Furman University
Electronic Journal of Undergraduate Mathematics
Volume **12, 5 – 20, 2007**

THE RELATIONSHIPS BETWEEN CG, BFGS, AND TWO LIMITED-MEMORY ALGORITHMS

ZHIWEI (TONY) QIN

Abstract. For the solution of linear systems, the conjugate gradient (CG) and BFGS are among the most popular and successful algorithms with their respective advantages. The limited-memory methods have been developed to combine the best of the two. We describe and examine CG, BFGS, and two limited-memory methods (L-BFGS and VSCG) in the context of linear systems. We focus on the relationships between each of the four algorithms, and we present numerical results to illustrate those relationships.

1. Introduction

Systems of linear equations arise in such diverse areas as digital signal processing, forecasting, and telecommunications. Hence solving linear systems in an efficient and robust manner has significant scientific and social impact. Here, we are concerned with finding the solution to

$$Ax = b, \tag{1.1}$$

where $x \in \mathbb{R}^n$, and A is an $n \times n$ symmetric positive definite matrix. In the past fifty years, many numerical algorithms have been proposed to achieve this goal. Among them, the most well-known and established are the conjugate gradient (CG) method and the family of quasi-Newton (QN) methods. Although quasi-Newton methods are normally thought of as nonlinear minimization algorithms, they can be used to solve systems of linear equations by instead applying them to the quadratic problem

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2} x^T A x - b^T x. \tag{1.2}$$

In this thesis, whenever the quasi-Newton family is conerned, we will focus on the BFGS method, which has been proved most effective among all the Quasi-Newton methods. The delicate relationship between CG and BFGS has been explored extensively in the past, and new limited-memory algorithms based on CG and BFGS have been proposed to address the problem of large memory requirement for BFGS. Two competing algorithms of this type are the L-BFGS method described by Nocedal [8] and the variable storage conjugate gradient (VSCG) method published by Buckley and LeNir [2]. In this thesis, we describe, in the context of linear systems, the CG, BFGS, and the limited-memory methods with a unified approach emphasizing the relationships between each of them. We compare their performances on

Received by the editors December 5, 2007.
2000 *Mathematics Subject Classification.* 65K10, 90C53.
Key words and phrases. Numerical optimization, Conguate gradient, BFGS, Quasi-Newton, Limited-memory.
The author would like to express his sincere gratitude and appreciation to his sponsor, Professor Michael Friedlander, for all the help and guidance throughout this project.

test matrices, in particular, highly ill-conditioned matrices, and we present the results of numerical experiments. We close with some recommendations on when to use the respective algorithms.

2. Notation

In this thesis, lower-case roman letters denote column vectors, and upper-case letters denote matrices. Greek letters are reserved for scalars. In the context of preconditioned CG (PCG), we use H_0 for the inverse of the preconditioner M, and H_k for the updated matrix based on H_0 at iteration k. In the context of BFGS, H_k denotes the k-th approximation to the inverse of the Hessian matrix. For both CG and BFGS at iteration k, x_k denotes the current approximate solution, and d_k is the search direction. We write

$$\begin{aligned} g_k &= \nabla f(x_k), \\ y_{k+1} &= g_{k+1} - g_k, \\ s_{k+1} &= x_{k+1} - x_k = \alpha_k d_k, \end{aligned}$$

where α_k is the step length determined by exact line-search as

$$\alpha_k = \frac{g_k^T H_0 g_k}{d_k^T A d_k}. \tag{2.1}$$

We will see later on why we use the same letters for CG and BFGS. The function $U(H_k, y_{k+1}, s_{k+1})$ will be used to represent the BFGS update formula for H_k, i.e.

$$H_{k+1} = U(H_k, y_{k+1}, s_{k+1}) \tag{2.2}$$

$$= H_k - \frac{s_{k+1} y_{k+1}^T H_k + H_k y_{k+1} s_{k+1}^T}{s_{k+1}^T y_{k+1}} + \left(1 + \frac{y_{k+1}^T H_k y_{k+1}}{s_{k+1}^T y_{k+1}}\right) \frac{s_{k+1} s_{k+1}^T}{s_{k+1}^T y_{k+1}}. \tag{2.3}$$

3. The Conjugate Gradient method

The linear Conjugate Gradient method was first introduced by Hestenes and Stiefel [5]. Here we present the most standard form.

Algorithm 3.1. CG ([9, Algorithm 5.2])
Initialization x_0, $g_0 = Ax_0 - b$, $d_0 = -g_0$, $k = 0$
while not converged

$$\begin{aligned} \alpha_k &= \frac{g_k^T g_k}{d_k^T A d_k} \\ x_{k+1} &= x_k + \alpha_k d_k \\ g_{k+1} &= g_k + \alpha_k A d_k \\ \beta_{k+1} &= \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k} \\ d_{k+1} &= -g_{k+1} + \beta_{k+1} d_k \\ k &= k + 1 \end{aligned}$$

end

The following theorem for linear CG will be useful later on and can be found in many textbooks, such as [9].

Theorem 3.1. *For linear CG at iteration k, suppose x_k is not yet the solution of (1.1), the following properties hold:*

$$g_k^T g_i = 0 \quad \text{for } i = 0, \cdots, k-1, \tag{3.1}$$

$$d_k^T A d_i = 0 \quad \text{for } i = 0, \cdots, k-1. \tag{3.2}$$

In addition, with exact line-search (2.1), we have

$$g_k^T d_i = 0 \quad \text{for } i = 0, \cdots, k-1. \tag{3.3}$$

Since $\alpha_i d_i = s_{i+1}$, it follows that

$$g_k^T s_{i+1} = 0. \tag{3.4}$$

CG is often employed with the preconditioner M to improve its performance, especially on ill-conditioned matrices. PCG simply transforms the original linear system that CG solves by a change of variable, $\hat{x} = Rx$, where $M = R^T R$. Now instead of solving $Ax = b$, we solve the new system

$$(R^{-T} A R^{-1})\hat{x} = R^{-T} b$$

with the hope that $R^{-T} A R^{-1}$ has better eigenvalue distribution than A. In practice, the preconditioner often comes in the form of $M = R^T R$, which is symmetric positive definite, and so is its inverse H_0. Equation (3.1) in the theorem above thus becomes

$$g_k^T H_0 g_i = 0 \quad \text{for } i = 0, \cdots, k-1. \tag{3.5}$$

In the algorithm we present here, we actually use H_0 instead of M to ease the comparison with BFGS. Of course, $H_0 = M$ when $M = I$, and PCG reduces to the standard CG method.

