applied selectively only to some iterations. Following is the algorithm for computing ω_k suggested by Osborne and Sun (They call it OCSSR1).

Algorithm 4.1 SSR1_SCALAR (IN: $B_k^{(\mathrm{OSSR1})}, y_k, s_k$; OUT: ω_k)

$a := y_k^T [B_k^{(\mathrm{OSSR1})}]^{-1} y_k$;
$b := y_k^T s_k$;
$c := s_k^T B_k^{(\mathrm{OSSR1})} s_k$;
IF $b > a$ THEN
  $\omega_k := 1$;
ELSE
  $\Delta := \sqrt{(c/b)^2 - c/a}$;
  $\theta_1 := c/b + \Delta$;
  $\theta_2 := c/b - \Delta$;

DEFINE $\psi(\theta) = \dfrac{c - b\theta}{b\theta - a\theta^2}$;
IF $\psi(\theta_1) \geq \psi(\theta_2)$ THEN
$\omega_k := 1/\theta_1$;
ELSE
$\omega_k := 1/\theta_2$;
END IF;
END IF;
return.

Generally, OSSR1 method is worse than the standard SR1 method. Table 4.3 shows the comparison of standard SR1 method with Osborne and Sun's. Both of them are implemented with Algorithm 2.5 as the solver to trust region subproblems and Algorithm 2.3 as the frame for updating trust region radius.

It is obvious that Conn, Gould and Toint (1991)'s theory requires $\omega_k \to 1$ for the superlinear convergence of methods defined by (4.2). But Osborne and Sun's choice for ω_k does not possess such property. Our experiment shows that the ω_k computed by Algorithm 4.1 frequently deviate from value one, even though x_k is already close enough to the solution.

But for some problems, OSSR1 performs great. Table 4.4 is a comparison for problem SROSENBR, a separable extension of Rosenbrock's function. Even in Table 4.3, we can see that the fun/itn ratio for OSSR1 looks more reasonable than the standard SR1's for the problems we choose. They are 1.60 and 1.47 respectively.

What we tried to explain in the past two paragraphs is that, although OSSR1 method is generally not superior to the standard SR1 method, it does help to improve the fun/itn ratio. It proves that to make the L-SR1 matrix positive definite can be the right direction to fix L-SR1's fatal defect we

Table 4.3: Comparisons of Standard SR1 to OSSR1

Pname	dim	itn	fun
ALLINITU	4	9/12	15/15
BARD	3	17/24	27/44
BEALE	2	15/17	19/24
BOX3	3	8/9	13/12
BRKMCC	2	5/5	8/8
BROWNBS	2	11/11	29/15
BROWNDEN	4	17/23	37/46
DENSCHNA	2	8/9	10/11
DENSCHNC	2	13/21	23/32
DENSCHNE	3	24/40	29/57
EXPFIT	2	13/15	24/22
HIMMELBB	2	14/6	25/12
HIMMELBG	2	6/7	9/8
S308	2	12/12	16/15
SISSER	2	15/20	17/22
SNAIL	2	9/11	13/13
total		196/242	314/356

Table 4.4: Comparison of Standard SR1 to OSSR1 for SROSENBR

	gradtol: 10^{-5}		gradtol: 10^{-9}	
dim	itn	fun	itn	fun
10	16/12	46/17	19/14	49/19
50	18/12	52/17	27/14	72/20

discovered in the previous section.

Actually, for limited memory methods, we don't expect any superlinear convergence. Whether $\omega_k \rightarrow 1$ or not is not an important issue any more. It can be quite possible that, what we get from the positive definiteness of OSSR1 update can be more than what we lose in the issue of $\omega_k \rightarrow 1$.

We implemented the limited memory version of OSSR1 method (L-OSSR1). The implementation uses the same trust region technique as that of L-SR1. The only difference is in matrix representation and subsequently the step computation. We use an $n \times n$ array to explicitly store the L-OSSR1 Hessian. So, it is a limited memory method implemented with full memory locations. This difference shouldn't have any impact for our test. We run the code on a set of randomly generated strictly convex quadratic problems (See Chapter 6 for the definition of them). The reason to use quadratic problems is for better understanding.

Table 4.5 is a comparison of L-SR1 with L-OSSR1. The parameter m is set to 4. For L-OSSR1 method, although we use the same scaling parameter, its real value is actually up to Algorithm 4.1 to decide. We can see that the fun/itn ratios are still outrageous (2.14 and 1.69 respectively for L-SR1 and L-OSSR1). But compared to the result shown in Table 4.3, L-OSSR1 outperforms L-SR1 this time.

The figures shown in Table 4.6 are of inspiration. They show the Hessian and step patterns of the last 63 iterations of the same experiments as in Table 4.5, where "P" has the same problem numbers as in Table 4.5. We can see that, even though an iterate is close to the solution and the objective function is strictly convex, the L-SR1 matrix can be indefinite at any time. This

Table 4.5: Comparison of L-SR1 with L-OSSR1 (M3/M5)

P	dim	itn	fun
1	5	7/8	20/12
2	5	8/10	16/21
3	10	27/25	46/34
4	10	39/44	79/75
5	15	38/35	63/53
6	15	67/78	145/130
7	20	36/39	76/52
8	20	72/70	151/123
9	50	61/63	128/98
10	50	190/168	440/312
total		545/540	1164/910

M3: L-SR1 with $\gamma_k = \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
M5: L-OSSR1 with $\gamma_k = \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;

causes the trust region radius to adjust more frequently than L-OSSR1 and therefore makes L-SR1 have more function evaluations, as shown in Table 4.5. As is expected, L-OSSR1's Hessians are always positive definite and produces more Newton steps.

4.3 Positive Initial Scalar

The Newton step is an important issue for the performance of any optimization algorithm. Without Newton steps, an algorithm can be seriously slowed down. In order to obtain a Newton step, the positive definiteness of Hessian is imperative. For L-BFGS method, the positive definiteness of its Hessian is guaranteed by line searches with both α- and β-conditions. For standard SR1 method, since its Hessian can converge to the true Hessian, the positive definiteness is also usually out of question. But for L-SR1 method, we don't have these guarantee any more. Our conclusion from the previous section is that, the L-OSSR1 method is a direction to consider, but not good enough to compete with L-BFGS method. In this section, we will talk about another alternative. First, let's look at the well-definedness of the standard SR1 matrix $B_k^{(\mathrm{SR1})}$.

4.3.1 Well-definedness of the SR1 Hessian

It is known that the SR1 Hessian matrix $B_{k+1}^{(\mathrm{SR1})}$ is well-defined only if both conditions (2.15) and (2.16) hold. While there is no theory to guarantee the correctness, we can construct some counter-examples against condition (2.15) or (2.16). The following two examples are designed only for the first iteration.

- **Example 1.** Consider the quadratic objective function of $2n$ dimensions

$$f(x) = \frac{1}{2}x^T G x$$

Table 4.6: Hessian and Step Patterns for L-SR1 and L-OSSR1

P	M	Patterns (→)
1	3	1011100 nennnss
	5	11111111 nennnnnn
3	3	10111110001000011000111010 nennnnnssrnsrseeessrrennsrs
	5	111111111111111111111111 nennnnnnsnsennnnnnnsnnsnn
5	3	11111100100110001110110110000101100001 nesnnrsrrssesrserensrsrnesrrrernrsrrre
	5	11111111111111111111111111111111111 nennnnnnnnnnnnssrnsesrensnsnesrnsrn
7	3	11111101100111110011001000010100 1101 nesnnssesssesernsressersrreesssseesse
	5	111111111111111111111111111111111111111 nennnnnnnnnsnnsnesnennnsnennnnnnnnsrrsn
9	3	111101101101000110111011010001000101010011100011001000 1001110 nrnnsesrensesrressessresresrseseserrresrressrsesrressresresrs
	5	11111111111111111111111111111111111111111111111111111111111111111 nrnnnnsrrnesrreseensennnsnsersrrnrrrnnsnnnsnnnrnnrsnennsrenrsrr

M3: L-SR1 with $\gamma_k = \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
M5: L-OSSR1 with $\gamma_k = \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
1: positive definite Hessian;
0: indefinite Hessian;
n: Newton step;
e: expanded step;
s: shrunk step;
r: other steps.

where $G = \text{diag}\,(h_1, h_2, \ldots, h_{2n})$ is a diagonal matrix with

$$h_1 = h_3 = \ldots = h_{2n-1} = \gamma + \delta$$

$$h_2 = h_4 = \ldots = h_{2n} = \gamma - \delta$$

where $\gamma > \delta > 0$. Denote $e = [1 \;\; 1 \;\; \ldots \;\; 1]^T$.

Let $x_0 = -G^{-1}e$ and $B_0^{(\text{SR1})} = \gamma I$. Then $s_0 = -[B_0^{(\text{SR1})}]^{-1}\nabla f(x_0) = \frac{1}{\gamma}e$ and $y_0 = Gs_0$. Therefore

$$(y_0 - B_0^{(\text{SR1})}s_0)^T s_0 = s_0^T(G - B_0^{(\text{SR1})})s_0 = 0.$$

• **Example 2.** Consider the quadratic objective function of $2n+1$ dimensions

$$f(x) = \frac{1}{2}x^T G x$$

where $G = \text{diag}\,(h_1, h_2, \ldots, h_{2n}, h_{2n+1})$ is a diagonal matrix with

$$h_1 = h_3 = \ldots = h_{2n-1} = h_{2n+1} = \gamma + \delta$$

$$h_2 = h_4 = \ldots = h_{2n} = \gamma - \delta$$

where $\gamma > \delta > 0$.

Given $\epsilon > 0$, denote $e(\epsilon) = [1 \;\; 1 \;\; \ldots \;\; 1 \;\; \epsilon]^T$. Let $x_0 = -G^{-1}e(\epsilon)$ and $B_0^{(\text{SR1})} = \gamma I$. Then $s_0 = -[B_0^{(\text{SR1})}]^{-1}\nabla f(x_0) = \frac{1}{\gamma}e(\epsilon)$ and $y_0 = Gs_0$. Therefore

$$\begin{aligned}\cos\theta_0 &= \frac{|(y_0 - B_0^{(\text{SR1})}s_0)^T s_0|}{\|y_0 - B_0^{(\text{SR1})}s_0\|\|s_0\|} \\ &= \frac{\epsilon^2}{2n + \epsilon^2}\end{aligned}$$

As $\epsilon \to 0$, $\cos\theta_0 \to 0$.

The first example shows that the denominator $(y_k - B_k^{(\text{SR1})}s_k)^T s_k$ of SR1 Hessian can be exactly zero at the very first iteration. The second example

shows that the first cosine value, $\cos\theta_0$, can be as close to zero as possible, or $\cos\theta_0 = o(1)$.

• **Example 3.** Consider the quadratic objective function of n dimensions

$$f(x) = \frac{1}{2}x^T G x$$

where G is a symmetric positive definite matrix.

Let $B_0^{(\mathrm{SR1})} = \gamma I$, where h is a positive scalar. If we look at the compact form of SR1 formula (2.20) with $m = k$, the kernel matrix M_k can be expressed by

$$M_k = S_k^T(G - \gamma I)S_k.$$

If the initial γ is taken to be equal to any of the eigenvalues of G, the matrix M_k will be singular and therefore formula (2.20) will become invalid.

The three examples above explain that no matter how "good" the initial scalar matrix looks like, the possibility for invalid SR1 matrix does exist. Thus, the two conditions (2.15) and (2.16) are crucial to the finiteness of SR1 method. We have no idea of conditions under which which (2.15) and (2.16) can be true. In Appendix I of this thesis, we prove, however, that if uniform distribution of both r_k and s_k is assumed, then the probability of $\cos\theta_k \leq \rho$ is approximately $n\rho$ if ρ is a small number. Table 4.7 lists the distribution of cosine values for some test problems. We can see that, overwhelming majority of cosine values lie in $[10^{-5}, 1]$ and seldomly beyond 10^{-7}.

Although it rarely happens, once the cosine value becomes really small, it can hurt the algorithm a lot. The safeguard check (2.17) is therefore strongly recommended. We take the value of ρ as 10^{-7} for double precision computation.

Table 4.7: SR1 Cosine Value Distribution

problems	dim	iteration	$[10^{-3}, 10^{0}]$	$[10^{-5}, 10^{-3}]$	$[10^{-7}, 10^{-5}]$	$[10^{-7}, 0]$
Beale	2	14	14	0	0	0
Biggs	6	32	31	1	0	0
Box 3-D	3	24	23	0	0	0
Brown	2	13	1	5	7	0
Cheby	9	19	19	0	0	0
Dennis	4	15	15	0	0	0
Gaussian	3	1	1	0	0	0
Gulf	3	29	23	5	0	0
Helical	3	32	31	1	0	0
Penalty I	10	17	17	0	0	0
Penalty II	10	27	27	0	0	0
Powell	100	78	77	1	0	0
Tridia	100	69	69	0	0	0
Trig	100	42	42	0	0	0
Var Dim	10	10	10	0	0	0
Watson	9	35	33	2	0	0
Wood	100	37	37	0	0	0
Total		494	470	15	7	0

4.3.2 Positive Definiteness of L-SR1 Matrix On the other hand side, the positive definiteness of L-SR1 matrix can be handled more elegantly. For SR1 method, the initial Hessian $B_0^{(\mathrm{SR1})}$ is something we don't know how to control at the very beginning, because of the lack of information. But for L-SR1 method, we have all the information pairs $\{y_{k-i}, s_{k-i}\}_{i=1}^{m}$ in memory. With careful choice of initial γ_k's, the behavior of the algorithm can be improved significantly.

Theorem 4.2

(1) If the matrix W_k defined by (2.22) is positive definite, there is a number $\underline{\gamma}_k > 0$ such that, if

$$0 < \gamma_k < \underline{\gamma}_k,$$

then $B_k^{(\mathrm{LSR1})}$ defined by (2.20) is a positive definite matrix.

(2) If the matrix $\tilde{W}_k$ defined by (2.24) is positive definite, there is a number $\bar{\gamma}_k > 0$ such that, if

$$\gamma_k > \bar{\gamma}_k,$$

then $B_k^{(\mathrm{LSR1})}$ defined by (2.20) is a positive definite matrix.

Proof:

As we know from Section 2.4, matrix M_k is defined by

$$M_k = W_k - \gamma_k S_k^T S_k. \tag{4.3}$$

Since W_k is positive definite, it is easy to conclude that there is a $\underline{\gamma}_k > 0$ such that, for any $\gamma_k \in (0, \underline{\gamma}_k)$, M_k is positive definite. Then by (2.20), $B_k^{(\mathrm{LSR1})}$ is positive definite.

With the same reason, if matrix $\tilde{W}_k$ defined by (2.24) is positive definite, there is a $\bar{\gamma}_k > 0$ such that, for any $\gamma_k \in (\bar{\gamma}_k, \infty)$, the matrix

$$\tilde{W}_k - \frac{1}{\gamma_k} Y_k^T Y_k \tag{4.4}$$

is positive definite. Then, by (2.23), $[B_k^{(\mathrm{LSR1})}]^{-1}$ and therefore $B_k^{(\mathrm{LSR1})}$ is positive definite.

