# Solving Limited-Memory BFGS Systems with Generalized Diagonal Updates

Jennifer Erway, *Member, IAENG,* and Roummel F. Marcia, *Member, IAENG*

***Abstract*—In this paper, we investigate a formula to solve systems of the form $(B_k + D)x = y$, where $B_k$ comes from a limited-memory BFGS quasi-Newton method and $D$ is a diagonal matrix with diagonal entries $d_{i,i} \geq \sigma$ for some $\sigma > 0$. These types of systems arise naturally in large-scale optimization. We show that provided a simple condition holds on $B_0$ and $\sigma$, the system $(B_k + D)x = y$ can be solved via a recursion formula that requies only vector inner products. This formula has complexity $M^2n$, where $M$ is the number of L-BFGS updates and $n \gg M$ is the dimension of $x$. We do not assume anything about the distribution of the values of the diagonal elements in $D$, and our approach is particularly for robust non-clustered values, which proves problematic for the conjugate gradient method.**

***Index Terms*—L-BFGS, quasi-Newton methods, diagonal modifications, Sherman-Morrison-Woodbury**

## I. Introduction

LIMITED-memory (L-BFGS) quasi-Newton methods are powerful tools within the field of optimization [1], [2], [3], [4] for solving problems when second derivative information is not available or computing the second derivative is too computationally expensive. In addition, L-BFGS matrices can be used to precondition iterative methods for solving large linear systems of equations. For both L-BFGS methods and preconditioning schemes, being able to solve linear systems with L-BFGS matrices are of utmost importance. While there is a well-known two-loop recursion (cf. [4], [5]) for solving linear systems with L-BFGS matrices, little is known about solving systems involving matrix modifications of L-BFGS matrices.

In this paper, we develop a recursion formula to solve linear systems with positive-definite diagonal modifications of L-BFGS matrices, i.e., systems of the form

$$(B_k + D)x = y, \tag{1}$$

where $B_k$ is the $k$-th step $n \times n$ limited-memory (L-BFGS) quasi-Newton matrix, $D$ is a positive-definite diagonal matrix with each diagonal element $d_{i,i} \geq \sigma$ for some $\sigma > 0$, and $x, y \in \mathbb{R}^n$. Systems of the form (1) arise naturally in constrained optimization (see, e.g., [6], [7], [8], [9]) and are often a block component of so-called *KKT systems* (see, e.g., [10]). In previous work [11], we developed a direct recursion formula for solving (1) in the special case when $D$ is a scalar multiple of the identity matrix, i.e., $D = \alpha I$ for some constant $\alpha > 0$, and we stated that the formula can be generalized to diagonal matrices. Here, we explicitly show how to generalize this approach to positive-definite diagonal matrices $D$. Furthermore, we compare this generalized formula to a popular direct method (the Matlab "backslash" command) and an indirect method (the conjugate gradients). Numerical results suggest that our approach offers significant computational time savings while maintaining high accuracy.

Manuscript received February 16, 2012; revised March 21, 2012. This work was supported in part by NSF grants DMS-08-11106 and DMS-0965711.

Jennifer B. Erway is with the Department of Mathematics, Wake Forest University, Winston-Salem, 27109, USA. E-mail: erwayjb@wfu.edu (see http://math.wfu.edu/∼jerway).

Roummel F. Marcia is with the Department of Applied Mathematics, University of California, Merced, 5200 N. Lake Road, Merced, 95340, USA. E-mail: rmarcia@ucmerced.edu (see http://faculty.ucmerced.edu/∼rmarcia).

## II. The limited-memory BFGS method

Let $f(x)\colon \mathbb{R}^n \to \mathbb{R}$ a continuously differentiable function. The BFGS quasi-Newton method for minimizing $f(x)$ works by minimizing a sequence of convex, quadratic models of $f(x)$. Specifically, the method generates a sequence of positive-definite matrices $\{B_k\}$ to approximate $\nabla^2 f(x)$ from a sequence of vectors $\{y_k\}$ and $\{s_k\}$ defined as

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \quad \text{and} \quad s_k = x_{x+1} - x_k,$$

respectively. (See, e.g., [12] for further background on quasi-Newton methods).

