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

\[(1.1) \quad Ax = b,\]

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

\[(1.2) \quad \min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Ax - b^T x.\]

In this thesis, whenever the quasi-Newton family is concerned, 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
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 $g_k = \nabla f(x_k)$, $y_{k+1} = g_{k+1} - g_k$, $s_{k+1} = x_{k+1} - x_k = \alpha_k d_k$, where $\alpha_k$ is the step length determined by exact line-search as

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

(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.

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

(2.2)

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

(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** $x_0$, $g_0 = A x_0 - b$, $d_0 = -g_0$, $k = 0$

**while** not converged

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

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

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

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

$$d_{k+1} = -g_{k+1} + \beta_{k+1} d_k$$

$$k = k + 1$$

**end**

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:

\[
\begin{align*}
g_k^T g_i &= 0 \quad \text{for } i = 0, \cdots, k - 1, \\
d_k^T A d_i &= 0 \quad \text{for } i = 0, \cdots, k - 1.
\end{align*}
\]

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

\[
g_k^T d_i = 0 \quad \text{for } i = 0, \cdots, k - 1.
\]

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

\[
g_k^T s_{i+1} = 0.
\]

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

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

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])

\begin{itemize}
  \item Initialization \( x_0, \) preconditioner \( H_0, \) \( g_0 = Ax_0 - b, \) \( d_0 = -H_0 g_0, \) \( k = 0 \)
  \item while not converged
    \begin{align*}
    \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{align*}
  \end{itemize}

We can see that the only differences between PCG and CG are the initial search direction \( d_0 \) and the “correction coefficient” \( \beta_{k+1} \). 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 \( \alpha_k \). 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 $\beta_{k+1}$. The Hestenes-Stiefel form (HS-CG) \cite{hestenes1952methods} with a preconditioner $H_0$ defines
\begin{equation}
\beta_{k+1}^{\text{HS}} = \frac{g^T_{k+1} H_0 y_{k+1}}{y^T_{k+1} d_{k}}.
\end{equation}
Here, we show that HS-CG is equivalent to FR-CG on quadratic functions. First, recall that
\begin{equation}
d_{k+1} = -H_0 g_{k+1} + \beta_{k+1}^{\text{FR}} d_k,
\end{equation}
for each $k$. Hence,
\begin{align*}
g^T_k d_k &= -g^T_k H_0 g_k + \beta_{k+1}^{\text{FR}} g^T_k d_{k-1} \\
&= -g^T_k H_0 g_k.
\end{align*}
Now,
\begin{align*}
\beta_{k+1}^{\text{HS}} &= \frac{g^T_{k+1} H_0 (g_{k+1} - g_k)}{(g_{k+1} - g_k)^T d_k} \\
&= \frac{g^T_{k+1} H_0 g_{k+1}}{g^T_k d_k} \\
&= \frac{g^T_{k+1} H_0 g_{k+1}}{g^T_k H_0 g_k} \\
&= \beta_{k+1}^{\text{FR}},
\end{align*}
which by Algorithm 3.2, we have
\begin{equation}
\beta_{k+1}^{\text{FR}} = \beta_{k+1}.
\end{equation}
Let us consider HS-CG. If
\begin{equation}
Q_k = H_0 - \frac{s_k y^T_k H_0}{s_k^T y_k},
\end{equation}
then
\begin{align*}
d_{k+1} &= -H_0 g_{k+1} + \left( \frac{g^T_{k+1} H_0 y_{k+1}}{y^T_{k+1} d_{k}} \right) d_k \\
&= -H_0 g_{k+1} + \left( \frac{g^T_{k+1} H_0 y_{k+1}}{y^T_{k+1} s_{k+1}} \right) s_{k+1} \\
&= -(H_0 - \frac{s_{k+1} y^T_{k+1} H_0}{s_{k+1}^T y_{k+1}}) g_{k+1} \\
&= -Q_{k+1} g_{k+1}.
\end{align*}
Note that $Q_k$ is not symmetric positive definite in most cases. Shanno \cite{shanno1981some} suggested that we can augment/pad $Q_k$ to get
\begin{equation}
H_k = H_0 - \frac{s_k y^T_k H_0 + H_0 y s^T_k}{s_k^T y_k} + \left( 1 + \frac{y^T_k H_0 y_k}{s_k^T y_k} \right) \frac{s_k y^T_k}{s_k^T y_k}.
\end{equation}
so that $H_k$ is symmetric positive definite. By definition of the function $U$ in (2.2) and (2.3),
\begin{equation}
H_k^{\text{PCG}} = U(H_0, y_k, s_k).
\end{equation}
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

\[
H_k g_k = H_0 g_k - \left( \frac{s_k y_k^T H_0 + H_0 y_k s_k^T}{s_k^T y_k} \right) g_k + \left( 1 + \frac{y_k^T H_0 y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k} g_k
\]

\[
= H_0 g_k - \frac{s_k y_k^T H_0 g_k}{s_k^T y_k} - \frac{H_0 y_k s_k^T g_k}{s_k^T y_k}
\]

\[
= H_0 g_k - \frac{\alpha_{k-1} d_{k-1} y_k^T H_0 y_k}{\alpha_{k-1} d_{k-1}^T y_k}
\]

\[
= H_0 g_k - d_{k-1} \left( \frac{y_k^T H_0 y_k}{d_{k-1}^T y_k} \right)
\]

\[
= H_0 g_k - \left( \frac{y_k^T H_0 y_k}{y_k^T d_{k-1}} \right) d_{k-1}
\]

\[
= -d_k.
\]