□

This theorem does not hint any method we can use to compute the biggest $\underline{\gamma}_k$ and smallest $\bar{\gamma}_k$. Its proof, however, does point out a direction to consider. We will let

$$\underline{\gamma}_k = \max \{\gamma > 0 : W_k - \gamma S_k^T S_k \text{ is positive definite } \} \tag{4.5}$$

if W_k is positive definite, and

$$\bar{\gamma}_k = \min \{\gamma > 0 : \tilde{W}_k - \frac{1}{\gamma} Y_k^T Y_k \text{ is positive definite } \} \tag{4.6}$$

if $\tilde{W}_k$ is positive definite. We will state the computation of $\underline{\gamma}_k$ and $\bar{\gamma}_k$ right after this subsection.

Recall that W_k is a symmetric matrix whose upper triangular resembles that of $Y_k^T S_k$ and $\tilde{W}_k$ is a symmetric matrix whose lower triangular resembles its counterpart of $Y_k^T S_k$. If we define the positive definiteness of an arbitrary $n \times n$ matrix A as $x^T A x > 0$ for any $x \in \mathcal{R}^n$ with $x \neq 0$, then the positiveness of both W_k and $\tilde{W}_k$ implies the positiveness of $Y_k^T S_k$. But the reverse is not true. For example, the small matrix

$$\begin{bmatrix} 3 & 4 \\ 1 & 3 \end{bmatrix}$$

is positive definite whereas

$$\begin{bmatrix} 3 & 4 \\ 4 & 3 \end{bmatrix}$$

is not.

Although it is hard to pose a general condition as to when W_k and/or $\tilde{W}_k$ are positive definite, we do have the following result when x_k is close to the solution. First, we define the concept of **uniformly linear independence.**[2]

Definition 4.3 Let m and n be two integers with $0 < m \leq n$. Given a sequence of n-dimensional vectors $\{v_i\}_{i=1}^{\infty}$. If there is a constant $c > 0$ such that, for any $k \geq 0$, the smallest singular value of the matrix

$$\begin{bmatrix} \frac{v_{k+1}}{\|v_{k+1}\|} & \frac{v_{k+1}}{\|v_{k+1}\|} & \cdots & \frac{v_{k+1}}{\|v_{k+1}\|} \end{bmatrix}$$

is greater than c, then we say the sequence $\{v_i\}_{i=1}^{\infty}$ is m-uniformly linearly independent.

Theorem 4.4 Assume $f(x)$ is a twice continuously differentiable function which has a local minimizer x_* where $\nabla f(x_*) = 0$ and $\nabla^2 f(x_*)$ is a positive definite matrix. Given a vector sequence $\{s_j\}_{j=0}^{\infty}$ and an initial point x_0. Let $\{x_j\}_{j=0}^{\infty}$ be the iterate sequence generated by $x_{j+1} = x_j + s_j$. If $\lim_{j\to\infty} x_j = x_*$ and the sequence $\{s_j\}_{j=0}^{\infty}$ is m-uniformly linearly independent, then there is a constant k_0, such that

(1) the matrices W_k defined by (2.22) and $\tilde{W}_k$ defined by (2.24) are both positive definite for all $k \geq k_0$.

(2) the two variables $\underline{\gamma}_k$ and $\bar{\gamma}_k$ defined by (4.5) and (4.6) are bounded away from zero and infinity.

[2]This concept here is slightly different from that used by Conn, Gould and Toint (1991), and Ortega and Rheinboldt (1970).

Proof:

For any $i \geq m$, let

$$\epsilon_i = \max_{0 \leq j \leq m} \{\|x_{i-j} - x_*\|\}.$$

Let G_* be the exact Hessian of $f(x)$ at minimizer x_*, $G_* = \nabla^2 f(x_*)$. By Taylor's expansion,

$$y_i = G_* s_i + O(\|s_i\|^2) + O(\|s_i\|\|x_i - x_*\|), \tag{4.7}$$

and therefore

$$y_i^T s_j = s_i^T G_* s_j + O(\epsilon_i \|s_i\|\|s_j\|).$$

Given $k \geq m$, If we let

$$D_k = \text{ diag } (\|s_{k-m}\|, \|s_{k-m+1}\|, \ldots, \|s_{k-1}\|),$$

then

$$W_k = D_k[D_k^{-1} S_k^T G_* S_k D_k^{-1} + O(\epsilon_k)]D_k. \tag{4.8}$$

Since G_* is symmetric positive definite, we know that the matrix

$$D_k^{-1} S_k^T G_* S_k D_k^{-1}$$

is positive definite and its minimum eigenvalue is bounded away from zero. Thus, as $\epsilon_k \to 0$, W_k will be eventually a positive definite matrix when k is greater than some k_0.

Similarly, for $\tilde{W}_k$, we also have

$$\tilde{W}_k = D_k[D_k^{-1} S_k^T G_* S_k D_k^{-1} + O(\epsilon_k)]D_k. \tag{4.9}$$

It will be eventually a positive definite matrix, too.

We now prove the second claim. We prove the result for $\underline{\gamma}_k$ first. Let

$$\begin{aligned} \underline{\omega}_* &= \min \{\text{all eigenvalues of } G_*\}, \\ \bar{\omega}_* &= \max \{\text{all eigenvalues of } G_*\}. \end{aligned}$$

By (4.8), we have

$$W_k - \gamma S_k^T S_k = D_k[D_k^{-1} S_k^T (G_* - \gamma I) S_k D_k^{-1} + O(\epsilon_k)] D_k.$$

Obviously, there is an integer k_1 such that the matrix $W_k - \gamma S_k^T S_k$ will be negative definite if $\gamma > 2\bar{\omega}_*$ and $k \geq k_1$. Also, there is another integer k_2 such that the matrix $W_k - \gamma S_k^T S_k$ will be always positive definite if $\gamma < \frac{1}{2}\underline{\omega}_*$ and $k \geq k_2$. Put together, by the definition of $\underline{\gamma}_k$, for $k \geq \max\{k_1, k_2\}$,

$$\frac{1}{2}\underline{\omega}_* \leq \underline{\gamma}_k \leq 2\bar{\omega}_*,$$

and therefore $\underline{\gamma}_k$ is bounded away from both zero and infinity.

At last, we prove the result for $\bar{\gamma}_k$. By (4.7), we have

$$Y_k = G_* S_k + O(\epsilon_k) D_k,$$

and therefore

$$Y_k^T Y_k = S_k^T G_*^2 S_k + S_k^T O(\epsilon_k) D_k + D_k O(\epsilon_k) S_k + D_k O(\epsilon_k^2) D_k.$$

Thus, by (4.9),

$$\begin{aligned} \tilde{W}_k - \frac{1}{\gamma} Y_k^T Y_k &= D_k \left[D_k^{-1} S_k^T \left(G_* - \frac{1}{\gamma} G_*^2 \right) S_k D_k^{-1} \right. \\ &\quad \left. + D_k^{-1} S_k^T O(\epsilon_k) + O(\epsilon_k) S_k D_k^{-1} + O(\epsilon_k^2) \right] D_k. \end{aligned}$$

The last three terms in the square brackets are negligible, compared to the first term when $\epsilon_k \to 0$. By a similar statement when we proved the result for $\underline{\gamma}_k$,

we conclude that, there is an integer k_3 such that for $k \geq k_3$,

$$\frac{1}{2}\underline{\omega}_* \leq \bar{\gamma}_k \leq 2\bar{\omega}_*,$$

and therefore $\bar{\gamma}_k$ is bounded away from both zero and infinity.

□

Theorem 4.4 says that, when the iterate x_k getting close enough to solution x_*, the two matrices W_k and $\tilde{W}_k$ are guaranteed to be positive definite. Therefore, by Theorem 4.2, we can have a positive definite L-SR1 matrix by choosing γ_k and thus makes Newton steps possible.

Actually, from the proof of Theorem 4.4, we can see that

$$\liminf_{k\to\infty} \underline{\gamma}_k = \underline{\omega}_* \quad \text{and} \quad \limsup_{k\to\infty} \bar{\gamma}_k = \bar{\omega}_*,$$

if $\{s_j\}$ is both m-uniformly linearly independent (Definition 4.3) and uniformly linearly independent (Conn, Gould and Toint (1991), and Ortega and Rheinboldt (1970)).

To conclude this subsection, it is interesting to notice that, with $\bar{\gamma}_k$ defined by (4.6) and $\gamma_k > \bar{\gamma}_k$, matrix $B_k^{(\mathrm{LSR1})}$ can be then positive definite, but not M_k. To make thing worse, we even have no idea if M_k is singular or nonsingular in such an ideal case. Fortunately, we can use the finiteness assurance technique (Section 3.2) to ensure the existence of M_k^{-1}.

4.3.3 Computation of the Positive Initial Scalars In Section 4.3.2, we discussed the positive definiteness of L-SR1 Hessian matrix and pointed out in Theorem 4.2 that if matrices W_k and $\tilde{W}_k$ are positive definite, there are two positive numbers $\underline{\gamma}_k > 0$ and $\bar{\gamma}_k > 0$, such that if

$$0 < \gamma_k < \underline{\gamma}_k \quad \text{or} \quad \gamma_k > \bar{\gamma}_k,$$

then the L-SR1 matrix is positive definite. We also stated there that $\underline{\gamma}_k$ can be computed by (4.5) and $\bar{\gamma}_k$ by (4.6).

The actual computation of $\underline{\gamma}_k > 0$ and $\bar{\gamma}_k > 0$ will incorporate the scaling matrix D_k defined by (3.14). For $\underline{\gamma}_k$, we can do eigen-decomposition to the matrix $D_k^{-1} W_k D_k^{-1}$ first,

$$D_k^{-1} W_k D_k^{-1} = C_k N_k C_k^T,$$

where C_k is an $m \times m$ orthonormal matrix and N_k a diagonal matrix. Then if all diagonal entries of N_k are positive, $\underline{\gamma}_k$ can be the reciprocal of the maximum eigenvalue of matrix

$$N_k^{-\frac{1}{2}} C_k^T D_k^{-1} S_k^T S_k D_k^{-1} C_k N_k^{-\frac{1}{2}}.$$

Similarly, for $\bar{\gamma}_k$, we can eigendecompose the matrix $D_k^{-1} \tilde{W}_k D_k^{-1}$ first,

$$D_k^{-1} \tilde{W}_k D_k^{-1} = \bar{C}_k \bar{N}_k \bar{C}_k^T,$$

where $\bar{C}_k$ is an $m \times m$ orthonormal matrix and $\bar{N}_k$ a diagonal matrix. Then if all diagonal entries of $\bar{N}_k$ are positive, $\bar{\gamma}_k$ can be the maximum eigenvalue of matrix

$$\bar{N}_k^{-\frac{1}{2}} \bar{C}_k^T D_k^{-1} Y_k^T Y_k D_k^{-1} \bar{C}_k \bar{N}_k^{-\frac{1}{2}}.$$

Algorithm 4.5 computes γ_k based on $\bar{\gamma}_k$ and Algorithm 4.6 computes γ_k based on $\underline{\gamma}_k$.

Algorithm 4.5 INIT_HESSIAN_2 (IN: $\gamma_{k-1}, D_k, W_k, \tilde{W}_k$; OUT: γ_k)

IF $k = 0$ THEN
  $\gamma_k := 1$;
ELSE
  $[\bar{C}_k, \bar{N}_k] :=$ eigendecomposition $(D_k^{-1} \tilde{W}_k D_k^{-1})$;

IF min_eigenvalue $(\bar{N}_k) > 0$ THEN
  $\bar{\gamma}_k :=$ max_eigenvalue $(\bar{N}_k^{-\frac{1}{2}} \bar{C}_k^T D_k^{-1} Y_k^T Y_k D_k^{-1} \bar{C}_k \bar{N}_k^{-\frac{1}{2}})$;
  $\gamma_k := 1.1 * \bar{\gamma}_k$;
ELSE IF $y_{k-1}^T s_{k-1} > 0$ THEN
  $\gamma_k := \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
ELSE
  $\gamma_k := \gamma_{k-1}$;
ENDIF;
END IF;
return.

□

Algorithm 4.6 INIT_HESSIAN_3 (IN: $\gamma_{k-1}, D_k, W_k, \tilde{W}_k$; OUT: γ_k)

IF $k = 0$ THEN
  $\gamma_k := 1$;
ELSE
  $[C_k, N_k] :=$ eigendecomposition $(D_k^{-1} W_k D_k^{-1})$;
  IF min_eigenvalue $(N_k) > 0$ THEN
    $\underline{\gamma}_k :=$ 1/ max_eigenvalue $(N_k^{-\frac{1}{2}} C_k^T D_k^{-1} S_k^T S_k D_k^{-1} C_k N_k^{-\frac{1}{2}})$;
    $\gamma_k := 0.9 * \underline{\gamma}_k$;
  ELSE IF $y_{k-1}^T s_{k-1} > 0$ THEN
    $\gamma_k := \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
  ELSE
    $\gamma_k := \gamma_{k-1}$;
  END IF;
ENDIF;
return.

□

We test Algorithm 4.5 and 4.6 on the same quadratic problems as we did in Section 4.2. Table 4.8 lists the result for $m = 4$. Obviously, Positive Initial L-SR1 with γ_k computed by 4.5 is better than all the others and its fun/itn ratio is just 1.20.

Table 4.9 is a comparison of step patterns, where "P" has the same

Table 4.8: Comparison of L-OSSR1 and PI L-SR1 (M5/M6/M7)

P	dim	itn	fun
1	5	8/7/9	12/11/13
2	5	10/7/10	21/13/19
3	10	25/29/29	34/37/45
4	10	44/45/45	75/65/79
5	15	35/31/39	53/37/60
6	15	78/60/81	130/71/143
7	20	39/33/31	52/40/45
8	20	70/64/99	123/82/180
9	50	63/56/57	98/64/94
10	50	168/155/209	312/165/374
total		540/487/609	910/585/1052

M5: L-OSSR1 with $\gamma_k = \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
M6: L-SR1 with $\gamma_k = 1.1 * \bar{\gamma}_k$;
M7: L-SR1 with $\gamma_k = 0.9 * \underline{\gamma}_k$;

problem numbers as in Table 4.8. We see that L-SR1 method with $\gamma_k = 1.1 * \bar{\gamma}_k$ has much more Newton steps than the other two algorithms. The L-SR1 method with $\gamma_k = 0.9 * \underline{\gamma}_k$ is interesting. Its L-SR1 matrices are positive definite. But it does not have lots of Newton steps. The possible reason is because of the small of $\underline{\gamma}_k$. Since B_k has $(n-m)$ eigenvalues of γ_k, B_k^{-1} can be dominated by $1/\gamma_k$ and therefore $-B_k^{-1}g_k$ can predict a too long descent direction. Moreover, the acceptance condition (2.2) is hard to be satisfied because the quadratic interpolation model $q_k(\delta)$ (see Section 3.4) is not accurate in this case. Thus, more iterations and function evaluations are inevitable.

We decide to use $\gamma_k = 1.1 * \bar{\gamma}_k$ for our L-SR1 method.