Algorithm 3.2. PCG ([9, Algorithm 5.3])

Initialization x_0, preconditioner H_0, $g_0 = Ax_0 - b$, $d_0 = -H_0 g_0$, $k = 0$

while not converged

$$\begin{aligned}
\alpha_k &= \frac{g_k^T H_0 g_k}{d_k^T A d_k} \\
x_{k+1} &= x_k + \alpha_k d_k \\
g_{k+1} &= g_k + \alpha_k A d_k \\
\beta_{k+1} &= \frac{g_{k+1}^T H_0 g_{k+1}}{g_k^T H_0 g_k} \\
d_{k+1} &= -H_0 g_{k+1} + \beta_{k+1} d_k \\
k &= k + 1
\end{aligned}$$

end

We can see that the only differences between PCG and CG are the initial search direction d_0 and the "correction coefficient" β_{k+1}. We also note that linear PCG in this form is exactly same as the preconditioned Fletcher-Reeves nonlinear CG (FR-CG) method [3], except that FR-CG requires a line-search for computing α_k. However, when applied on a quadratic function, we can assume that all line-searches are exact, and we keep this assumption for the remaining parts of this thesis unless otherwise specified. There are many other forms of the nonlinear CG method,

which differ from FR-CG only on the choice of β_{k+1}. The Hestenes-Stiefel form (HS-CG) [5] with a preconditioner H_0 defines

$$\beta_{k+1}^{\text{HS}} = \frac{g_{k+1}^T H_0 y_{k+1}}{y_{k+1}^T d_k}. \tag{3.6}$$

Here, we show that HS-CG is equivalent to FR-CG on quadratic functions. First, recall that

$$d_{k+1} = -H_0 g_{k+1} + \beta_{k+1}^{\text{FR}} d_k, \quad \text{for each } k. \tag{3.7}$$

Hence,

$$\begin{aligned} g_k^T d_k &= -g_k^T H_0 g_k + \beta_k^{FR} g_k^T d_{k-1} \\ &= -g_k^T H_0 g_k. \end{aligned}$$

Now,

$$\begin{aligned} \beta_{k+1}^{\text{HS}} &= \frac{g_{k+1}^T H_0 (g_{k+1} - g_k)}{(g_{k+1} - g_k)^T d_k} \\ &= \frac{g_{k+1}^T H_0 g_{k+1}}{-g_k^T d_k} \\ &= \frac{g_{k+1}^T H_0 g^{k+1}}{g_k^T H_0 g_k} \\ &= \beta_{k+1}^{\text{FR}}, \end{aligned}$$

which by Algorithm 3.2, we have

$$\beta_{k+1}^{\text{FR}} = \beta_{k+1}.$$

Let us consider HS-CG. If

$$Q_k = H_0 - \frac{s_k y_k^T H_0}{s_k^T y_k},$$

then

$$\begin{aligned} d_{k+1} &= -H_0 g_{k+1} + \left(\frac{g_{k+1}^T H_0 y_{k+1}}{y_{k+1}^T d_k} \right) d_k \\ &= -H_0 g_{k+1} + \left(\frac{g_{k+1}^T H_0 y_{k+1}}{y_{k+1}^T s_{k+1}} \right) s_{k+1} \\ &= -(H_0 - \frac{s_{k+1} y_{k+1}^T H_0}{s_{k+1}^T y_{k+1}}) g_{k+1} \\ &= -Q_{k+1} g_{k+1} \\ d_k &= -Q_k g_k. \end{aligned}$$

Note that Q_k is not symmetric positive definite in most cases. Shanno [11] suggested that we can augment/pad Q_k to get

$$H_k = H_0 - \frac{s_k y_k^T H_0 + H_0 y_k s_k^T}{s_k^T y_k} + \left(1 + \frac{y_k^T H_0 y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k}, \tag{3.8}$$

so that H_k is symmetric positive definite. By definition of the function U in (2.2) and (2.3),

$$H_k^{\text{PCG}} = U(H_0, y_k, s_k). \tag{3.9}$$

Buckley and LeNir [2] claim that $-Q_k g_k$ and $-H_k g_k$ generate the identical search directions. We give the detailed derivation here. With exact line-search and (3.4), we have

$$\begin{aligned}
H_k g_k &= H_0 g_k - \left(\frac{s_k y_k^T H_0 + H_0 y_k s_k^T}{s_k^T y_k}\right) g_k + \left(1 + \frac{y_k^T H_0 y_k}{s_k^T y_k}\right) \frac{s_k s_k^T}{s_k^T y_k} g_k \\
&= H_0 g_k - \frac{s_k y_k^T H_0 g_k}{s_k^T y_k} - \frac{H_0 y_k s_k^T g_k}{s_k^T y_k} \\
&= H_0 g_k - \frac{\alpha_{k-1} d_{k-1} y_k^T H_0 g_k}{\alpha_{k-1} d_{k-1}^T y_k} \\
&= H_0 g_k - d_{k-1} \left(\frac{y_k^T H_0 g_k}{d_{k-1}^T y_k}\right) \\
&= H_0 g_k - \left(\frac{g_k^T H_0 y_k}{y_k^T d_{k-1}}\right) d_{k-1} \\
&= -d_k.
\end{aligned}$$

Therefore,

$$Q_k g_k = H_k g_k,$$

which means that we did not change anything by replacing Q_k with H_k. So far, we have established that the PCG search direction can be written in the QN-like form $d_k = -H_k g_k$.