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,

\[
H_k = H_0 g_k - \left( \frac{s_k y_k^T H_0 + H_0 y_k s_k^T}{s_k^T y_k} \right) g_k + \left( 1 + \frac{y_k^T H_0 y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k} g_k
\]

\[
= H_0 - \frac{s_k y_k^T H_0}{s_k^T y_k} - \frac{H_0 y_k s_k^T}{s_k^T y_k} + \frac{s_k (y_k^T H_0 y_k)}{s_k^T y_k} - \frac{s_k s_k^T}{s_k^T y_k}
\]

\[
= \left( I - \frac{s_k y_k^T}{s_k^T y_k} \right) H_0 \left( I - \frac{y_k s_k^T}{s_k^T y_k} \right) + \frac{s_k s_k^T}{s_k^T y_k}.
\]

Thus,

\[
H_k = V_k^T H_0 V_k + \rho_k s_k s_k^T,
\]

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:

$$B_{k+1} = B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^Ts_{k+1}} - \frac{B_k s_{k+1} s_{k+1}^T B_k}{s_{k+1}^T B_k s_{k+1}}$$

$$= B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^Ts_{k+1}} + \frac{B_k s_{k+1} s_{k+1}^T B_k}{s_{k+1}^T B_k s_{k+1}}$$

$$= B_k + \frac{y_{k+1}y_{k+1}^T}{y_{k+1}^Ts_{k+1}} + \frac{\alpha_k B_k d_k \alpha_k d_k^T B_k}{\alpha_k g_k^T \alpha_k d_k}$$

because $B_k d_k = -g_k$

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

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

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

$$H_{k+1} = B_{k+1}^{-1}$$

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

$$= V_{k+1} H_k V_{k+1} + \rho_{k+1} s_{k+1} s_{k+1}^T$$

by (3.10)

$$= U(H_k, y_{k+1}, s_{k+1})$$

$$= U(H_{k-1}, y_{k}, s_{k})$$

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

Solve \( B_k d_k = -g_k \) for \( d \)
\[
\alpha_k = -\frac{g_k^T d_k}{d_k^T A d_k}
\]

\( s_{k+1} = \alpha_k d_k \)

\( x_{k+1} = x_k + s_{k+1} \)

\( y_{k+1} = A s_{k+1} \)

\( g_{k+1} = g_k + y_{k+1} \)

\( B_{k+1} = B_k + \frac{y_{k+1} y_{k+1}^T}{y_{k+1}^T y_{k+1}} s_{k+1} + \frac{g_{k+1} g_{k+1}^T}{g_{k+1}^T d_k} \)

\( k = k + 1 \)

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 \( \left( \frac{y_{k+1}}{\sqrt{y_{k+1}^T y_{k+1}}} s_{k+1} \right) \) 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.

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

\[
R_{k+1}^T R_{k+1} = R_k^T R_k + \left( \frac{y_{k+1}}{\sqrt{y_{k+1}^T y_{k+1}}} \right) \left( \frac{y_{k+1}}{\sqrt{y_{k+1}^T y_{k+1}}} \right)^T
- \left( \frac{g_k}{\sqrt{g_k^T d_k}} \right) \left( \frac{g_k}{\sqrt{g_k^T d_k}} \right)^T.
\]

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

\( g_k = -B_k d_k \)

\( g_k^T = -d_k^T B_k \)

\( g_k^T d_k = -d_k^T B_k d_k \).

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

\[
d_j^{\text{CG}} = d_j^{\text{BFGS}}, \quad j = 1, 2, \ldots, n.
\]
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
\[ H_{j}^{\text{BFGS}} g_k = H_0 g_k, \quad 0 \leq j < k \leq n \]
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
\[ (5.3) \]
\[ H_j^{\text{BFGS}} g_k = H_{j-1} g_k - \frac{s_j y_j^T H_{j-1} + H_{j-1} y_j s_j^T}{s_j y_j} g_k + \left( 1 + \frac{y_{k+1}^T H_k y_{k+1}}{s_{k+1}^T y_{k+1}} \right) \frac{s_{k+1} y_{k+1}^T g_k}{s_{k+1}^T y_{k+1}}. \]
In addition,
\[ y_j^T H_{j-1}^{\text{BFGS}} g_k = (g_j - g_{j-1})^T H_{j-1}^{\text{BFGS}} g_k \]
\[ = g_j H_{j-1} g_k - g_{j-1} H_{j-1} g_k \]
\[ = g_j H_0 g_k - g_{j-1} H_0 g_k \]
\[ = 0 \quad [\text{by (3.5)}]. \]
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 $H_k^{BFGS}$ (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