Table 4.9: Step Patterns for L-OSSR1 and PI L-SR1

P	M	Patterns (→)
1	5	nennnnnn
	6	nennnnn
	7	nennnnnnn
3	5	nennnnnnsnsennnnnnnsnnsnn
	6	nennnnnnnnnnnnsnnnnnsnnnnnnsn
	7	nennnnsrnesernnsesernsnnnsesn
5	5	nennnnnnnnnnnnssrnsesrensnsnesrnsrn
	6	nennnnnnnnnnnnnnnnnnnnnnnnnnsn
	7	nennsrnnsesesrnrnseennsnnnnssennnnnsnes
7	5	nennnnnnnnnsnnsnesnennnsnennnnnnnnsrrsn
	6	nennnnnnnnnnnnnsnnnnssnnnnnsrnnnn
	7	nennnnnnnsensrnnrnnnsnnnsesesrn
9	5	rnnnnsrrnesrreseensennnsnsersrrnrrrnnsnnnsnnnrnnrsnennsrenrsrr
	6	nrnnnnnnnnnnnnnnnnnsnnnnnnsnnnnnnnnnnnnnnnnnsnnnsnnnnnnnsn
	7	nnnnnnnssrnnennsrrnrsennnsrnsernsrensenrnsreesensensnsrre

M5: L-OSSR1 with $\gamma_k = \|y_{k-1}\|^2 / y_{k-1}^T s_{k-1}$;
M6: L-SR1 with $\gamma_k = 1.1 * \bar{\gamma}_k$;
M7: L-SR1 with $\gamma_k = 0.9 * \underline{\gamma}_k$;
n: Newton step;
e: expanded step;
s: shrunk step;
r: other steps.

CHAPTER 5

COST AND CONVERGENCE ANALYSIS

The L-SR1 algorithm we gave in Section 3.1 is a logical algorithm. In this chapter, we will talk about physical memory requirement and computing time analysis. We assume that $m \ll n$. So, any expense with magnitude of m^p ($p \geq 1$), is negligible. We count only those in magnitude of about n.

We also analyze in this chapter the global and local R-linear convergence of L-SR1 algorithm. The conclusion we get is actually for any general Hessian matrix, as long as it is bounded. The L-SR1 method is just a special case.

5.1 Memory Requirement

Matrices and vectors which reside in memory "permanently" are listed in Table 5.1. They are updated in the beginning of each iteration and then stay in memory unchangedly throughout the iteration. In the same table, there are also two working vectors, s_- and x_-. All other matrices and vectors appear in Algorithm 3.1 can be derived from these matrices and vectors. Besides, there are some "permanent" variables. We don't count them because their number is negligible compared to n.

Symbols in the row titled "Locations" of Table 5.1 are supposed to mean physical representations for their corresponding matrices and vectors. They are purely for the convenience of description.

We mentioned the index table I_k in Subsection 3.2.2. Matrices Y_k, S_k,

Table 5.1: Memory Allocations for updating Matrices/Vectors

Matrices	Y_k	S_k	$Y_k^T Y_k$	$S_k^T S_k$	$Y_k^T S_k$
Locations	$\mathcal{L}_Y$	$\mathcal{L}_S$	$\mathcal{L}_{Y'Y}$	$\mathcal{L}_{S'S}$	$\mathcal{L}_{Y'S}$
Size	$m \cdot n$	$m \cdot n$	$m \cdot m$	$m \cdot m$	$m \cdot m$

Vectors	$Y_k^T g_k$	$S_k^T g_k$	D_k	g_k	s_k	s_-	x_k	x_t	I_k
Locations	$\mathcal{L}_{Y'g}$	$\mathcal{L}_{S'g}$	$\mathcal{L}_D$	$\mathcal{L}_g$	$\mathcal{L}_s$	$\mathcal{L}_{s_-}$	$\mathcal{L}_x$	$\mathcal{L}_{x_t}$	$\mathcal{L}_I$
Size	m	m	m	n	n	n	n	n	m

$Y_k^T Y_k$, $S_k^T S_k$ and $Y_k^T S_k$ are all referenced through the index table, which itself is an m-vector. For example, the j-th columns of Y_k and S_k are stored physically in the $\mathcal{L}_I[j]$-th columns of arrays $\mathcal{L}_Y$ and $\mathcal{L}_S$ respectively. The (i,j)-th elements of $Y_k^T Y_k$, $S_k^T S_k$ and $Y_k^T S_k$ are stored physically in the $(\mathcal{L}_I[i], \mathcal{L}_I[j])$-th positions of arrays $\mathcal{L}_{Y'Y}$, $\mathcal{L}_{S'S}$ and $\mathcal{L}_{Y'S}$ respectively.

The diagonal matrix D_k is stored as a vector, also through the index table I_k. So, the i-th diagonal element of D_k is stored physically in $\mathcal{L}_D[\mathcal{L}_I[i]]$.

To analyze, we divide Algorithm 3.1 into three stages: matrix operation stage (Steps 3-4), trust region stage (Steps 5-21) and updating stage (Steps 22-24).

• **Matrix Operation Stage:**

For this stage, if we have the two $n \times m$ matrices Y_k and S_k, and the scalar γ_k in memory, then logically, the whole matrix B_k and therefore all its related computations can be retrieved through the $2mn + 1$ memory locations. But for fast computation, several other matrices and vectors are carried on from iteration to iteration. While their information is unnecessary for the accuracy of the result, their presences can save lots of computing time. They are $m \times m$ matrices $Y_k^T Y_k$, $S_k^T S_k$ and $Y_k^T S_k$, m-vectors $Y_k^T g_k$, $S_k^T g_k$ and

D_k.

Vector g_k is special in this stage. Although it appears here, what we all need is actually the product $Q_k^T g_k$, which can be computed by

$$Q_k^T g_k = Y_k^T g_k - \gamma_k S_k^T g_k. \tag{5.1}$$

So, for now, we don't have to allocate any memory for g_k.

All the other matrices and vectors in matrix operation stage are then either negligible or can be retrieved through the matrices/vectors listed in Table 5.1. So totally, we need just $2mn$ memory locations in matrix operation stage, which are for matrices Y_k and S_k.

• **Trust Region Stage:**

This stage is a little bit complicated. It can be divided into two sub-stages. In the contraction sub-stage (Steps 5-12), we need Y_k, S_k, g_k and s_k for Algorithm 3.7 to compute the current trial step s_k and then x_k and x_t for the trial point $x_k + s_k$. In expansion sub-stage (Steps 13-20), we store the last acceptable step in s_- and keep trying new steps with Y_k, S_k, g_k, s_k, x_k and x_t.

All the other matrices and vectors in trust region stage are either negligible or already allocated in matrix operation stage. So totally, we need $5n$ addition memory locations in trust region stage, which are for vectors g_k, s_k, x_k, x_t and s_-.

• **Updating Stage:**

This last stage does not need any additional memory. We can copy g_k to y_{k-m}'s slot in Y_k first and then store g_{k+1} into g_k's location.

Thus, totally, L-SR1 algorithm 3.1 requires $2mn + 5n$ memory locations. See Table 5.1 for the matrices/vectors and their corresponding memory sizes. We will use the table for time cost analysis in the next section.

5.2 Computing Time Analysis

We define the time cost as the number of multiplications/divisions per iteration. Also, since $m \ll n$, any expense with magnitude of m^p $(p \geq 1)$, is negligible. We count only those in magnitude of about n.

Similar to the analysis of memory requirement, we will divide an iteration of Algorithm 3.1 into three stages: matrix operation stage (Steps 3-4), trust region stage (Steps 5-21) and updating stage (Steps 22-24). We will analyze the time cost stage by stage.

- **Matrix Operation Stage:**

At this stage, the computation of initial scaling parameter γ_k, the finiteness assurance and all the eigendecompositions can be very cheap. They deal with $m \times m$ matrices only and therefore cost $O(m^3)$. Matrices W_k and $\bar{W}_k$ can be easily formed from $Y_k^T S_k$, and matrix $Q_k^T Q_k$ can be got from

$$Q_k^T Q_k = Y_k^T Y_k + \gamma_k^2 * S_k^T S_k - \gamma_k * [Y_k^T S_k + (Y_k^T S_k)^T],$$

which costs $O(m^2)$. The vector $Q_k^T g_k$ can be computed by (5.1) and also costs $O(m)$. Sometimes, the computation of γ_k involves $\|y_{k-1}\|^2$ and $y_{k-1}^T s_{k-1}$. They are actually the last diagonal elements of matrices $Y_k^T Y_k$ and $Y_k^T S_k$, respectively. We can see that all the other operations in this stage cost at most $O(m^3)$.

- **Trust Region Stage:**

At this stage, we have an inner loop (Step 8 through 11) to find an acceptable point and sometimes another inner loop (Step 14 through 18) to find a further acceptable point. Every time before we test if a point is acceptable or not, we have to compute the step s_k by Algorithm 3.7, where we can rewrite

s_k's formula as

$$s_k = -\frac{1}{\gamma_k + \nu_k}[g_k - Y_k h_k + S_k(\gamma_k h_k)], \tag{5.2}$$

which needs $(2m+1)n$ multiplications. The directional derivative $g_k^T s_k$ in the acceptance test does not need any major extra multiplication, if we note that

$$g_k^T s_k = -\frac{1}{\gamma_k + \nu_k}\left(\|g_k\|^2 - h_k^T Y_k^T g_k + \gamma_k h_k^T S_k^T g_k\right).$$

So, if the number of function evaluations (which is equal to the number of inner loops) at iteration k is p_k, then the time cost for trust region stage can be $p_k(2m+1)n$.

• **Updating Stage:**

The updating stage updates the matrices and vectors listed in Table 5.1. As mentioned before (Subsection 3.2.2 and Section 5.1), most of these matrices/vectors are maintained through the index vector I_k. So, we don't have any major expense in making them in order. All we concern for this stage is the updates involving computations. Specifically, they are $Y_k^T g_k$, $S_k^T g_k$, $Y_k^T Y_k$, $S_k^T S_k$ and $Y_k^T S_k$. We analyze their expense one by one as follows.

(1) $Y_k^T g_k$ and $S_k^T g_k$: Just compute the two vectors $Y_{k+1}^T g_{k+1}$ and $S_{k+1}^T g_{k+1}$ and then store them in $\mathcal{L}_{Y'g}$ and $\mathcal{L}_{S'g}$ respectively. Cost: $2mn$. But before moving the new information in, we need to preserve the old information so that we can use it for other updatings.

(2) $Y_k^T Y_k$: The updating of $Y_k^T Y_k$ to $Y_{k+1}^T Y_{k+1}$ involves the computation of $\|y_k\|^2$ and the last $(m-1)$ components of vector $Y_k^T y_k$. Since

$$\begin{aligned} \|y_k\|^2 &= -\|g_{k+1}\|^2 + 2y_k^T g_{k+1} + \|g_k\|^2, \\ Y_k^T y_k &= Y_k^T g_{k+1} - Y_k^T g_k, \end{aligned}$$

where $y_k^T g_{k+1}$ is just the last element of $Y_{k+1}^T g_{k+1}$, and the last $(m-1)$

elements of $Y_k^T g_{k+1}$ is just the first $(m-1)$ elements of $Y_{k+1}^T g_{k+1}$, no extra expense is needed.

(3) $S_k^T S_k$: The updating of $S_k^T S_k$ involves the computation of the last $(m-1)$ elements of $S_k^T s_k$. By (5.2),

$$S_k^T s_k = \frac{1}{\gamma_k + \nu_k}\left[(Y_k^T S_k)^T h_k - \gamma_k S_k^T S_k h_k - S_k^T g_k\right].$$

So, the expense is just $O(m^2)$.

(4) $Y_k^T S_k$: The updating of $Y_k^T S_k$ involves the computation of $y_k^T s_k$, and the last $(m-1)$ elements of both $Y_k^T s_k$ and $S_k^T y_k$. Since

$$\begin{aligned} y_k^T s_k &= s_k^T g_{k+1} - g_k^T s_k, \\ S_k^T y_k &= S_k^T g_{k+1} - S_k^T g_k, \\ Y_k^T s_k &= \frac{1}{\gamma_k + \nu_k}\left[Y_k^T Y_k h_k - \gamma_k Y_k^T S_k h_k - Y_k^T g_k\right], \end{aligned}$$

and the last $(m-1)$ elements of $S_k^T g_{k+1}$ is just the first $(m-1)$ elements of $S_{k+1}^T g_{k+1}$, only $O(m^2)$ multiplications are needed.

Totally, we need $2mn$ multiplications for updating stage. All together, we need $[2m + (2m+1)p_k]n$ multiplications for k-th iteration and $2mn \cdot n_{it} + (2m+1) \cdot n_{fn}$ multiplications for solving a certain problem, where n_{it} is the number of iterations and n_{fn} is the number of function evaluations.

5.3 Convergence Analysis

We prove the global convergence and local Q-linear convergence for L-SR1 method in this section. Our results are actually true in general for any method as long as the Hessian approximations B_k are uniformly bounded.

We will have a lemma for the boundedness of L-SR1 matrices first. Then in Subsection 5.3.1, we prove the global convergence of trust region

method (Algorithm 2.3) and in Subsection 5.3.2 we prove the Q-linear local convergence.

Define the level set $\mathcal{D} = \{x \in \mathcal{R}^n : f(x) \leq f(x_0)\}$. We assume $\mathcal{D}$ is convex and compact. Thus, if function $f(x)$ is twice continuously differentiable on $\mathcal{D}$, its Hessian is then bounded. We may assume $\|\nabla^2 f(x)\| \leq C_0$ for all $x \in \mathcal{D}$.

Lemma 5.1 Let $f(x)$ be twice continuously differentiable on the compact set $\mathcal{D}$ and $\{s_k\}$ be a sequence of vectors in $\mathcal{R}^n$. If the initial γ_k is bounded, then the L-SR1 Hessian B_k built by (2.20) with finiteness assurance and index adjustment (Algorithm 3.5) is bounded.

Proof:

What we will prove is actually a more general result: the matrix defined by (3.1) is bounded by

$$\|B_{k,j}\| \leq \left(1 + \frac{1}{\rho}\right)^j \gamma_k + \left[\left(1 + \frac{1}{\rho}\right)^j - 1\right] C_0, \tag{5.3}$$

We prove it by induction.

The correctness for $j = 0$ is trivial. Now we assume (5.3) is true for some $j \geq 0$. By Algorithm 3.5, we then have two possibilities. If we find a candidate index which passes the finiteness assurance, that means condition (3.5) is made true. Then

$$\begin{aligned} \|B_{k,j+1}\| &\leq \|B_{k,j}\| + \frac{\|y_{k-m+j} - B_{k,j} s_{k-m+j}\|}{\rho \|s_{k-m+j}\|} \\ &\leq \left(1 + \frac{1}{\rho}\right) \|B_{k,j}\| + \frac{1}{\rho} C_0 \\ &\leq \left(1 + \frac{1}{\rho}\right)^{j+1} \gamma_k + \left[\left(1 + \frac{1}{\rho}\right)^{j+1} - 1\right] C_0. \end{aligned}$$

If we fail to find such a candidate index, then Algorithm 3.5 resets m to $j + 1$ and returns.