Moreover,

$$\begin{aligned}
H_k &= H_0 g_k - \left(\frac{s_k y_k^T H_0 + H_0 y_k s_k^T}{s_k^T y_k}\right) g_k + \left(1 + \frac{y_k^T H_0 y_k}{s_k^T y_k}\right) \frac{s_k s_k^T}{s_k^T y_k} g_k \\
&= H_0 - \frac{s_k y_k^T H_0}{s_k^T y_k} - \frac{H_0 y_k s_k^T}{s_k^T y_k} + \frac{s_k (y_k^T H_0 y_k) s_k^T}{\|s_k^T y_k\|^2} + \frac{s_k s_k^T}{y_k^T s_k} \\
&= \left(I - \frac{s_k y_k^T}{s_k^T y_k}\right) H_0 \left(I - \frac{y_k s_k^T}{s_k^T y_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}.
\end{aligned}$$

Thus,

$$H_k = V_k^T H_0 V_k + \rho_k s_k s_k^T, \tag{3.10}$$

where $V_k = I - \rho_k y_k s_k^T$, and $\rho_k = \frac{1}{y_k^T s_k}$. This expression is used in the L-BFGS update later on.

4. The BFGS method

The BFGS method has proved the most effective among all quasi-Newton algorithms [9]. The essence of BFGS can be summarized by a rank-two update on the

approximate Hessian matrix B_k as follows:

$$\begin{aligned} B_{k+1} &= B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^T s_{k+1}} - \frac{B_k s_{k+1} s_{k+1}^T B_k}{s_{k+1}^T B_k s_{k+1}} \\ &= B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^T s_{k+1}} + \frac{B_k s_{k+1} s_{k+1}^T B_k}{-(s_{k+1}^T B_k) s_{k+1}} \\ &= B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^T s_{k+1}} + \frac{\alpha_k B_k d_k \alpha_k d_k^T B_k}{\alpha_k g_k^T \alpha_k d_k} \quad \text{because } B_k d_k = -g_k \\ &= B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^T s_{k+1}} + \frac{g_k g_k^T}{g_k^T d_k}. \end{aligned}$$

By applying twice a special case of the Sherman-Morrison-Woodbury formula [4],

$$(A - uv^T)^{-1} = A^{-1} + \alpha A^{-1} u v^T A^{-1}, \tag{4.1}$$

where $\alpha = (1 - v^T A^{-1} u)^{-1}$, and u and v are column vectors, it can be shown that the approximate inverse Hessian satisfies

$$\begin{aligned} H_{k+1} &= B_{k+1}^{-1} \\ &= H_k - \frac{s_{k+1}y_{k+1}^T H_k + H_k y_{k+1} s_{k+1}^T}{s_{k+1}^T y_{k+1}} + \left(1 + \frac{y_{k+1}^T H_k y_{k+1}}{s_{k+1}^T y_{k+1}}\right) \frac{s_{k+1}s_{k+1}^T}{s_{k+1}^T y_{k+1}} \\ &= V_{k+1}^T H_k V_{k+1} + \rho_{k+1} s_{k+1} s_{k+1}^T \quad [\text{by (3.10)}] \qquad (4.2) \\ &= U(H_k, y_{k+1}, s_{k+1}) \\ H_k^{\text{BFGS}} &= U(H_{k-1}, y_k, s_k). \qquad (4.3) \end{aligned}$$

By comparing the expressions for H_k^{CG} (3.9) and H_k^{BFGS} (4.3), it is clear that PCG is really a special case of BFGS in which a fixed matrix H_0 is updated at each iteration. Since BFGS stores at each iteration the approximate inverse Hessian matrix H_k^{BFGS} which is usually dense while PCG does not, PCG can thus be interpreted as the memory-less version of BFGS.

For implementation purpose, we follow the standard algorithm that makes use of the approximate Hessian matrix B_k as it allows us to update B_k by two rank-one updates to its Cholesky factor R_k [10]. Hence we just need to store R_k instead of B_k, saving half amount of memory.

Algorithm 4.1. BFGS (on the quadratic function in (1.2))
Initialization x_0, B_0, $g_0 = Ax_0 - b$, $k = 0$

while not converged

$$\begin{aligned}
\text{Solve } B_k d_k &= -g_k \text{ for } d \\
\alpha_k &= -\frac{g_k^T d_k}{d_k^T A d_k} \\
s_{k+1} &= \alpha_k d_k \\
x_{k+1} &= x_k + s_{k+1} \\
y_{k+1} &= A s_{k+1} \\
g_{k+1} &= g_k + y_{k+1} \\
B_{k+1} &= B_k + \frac{y_{k+1} y_{k+1}^T}{y_{k+1}^T s_{k+1}} + \frac{g_k g_k^T}{g_k^T d_k} \\
k &= k+1
\end{aligned}$$

end

As mentioned before, we use R_k instead of B_k in practice. The rank-two update to B_k is accomplished through a rank-one update of $\left(\frac{y_{k+1}}{\sqrt{y_{k+1}^T s_{k+1}}}\right)$ to R_k, followed by a rank-one downdate of $\left(\frac{g_k}{\sqrt{\|g_k^T d_k\|}}\right)$ to $R_k^{'}$, the updated R_k, i.e.

$$B_{k+1} = B_k + \frac{y_{k+1} y_{k+1}^T}{y_{k+1}^T s_{k+1}} + \frac{g_k g_k^T}{g_k^T d_k}$$

$$\begin{aligned}
R_{k+1}^T R_{k+1} = R_k^T R_k &+ \left(\frac{y_{k+1}}{\sqrt{y_{k+1}^T s_{k+1}}}\right)\left(\frac{y_{k+1}}{\sqrt{y_{k+1}^T s_{k+1}}}\right)^T \\
&- \left(\frac{g_k}{\sqrt{\|g_k^T d_k\|}}\right)\left(\frac{g_k}{\sqrt{\|g_k^T d_k\|}}\right)^T .
\end{aligned} \tag{4.4}$$

The minus sign in (4.4) is because $g_k^T d_k < 0$ as shown below:

$$\begin{aligned}
g_k &= -B_k d_k \\
g_k^T &= -d_k^T B_k \\
g_k^T d_k &= -d_k^T B_k d_k .
\end{aligned}$$

Since B_k is symmetric positive definite, $-d_k^T B_k d_k < 0$.

5. The relationship between CG and BFGS on quadratic functions

In the previous sections, we established the general relationship that PCG is a special case of BFGS. In fact, we can further strengthen this relationship in the context of linear systems, i.e. on quadratic functions.