The L-BFGS quasi-Newton method can be viewed as the BFGS quasi-Newton method where only at most $M$ ($M \ll n$) recently computed updates are stored and used to update the initial matrix $B_0$. The number of updates $M$ is generally kept very small; for example, Byrd et al. [1] suggest $M \in [3, 7]$. The L-BFGS quasi-Newton approximation to $\nabla^2 f(x)$ is implicitly updated as follows:

$$B_k \;=\; B_0 \;-\; \sum_{i=0}^{k-1} \frac{1}{s_i^T B_i s_i} B_i s_i s_i^T B_i \;+\; \sum_{i=0}^{k-1} \frac{1}{y_i^T s_i} y_i y_i^T, \tag{2}$$

where $B_0 = \gamma_k^{-1} I$ with $\gamma_k > 0$ a constant. In practice, $\gamma_k$ is often taken to be $\gamma_k \triangleq s_{k-1}^T y_{k-1} / \|y_{k-1}\|_2^2$ (see, e.g., [2] or [4]). Using the Sherman-Woodbury-Morrison formula, the inverse of $B_k$ is given by

$$\begin{aligned} B_k^{-1} \;=\; & B_{k-1}^{-1} + \frac{s_{k-1}^T y_{k-1} + y_{k-1}^T B_{k-1}^{-1} y_{k-1}}{(s_{k-1}^T y_{k-1})^2} s_{k-1} s_{k-1}^T \\ & - \frac{1}{s_{k-1}^T y_{k-1}} \left( B_{k-1}^{-1} y_{k-1} s_{k-1}^T + s_{k-1} y_{k-1}^T B_{k-1}^{-1} \right). \end{aligned} \tag{3}$$

There is an alternative representation of $B_k^{-1}$ from which a two-term recursion can be established:

$$\begin{aligned} B_k^{-1} \;=\; & (V_{k-1}^T \cdots V_0^T) B_0^{-1} (V_0 \cdots V_{k-1}) \\ & + \sum_{j=1}^{k-1} (V_{k-1}^T \cdots V_j^T) s_{j-1} s_{j-1}^T (V_j \cdots V_{k-1}) \\ & + \frac{1}{y_{k-1}^T s_{k-1}} s_{k-1} s_{k-1}^T, \end{aligned} \tag{4}$$

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

where

$$V_j = I - \frac{1}{y_j^T s_j} y_j s_j^T$$

(see, e.g., [5]). For the L-BFGS method, there is an efficient way for computing products with $B_k^{-1}$. Given a vector $z$, the following algorithm (cf. [4], [5]) terminates with $q \triangleq B_k^{-1} z$:

**Algorithm 2.1:** Two-loop recursion to compute $q = B_k^{-1} z$

$q \leftarrow z$;
**for** $i = k-1, \ldots, 0$
  $\alpha_i \leftarrow (s_i^T q)/(y_i^T s_i)$;
  $q \leftarrow q - \alpha_i y_i$;
**end**
$q \leftarrow B_0^{-1} q$;
**for** $i = 0, \ldots, k-1$
  $\beta \leftarrow (y_i^T q)/(y_i^T s_i)$;
  $q \leftarrow q + (\alpha_i - \beta) s_i$:
**end**

The first loop in Algorithm 2.1 for $B_k^{-1} z$ computes and stores the products $(V_j \cdots V_{k-1})z$ for $j = 0, \ldots, k-1$. The algorithm then applies $B_0^1$. Finally, the second loop computes the remainder of (4). This algorithm can be made more efficient by pre-computing and storing $1/y_i^T s_i$ for $0 \le i \le k-1$. The two-term recursion formula requires at most $\mathcal{O}(Mn)$ multiplications and additions. There is compact matrix representation of the L-BFGS that can make computing products with the L-BFGS quasi-Newton matrix equally efficient (see, e.g., [5]).

It is important to note that computing the inverse of $B_k + D$ is not equivalent to simply replacing $B_0^{-1}$ in the two-loop recursion with $(B_0 + D)^{-1}$ (c.f. Figure 1 for an illustration). To see this, notice that replacing $B_0^{-1}$ in (4) with $(B_0 + D)^{-1}$ would apply the updates $V_i$ to the full quantity $(B_0 + D)^{-1}$ instead of only $B_0^{-1}$. The main contribution of this paper generalizes the recursion formula in [11] for computing $(B_k + D)^{-1} z$ in an efficient manner using only vector inner products.