Anyway, we can finally arrive at

$$\|B_{k,m-1}\| \leq \left(1+\frac{1}{\rho}\right)^{m-1}\gamma_k + \left[\left(1+\frac{1}{\rho}\right)^{m-1} - 1\right] C_0.$$

Since $B_k = B_{k,m-1}$, we complete the proof.

□

5.3.1 Global Convergence In this subsection, we prove that the trust region algorithm (Algorithm 2.3) with Hessian approximations B_k uniformly bounded generates a sequence of iterates $\{x_k\}$ such that $\lim \nabla f(x_k) = 0$.

The proof of the following lemma is due to Powell. We make some minor changes.

Lemma 5.2 Let B_k be any $n \times n$ symmetric matrix and s_k be an optimal solution to trust region problem (2.8) solved by Algorithm 2.5. Then for all $k \geq 0$, we have

$$\left| g_k^T s_k + \frac{1}{2} s_k^T B_k s_k \right| \geq \frac{1}{2}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}, \tag{5.4}$$

and

$$|g_k^T s_k| \geq \frac{1}{4}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}. \tag{5.5}$$

Proof:

Define

$$\mathcal{Q}_k(s) = f(x_k) + g_k^T s + \frac{1}{2} s^T B_k s.$$

At first, we prove statement (5.4) or

$$\mathcal{Q}_k(0) - \mathcal{Q}_k(s_k) \geq \frac{1}{2}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}.$$

Let

$$s_c = -\tau_k g_k$$

where

$$\tau_k = \begin{cases} \dfrac{\delta_k}{\|g_k\|}, & \text{if } g_k^T B_k g_k \leq 0 \\ \min\left\{ \dfrac{\|g_k\|^2}{g_k^T B_k g_k}, \dfrac{\delta_k}{\|g_k\|} \right\}, & \text{otherwise.} \end{cases}$$

Since $\|s_c\| \leq \delta_k$ and s_k is the optimal solution to (2.8), we must have that

$$\mathcal{Q}_k(s_k) \leq \mathcal{Q}_k(s_c).$$

If $g_k^T B_k g_k \leq 0$, then

$$\begin{aligned} \mathcal{Q}_k(s_c) &\leq f(x_k) - \delta_k \|g_k\| \\ &\leq f(x_k) - \frac{1}{2}\|g_k\| \min\left\{ \delta_k, \frac{\|g_k\|}{\|B_k\|} \right\}. \end{aligned}$$

Otherwise if $\dfrac{\|g_k\|^2}{g_k^T B_k g_k} \geq \dfrac{\delta_k}{\|g_k\|}$, then

$$\begin{aligned} \mathcal{Q}_k(s_c) &= f(x_k) - \delta_k \|g_k\| + \frac{1}{2}\left(\frac{\delta_k}{\|g_k\|} \right)^2 g_k^T B_k g_k \\ &\leq f(x_k) - \frac{1}{2}\delta_k \|g_k\| \\ &\leq f(x_k) - \frac{1}{2}\|g_k\| \min\left\{ \delta_k, \frac{\|g_k\|}{\|B_k\|} \right\}. \end{aligned}$$

Finally, if $g_k^T B_k g_k > 0$ and $\dfrac{\|g_k\|^2}{g_k^T B_k g_k} < \dfrac{\delta_k}{\|g_k\|}$, then

$$\begin{aligned} \mathcal{Q}_k(s_c) &= f(x_k) - \frac{\|g_k\|^4}{2 g_k^T B_k g_k} \\ &\leq f(x_k) - \frac{\|g_k\|^2}{2\|B_k\|} \\ &\leq f(x_k) - \frac{1}{2}\|g_k\| \min\left\{ \delta_k, \frac{\|g_k\|}{\|B_k\|} \right\}. \end{aligned}$$

Thus, in all cases, we have

$$\mathcal{Q}_k(s_c) \leq f(x_k) - \frac{1}{2}\|g_k\| \min\left\{ \delta_k, \frac{\|g_k\|}{\|B_k\|} \right\}$$

and therefore

$$\begin{aligned} \mathcal{Q}_k(0) - \mathcal{Q}_k(s_k) &\geq \mathcal{Q}_k(0) - \mathcal{Q}_k(s_c) \\ &\geq \frac{1}{2}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}. \end{aligned}$$

Now we prove (5.5). If B_k is semi-positive definite, $g_k^T s_k$ has different sign from $s_k^T B_k s_k$. Then it is obvious that

$$\begin{aligned} |g_k^T s_k| &\geq \left| g_k^T s_k + \frac{1}{2} s_k^T B_k s_k \right| \\ &= \mathcal{Q}_k(0) - \mathcal{Q}_k(s_k) \\ &\geq \frac{1}{2}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}. \end{aligned}$$

If B_k has negative eigenvalues, we may assume the smallest one is $\omega_{min} < 0$. Then the matrix $(B_k + |\omega_{min}|I)$ is semi-positive definite and the trust region problem (2.8) is equivalent to

$$\begin{aligned} &\text{minimize} \quad f(x_k) + g_k^T s + \tfrac{1}{2} s^T (B_k + |\omega_{min}|I) s \\ &\text{such that} \quad \|s\| \leq \delta_k, \end{aligned} \tag{5.6}$$

if we solve both (2.8) and (5.6) by Algorithm 2.5. Thus, in this case,

$$|g_k^T s_k| \geq \frac{1}{2}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k + |\omega_{min}|I\|}\right\}.$$

Since $|\omega_{min}| \leq \|B_k\|$, we then have

$$|g_k^T s_k| \geq \frac{1}{4}\|g_k\| \min\left\{\delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}.$$

The proof is finished.

□

In Algorithm 2.3, Steps 3-6 are the contraction loop. The trust region radius δ_k is decreased in the loop until the acceptance condition (2.2) is met.

As we can imagine, if δ_k and subsequently $\|s_k\|$ go to zero too fast, x_k will be stuck away from the solution. Fortunately, we can prove that δ_k does not go to zero faster than $\|g_k\|$.

Lemma 5.3 Let $f(x)$ be an $\mathcal{R}^n \to \mathcal{R}^1$ twice continuously differentiable function on $\mathcal{D}$. We solve the trust region subproblem (2.8) by Algorithm 2.3 with step computed by Algorithm 2.5 and trust region radius reduced by backtracking (Algorithm 3.8). If the Hessian approximations B_k are symmetric and bounded, then there is a constant $a_1 > 0$ such that the δ_k at the end of Algorithm 2.3 satisfies

$$\delta_k \geq a_1 \|g_k\|$$

for all $k \geq 0$.

Proof:

First we prove that, there is a constant $a_1' > 0$ such that, when the contraction phase stops,

$$\delta_k \geq \min\{a_1' \|g_k\|, \delta_{k-1}\}.$$

By Taylor's theorem, we know that if $\nabla^2 f(x)$ is bounded by C_0, then

$$\begin{aligned} f(x_k + s_k) - f(x_k) - g_k^T s_k &\leq C_0 \|s_k\|^2 \\ &\leq C_0 \delta_k^2. \end{aligned}$$

On the other hand, by (5.5), if $\|B_k\| \leq C_1$, we have

$$1 \leq -\frac{4C_1 g_k^T s_k}{\|g_k\| \min\{C_1 \delta_k, \|g_k\|\}}.$$

Thus,

$$f(x_k + s_k) \leq f(x_k) + \left(1 - \frac{4C_0 C_1 \delta_k^2}{\|g_k\| \min\{C_1 \delta_k, \|g_k\|\}}\right) g_k^T s_k.$$

It is easy to see that there exists a constant $a_1' > 0$ such that, for any δ: $0 < \delta \leq 10a_1'\|g_k\|$, we have

$$0 < \frac{4C_0C_1\delta^2}{\|g_k\| \min\{C_1\delta, \|g_k\|\}} \leq 1 - \alpha.$$

That means the contraction of trust region radius at Step 4 of Algorithm 2.3 will stop once δ_k is reduced to be smaller than $10a_1'\|g_k\|$ or even earlier. Precisely, when the contraction loop (Step 3-6 of Algorithm 2.3) stops,

$$\delta_k \geq \min\{a_1'\|g_k\|, \delta_{k-1}\}.$$

Then, we prove that, there is a constant $a_1'' > 0$ such that, if $\delta_{k-1} < a_1'\|g_k\|$ then the expanding phase will terminate after

$$\delta_k \geq a_1''\|g_k\|.$$

This time, we have

$$\begin{aligned} f(x_k + s_k) - f(x_k) - g_k^T s_k - \frac{1}{2}s_k^T B_k s_k &\leq (C_0 + C_1)\|s_k\|^2 \\ &\leq (C_0 + C_1)\delta_k^2. \end{aligned}$$

On the other hand, by (5.4), we get

$$1 \leq -\frac{2C_1(g_k^T s_k + \frac{1}{2}s_k^T B_k s_k)}{\|g_k\| \min\{C_1\delta_k, \|g_k\|\}}.$$

So,

$$f(x_k) - f(x_k + s_k) \geq \left(1 - \frac{2C_1(C_0 + C_1)\delta_k^2}{\|g_k\| \min\{C_1\delta_k, \|g_k\|\}}\right)\left(-g_k^T s_k - \frac{1}{2}s_k^T B_k s_k\right),$$

or equivalently,

$$\Delta f_k \geq \left(1 - \frac{2C_1(C_0 + C_1)\delta_k^2}{\|g_k\| \min\{C_1\delta_k, \|g_k\|\}}\right)\Delta f_k^{(pred)}.$$

It is easy to see that there exists a constant $a_1'' > 0$ such that, for any δ: $0 < \delta \leq a_1''\|g_k\|$, we have

$$0 < \frac{2C_1(C_0 + C_1)\delta^2}{\|g_k\| \min\{C_1\delta, \|g_k\|\}} \leq 1 - \mu.$$

That means the expansion of trust region radius at Step 10 of Algorithm 2.3 will continue as long as δ_k is smaller than $a_1''\|g_k\|$. Precisely, when the expansion loop (Step 8-12 of Algorithm 2.3) stops,

$$\delta_k \geq a_1''\|g_k\|,$$

or s_k is a Newton step. In the case of a Newton step,

$$\delta_k \geq \|B_k^{-1}g_k\| \geq \frac{1}{C_1}\|g_k\|.$$

Let $a_1 = \min\{a_1', a_1'', 1/C_1\}$, we get

$$\delta_k \geq a_1\|g_k\|.$$

□

The following lemma indicates that the improvement of function values from iteration to iteration can be at least proportional to $\|g_k\|^2$.

Lemma 5.4 Under the same assumption of Lemma 5.3, there is a constant $a_2 > 0$ such that

$$f(x_{k+1}) \leq f(x_k) - a_2\|g_k\|^2 \tag{5.7}$$

is true for all $k \geq 0$.

Proof:

At the end of Algorithm 2.3, either

$$f(x_{k+1}) \leq f(x_k) + \alpha g_k^T s_k, \tag{5.8}$$

or

$$f(x_{k+1}) \leq f(x_k + s_-) \leq f(x_k) + \mu \left(g_k^T s_- + \frac{1}{2} s_-^T B_k s_- \right), \qquad (5.9)$$

where s_- is the step before the last step in the expansion stage (Step 8-12 of Algorithm 2.3).

By Lemma 5.2 and 5.3, we have

$$g_k^T s_k \leq -a_2' \|g_k\|^2$$

where $a_2' = \frac{1}{4} \min\{a_1, 1/C_1\}$. So if (5.8) holds,

$$f(x_{k+1}) \leq f(x_k) - \alpha a_2' \|g_k\|^2.$$

If (5.9) holds, by Lemma 5.2, 5.3 and the fact that $\delta_k = 2\delta_-$ at the end of Algorithm 2.3,

$$\begin{aligned} g_k^T s_- + \frac{1}{2} s_-^T B_k s_- &\leq -\frac{1}{2} \|g_k\| \min\left\{ \delta_-, \frac{\|g_k\|}{\|B_k\|} \right\} \\ &\leq -a_2' \|g_k\|^2. \end{aligned}$$

Thus, we have

$$f(x_{k+1}) \leq f(x_k) - \mu a_2 \|g_k\|^2.$$

Anyway, (5.7) is true if we let $a_2 = \min\{\alpha, \mu\} a_2'$.

□

Now, we can prove the global convergence of Algorithm 2.3. Usually, global convergence means $\lim \nabla f(x_k) = 0$. Our result here is actually stronger: $\sum \|\nabla f(x_k)\|^2 < \infty$.

Theorem 5.5 Let $\{x_k\}$ be a sequence produced by Algorithm 2.1 with the step s_k computed by trust region method (Algorithm 2.3). Under the same conditions as Lemma 5.3, we have

$$\sum_{k=0}^{\infty} \|\nabla f(x_k)\|^2 < \infty,$$

and therefore

$$\lim_{k\to\infty} \nabla f(x_k) = 0.$$

Proof:

If we sum up (5.7) for k from 0 to some sufficient big k_b, we get

$$f(x_{k_b+1}) \leq f(x_0) + a_2 \sum_{k=0}^{k_b} \|g_k\|^2.$$

Since $f(x)$ is bounded below and k_b can be as big as possible, we must then have

$$\sum_{k=0}^{\infty} \|g_k\|^2 < \infty,$$

and therefore

$$\lim_{k\to\infty} \nabla f(x_k) = 0.$$

□

By Lemma 5.1 and Theorem 5.5, we get the following global convergence for the L-SR1 method immediately.

Corollary 5.6 Let $f(x)$ be an $\mathcal{R}^n \to \mathcal{R}^1$ twice continuously differentiable function on the compact set $\mathcal{D}$. If the initial scaling parameters γ_k are uniformly bounded, then the L-SR1 method (Algorithm 3.1) has global convergence, i.e.

$$\lim_{k\to\infty} \nabla f(x_k) = 0.$$

□

Note that, if we use Algorithm 2.4 for the trust region update in the L-SR1 algorithm, Theorem 5.5 and Corollary 5.6 are still true. The proof can be a little bit different. We omit it.

5.3.2 Local Convergence In Subsection 5.3.1, we proved the convergence of g_k to zero. It is then reasonable to assume that the iterates x_k are convergent to some local minimizer x_*. In this subsection, we prove that if x_k does converge to x_*, the rate of convergence can be Q-linear. First, we have the following lemma.

Lemma 5.7 Let x_* be a local minimizer of function $f(x)$ and the Hessian $\nabla^2 f(x)$ be Lipschitz continuous and positive definite in a neighborhood of x_*. If the sequence $\{x_k\}$ converges to x_*, then

(1) there are two constants $a_4, a_5 > 0$, such that

$$f(x) \leq f(x_*) + a_5 \|\nabla f(x)\|^2 \tag{5.10}$$

for all $x : \|x - x_*\| \leq a_6$;

(2) there exists a norm $\|\cdot\|_G$ such that, for any $\epsilon \in (0,1)$ there is an $a_\epsilon > 0$ such that

$$(1-\epsilon)\|x - x_*\|_G^2 \leq f(x) - f(x_*) \leq (1+\epsilon)\|x - x_*\|_G^2 \tag{5.11}$$

for all $x : \|x - x_*\| \leq a_\epsilon$.