Lemma 5.1. *When the PCG and BFGS algorithms are applied to the quadratic function* (1.2) *using the same starting point x_0 and initial symmetric positive definite matrix H_0, then*

$$d_j^{\text{CG}} = d_j^{\text{BFGS}}, \quad j = 1, 2, \cdots, n. \tag{5.1}$$

The detailed proof was given by Nazareth [7]. While we will not repeat the major part of his proof here, we would like to provide the proof by induction that

$$H_j^{\text{BFGS}} g_k = H_0 g_k, \quad 0 \le j < k \le n \tag{5.2}$$

for which he omitted the details.

Proof. When $j = 0$,

$$H_0^{\text{BFGS}} g_k = H_0 g_k.$$

Now, assume that

$$H_{j-1}^{\text{BFGS}} g_k = H_0 g_k,$$

so that

$$H_j^{\text{BFGS}} g_k = H_{j-1} g_k - \frac{s_j y_j^T H_{j-1} + H_{j-1} y_j s_j^T}{s_j^T y_j} g_k + \left(1 + \frac{y_{k+1}^T H_k y_{k+1}}{s_{k+1}^T y_{k+1}}\right) \frac{s_{k+1} s_{k+1}^T}{s_{k+1}^T y_{k+1}} g_k. \tag{5.3}$$

In addition,

$$\begin{aligned} y_j^T H_{j-1}^{\text{BFGS}} g_k &= (g_j - g_{j-1})^T H_{j-1}^{\text{BFGS}} g_k \\ &= g_j H_{j-1} g_k - g_{j-1} H_{j-1} g_k \\ &= g_j H_0 g_k - g_{j-1} H_0 g_k \\ &= 0 \quad \text{[by (3.5)]}. \end{aligned} \tag{5.4}$$

Applying (5.4) and (3.4) to (5.3), we get

$$H_j^{\text{BFGS}} g_k = H_{j-1}^{\text{BFGS}} g_k = H_0 g_k.$$

□

This equivalence relation is further extended by Buckley in [1].

6. Limited-memory methods based on PCG and BFGS

Before we talk about the limited-memory methods, it is probably sensible to consider the memory requirements for PCG and BFGS. For PCG, we need to store only several n-vectors, hence the storage requirement is $O(n)$. For BFGS, we have to save H_k or the Cholesky factor R_k of B_k, which accounts for an $O(n^2)$ memory requirement.

The limited-memory methods are designed for the situation where the amount of available storage is not enough for BFGS, but exceeds the requirement for PCG. The motivation is that by utilizing more memory than PCG, we expect to achieve performance that is superior to PCG, though inferior to BFGS.

Both of the limited-memory methods we consider here have an input parameter m, which specifies the number of stored vector pairs for the L-BFGS method and the number of BFGS iterations for each invocation of the Quasi-Newton phase in the VSCG algorithm.

6.1. **The L-BFGS method.** The L-BFGS method was developed by Nocedal [8] based on the fact that we can construct H_k^{BFGS} from H_0 by applying k times the updates with the vector pairs $(y_j, s_j), j = 1, \cdots, k$. We can easily deduce this fact from the recursive relation for in H_k^{BFGS} (4.3).

Moreover, we need not compute H_k^{BFGS} explicitly, but we instead compute $H_k g_k$. This can be accomplished by the two-loop recursion described by Nocedal (see [8], and [9, Algorithm 9.1]).

In L-BFGS, we store the m most recent pairs of (y_i, s_i). When the storage limit is reached, we discard the oldest vector pair before saving the newest one, i.e., we keep $((y_k, s_k), (y_{k-1}, s_{k-1}), \cdots, (y_{k-m+1}, s_{k-m+1}))$. For computing $H_k g_k$, we use the two-loop recursion. The two-loop recursion is based on the expression (4.2). We can repeatedly apply (4.2) in the L-BFGS context and we have

$$\begin{aligned} H_k^{\text{BFGS}} = & (V_k^T \cdots V_{k-m+1}^T) H_k^0 (V_{k-m+1} \cdots V_k) \\ & + \rho_{k-m+1} (V_k^T \cdots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_k) \\ & + \rho_{k-m+2} (V_k^T \cdots V_{k-m+3}^T) s_{k-m+2} s_{k-m+2}^T (V_{k-m+3} \cdots V_k) \\ & \cdots \\ & + \rho_k s_k s_k^T, \end{aligned} \tag{6.1}$$

which forms the basis of the two-loop recursion. The formal L-BFGS algorithm that we have implemented can be found in [9, Algorithm 9.2].

PCG can be understood as a special case of BFGS, hence it is not surprising that PCG can be interpreted as a special case of L-BFGS as well. Indeed, from (3.9), we can reconstruct H_k^{CG} from H_0 by an update with (y_k, s_k). As a result, we can interpret PCG as L-BFGS with $m = 1$ and $H_k^0 = H_0$ (i.e. the initial matrix of each Quasi-Newton phase is set to H_0). By similar reasoning, we see that BFGS is, in fact, L-BFGS with $m = \infty$ and $H_k^0 = H_0$.

6.2. **The VSCG method.** With the relationship between PCG and BFGS firmly established, we are now ready to consider the VSCG method, which was proposed by Buckley and LeNir [2]. The VSCG algorithm combines cycles of BFGS with CG iterations in an intelligent manner. The basic reasoning is that the symmetric positive definite H_k from BFGS approximates the inverse of the Hessian, which is just A^{-1} if $f(x)$ is the quadratic function in (1.2) So it is reasonable to apply H_m, which is generated by m iterations of BFGS, as the preconditioner to CG so as to improve the performance of CG. When the BFGS iterations are invoked is determined by a CG restart criterion. Here, we present the basic algorithm for VSCG.

Algorithm 6.1. VSCG
Initialization $x_0, g_0 = \nabla f(x_0), d_0 = -H_0 g_0$
while not converged
BFGS-part: Choose/reset initial matrix H_k^0 to be s.p.d. H_0.
for $i = 1, \cdots, m-1, m$

$$\begin{aligned} H_i &= U(H_{i-1}, s_i, y_i) \\ d_i &= -H_i g_i \\ x_{i+1} &= x_i + \alpha_i d_i \end{aligned}$$

end (for)

CG-part: Continue from x_{m+1}, use H_m as the preconditioner
for $i = m+1, m+2, \cdots$ until a restart is necessary

$$\begin{aligned} H_i &= U(H_m, s_i, y_i) \\ d_i &= -H_i g_i \\ x_{i+1} &= x_i + \alpha_i d_i \end{aligned}$$

end (for)
end (while)

In VSCG, although m denotes the number of iterations allowed for each invocation of the BFGS-part, it in fact carries the same meaning as that in L-BFGS. Since the preconditioner H_m is constructed from the m (s_i, y_i) pairs, we have to store them for the CG-part. In other words, m is the number of vector pairs saved.