## III. The recursion formula

Consider the problem of finding the inverse of $B_k + D$, where $B_k$ is an L-BFGS quasi-Newton matrix and $D$ is a positive-definite diagonal matrix with each diagional element $d_{i,i} \ge \sigma$ for some constant $\sigma > 0$. The Sherman-Morrison-Woodbury (SMW) formula gives the following formula for computing the inverse of $B + uv^T$, where $B$ is invertible and $uv^T$ is a rank-one update (see [13]):

$$\begin{aligned}(B + uv^T)^{-1} &= B^{-1} - \frac{1}{1 + v^T B^{-1} u} B^{-1} u v^T B^{-1} \\ &= B^{-1} - \frac{1}{1 + \operatorname{trace}(B^{-1} u v^T)} B^{-1} u v^T B^{-1}\end{aligned}$$

where $u$ and $v$ are both $n$-vectors. For simplicity, consider computing the inverse of an L-BFGS quasi-Newton matrix after only one update, i.e., the inverse of $B_1 + D$. First, we simplify (2) by defining

$$a_i = \frac{B_i s_i}{\sqrt{s_i^T B_i s_i}} \quad \text{and} \quad b_i = \frac{y_i}{\sqrt{y_i^T s_i}}. \tag{5}$$

Then, $B_1 + D$ becomes

$$B_1 + D = (\gamma_1^{-1} I + D) - a_0 a_0^T + b_0 b_0^T$$

We apply the SMW formula twice to compute $(\gamma_1^{-1} + D)^{-1}$. More specifically, let

$$\begin{aligned}C_0 &= (\gamma_1^{-1} I + D), \\ C_1 &= (\gamma_1^{-1} I + D) - a_0 a_0^T, \\ C_2 &= (\gamma_1^{-1} I + D) - a_0 a_0^T + b_0 b_0^T.\end{aligned}$$

Applying the SMW formula yields:

$$\begin{aligned}C_1^{-1} &= C_0^{-1} + \frac{1}{1 - \operatorname{trace}(C_0^{-1} a_0 a_0^T)} C_0^{-1} a_0 a_0^T C_0^{-1} \\ C_2^{-1} &= C_1^{-1} - \frac{1}{1 + \operatorname{trace}(C_1^{-1} b_0 b_0^T)} C_1^{-1} b_0 b_0^T C_1^{-1},\end{aligned}$$

giving an expression for $(B_1 + D)^{-1} = C_2^{-1}$, as desired. This is the basis for the following recursion method that appears in [14]:

**Theorem 1.** *Let $G$ and $G + H$ be nonsingular matrices and let $H$ have positive rank $M$. Let*

$$H = E_0 + E_1 + \cdots + E_{M-1},$$

*where each $E_k$ has rank one and*

$$C_{k+1} = G + E_0 + \cdots + E_k$$

*is nonsingular for $k = 0, \ldots M-1$. Then if $C_0 = G$, then for $k = 0, 1, \ldots, M-1$, the inverse of $C_{k+1}^{-1}$ can be computed recursively and is given by*

$$C_{k+1}^{-1} = C_k^{-1} - \tau_k C_k^{-1} E_k C_k^{-1},$$

*where*

$$\tau_k = \frac{1}{1 + \operatorname{trace}\left(C_k^{-1} E_k\right)}.$$

*In particular,*

$$(G + H)^{-1} = C_{M-1}^{-1} - \tau_{M-1} C_{M-1}^{-1} E_{M-1} C_{M-1}^{-1}.$$

*Proof:* See [14]. ∎

We now show that applying the above recursion method to $B_k + D$, the product $(B_k + D)^{-1} z$ can be computed recursively, assuming $\gamma_k \sigma$ is bounded away from zero. The proof follows [11] very closely.

**Theorem 2.** *Let $B_0 = \gamma_k^{-1} I$, where $\gamma_k > 0$, and let $D$ be a diagonal matrix with $d_{i,i} \ge \sigma$ for some $\sigma > 0$ and $\gamma_k \sigma > \epsilon$ for some $\epsilon > 0$. Let*

$$\begin{aligned}E_0 &= -a_0 a_0^T, \\ E_1 &= b_0 b_0^T, \\ &\vdots \\ E_{2k-2} &= -a_{k-1} a_{k-1}^T, \\ E_{2k-1} &= b_{k-1} b_{k-1}^T,\end{aligned}$$

*where $a_i$ and $b_i$ for $i = 0, 1, \ldots, k-1$, are given in (5). Finally, let $C_0 = B_0 + D$ and*