Proof:

We prove (5.10) first. Let $G_* = \nabla^2 f(x_*)$. Since $\nabla^2 f(x)$ is continuous, we may expand $\nabla f(x)$ at x_* by

$$\nabla f(x) = G_*(x - x_*) + o(\|x - x_*\|).$$

The Hessian G_* is positive definite. There is a constant $a_6' > 0$ such that

$$\|\nabla f(x)\| \geq \frac{\|x - x_*\|}{2\|G_*^{-1}\|}$$

for all $x : \|x - x_*\| \leq a_6'$.

On the other hand, if we expand $f(x)$ at x_* to its second order, we have

$$f(x) = f(x_*) + \frac{1}{2}(x - x_*)^T G_*(x - x_*) + o(\|x - x_*\|^2). \tag{5.12}$$

Obviously, there is a constant $a_6'' > 0$ such that,

$$f(x) \leq f(x_*) + \|G_*\|\|x - x_*\|^2$$

for all $x : \|x - x_*\| \leq a_6''$.

Let $a_6 = \min\{a_6', a_6''\}$ and $a_5 = 4\|G_*\|\|G_*^{-1}\|^2$. We then have (5.10).

Next, we prove (5.11). By assumption, G_* is positive definite. We can define the vector norm $\|\cdot\|_G$ in $\mathcal{R}^n$ by

$$\|v\|_G = \frac{1}{2} v^T G_* v.$$

Thus, (5.12) can be equivalent to

$$f(x) = f(x_*) + \|x - x_*\|_G^2 + o(\|x - x_*\|_G^2).$$

It is easy to see that, given $\epsilon \in (0, 1)$, there is a constant $a_\epsilon > 0$ dependent on ϵ such that (5.11) is true for all $x : \|x - x_*\| \leq a_\epsilon$.

□

Now we can prove the Q-linear convergence of the trust region algorithm (Algorithm 2.3).

Theorem 5.8 Let x_* be a local minimizer of $f(x)$, at which assumptions of Lemma 5.7 hold. Let $\{x_k\}$ be a sequence produced by Algorithm 2.1 with the step s_k computed by trust region method (Algorithm 2.3). If $\{x_k\}$ converges to x_* and the conditions of Lemma 5.3 are true, then the rate of convergence is Q-linear, that is, there is a constant $c \in [0, 1)$ and a norm $\|\cdot\|_G$ such that

$$\frac{\|x_{k+1} - x_*\|_G}{\|x_k - x_*\|_G} \leq c$$

for all k sufficiently large.

Proof:

Since the convergence of $\{x_k\}$ to x_* is assumed, there is a constant k' such that $\|x_k - x_*\| \leq a_6$ for all $k \geq k'$, where $a_6 > 0$ is the constant defined by Lemma 5.7.

Then by Lemma 5.4 and 5.7, we have

$$\frac{1}{a_5}(f(x_k) - f(x_*)) \leq \|g_k\|^2 \leq \frac{1}{a_2}(f(x_k) - f(x_{k+1}))$$

for all $k \geq k'$. That is,

$$f(x_{k+1}) - f(x_*) \leq \left(1 - \frac{a_2}{a_5}\right)[f(x_k) - f(x_*)]. \tag{5.13}$$

Note, since $\{f(x_k)\}$ is a monotonously decreasing sequence converging to $f(x_*)$, we must have

$$0 < \frac{a_2}{a_5} \leq 1.$$

Therefore there exists an $\epsilon \in (0, 1)$ such that

$$0 \leq \frac{1+\epsilon}{1-\epsilon}\left(1 - \frac{a_2}{a_5}\right) < 1.$$

We let

$$c \equiv \sqrt{\frac{1+\epsilon}{1-\epsilon}\left(1 - \frac{a_2}{a_5}\right)}.$$

By the second claim of Lemma 5.7, there is a constant $a_\epsilon > 0$ such that (5.11) holds for all $x : \|x - x_*\| \leq a_\epsilon$. By the convergence of $\{x_k\}$, there is a constant $k_0 \geq k'$ such that $\|x_k - x_*\| \leq a_\epsilon$ for all $k \geq k_0$. Thus, if we combine (5.11) and (5.13) together for $k \geq k_0$, we will get

$$\|x_{k+1} - x_*\|_G^2 \leq \frac{1+\epsilon}{1-\epsilon}\left(1 - \frac{a_2}{a_5}\right)\|x_k - x_*\|_G^2,$$

or

$$\frac{\|x_{k+1} - x_*\|_G}{\|x_k - x_*\|_G} \leq c.$$

□

By Lemma 5.1 and Theorem 5.8, we get the following local convergence for the L-SR1 method immediately.

Corollary 5.9 Let x_* be a local minimizer of $f(x)$, at which assumptions of Lemma 5.7 hold. Let $\{x_k\}$ be a sequence produced by the L-SR1 method (Algorithm 3.1). If $\{x_k\}$ converges to x_* and the initial scaling parameters γ_k are uniformly bounded, then $\{x_k\}$ converges Q-linearly, that is, there is a constant $c \in [0, 1)$ and a norm $\|\cdot\|_G$ such that

$$\frac{\|x_{k+1} - x_*\|_G}{\|x_k - x_*\|_G} \leq c$$

for all k sufficiently large.

□

CHAPTER 6

NUMERICAL EXPERIMENT

We implemented the L-SR1 algorithm on SUN/SPARK workstation with FORTRAN 77 programming language. Mathematics library BLAS is used for basic matrix/vector operations and LAPACK routines are used for basic algebraic computations.

6.1 Test Problems

Most of our test problems are selected from CUTE library. CUTE is a testing environment developed by Conn, Gould and Toint (1994) for numerical optimization algorithms. It consists of two parts: problems set and utility tools. The current version collects 794 test problems. All problems are written in the standard input format (SIF). CUTE supplies utilities to build interfaces between this input format and other software packages. Usually, a user does not have to know much about the encoding technique used in writing SIF.

All CUTE problems are well classified. We can select the desired problems from CUTE database by using a tool called `select`, which allows users to choose unconstrained problems with different sizes. There are two kinds of CUTE problems: fixed-size and variable-size. For a variable-size problem, there is a corresponding SIF file, where all its candidate sizes are listed. We can set the size manually by editing this file.

We have three sets of problems selected from CUTE.

- Set I: These are problems we used to compare standard SR1 method

with Osborne and Sun's scaled SR1 method. Their sizes are all small (about 2-4), except for Problem SROSENBR (10,50). The results listed in Table 4.3 and 4.4 were run on these problems.

- Set II: These are problems we used for our preliminary test and study of L-SR1 method. They consist of all CUTE unconstrained problems which can have size 100. The results listed in Table 4.1 were run on these problems.
- Set III: These are problems we used for our final comparisons between L-SR1 method and L-BFGS method. They consist of all CUTE unconstrained problems which can have size between 1000 and 5000. The only exception is JIMACK, which takes too long to make function and gradient evaluations and therefore is excluded. For problems which have more than one candidate dimensions between 1000 and 5000, we choose the one closer 5000. There are totally 55 problems in Set III. The results listed in Table 6.2 through 6.14 are based on Set III problems.

In addition to CUTE problems, we also construct some strictly convex quadratic problems with their eigenvalues generated randomly. The purpose of these problems is to find the behavior of a given algorithm, and therefore assist our analysis and design. We never use them to formally compare any two algorithms. For that goal, problems in this have the simplest structure,

$$f(x) = x^T G x$$

where G is a diagonal matrix whose diagonal entries are generated randomly. Let `random` be a random number from $[0, 1]$. For each dimension

$n = 5, 10, 15, 20$ and 50, we have two problems with their diagonal entries generated by $10^{-2} + \texttt{random}^2$ and $10^{-3} + \texttt{random}^3$, respectively. Therefore there are totally 10 problems in this set. We may denote this set as Set IV.

6.2 Comparison of L-SR1 Method with L-BFGS Method

We finally tried the L-SR1 with γ_k computed by Algorithm 4.5 and compared it with the result by L-BFGS method. The stopping criterion in Step 2 of Algorithm 3.1 is

$$\max_{1 \le i \le n} \left| g_k^{(i)} \max\left\{ |x_k^{(i)}|, 1 \right\} \right| \le gradtol * \max\{f(x_k), 1\}$$

where we set $gradtol = 10^{-5}$. We also changed the L-BFGS code so that it has the same stopping criterion.

We set the maximum number of iterations to 2000. Any problem which does not converge within 2000 iterations is considered a failure. We test Set III problems with $m = 4, 5, 7$ and 10. The results are listed in Table 6.3 and 6.4 for $m = 4$, Table 6.5 and 6.6 for $m = 5$, Table 6.9 and 6.10 for $m = 7$, and Table 6.13 and 6.14 for $m = 10$, where the meaning of the field "cod" is

0: the two methods converge to the same solution;
1: the two methods converge to different solutions;
2: maximum number of iterations reached (failure);
3: the algorithm can not proceed (failure).

A summary of the results is listed in Table 6.1, where the fields "tot_itn", "tot_fun" and "tot_grd" are counted over those problems for which both L-SR1 and L-BFGS are successful (cod=0) and converge to the same solution; "twin" means the number of problems for which both L-SR1 and L-BFGS

Table 6.1: L-SR1/L-BFGS for Large Problems (Summary)

m	tot_itn	tot_fun	tot_grd	twin	tie	win	fail
4	5317/5748	8011/6345	5358/6345	2	8	26/19	10/10
5	8578/9179	11827/9814	8622/9814	1	10	27/17	8/10
6	10561/10699	14079/11362	10605/11362	2	7	26/20	7/9
7	9587/10006	12398/10546	9631/10546	3	9	23/20	7/8
8	11247/11507	17679/12380	11293/12380	2	10	24/20	7/8
10	6881/6808	9887/7330	6924/7330	2	8	18/27	9/8

are successful but converges to different solutions (cod=1); "tie" is the number of problems for which L-SR1 and L-BFGS converge to the same solution with exactly the same number of iterations, or both of the two methods fail; "win" counts the number of problems for which both methods have the same solution (cod=0) and the current method has few iterations than the other, or the current method is successful (cod=0) and the other fails (cod=2,3); "fail" is the number of problems which fail the test (cod=2,3).

It looks like that both methods are competitive with each other. The L-SR1 needs a little fewer iterations than the L-BFGS method for $m = 4$ through 8. But in all cases, L-SR1 method needs more function evaluations. As with the number of gradient evaluations, the comparison can be unfair for the L-BFGS method. The L-BFGS code evaluates the gradient every time when the objective function is evaluated, even though it is not necessary sometimes. However, the L-BFGS method needs at least the same number of gradient evaluations as that of iterations. So, talking about the number of gradient evaluations, the L-SR1 method wins.

Table 6.2 is a comparison of the L-SR1 method to itself with different values of m, where the fields of "tot_itn", "tot_fun", "tot_grd", "twin", "tie", "win" and "fail" have almost the same meanings as those of Table 6.1 except

Table 6.2: L-SR1/L-SR1 for Different m's (Summary)

m/m	tot_itn	tot_fun	twin	tie	win	fail
4/5	5736/5692	8595/8809	1	21	13/20	10/8
5/6	8949/9079	12345/12754	1	24	17/13	8/7
6/7	11135/10594	14988/14179	0	17	19/19	7/7
7/8	10594/10027	14179/13903	0	22	18/15	7/7
8/10	7177/7475	10686/10858	0	21	24/10	7/9

for that the comparison in Table 6.2 is between two different runs of the same L-SR1 algorithm.

We can see that, as the value of m grows, the performance of the algorithm is not always improved. In the "win" column, it appears that smaller m wins more problems than bigger m. We do not know if it is just an accident or reflects some pattern of the L-SR1 method.

Table 6.3: L-SR1/L-BFGS for Large Problems ($m = 4$)

Pname	dim	itn	fun	grd	cod
ARWHEAD	5000	8/	14/	9/	0/3
BDQRTIC	1000	163/73	280/92	164/92	0/0
BROYDN7D	1000	29/351	44/359	30/359	1/1
BRYBND	5000	98/29	127/33	99/33	0/0
CHAINWOO	1000	382/	526/	383/	0/2
COSINE	1000	7/6	9/16	8/16	0/0
CRAGGLVY	5000	51/55	62/65	52/65	0/0
DIXMAANA	3000	7/10	11/12	8/12	0/0
DIXMAANB	3000	10/11	12/13	11/13	0/0
DIXMAANC	3000	13/12	16/14	14/14	0/0
DIXMAAND	3000	12/13	15/15	13/15	0/0
DIXMAANE	3000	262/223	312/241	263/241	0/0
DIXMAANF	3000	167/203	181/211	168/211	0/0
DIXMAANG	3000	175/208	200/222	176/222	0/0
DIXMAANH	3000	167/168	200/177	168/177	0/0
DIXMAANI	3000	919/1085	1157/1129	920/1129	0/0
DIXMAANJ	3000	150/159	169/169	151/169	0/0
DIXMAANK	3000	134/150	153/160	135/160	0/0
DIXMAANL	3000	122/137	139/144	123/144	0/0
DQDRTIC	5000	6/13	10/21	7/21	0/0
DQRTIC	5000	65/56	74/64	66/64	0/0
EDENSCH	2000	18/20	31/24	19/24	0/0
EG2	1000	7/4	19/5	8/5	0/0
ENGVAL1	5000	12/13	15/15	13/15	0/0
FLETCBV2	1000	1/479	2/498	2/498	0/0
FLETCBV3	1000	/	/	/	2/2
FMINSURF	1024	244/194	347/200	245/200	0/0
FREUROTH	5000	15/14	39/20	16/20	0/0

Table 6.4: L-SR1/L-BFGS for Large Problems ($m = 4$), continue

Pname	dim	itn	fun	grd	cod
INDEF	1000	/	/	/	3/3
LIARWHD	5000	21/23	38/28	22/28	0/0
MOREBV	5000	31/26	35/28	32/28	0/0
MSQRTALS	1024	/	/	/	2/2
MSQRTBLS	1024	/1919	/2013	/2013	2/0
NCB20	1010	570/366	991/429	571/429	0/0
NONCVXU2	1000	/1426	/1498	/1498	2/0
NONCVXUN	1000	/	/	/	2/2
NONDIA	5000	9/20	16/24	10/24	0/0
NONDQUAR	5000	1008/1123	1797/1241	1009/1241	0/0
NONMSQRT	1024	/	/	/	2/2
PENALTY1	1000	92/54	241/66	93/66	0/0
POWELLSG	5000	37/49	61/53	38/53	0/0
POWER	1000	146/135	243/146	147/146	0/0
QUARTC	5000	65/56	74/64	66/64	0/0
SCHMVETT	5000	11/13	18/15	12/15	0/0
SINQUAD	5000	123/265	445/379	124/379	0/0
SPARSINE	1000	/	/	/	2/2
SPARSQUR	1000	29/24	34/25	30/25	0/0
SPMSRTLS	4999	219/194	237/206	220/206	0/0
SROSENBR	5000	12/16	17/17	13/17	0/0
TESTQUAD	1000	/	/	/	2/2
TOINTGSS	5000	4/14	12/19	5/19	0/0
TQUARTIC	5000	16/19	52/25	17/25	0/0
TRIDIA	5000	/	/	/	2/2
VAREIGVL	5000	99/16	120/20	100/20	0/0
WOODS	1000	105/16	242/20	106/20	1/1