It turns out that VSCG is related to CG and BFGS in the same way as L-BFGS. Intuitively, this is not surprising since the parameter m in both algorithms represents the same thing. When $m = 1$, VSCG reduces to Beale's recurrence with padding [2]. (Recall that with exact line-search, padding does not affect the search directions generated.) Now, Beale's recurrence is in turn equivalent to PCG on quadratic functions with exact line-search [2]. When $m = \infty$, obviously the CG-part of VSCG will never be executed, hence VSCG is just BFGS.

On quadratic functions, as we have discussed in the previous section, PCG and BFGS are equivalent. Therefore, it follows that L-BFGS and VSCG are also equivalent to PCG and BFGS on quadratics. We will make that observation in the next section.

The storage requirements for L-BFGS and VSCG are both $O(mn)$ since both algorithms require storing m n-vector pairs.

6.3. **Implementation issues.** One issue regarding the VSCG implementation that is worth discussing is the way we store and discard the (y_i, s_i) vector pairs. The original approach adopted by Buckley and LeNir is to discard all m vector pairs at the first step of each run of the BFGS part, i.e. resetting the preconditioner to H_0, and then start afresh. In the numerical results that we present in the next section, we will use VSCG2 to represent the implementation with this approach.

We have also tried to adopt the L-BFGS strategy by discarding only the oldest vector pair (and add in the newest one) at each restart of CG. Test experience showed that this approach is almost the same as the original one. Intuitively, that makes sense because after m BFGS iterations, all the m old vector pair would have been discarded, and therefore their effect is the preconditioner H_m which is to be applied to the CG-part. We will not show the numerical results for this approach.

In a personal communication, Friedlander suggested to keep the diagonal of H_m before discarding all the m vector pairs at a new restart and then proceed as the original approach with H_0 being a diagonal matrix whose diagonal is that of the previous H_m. The rationale behind is that the diagonal of H_m contains the most information of the matrix. We represent the implementation with this strategy as VSCG4.

VSCG requires a restart criterion for the CG-part of the algorithm. Buckley and LeNir [2] proposed to use

$$\tau = \frac{g_i^T H_m g_{i-1}}{g_{i-1}^T H_m g_{i-1}}. \tag{6.2}$$

However, τ is always 0 on quadratic functions by (3.5), hence (6.2) is not applicable to linear problems. In our implementation, we invoke the BFGS-part whenever the number of iterations is a multiple of n.

In our VSCG implementation, we also use the L-BFGS two-loop recursion to compute $H_k g_k$ for simplicity, although Buckley and LeNir [2] described a slightly different way to do that.

Since the initial matrix H_0 is usually diagonal, we simply use a vector to represent H_0 so that $H_0 q$ is just the result of an element-wise multiplication of H_0 and q. In our implementations of all the algorithms under consideration, H_0 is always set to the identity matrix.

7. Numerical results

7.1. **Explanation of set-up.** Our implementations of the algorithms are in MATLAB. The source code is available upon request. The algorithms are tested over 30 test matrices from *Matrix Market* [6], and their performances, i.e. number of iterations for convergence, are reflected in the table and graphs. We classify the test matrices by their condition numbers according to the following table:

category	condition number
extremely ill-conditioned	$\geq 10^{10}$
highly ill-conditioned	$10^7 - 10^9$
moderately ill-conditioned	$10^4 - 10^6$
well-conditioned	$\leq 10^3$

The information for the test matrices can be found in Appendix A or in the source file loadTestMatrices.m.

The table of results is organized as follows: The first column contains the indices of the test matrices. The second and last columns contain the results for PCG and BFGS respectively. The remaining columns show the results for the limited-memory methods with different values of m.

Each test matrix occupies three rows, which corresponds to L-BFGS, VSCG2, and VSCG4 in that order. The results for PCG and BFGS are put in each row to server as benchmarks.

The values of m are not set to specific numbers. Instead, percentages are used to reflect the amount of storage allowed relative to the total size of the matrix. When m is at 100%, it is set to the value $l = min(n, 200)$. Similarly at 10%, for example, $m = l/10$.

7.2. **Table and graphs.**

7.3. **Observations.** In the previous sections, we have established that PCG, BFGS, L-BFGS, and VSCG are all equivalent on quadratic functions with exact line-search. We can see from the numerical results that this is true provided that the matrix in problem is well-conditioned (e.g. matrices 2,7 in Figure 1). So, PCG should be the first choice for solving well-conditioned linear systems.