$$
H_k^{BFGS} = (V_k^T \cdots V_{k-m+1}^T) H_k^0 (V_{k-m+1} \cdots V_k)
+ \rho_{k-m+1} (V_k^T \cdots V_{k-m+2}^T s_{k-m+2} s_{k-m+1}^T V_{k-m+2} \cdots V_k)
+ \rho_{k-m+2} (V_k^T \cdots V_{k-m+3}^T s_{k-m+3} s_{k-m+2}^T V_{k-m+3} \cdots V_k)
+ \cdots
+ \rho_k s_k s_k^T,
$$

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 $i = 1, \cdots, m - 1, m$

$$
H_i = U(H_{i-1}, s_i, y_i)
$$

$$
d_i = -H_i g_i
$$

$$
x_{i+1} = x_i + \alpha_i d_i
$$
end (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

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

end (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
\[ \tau = \frac{g_i^T H_m g_i - 1}{g_i^T H_m g_i - 1}. \]
However, \( \tau \) 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:

<table>
<thead>
<tr>
<th>category</th>
<th>condition number</th>
</tr>
</thead>
<tbody>
<tr>
<td>extremely ill-conditioned</td>
<td>( \geq 10^{10} )</td>
</tr>
<tr>
<td>high ill-conditioned</td>
<td>( 10^7 - 10^9 )</td>
</tr>
<tr>
<td>moderately ill-conditioned</td>
<td>( 10^4 - 10^6 )</td>
</tr>
<tr>
<td>well-conditioned</td>
<td>( \leq 10^3 )</td>
</tr>
</tbody>
</table>

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 serve 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

<table>
<thead>
<tr>
<th>Mat</th>
<th>FCG</th>
<th>m-1</th>
<th>m-20%</th>
<th>m-40%</th>
<th>m-50%</th>
<th>m-80%</th>
<th>m-100%</th>
<th>BHGS</th>
<th>Condition</th>
<th>Num.</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>564</td>
<td>1499</td>
<td>1132</td>
<td>1034</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>154</td>
<td>126</td>
<td>342</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>564</td>
<td>686</td>
<td>566</td>
<td>430</td>
<td>132</td>
<td>128</td>
<td>128</td>
<td>154</td>
<td>126</td>
<td>342</td>
<td>128</td>
</tr>
<tr>
<td>3</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>673</td>
<td>629</td>
<td>555</td>
<td>460</td>
<td>264</td>
<td>109</td>
<td>109</td>
<td>890</td>
<td>890</td>
<td>4322</td>
<td>376</td>
</tr>
<tr>
<td>6</td>
<td>673</td>
<td>2380</td>
<td>1082</td>
<td>113</td>
<td>113</td>
<td>109</td>
<td>109</td>
<td>919</td>
<td>919</td>
<td>376</td>
<td>376</td>
</tr>
<tr>
<td>7</td>
<td>673</td>
<td>1093</td>
<td>442</td>
<td>276</td>
<td>161</td>
<td>109</td>
<td>109</td>
<td>919</td>
<td>919</td>
<td>376</td>
<td>376</td>
</tr>
<tr>
<td>8</td>
<td>1044</td>
<td>1026</td>
<td>1902</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
</tr>
<tr>
<td>9</td>
<td>1044</td>
<td>1027</td>
<td>994</td>
<td>467</td>
<td>198</td>
<td>189</td>
<td>189</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
<td>1899</td>
</tr>
<tr>
<td>10</td>
<td>3664</td>
<td>3594</td>
<td>3026</td>
<td>2977</td>
<td>985</td>
<td>1075</td>
<td>915</td>
<td>428</td>
<td>4.10E+09</td>
<td>729</td>
<td>2673</td>
</tr>
<tr>
<td>11</td>
<td>3664</td>
<td>5000</td>
<td>2603</td>
<td>492</td>
<td>513</td>
<td>487</td>
<td>485</td>
<td>482</td>
<td>4.10E+09</td>
<td>729</td>
<td>2673</td>
</tr>
<tr>
<td>12</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>13</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>14</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>15</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>16</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>17</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>18</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>19</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>20</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>21</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>22</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>23</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>24</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>25</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>26</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>27</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>28</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>29</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>30</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>31</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>32</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>33</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>34</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
<tr>
<td>35</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1306</td>
<td>1306</td>
<td>1306</td>
</tr>
</tbody>
</table>

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,

\begin{equation}
    g_i^T H_j y_i = g_i^T H_0 y_i, \quad j > 0, i > j
\end{equation}

which reduces to

\begin{equation}
    H_i g_i = H_0 g_i, \quad i > j
\end{equation}

Here, $H_j y_i$ 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