Table 6.5: L-SR1/L-BFGS for Large Problems ($m = 5$)

Pname	dim	itn	fun	grd	cod
ARWHEAD	5000	8/	14/	9/	0/3
BDQRTIC	1000	111/94	188/113	112/113	0/0
BROYDN7D	1000	35/344	46/355	36/355	1/1
BRYBND	5000	98/27	130/30	99/30	0/0
CHAINWOO	1000	346/	494/	347/	0/2
COSINE	1000	7/6	9/16	8/16	0/0
CRAGGLVY	5000	42/49	64/59	43/59	0/0
DIXMAANA	3000	7/10	11/12	8/12	0/0
DIXMAANB	3000	10/11	12/13	11/13	0/0
DIXMAANC	3000	13/12	16/14	14/14	0/0
DIXMAAND	3000	13/13	16/15	14/15	0/0
DIXMAANE	3000	236/224	245/231	237/231	0/0
DIXMAANF	3000	171/200	173/205	172/205	0/0
DIXMAANG	3000	145/176	148/184	146/184	0/0
DIXMAANH	3000	154/199	184/205	155/205	0/0
DIXMAANI	3000	851/1109	883/1144	852/1144	0/0
DIXMAANJ	3000	135/140	137/144	136/144	0/0
DIXMAANK	3000	119/124	127/130	120/130	0/0
DIXMAANL	3000	98/124	101/129	99/129	0/0
DQDRTIC	5000	6/13	10/20	7/20	0/0
DQRTIC	5000	90/56	125/64	91/64	0/0
EDENSCH	2000	19/21	35/25	20/25	0/0
EG2	1000	7/4	19/5	8/5	0/0
ENGVAL1	5000	11/13	14/15	12/15	0/0
FLETCBV2	1000	1/504	2/521	2/521	0/0
FLETCBV3	1000	/	/	/	2/2
FMINSURF	1024	266/198	363/207	267/207	0/0
FREUROTH	5000	13/14	25/21	14/21	0/0

Table 6.6: L-SR1/L-BFGS for Large Problems ($m = 5$), continue

Pname	dim	itn	fun	grd	cod
INDEF	1000	/	/	/	3/3
LIARWHD	5000	20/24	43/27	21/27	0/0
MOREBV	5000	27/27	31/29	28/29	0/0
MSQRTALS	1024	/	/	/	2/2
MSQRTBLS	1024	/	/	/	2/2
NCB20	1010	484/360	900/416	485/416	0/0
NONCVXU2	1000	1570/1957	1771/2022	1571/2022	0/0
NONCVXUN	1000	/	/	/	2/2
NONDIA	5000	9/18	16/23	10/23	0/0
NONDQUAR	5000	1223/963	2588/1088	1224/1088	0/0
NONMSQRT	1024	/	/	/	2/2
PENALTY1	1000	101/50	265/60	102/60	0/0
POWELLSG	5000	30/39	65/46	31/46	0/0
POWER	1000	155/131	245/136	156/136	0/0
QUARTC	5000	90/56	125/64	91/64	0/0
SCHMVETT	5000	12/13	19/15	13/15	0/0
SINQUAD	5000	158/176	472/234	159/234	0/0
SPARSINE	1000	/	/	/	2/2
SPARSQUR	1000	34/24	39/25	35/25	0/0
SPMSRTLS	4999	210/190	226/205	211/205	0/0
SROSENBR	5000	12/18	17/20	13/20	0/0
TESTQUAD	1000	/	/	/	2/2
TOINTGSS	5000	4/14	12/18	5/18	0/0
TQUARTIC	5000	16/18	49/26	17/26	0/0
TRIDIA	5000	1687/1727	1765/1797	1688/1797	0/0
VAREIGVL	5000	95/16	106/20	96/20	0/0
WOODS	1000	18/17	36/21	19/21	0/0

Table 6.7: L-SR1/L-BFGS for Large Problems ($m = 6$)

Pname	dim	itn	fun	grd	cod
ARWHEAD	5000	8/14	14/35	9/35	0/3
BDQRTIC	1000	113/91	182/103	114/103	0/0
BROYDN7D	1000	32/347	43/356	33/356	1/1
BRYBND	5000	99/27	126/30	100/30	0/0
CHAINWOO	1000	435/	594/	436/	0/2
COSINE	1000	7/6	9/16	8/16	0/0
CRAGGLVY	5000	39/50	58/59	40/59	0/0
DIXMAANA	3000	7/11	11/13	8/13	0/0
DIXMAANB	3000	10/11	12/13	11/13	0/0
DIXMAANC	3000	13/12	16/14	14/14	0/0
DIXMAAND	3000	13/14	16/16	14/16	0/0
DIXMAANE	3000	232/227	234/234	233/234	0/0
DIXMAANF	3000	172/208	174/215	173/215	0/0
DIXMAANG	3000	155/168	158/175	156/175	0/0
DIXMAANH	3000	158/193	203/204	159/204	0/0
DIXMAANI	3000	571/1163	573/1201	572/1201	0/0
DIXMAANJ	3000	138/139	140/144	139/144	0/0
DIXMAANK	3000	123/127	126/132	124/132	0/0
DIXMAANL	3000	61/139	64/146	62/146	0/0
DQDRTIC	5000	6/13	10/20	7/20	0/0
DQRTIC	5000	88/56	97/64	89/64	0/0
EDENSCH	2000	25/20	42/25	26/25	0/0
EG2	1000	7/4	19/5	8/5	0/0
ENGVAL1	5000	11/13	14/15	12/15	0/0
FLETCBV2	1000	1/721	2/749	2/749	0/0
FLETCBV3	1000	/	/	/	2/3
FMINSURF	1024	251/209	411/215	252/215	0/0
FREUROTH	5000	13/15	28/22	14/22	0/0

Table 6.8: L-SR1/L-BFGS for Large Problems ($m = 6$), continue

Pname	dim	itn	fun	grd	cod
INDEF	1000	/	/	/	3/3
LIARWHD	5000	21/25	47/29	22/29	0/0
MOREBV	5000	26/28	30/30	27/30	0/0
MSQRTALS	1024	/	/	/	2/2
MSQRTBLS	1024	1957/1928	1976/1986	1958/1986	0/0
NCB20	1010	566/325	1087/375	567/375	0/0
NONCVXU2	1000	1725/1431	1986/1479	1726/1479	0/0
NONCVXUN	1000	/	/	/	2/2
NONDIA	5000	9/20	16/24	10/24	0/0
NONDQUAR	5000	1356/1141	2959/1270	1357/1270	0/0
NONMSQRT	1024	/	/	/	2/2
PENALTY1	1000	116/50	288/60	117/60	0/0
POWELLSG	5000	30/42	60/49	31/49	0/0
POWER	1000	165/124	241/127	166/127	0/0
QUARTC	5000	88/56	97/64	89/64	0/0
SCHMVETT	5000	12/13	20/15	13/15	0/0
SINQUAD	5000	149/222	436/296	150/296	0/0
SPARSINE	1000	/	/	/	2/2
SPARSQUR	1000	36/24	41/25	37/25	0/0
SPMSRTLS	4999	201/188	213/197	202/197	0/0
SROSENBR	5000	12/15	17/16	13/16	0/0
TESTQUAD	1000	/	/	/	2/2
TOINTGSS	5000	4/13	12/17	5/17	0/0
TQUARTIC	5000	16/19	52/27	17/27	0/0
TRIDIA	5000	1663/1382	1675/1426	1664/1426	0/0
VAREIGVL	5000	96/16	101/20	97/20	0/0
WOODS	1000	99/17	258/21	100/21	1/1

Table 6.9: L-SR1/L-BFGS for Large Problems ($m = 7$)

Pname	dim	itn	fun	grd	cod
ARWHEAD	5000	8/14	14/17	9/17	0/0
BDQRTIC	1000	56/48	101/56	57/56	0/0
BROYDN7D	1000	31/351	53/360	32/360	1/1
BRYBND	5000	90/27	114/30	91/30	0/0
CHAINWOO	1000	398/3670	575/3957	399/3957	1/1
COSINE	1000	7/6	9/16	8/16	0/0
CRAGGLVY	5000	42/52	77/61	43/61	0/0
DIXMAANA	3000	7/11	11/13	8/13	0/0
DIXMAANB	3000	11/11	13/13	12/13	0/0
DIXMAANC	3000	14/12	17/14	15/14	0/0
DIXMAAND	3000	13/14	16/16	14/16	0/0
DIXMAANE	3000	211/223	213/231	212/231	0/0
DIXMAANF	3000	186/162	188/170	187/170	0/0
DIXMAANG	3000	162/190	165/197	163/197	0/0
DIXMAANH	3000	170/158	227/163	171/163	0/0
DIXMAANI	3000	785/1127	787/1160	786/1160	0/0
DIXMAANJ	3000	132/139	134/142	133/142	0/0
DIXMAANK	3000	138/112	141/119	139/119	0/0
DIXMAANL	3000	55/111	58/116	56/116	0/0
DQDRTIC	5000	6/12	10/20	7/20	0/0
DQRTIC	5000	96/56	114/64	97/64	0/0
EDENSCH	2000	23/21	40/25	24/25	0/0
EG2	1000	7/4	19/5	8/5	0/0
ENGVAL1	5000	12/13	15/15	13/15	0/0
FLETCBV2	1000	1/526	2/537	2/537	0/0
FLETCBV3	1000	/	/	/	2/2
FMINSURF	1024	274/210	433/219	275/219	0/0
FREUROTH	5000	14/14	29/21	15/21	0/0

Table 6.10: L-SR1/L-BFGS for Large Problems ($m = 7$), continue

Pname	dim	itn	fun	grd	cod
INDEF	1000	/	/	/	3/3
LIARWHD	5000	22/23	40/27	23/27	0/0
MOREBV	5000	28/26	32/28	29/28	0/0
MSQRTALS	1024	/	/	/	2/2
MSQRTBLS	1024	1779/1813	1801/1853	1780/1853	0/0
NCB20	1010	482/305	931/358	483/358	1/1
NONCVXU2	1000	1487/1627	1782/1671	1488/1671	0/0
NONCVXUN	1000	/	/	/	2/2
NONDIA	5000	9/21	16/25	10/25	0/0
NONDQUAR	5000	966/914	2297/1015	967/1015	0/0
NONMSQRT	1024	/	/	/	2/2
PENALTY1	1000	112/50	280/60	113/60	0/0
POWELLSG	5000	29/32	60/38	30/38	0/0
POWER	1000	168/125	233/130	169/130	0/0
QUARTC	5000	96/56	114/64	97/64	0/0
SCHMVETT	5000	11/13	19/15	12/15	0/0
SINQUAD	5000	226/234	557/313	227/313	0/0
SPARSINE	1000	/	/	/	2/2
SPARSQUR	1000	39/24	44/25	40/25	0/0
SPMSRTLS	4999	193/190	212/200	194/200	0/0
SROSENBR	5000	12/17	17/18	13/18	0/0
TESTQUAD	1000	/	/	/	2/2
TOINTGSS	5000	4/14	12/18	5/18	0/0
TQUARTIC	5000	14/19	46/27	15/27	0/0
TRIDIA	5000	1777/1519	1789/1559	1778/1559	0/0
VAREIGVL	5000	95/16	100/20	96/20	0/0
WOODS	1000	96/17	222/21	97/21	1/1

Table 6.11: L-SR1/L-BFGS for Large Problems ($m = 8$)

Pname	dim	itn	fun	grd	cod
ARWHEAD	5000	8/15	14/18	9/18	0/0
BDQRTIC	1000	98/55	153/63	99/63	0/0
BROYDN7D	1000	34/351	55/367	35/367	1/1
BRYBND	5000	97/28	123/32	98/32	0/0
CHAINWOO	1000	474/	730/	475/	0/2
COSINE	1000	7/6	9/16	8/16	0/0
CRAGGLVY	5000	44/46	80/54	45/54	0/0
DIXMAANA	3000	7/11	11/13	8/13	0/0
DIXMAANB	3000	11/11	13/13	12/13	0/0
DIXMAANC	3000	14/12	17/14	15/14	0/0
DIXMAAND	3000	14/14	17/16	15/16	0/0
DIXMAANE	3000	211/222	213/232	212/232	0/0
DIXMAANF	3000	172/158	174/168	173/168	0/0
DIXMAANG	3000	141/160	144/164	142/164	0/0
DIXMAANH	3000	176/161	236/168	177/168	0/0
DIXMAANI	3000	754/1105	756/1133	755/1133	0/0
DIXMAANJ	3000	159/148	161/155	160/155	0/0
DIXMAANK	3000	134/115	144/121	135/121	0/0
DIXMAANL	3000	62/121	65/127	63/127	0/0
DQDRTIC	5000	6/12	10/19	7/19	0/0
DQRTIC	5000	122/56	247/64	123/64	0/0
EDENSCH	2000	23/20	44/25	24/25	0/0
EG2	1000	7/4	19/5	8/5	0/0
ENGVAL1	5000	12/13	24/15	13/15	0/0
FLETCBV2	1000	1/505	2/518	2/518	0/0
FLETCBV3	1000	/	/	/	2/2
FMINSURF	1024	258/200	441/208	259/208	0/0
FREUROTH	5000	13/14	28/21	14/21	0/0

Table 6.12: L-SR1/L-BFGS for Large Problems ($m = 8$), continue

Pname	dim	itn	fun	grd	cod
INDEF	1000	/	/	/	3/3
LIARWHD	5000	22/24	39/28	23/28	0/0
MOREBV	5000	27/27	31/29	28/29	0/0
MSQRTALS	1024	/	/	/	2/2
MSQRTBLS	1024	1747/1858	1770/1904	1748/1904	0/0
NCB20	1010	341/359	713/409	342/409	0/0
NONCVXU2	1000	1103/1536	1447/1570	1104/1570	0/0
NONCVXUN	1000	/	/	/	2/2
NONDIA	5000	9/21	16/25	10/25	0/0
NONDQUAR	5000	836/1237	2233/1385	837/1385	0/0
NONMSQRT	1024	/	/	/	2/2
PENALTY1	1000	110/50	272/60	111/60	0/0
POWELLSG	5000	30/39	57/44	31/44	0/0
POWER	1000	164/130	205/133	165/133	0/0
QUARTC	5000	122/56	247/64	123/64	0/0
SCHMVETT	5000	12/13	20/15	13/15	0/0
SINQUAD	5000	186/240	455/330	187/330	0/0
SPARSINE	1000	/	/	/	2/2
SPARSQUR	1000	42/24	47/25	43/25	0/0
SPMSRTLS	4999	205/184	220/192	206/192	0/0
SROSENBR	5000	12/17	17/18	13/18	0/0
TESTQUAD	1000	/	/	/	2/2
TOINTGSS	5000	4/14	12/18	5/18	0/0
TQUARTIC	5000	16/19	51/27	17/27	0/0
TRIDIA	5000	1795/1336	1807/1383	1796/1383	0/0
VAREIGVL	5000	87/16	92/20	88/20	0/0
WOODS	1000	98/16	222/21	99/21	1/1

Table 6.13: L-SR1/L-BFGS for Large Problems ($m = 10$)