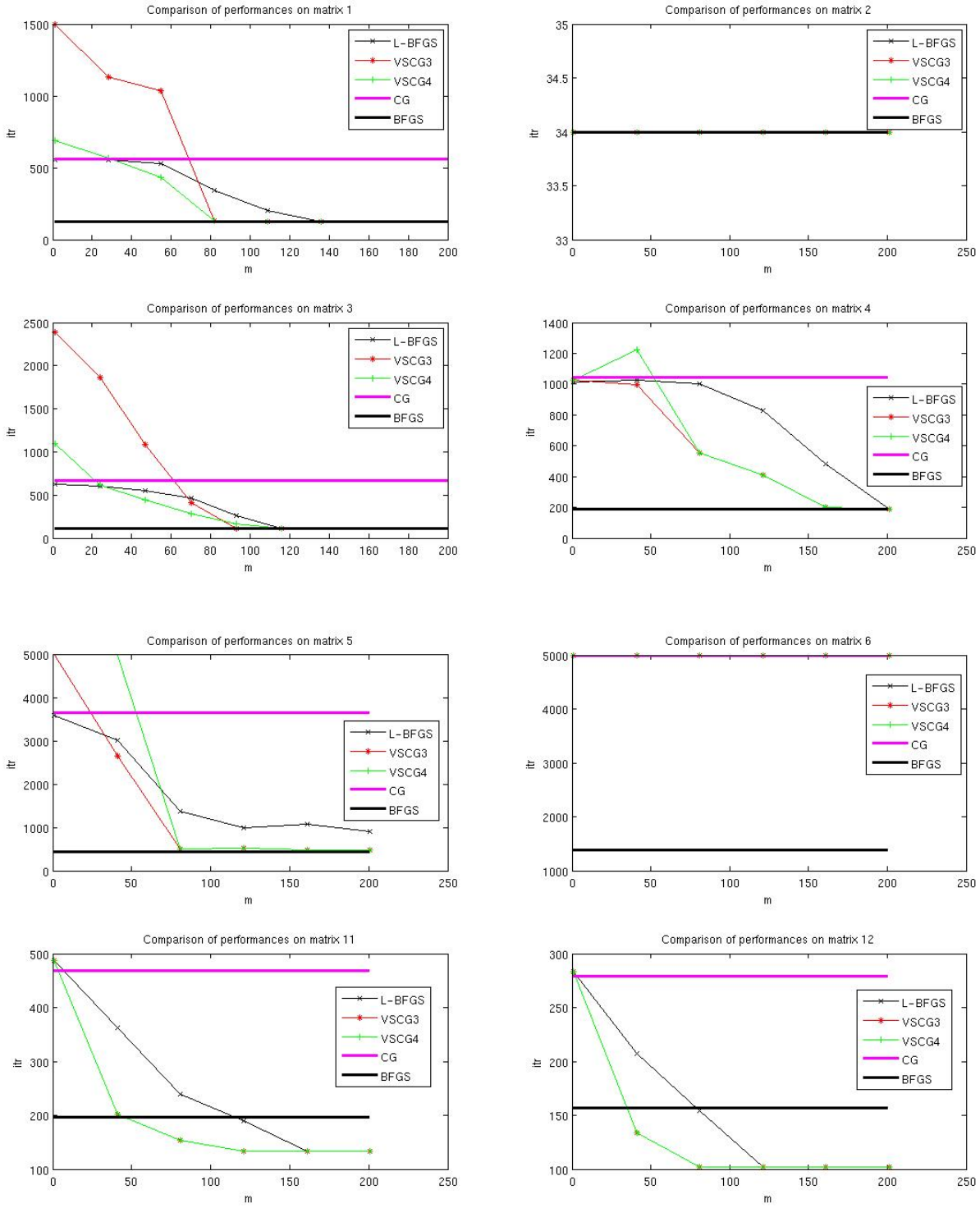

FIGURE 1. Performance graph for matrices 1-6, 11, and 12.

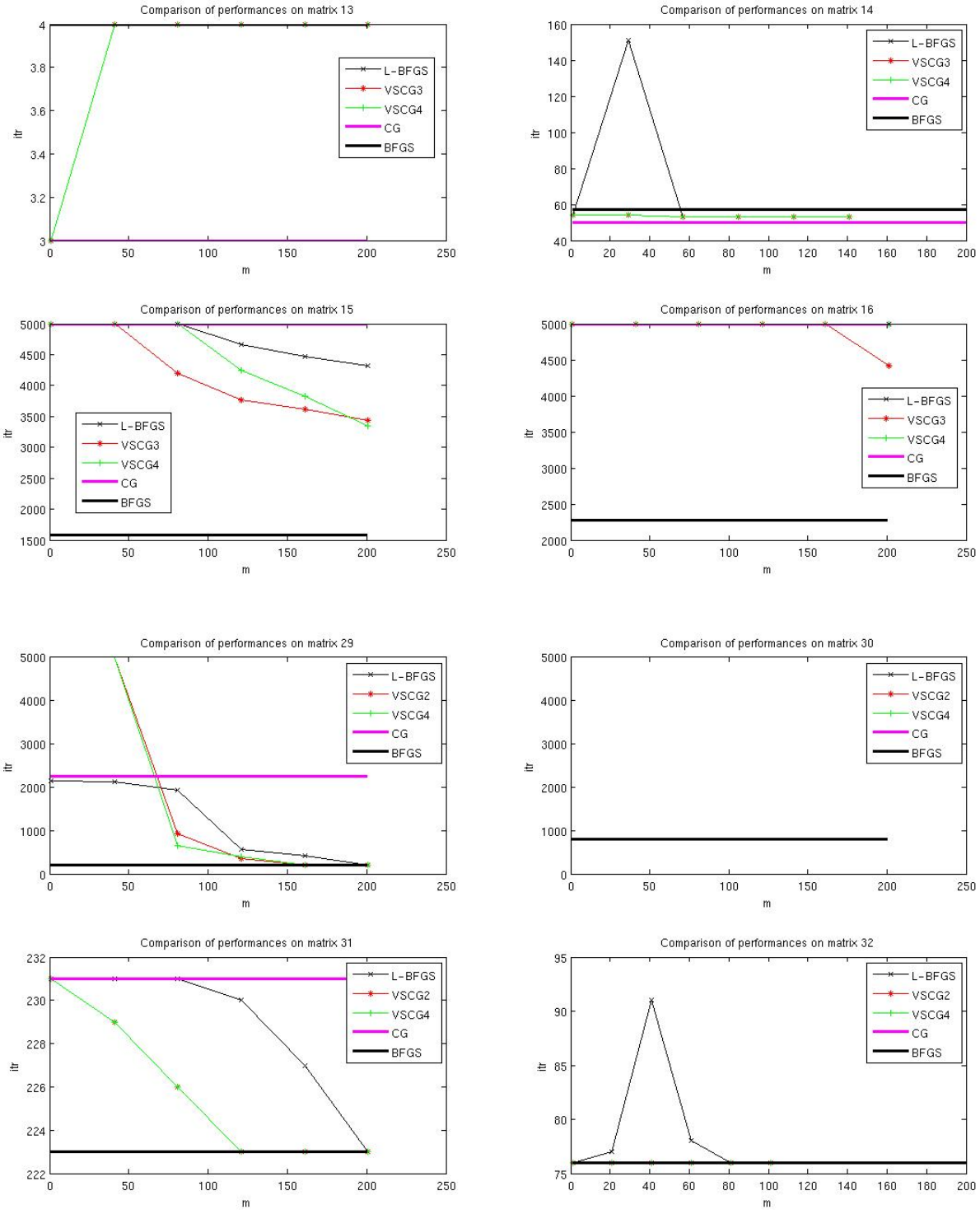

FIGURE 2. Performance graph for matrices 13-16, and 29-32.