Pname	dim	itn	fun	grd	cod
ARWHEAD	5000	8/15	14/17	9/17	0/0
BDQRTIC	1000	57/72	85/82	58/82	0/0
BROYDN7D	1000	32/353	59/366	33/366	1/1
BRYBND	5000	98/28	126/31	99/31	0/0
CHAINWOO	1000	475/	720/	476/	0/2
COSINE	1000	7/6	9/16	8/16	0/0
CRAGGLVY	5000	55/51	113/63	56/63	0/0
DIXMAANA	3000	7/11	11/13	8/13	0/0
DIXMAANB	3000	11/11	13/13	12/13	0/0
DIXMAANC	3000	15/12	18/14	16/14	0/0
DIXMAAND	3000	15/14	18/16	16/16	0/0
DIXMAANE	3000	261/216	263/225	262/225	0/0
DIXMAANF	3000	190/162	192/169	191/169	0/0
DIXMAANG	3000	221/162	231/167	222/167	0/0
DIXMAANH	3000	194/157	290/162	195/162	0/0
DIXMAANI	3000	778/1047	780/1072	779/1072	0/0
DIXMAANJ	3000	174/136	176/142	175/142	0/0
DIXMAANK	3000	139/133	154/140	140/140	0/0
DIXMAANL	3000	62/120	65/126	63/126	0/0
DQDRTIC	5000	6/12	10/19	7/19	0/0
DQRTIC	5000	123/56	211/64	124/64	0/0
EDENSCH	2000	22/21	44/26	23/26	0/0
EG2	1000	7/4	19/5	8/5	0/0
ENGVAL1	5000	15/13	18/15	16/15	0/0
FLETCBV2	1000	1/509	2/520	2/520	0/0
FLETCBV3	1000	/	/	/	2/2
FMINSURF	1024	268/198	526/210	269/210	0/0
FREUROTH	5000	13/14	28/21	14/21	0/0

Table 6.14: L-SR1/L-BFGS for Large Problems ($m = 10$), continue

Pname	dim	itn	fun	grd	cod
INDEF	1000	/	/	/	3/3
LIARWHD	5000	22/23	39/27	23/27	0/0
MOREBV	5000	27/26	31/28	28/28	0/0
MSQRTALS	1024	/	/	/	2/2
MSQRTBLS	1024	/1766	/1830	/1830	2/0
NCB20	1010	423/296	950/328	424/328	0/0
NONCVXU2	1000	/1502	/1542	/1542	2/0
NONCVXUN	1000	/	/	/	2/2
NONDIA	5000	9/21	16/25	10/25	0/0
NONDQUAR	5000	583/1102	1659/1241	584/1241	0/0
NONMSQRT	1024	/	/	/	2/2
PENALTY1	1000	114/50	290/60	115/60	0/0
POWELLSG	5000	28/42	48/45	29/45	0/0
POWER	1000	168/123	212/126	169/126	0/0
QUARTC	5000	123/56	211/64	124/64	0/0
SCHMVETT	5000	11/13	18/15	12/15	0/0
SINQUAD	5000	174/236	437/317	175/317	0/0
SPARSINE	1000	/	/	/	2/2
SPARSQUR	1000	47/24	71/25	48/25	0/0
SPMSRTLS	4999	390/186	409/198	391/198	0/0
SROSENBR	5000	12/17	17/18	13/18	0/0
TESTQUAD	1000	/	/	/	2/2
TOINTGSS	5000	4/14	12/18	5/18	0/0
TQUARTIC	5000	15/19	50/27	16/27	0/0
TRIDIA	5000	1898/1364	1910/1400	1899/1400	0/0
VAREIGVL	5000	86/16	91/20	87/20	0/0
WOODS	1000	87/16	192/21	88/21	1/1

CHAPTER 7

SUMMARY

In the past twenty years, limited memory methods have been becoming very popular. At first, only the BFGS method can be efficiently implemented with the limited memory idea. Then the work by Byrd, Nocedal and Schnabel (1992) sparked another surge of interest in limited memory methods. With the compact representation of quasi-Newton matrices, we can now virtually implement a limited memory version of any methods in the Broyden family. Not only the line search method we can use as global convergence strategy, but also the trust region method.

The behavior of the L-BFGS method has been studied intensively in the past twenty years. It is stable and fast for most of test problems, as we have noted in this study. As to the L-SR1 method, we can now say it is at least a prospective method worth further studying. For small m ($4 \leq m \leq 8$), it can be as good as the L-BFGS method, although it usually requires more function evaluations.

Like all other methods, the performance of the L-SR1 method can be affected by many factors. While we don't know much about all these factors, one thing is for sure, the initial scaling parameter γ_k. It has the tendency that a bigger γ_k is usually better than a small one. Among the five choices, $y_{k-1}^T s_{k-1}/\|s_{k-1}\|^2$, $\|y_{k-1}\|/\|s_{k-1}\|$, $\|y_{k-1}\|^2/y_{k-1}^T s_{k-1}$, $0.9*\underline{\gamma}_k$ and $1.1*\bar{\gamma}_k$, $1.1*\bar{\gamma}_k$ tends to be the best. It generates more Newton steps, requires fewer function

evaluations and converges faster than the other four choices.

The choice of $\gamma_k = 1.1 * \bar{\gamma}_k$ is worth pondering, as $1.1 * \bar{\gamma}_k$ tends to be bigger than the biggest eigenvalue of the Hessian $\nabla^2 f(x_k)$. When we start with $\gamma_k I$ with such a choice of γ_k, every eigenvalue of $\gamma_k I$ is "bigger" than all eigenvalues of the true Hessian $\nabla^2 f(x_k)$. After we apply SR1 update m times, m of these eigenvalues are adjusted "down" to where they are supposed to be in the subspace spanned by $\{s_{k-j}\}_{j=1}^{m}$. Now the scenario is that, "bigger" eigenvalues reflect the unknown subspace and "smaller" eigenvalues reflect the subspace which has been well represented in the Hessian approximation. Since the direction $-(B_k + \nu_k)^{-1} g_k$ tends to go along in the subspace of small eigenvalues, we can imagine that the predicted function reduction along this direction "matches" well the real function reduction. Thus, a Newton step is more likely accepted as an acceptable step.

BIBLIOGRAPHY

[1] Buckley, A.G. **A Combined Conjugate Gradient Quasi-Newton Minimization Algorithm,** Mathematical Programming 15 (1978a) 200-210.

[2] Buckley, A.G. **Extending the Relationship between the Conjugate Gradient and BFGS Algorithms,** Mathematical Programming 15 (1978b) 343-348.

[3] Buckley, A.G. and A. LeNir. **QN-like Variable Storage Conjugate Gradients,** Mathematical Programming 27 (1983) 155-175.

[4] Byrd, R.H., J. Nocedal and R. Schnabel. **Representations of Quasi-Newton Matrices and Their Use in Limited Memory Methods,** Department of Electrical Engineering and Computer Science, Northwestern University, Technical Report NAM-03, March 1992.

[5] Byrd, R.H., Humaid Fayez Khalfan and R. Schnabel. **Analysis of a Symmetric Rank-One Trust Region Method,** Department of Computer Science, University of Colorado at Boulder, CU-CS-657-93, July 1993.

[6] Coleman, T.F. **Large Sparse Numerical Optimization,** Notes in Computer Science, Vol.165, Springer-Verlag, 1984.

[7] Conn, A.R., N.I.M. Gould and Ph.L. Toint. **Convergence of Quasi-Newton Matrices Generated by the Symmetric Rank One Update,** Mathematical Programming 50 (1991) 177-195.

[8] Davidon, William. **Optimally Conditioned Optimization Algorithms without Line Searches,** Mathematical Programming 9 (1975) 1-30.

[9] Dennis, J.E. Jr. and R.B. Schnabel. **Numerical Methods for Unconstrained Optimization and Nonlinear Equations,** Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1983.

[10] Edwards, C.H. Jr. **Advanced Calculus of Several Variables,** Academic Press, 1973

[11] Fletcher, R. **Practical Methods of Optimization,** John Wiley & Sons, 1987.

[12] Gibson, Tamara, Dianne P. O'leary and Larry Nazareth. **BFGS with Update Skipping and Varying Memory,** University of Maryland, College Park, CS-TR-3663, July 1996.

[13] Gilbert, J.C. and C. Lemaréchal. **Some Numerical Experiments with Variable-storage Quasi-Newton Algorithm,** Mathematical Programming 45 (1989) 407-435.

[14] Khalfan, Humaid, R.H. Byrd and R.B. Schnabel. **A Theoretical and Experimental Study of the Symmetric Rank One Update,** Department of Computer Science, University of Colorado at Boulder, CU-CS-489-90, December 1990.

[15] Kelly, C.T. and E.W. Sachs **Local Convergence of the Symmetric Rank-One Iteration,** May, 1995.

[16] Liu, D.C. and J. Nocedal. **On the Limited Memory BFGS Method for Large Scale Optimization,** Mathematical Programming 45 (1989) 503-528.

[17] Moré, J.J., B.S. Garbow and K.E. Hillstrom. **Testing Unconstrained Optimization Software,** ACM Transactions on Mathematical Software 7, 1981, 17-41.

[18] Nash, S.G. **Solving Nonlinear Programming Problems Using Truncated-Newton Techniques,** Numerical Optimization 1984, P.T. Boggs, R.H. Byrd and R.B. Schnabel, eds., SIAM, Philadelphia, 1985, 119-136.

[19] Nash, S.G. and J. Nocedal. **A Numerical Study of the Limited Memory BFGS Method and the Truncated-Newton Method for Large Scale Optimization,** SIAM J. Optimization, Vol.1, No.3, August 1991, 358-372.

[20] Nazareth, L. **A Relationship between the BFGS and Conjugate Gradient Algorithms and its Implications for New Algorithms,** SIAM J. Numerical Analysis, Vol.16, No.5, October 1979, 794-800.

[21] Nocedal, J. **Updating Quasi-Newton Matrices with Limited Storage,** Mathematics of Computation, Vol.35, No.151, July 1980, 773-782.

[22] Nocedal, J. **Theory of Algorithms for Unconstrained Optimization,** Acta Numerica, Vol.1, 1992, 199-242.

[23] Osborne, M.R. and L.P. Sun. **A New Approach to the Symmetric Rank-one Updating Algorithm,** Report NMO/01, Department of Statistics, IAS, Australian National University, March 1988.

[24] Ortega, J.M. and W.C. Rheinboldt. **Iterative Solution of Nonlinear Equations in Several Variables,** Academic Press, New York, 1970.

[25] Powell, M.J.D. **A New Algorithm for Unconstrained Optimization,** Nonlinear Programming, J.B. Rosen, O.L. Mangasarian and K. Ritter,, eds., Academic Press, New York, 1970, 31-65.

[26] Powell, M.J.D. **Convergence Properties of a Class of Minimization Algorithms,** Nonlinear Programming 2, O.L. Mangasarian, R.R. Meyer and S.M. Robinson, eds., Academic Press, New York, 1975, 1-27.

[27] Powell, M.J.D. **Updating Conjugate Directions by the BFGS Formula,** DAMTP 1985/NA11, 1-28.

[28] Sally,Paul J. **The Symmetric Rank One Update for Large Scale Optimization,** Department of Electrical Engineering and Computer Science, Northwestern University, May, 1992

[29] Schlick, Tamar and Aaron Fogelson. **TNPACK – A Truncated Newton Minimization Package for Large-Scale Problems: I. Algorithm and Usage,** ACM Transactions on Mathematical Software, Vol.18, No.1, March 1992, 46-70.

[30] Shanno, D.F. **Conjugate Gradient Methods with Inexact Searches,** Mathematics of Operations Research Vol.3, No.3, August 1978, 244-256.

[31] Shultz, G.A., R.B. Schnabel and R.H. Byrd. **A Family of Trust-Region-Based Algorithms for Unconstrained Minimization with Strong Global Convergence Properties,** SIAM J. Numerical Analysis, Vol.22, No.1, February 1985, 47-67.

[32] Sorensen, D.C. **Newton's Method with a Model Trust Region Modification,** SIAM J. Numerical Analysis, Vol.19, No.2, April 1982, 409-426.

APPENDIX A

PROBABILITY OF SR1 COSINE VALUE

Theorem A.1 Given $\theta \in (0, \pi/2]$ and $e \in \mathcal{R}^n$. If we denote the angle between e and vector $x \in \mathcal{R}^n$ by φ_x and define the cosine value of φ_x by

$$\cos \varphi_x = x^T e/(\|x\|\|e\|),$$

then the probability that $|\cos \varphi_x| \leq \cos \theta$ can be estimated by

$$P_\theta = \frac{\int_\theta^{\frac{\pi}{2}} \sin^{n-2} t \, dt}{\int_0^{\frac{\pi}{2}} \sin^{n-2} t \, dt} \leq (n-1) \cos \theta,$$

if the uniform distribution of x in $\mathcal{R}^n$ is assumed.

Proof:

Without generality, we can assume that $e \in \mathcal{R}^n$ is the unit vector with its last element being one and all the others zero, i.e.

$$e = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

Let $\mathcal{S}_\theta \subset \mathcal{R}^n$ be the intersection of the unit sphere with the cone of central angle θ, i.e.

$$\mathcal{S}_\theta = \{x \in \mathcal{R}^n : \|x\| = 1, x^T e \geq \cos \theta\}.$$

Obviously, $\mathcal{S}_\pi$ is the unit sphere.

We use the notation $\mathbf{P}_{\mathcal{G}}(\mathcal{B})$ to denote the probability of events in $\mathcal{B}$ over the events space $\mathcal{G}$. As we know from the theory of probability,

$$\mathbf{P}_{\mathcal{G}}(\mathcal{B}) = \frac{\mathbf{M}(\mathcal{B})}{\mathbf{M}(\mathcal{G})},$$

where $\mathbf{M}$ is the measure defined over the space $\mathcal{G}$. If $\mathcal{G}$ is an $(n-1)$-dimensional surface in $\mathcal{R}^n$, then $\mathbf{M}(\mathcal{B})$ denotes the area of $\mathcal{B}$.

Thus,

$$\begin{aligned} P_\theta &= \mathbf{P}_{\mathcal{R}^n}(\{x : |\cos\varphi_x| \le \cos\theta\}) \\ &= \mathbf{P}_{\mathcal{S}_\pi}(\{x : |x^T e| \le \cos\theta, \|x\| = 1\}) \\ &= 1 - 2 \cdot \mathbf{P}_{\mathcal{S}_\pi}(\mathcal{S}_\theta) \\ &= 1 - \frac{2 \cdot \mathbf{M}(\mathcal{S}_\theta)}{\mathbf{M}(\mathcal{S}_\pi)}. \end{aligned}$$

Consider the mapping $x = \mathbf{m}(\phi) : \mathcal{R}^{n-1} \to \mathcal{R}^n$:

$$\mathbf{m}(\phi) = \begin{bmatrix} \sin\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ \cos\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ \vdots \\ \cos\phi_{n-2} \sin\phi_{n-1} \\ \cos\phi_{n-1} \end{bmatrix}, \quad \phi = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{n-1} \end{bmatrix},$$

and the domain $\mathcal{D}_\theta \subset \mathcal{R}^{n-1}$:

$$\mathcal{D}_\theta = (0, 2\pi] \times (0, \pi] \cdots (0, \pi] \times (0, \theta].$$

From Mathematical Analysis we know that $\mathbf{m}$ is a one-to-one mapping between $\mathcal{D}_\theta$ and $\mathcal{S}_\theta$. We now try to compute $\mathbf{M}(\mathcal{S}_\theta)$ through the transformation $\mathbf{m}$.