TABLE 1. Numerical results

Mat	PCG	m=1	m=20%	m=40%	m=60%	m=80%	m=100%	BFGS	Condition Num.	Size
1	564	555	554	526	341	202	128	128	5.60E+06	132 x 132, 1890 entries
1	564	1498	1132	1034	132	128	128	128		
1	564	686	566	430	132	128	128	128		
2	34	34	34	34	34	34	34	34	3.80E+02	900 x 900, 4322 entries
2	34	34	34	34	34	34	34	34		
2	34	34	34	34	34	34	34	34		
3	673	629	606	555	465	264	109	109	9.50E+06	112 x 112, 376 entries
3	673	2388	1862	1081	411	113	109	109		
3	673	1093	609	442	278	161	109	109		
4	1044	1013	1026	1003	826	484	189	189	8.00E+06	675 x 675, 1965 entries
4	1044	1027	994	555	407	198	189	189		
4	1044	1027	1222	555	407	198	189	189		
5	3664	3594	3026	1377	985	1075	915	428	4.10E+09	729 x 729, 2673 entries
5	3664	5000	2663	492	513	487	485	428		
5	3664	5000	5000	492	513	487	485	428		
6	5000	5000	5000	5000	5000	5000	5000	1393	1.30E+10	1806 x 1806, 32630 entries
6	5000	5000	5000	5000	5000	5000	5000	1393		
6	5000	5000	5000	5000	5000	5000	5000	1393		
11	469	488	362	239	189	134	134	196	2.30E+05	817 x 817, 817 entries
11	469	488	201	153	134	134	134	196		
11	469	488	201	153	134	134	134	196		
12	279	283	207	154	102	102	102	157	2.60E+05	485 x 485, 485 entries
12	279	283	134	102	102	102	102	157		
12	279	283	134	102	102	102	102	157		
13	3	3	4	4	4	4	4	4	24	3600 x 3600, 3600 entries
13	3	3	4	4	4	4	4	4		
13	3	3	4	4	4	4	4	4		
14	50	54	151	53	53	53	53	57	9.40E+02	138 x 138, 138 entries
14	50	54	54	53	53	53	53	57		
14	50	54	54	53	53	53	53	57		
15	5000	5000	5000	5000	4662	4473	4317	1584	9.50E+08	3134 x 3134, 3134 entries
15	5000	5000	5000	4205	3770	3612	3430	1584		
15	5000	5000	5000	5000	4247	3831	3347	1584		
16	5000	5000	5000	5000	5000	5000	5000	2276	1.80E+13	3562 x 3562, 3562 entries
16	5000	5000	5000	5000	5000	5000	4412	2276		
16	5000	5000	5000	5000	5000	5000	5000	2276		
18	909	901	900	870	825	772	733	527	7.70E+04	1224 x 1224, 28675 entries
18	909	901	825	720	642	603	576	527		
18	909	901	825	720	642	603	576	527		
19	2085	2094	2706	2899	2767	2584	2484	1147	2.60E+05	1922 x 1922, 1922 entries
19	2085	2133	1765	1598	1443	1320	1179	1147		
19	2085	2133	1765	1598	1443	1320	1179	1147		
20	5000	5000	5000	5000	5000	5000	5000	2385	3.21E+06	5489 x 5489, 143300 entries
20	5000	5000	5000	5000	5000	5000	5000	2385		
20	5000	5000	5000	5000	5000	5000	5000	2385		
29	2250	2154	2130	1942	560	415	198	198	2.50E+07	237 x 237, 627 entries
29	2250	5000	5000	937	359	198	198	198		
29	2250	5000	5000	659	386	198	198	198		
30	5000	5000	5000	5000	5000	5000	5000	798	6.30E+09	957 x 957, 2547 entries
30	5000	5000	5000	5000	5000	5000	5000	798		
30	5000	5000	5000	5000	5000	5000	5000	798		
31	231	231	231	231	230	227	223	223	7.30E+04	960 x 960, 8402 entries
31	231	231	229	226	223	223	223	223		
31	231	231	229	226	223	223	223	223		
32	76	76	77	91	78	76	76	76	2.70E+03	100 x 100, 347 entries
32	76	76	76	76	76	76	76	76		
32	76	76	76	76	76	76	76	76		
33	428	429	426	422	416	412	403	372	2.90E+04	468 x 468, 2820 entries
33	428	429	409	401	392	382	379	372		
33	428	429	409	401	392	382	379	372		
34	1044	1013	1026	1003	826	484	189	189	8.00E+06	675 x 675, 1965 entries
34	1044	1027	994	555	407	198	189	189		
34	1044	1027	1222	555	407	198	189	189		
35	3664	3594	3026	1377	985	1075	915	428	4.10E+09	729 x 729, 2673 entries
35	3664	5000	2663	492	513	487	485	428		
35	3664	5000	5000	492	513	487	485	428		

On ill-conditioned matrices, however, there is significant difference in the performance of PCG and BFGS. The gap widens as the matrix becomes more ill-conditioned and larger in size. The performance of the limited-memory methods

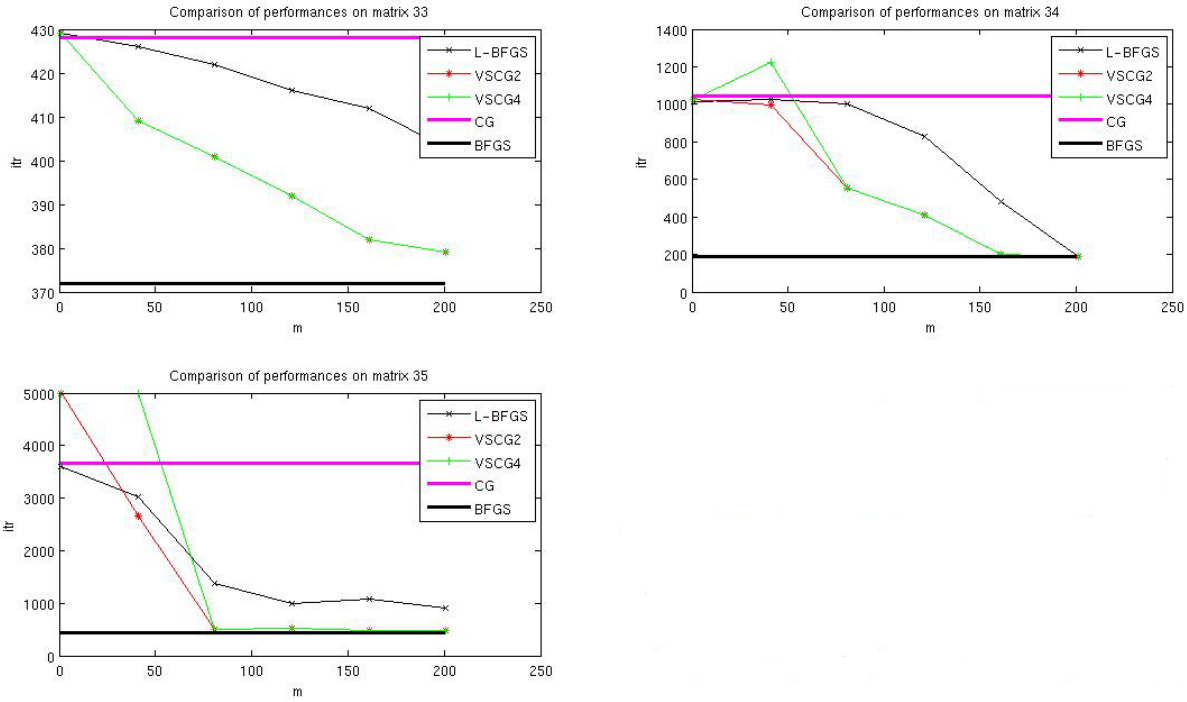

FIGURE 3. Performance graph for matrices 33-35.

resembles that in the nonlinear case; as the value of m increases, the number of iterations decreases until it reaches the level of BFGS.

The number of iterations for L-BFGS with $m = 1$ is identical or close to PCG as expected, but in many test cases (e.g. matrices 3,5 in Figure 1, and 29 in Figure 2), that is not true for VSCG2 and VSCG4. Here, we attempt to provide a possible explanation. As we mentioned in the previous section, the equivalence of VSCG with $m = 1$ and PCG on quadratic functions relies on the equivalence of Beale's recurrence and PCG. Buckley and LeNir [2] established the second equivalence by showing that on quadratics,

$$g_i^T H_j y_i = g_i^T H_0 y_i, \quad j > 0, i > j \tag{7.1}$$

which reduces to

$$H_i g_i = H_0 g_i, \quad i > j \tag{7.2}$$

Here, $H_j g_i$ comes from PCG, and $H_0 g_i$ comes from Beale's recurrence. We recall that the equivalence of PCG and BFGS on quadratics is based on the same result (5.2), but we just observed that PCG requires much more iterations than BFGS does to reach convergence on ill-conditioned matrices. That should explain why VSCG with $m = 1$ performs not as well as PCG. As such, if the linear system is highly ill-conditioned and the amount of storage is very limited, we recommend using L-BFGS to solve the problems.

It is worthwhile to note that when m reaches the 40% level, VSCG4 almost never perform worse than L-BFGS, and in some cases (e.g. matrices 4,5 in Figure 1, and 34 in Figure 2), the superiority is significant. The performances of VSCG2 and VSCG4 are generally identical, but VSCG4 shows superiority in several cases (e.g. matrix 1,3 in Figure 1).

Our final observation is that BFGS shows the most robustness on very large and extremely ill-conditioned matrices (e.g. matrix 20,30 in Figure 2). In those cases, all the other three methods fail to converge within 5000 iterations, but BFGS succeeds.

8. Conclusion

We have described PCG, BFGS, and the limited-memory methods in the context of linear systems, and we have also streamlined the relationships between each of the algorithms. The numerical results that we have presented demonstrate that CG is the best choice for well-conditioned problems because of its low memory requirement. On large, highly ill-conditioned problems, BFGS may be our only choice. Nevertheless, on moderately ill-conditioned problems with moderate sizes, we may be able to take advantage of the limited-memory algorithms, depending on the amount of storage we have available.

References

[1] Buckley, A. "Extending the relationship between the conjugate gradient and BFGS algorithms", *Mathematical Programming* 15 (1978) 343-348.
[2] Buckley, A. AND LeNir, A. "QN-like variable storage conjugate gradients", *Mathematical Programming* 27 (1983) 155-175.
[3] Fletcher, R. AND Reeves, C.M. "Function minimization by conjugate gradients", *Computer Journal* 7 (1964) 149-154.
[4] Hager, W.W. "Updating the inverse of a matrix", *SIAM Review* 31 (1989) 221-239.
[5] Hestenes, M.R. AND Stiefel, E. "Methods of conjugate gradients for solving linear systems", *Journal of Research of the National Bureau of Standards* 49 (1952) 409-436.
[6] Matrix Market, http://math.nist.gov/MatrixMarket/.
[7] Nazareth, L. "A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms", *SIAM Journal on Numerical Analysis* 16 (1979) 794-800.
[8] Nocedal, J. "Updating Quasi-Newton matrices with limited storage", *Mathematics of Computation* 35 (1980) 773-782.
[9] Nocedal, J. AND Wright, S. "Numerical Optimization", Springer-Verlag, New York, NY, 1999.
[10] Schoenberg, R. "Optimization with the Quasi-Newton method", Aptech Systems, Maple Valley, WA, 2001.
[11] Shanno, D.F. "Conjugate gradient methods with inexact searches", *Mathematics of Operations Research* 3 (1978) 244-256.

Department of Mathematics,, University of British Columbia, BC Canada
E-mail address: tonyqin@interchange.ubc.ca

Sponsor: Michael P. Friedlander, Department of Mathematics,, University of British Columbia, BC Canada
E-mail address: mpf@cs.ubc.ca