Be definition (see Edwards (1973) for reference),

$$\mathbf{M}(\mathcal{S}_\theta) = \int_{\mathcal{D}_\theta} \sqrt{\det(G_\theta)}\, d\phi$$

where G_θ is an $(n-1)\times(n-1)$ matrix whose ij-th element is

$$g_{ij} = \left\langle \frac{\partial \mathbf{m}}{\partial \phi_i}, \frac{\partial \mathbf{m}}{\partial \phi_j} \right\rangle.$$

It is not hard to verify that

$$\frac{\partial \mathbf{m}}{\partial \phi_1} = \begin{bmatrix} \cos\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ -\sin\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ 0 \end{bmatrix} \begin{matrix} \leftarrow \text{ 1-th} \\ \leftarrow \text{ 2-th} \\ \leftarrow \text{ all the rest,} \end{matrix}$$

$$\frac{\partial \mathbf{m}}{\partial \phi_{n-1}} = \frac{\cos\phi_{n-1}}{\sin\phi_{n-1}} \begin{bmatrix} \sin\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ \cos\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ \vdots \\ \cos\phi_{n-2} \sin\phi_{n-1} \\ -\frac{1}{\cos\phi_{n-1}} \sin^2\phi_{n-1} \end{bmatrix} \begin{matrix} \leftarrow \text{ 1-th} \\ \leftarrow \text{ 2-th} \\ \vdots \\ \leftarrow (n-1)\text{-th} \\ \leftarrow n\text{-th,} \end{matrix}$$

and for $i = 2, 3, \ldots, n-2$,

$$\frac{\partial \mathbf{m}}{\partial \phi_i} = \frac{\cos\phi_i}{\sin\phi_i} \begin{bmatrix} \sin\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ \cos\phi_1 \sin\phi_2 \cdots \sin\phi_{n-1} \\ \vdots \\ \cos\phi_{i-1} \sin\phi_i \cdots \sin\phi_{n-1} \\ -\frac{1}{\cos\phi_i} \sin^2\phi_i \cdots \sin\phi_{n-1} \\ 0 \end{bmatrix} \begin{matrix} \leftarrow \text{ 1-th} \\ \leftarrow \text{ 2-th} \\ \vdots \\ \leftarrow i\text{-th} \\ \leftarrow (i+1)\text{-th} \\ \leftarrow \text{ all the rest.} \end{matrix}$$

So, G_θ is a diagonal matrix whose i-th diagonal entry can be expressed as

$$g_{ii} = \sin^2\phi_{i+1} \cdots \sin^2\phi_{n-1},$$

for $i = 1, 2, \ldots, n-2$ and the last entry $g_{n-1,n-1} = 1$. Thus

$$\det(G_\theta) = \left(\sin^1\phi_2 \cdots \sin^{n-3}\phi_{n-2} \sin^{n-2}\phi_{n-1}\right)^2.$$

Therefore,

$$\begin{aligned}
&\mathbf{M}(\mathcal{S}_\theta) = \int_{\mathcal{D}_\theta} \sqrt{\det(G_\theta)}\, d\phi \\
= &\int_0^{2\pi}\int_0^{\pi}\cdots\int_0^{\pi}\int_0^{\theta} \sin^1\phi_2 \cdots \sin^{n-3}\phi_{n-2}\sin^{n-2}\phi_{n-1}\, d\phi_{n-1}\, d\phi_{n-2}\cdots d\phi_2\, d\phi_1 \\
= &\ 2\pi\int_0^{\pi}\sin\phi_2\, d\phi_2 \cdots \int_0^{\pi}\sin^{n-3}\phi_{n-2}\, d\phi_{n-2}\int_0^{\theta}\sin^{n-2}\phi_{n-1}\, d\phi_{n-1},
\end{aligned}$$

and

$$\begin{aligned}
P_\theta &= 1 - 2\cdot\frac{\int_0^{\theta}\sin^{n-2}\phi_{n-1}d\phi_{n-1}}{\int_0^{\pi}\sin^{n-2}\phi_{n-1}d\phi_{n-1}} \\
&= \frac{\int_\theta^{\frac{\pi}{2}}\sin^{n-2}t\, dt}{\int_0^{\frac{\pi}{2}}\sin^{n-2}t\, dt} \\
&\le \frac{\int_\theta^{\frac{\pi}{2}} dt}{\int_0^{\frac{\pi}{2}}\left(\frac{2}{\pi}t\right)^{n-2} dt} \\
&= (n-1)\left(1-\frac{2}{\pi}\theta\right) \\
&\le (n-1)\cos\theta.
\end{aligned}$$

□

APPENDIX B

AN ALTERNATIVE EIGENDECOMPOSITION OF L-SR1 MATRIX

Given the L-SR1 Hessian (2.20), we talked about its eigendecomposition in Section 3.3. There is, however, another way to eigendecompose the L-SR1 Hessian. Although both methods give the same decomposition, they have different numerical stability.

Let D_k be an $m \times m$ diagonal scaling matrix,

$$D_k = \text{diag}(\|s_{k-m}\|, \ldots, \|s_{k-2}\|, \|s_{k-1}\|).$$

If t is the rank of the $n \times m$ matrix Q_k, then we can do eigendecomposition to the $m \times m$ matrix (we may call it the scaled $Q_k^T Q_k$).

$$D_k^{-1} Q_k^T Q_k D_k^{-1} = U_k \Sigma_k^2 U_k^T$$

to get Σ_k and U_k, where Σ_k is an $t \times t$ diagonal matrix with all diagonal entries positive, and U_k is an $m \times t$ full rank orthogonal matrix.

The next step is to do eigendecomposition again to the compacted $t \times t$ matrix

$$\Sigma_k U_k^T D_k M_k^{-1} D_k U_k \Sigma_k = V_k \Lambda_k V_k^T,$$

where Λ_k is $t \times t$ diagonal and V_k is $t \times t$ orthogonal.

Now we have all the information required by trust region method to compute the step s_k. Let

$$P_k = Q_k D_k^{-1} U_k \Sigma_k^{-1} V_k,$$

and Z_k be an orthogonal base in the null space of Q_k, $Q_k^T Z_k = 0$. Then we assert that the $n \times n$ matrix

$$K_k = [P_k \;\; Z_k],$$

is orthogonal, and that B_k has the eigendecomposition:

$$B_k = \gamma_k I + Q_k M_k^{-1} Q_k^T = K_k \begin{bmatrix} \gamma_k I + \Lambda_k & 0 \\ 0 & \gamma_k I \end{bmatrix} K_k^T.$$

APPENDIX C

PSEUDO-CODE OF THE L-SR1 ALGORITHM

main program:

CALL read_in;

CALL set_ready;

DO WHILE ($done < 0$)

    CALL pre_iterate;

    CALL new_iterate;

    CALL stop_check;

END DO

CALL report.

subroutine pre_iterate:

$itn = itn + 1$;

$m = m + 1$;

IF $(m = 0)$ RETURN;

$m = m - 1$;

update m and IND_{orig};

$IND = IND_{orig}$;

$y_c = g_+ - g_c$;

$\gamma = y_c^T y_c / y_c^T s_c$;

update D, S, Y, STS, YTY and YTS;

$x_c = x_+$;

$f_c = f_+$;

$g_c = g_+$;

CALL eigen_decomp;

$\Omega = \gamma I + \Lambda$;

$PG = V^T * \Sigma^{-1} * U^T * D^{-1} * Q^T * g_c$;

$ZG = \|g_c\|^2 - \|PG\|^2$;

subroutine new_iterate:

/* find an acceptable point first */

$\delta_{orig} = \delta$;

$control = 1$;

DO WHILE $(control < 10)$

CALL new_sc$(\delta, s_c, \nu, newton)$;

$x_+ = x_c + s_c$;

$f_+ = f(x_+)$;

$\Delta f = f_+ - f_c$;

IF $(\Delta f < \alpha * g_c^T * s_c)$ THEN

IF $(newton)$ THEN

$control = 13$;

$\delta = \|s_c\|$;

ELSE

$control = 10 + control$;

END IF

ELSE

CALL redc_delta(δ);

IF $(\delta < steptol)$ THEN

$control = 100$; /* terminate the program */

ELSE

$control = 2$;

END IF

END IF

END DO

$\delta = \max(\delta, 0.1 * \delta_{orig})$;

/* try to make a larger stride */

IF $(control = 11)$ THEN

$\Delta f_p = 0.5 * (g_c^T * s_c - \nu * s_c^T * s_c)$;

IF $(|\Delta f_p - \Delta f| \leq \beta |\Delta f|)$ $control = 51$;

DO WHILE $(control > 50)$

$\delta_t = \delta$;

$x_t = x_+$;

$s_t = s_c$;

$f_t = f_+$;

$\nu_t = \nu$;

$\delta = 2 * \delta$;

CALL new_sc$(\delta, s_c, \nu, newton)$;

$x_+ = x_c + s_c$;

$f_+ = f(x_+)$;

$\Delta f = f_+ - f_c$;

$\Delta f_p = 0.5 * (g_c^T * s_c - \nu_t * s_c^T * s_c)$;

IF $(newton$ .OR. $f_+ \geq f_t$ .OR. $\Delta f > \alpha * g_c^T * s_c$ .OR.

$(|\Delta f_p - \Delta f| > \beta|\Delta f|$ .AND. $\Delta f \geq g_c^T * s_c))$ THEN

$control = control - 10;$

IF $(f_+ \geq f_t)$ THEN

$\delta = \delta_t;$

$x_+ = x_t;$

$s_c = s_t;$

$f_+ = f_t;$

$\nu = \nu_t;$

ELSE IF $(newton)$ THEN

$\delta = \|s_c\|;$

END IF

ELSE

$control = 52;$

END IF

END DO

END IF

$g_+ = \nabla f(x_+);$

subroutine redc_delta:

$\Delta f = f_+ - f_c - g_c^T s_c;$

IF $(\Delta f \neq 0)$ THEN

$\delta_{new} = -0.5 * g_c^T s_c * \|s_c\|/\Delta f;$

IF $(\delta_{new} > 0)$ THEN

IF $(\delta_{new} < 0.1 * \delta)$ THEN

$\delta = 0.1 * \delta;$

ELSE IF $(\delta_{new} > 0.5 * \delta)$ THEN

$\delta = 0.5 * \delta$;

ELSE

$\delta = \delta_{new}$;

END IF

END IF

END IF

subroutine eigen_decomp:

call effect$(\gamma, IND, D, YTY, YTS, STS, DL, H, QTQ)$;

$U * \Sigma * U^T =$ eigendecomposition(QTQ);

$r =$ number_of_positive(Σ);

$\Sigma =$ square_root_of_positive(Σ);

$DM = \Sigma * U^T * DL^T * H^{-1} * DL * U * \Sigma$;

$V * \Lambda * V^T =$ eigendecomposition(DM);

subroutine effect:

/* QTQ is for $D_k^{-1} Q_k^T Q_k D_k^{-1}$ */

/* DM is for $D_k^{-1} M_k D_k^{-1}$ */

$H, DL, DM, QTQ = \emptyset$;

$k, last = 1$;

DO WHILE $(k \leq m)$

$v = D^{-1} * [YTY + \gamma^2 * STS - \gamma * (YTS + YTS^T)]_{1:k-1,k} / D_{k,k}$;

$\alpha = [YTY + \gamma^2 * STS - \gamma * (YTS + YTS^T)]_{k,k} / D_{k,k}^2$;

$QTQ = \begin{bmatrix} QTQ & v \\ v^T & \alpha \end{bmatrix}$; /* scaled QTQ */

$v = D^{-1} * [W - \gamma * STS]_{1:k-1,k} / D_{k,k}$;

$\alpha = [W - \gamma * STS]_{k,k} / D_{k,k}^2$;

$$DM = \begin{bmatrix} DM & v \\ v^T & \alpha \end{bmatrix}; \quad /* \text{ scaled } DM \ */$$

$l = -DL^T * H^{-1} * DL * DM_{1:k-1,k}$;

$h = DM_{k,k} + l^T * DM_{1:k-1,k}$;

$H = \text{diag}(H, h)$; /* scaled H */

$$DL = \begin{bmatrix} DL & 0 \\ l^T & 1 \end{bmatrix}; \quad /* \text{ scaled } DL \ */$$

$$\cos^2 = h^2 / \left([\, l^T \ 1 \,][QTQ]_{1:k,1:k} \begin{bmatrix} l \\ 1 \end{bmatrix} \right);$$

IF ($|\cos| > \epsilon$) THEN

$k, last = k + 1$;

ELSE

$last = last + 1$;

switch $IND(k)$ with $IND(last)$;

END IF

END DO

subroutine new_sc:

define $\|s(\nu)\|^2 = \|(\Omega + \nu I)^{-1} * PG\|^2 + ZG/(\gamma + \nu)^2$;

IF ($r = 0$) THEN

$\nu = \sqrt{ZG}/\delta - \gamma$;

IF ($\nu < 0$) THEN

$\nu = 0$;

$newton = 1$;

ELSE

$newton = 0$;

END IF

ELSE IF (all $\Omega > 0$) THEN

$\nu_0 = 0$;

IF ($\|s(\nu_0)\|^2 > \delta^2$) THEN

$\nu_0 = \sqrt{PG^T * PG + ZG}/\delta - (\gamma + \min(\Omega))/2$;

IF ($\nu_0 < 0$) $\nu_0 = 0$;

solve $\|s(\nu)\|^2 = \delta^2$ for ν with ν_0 given;

$newton = 0$;

ELSE

$\nu = 0$;

$newton = 1$;

END IF

ELSE

$\nu_0 = \sqrt{PG^T * PG + ZG}/\delta - \min(\Omega)$;

solve $\|s(\nu)\|^2 = \delta^2$ for ν with ν_0 given;

$newton = 0$;

END IF

$s_c = s(\nu)$;

subroutine stop_check:

IF ($\|g_+\|_\infty \leq gradtol$)

$done = 0$;

ELSE IF ($\|s_c\|_\infty \leq steptol$)

$done = 1$;

ELSE IF ($itn \geq maxitn$)

$done = 2$;

ELSE IF ($\delta \leq steptol$)

$done = 3$;

ELSE

$done = -1$;

END IF

subroutine read_in:

read in n, m_max;

read in initial point x_0;

read in controling parameters $steptol, gradtol, maxitn$;

subroutine set_ready:

$\alpha = 10^{-4}$;

$\beta = 10$;

$\gamma = 1$;

$itn = 0$;

$m = -1$;

$x_c = x_0$;

$f_c = f(x_c)$;

$g_c = \nabla f(x_c)$;

$\delta = 0.01 * \|g_c\|$;

establish D, Y, S, YTY, STS, YTS and IND;

IF ($\|g_c\|_\infty \leq gradtol$)

$done = 0$;

ELSE

$done = -1$;

END IF