$$C_{j+1} = B_0 + D + E_0 + E_1 + \cdots + E_j,$$

*for $j = 0, 1, \ldots, 2k-1$. Then*

$$C_{j+1}^{-1} = C_j^{-1} - \tau_j C_j^{-1} E_j C_j^{-1}, \tag{6}$$

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

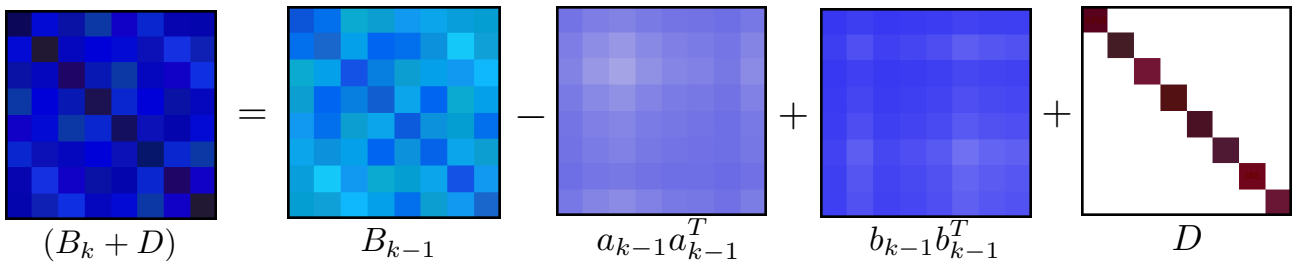


Fig. 1. Pictorial representation of the L-BFGS update method. The matrix $B_k$ is obtained from the previous matrix $B_k$ by adding two rank-one matrices, $a_{k-1}a_{k-1}^T$ and $b_{k-1}b_{k-1}^T$, where $a_{k-1}$ and $b_{k-1}$ are given in (5). The inverse of $B_k$ is similarly obtained by adding two rank-one updates to $B_{k-1}^{-1}$ (see (3)). However, the inverse of $B_k + D$ cannot be obtained by adding two rank-one updates to $(B_{k-1} + D)^{-1}$.

*where*

$$\tau_j = \frac{1}{1 + \operatorname{trace}\left(C_j^{-1}E_j\right)}. \tag{7}$$

*In particular, since* $C_{2k} = B_k + D$,

$$(B_k + D)^{-1} = C_{2k-1}^{-1} - \tau_{2k-1}C_{2k-1}^{-1}E_{2k-1}C_{2k-1}^{-1}. \tag{8}$$

*Proof:* Notice that this theorem follows from Theorem 1, provided we satisfy its assumptions. First, we let

$$G = B_0 + D \qquad \text{and} \qquad H = E_0 + E_1 + \cdots + E_{2k-1}.$$

Note that $G = B_0 + D$ is nonsingular because $B_0$ and $D$ are positive definite and that $G+H = B_k+D$ is also nonsingular since $B_k$ and $D$ are positive definite. Thus, it remains only to show that $C_j$, which is given by

$$C_j = G + \sum_{i=0}^{j-1} E_i = \left(B_0 + \sum_{i=0}^{j-1} E_i\right) + D,$$

is nonsingular for $j = 1, \ldots 2k$, for which we use induction on $j$.

For the base case $j = 1$, since

$$C_1 = C_0 - a_0a_0^T = C_0(I - C_0^{-1}a_0a_0^T),$$

the determinant of $C_1$ and $C_0$ are related as follows [15]:

$$\det(C_1) = \det(C_0)(1 - a_0^TC_0^{-1}a_0).$$

In other words, $C_1$ is invertible if $C_0$ is invertible and $a_0C_0^{-1}a_0 \neq 1$. Since $C_0$ is positive definite, it is therefore invertible. To show the latter condition, we use the definition of $a_0 = B_0s_0/\sqrt{s_0^TB_0s_0}$ together with $C_0^{-1} = (\gamma_k^{-1}I + D)^{-1}$ to obtain the following:

$$\begin{aligned}
a_0^TC_0^{-1}a_0 &= \frac{1}{s_0^TB_0s_0}s_0^TB_0^TC_0^{-1}B_0s_0 \\
&= \frac{\gamma_k^{-2}}{\gamma_k^{-1}s_0^Ts_0}s_0^T\left(\gamma_k^{-1}I + D\right)^{-1}s_0 \\
&= \frac{1}{\gamma_ks_0^Ts_0}\left(\sum_{i=1}^{n}\frac{1}{\gamma_k^{-1} + d_{i,i}}(s_0)_i^2\right) \\
&\leq \frac{1}{\gamma_ks_0^Ts_0}\left(\frac{1}{\gamma_k^{-1} + \sigma}s_0^Ts_0\right) \\
&= \frac{1}{1 + \gamma_k\sigma}.
\end{aligned} \tag{9}$$

By hypothesis, $\gamma_k\sigma > \epsilon$, which implies that $a_0^TC_0^{-1}a_0 < 1$ and that $\det(C_1) \neq 0$. Therefore, $C_1$ must be invertible.

Now we assume that $C_j$ is invertible and show that $C_{j+1}$ is invertible. If $j$ is odd, then $j+1 = 2i$ for some $i$ and $C_{j+1} = B_i + D$, which is positive definite because both $B_i$ and $D$ are positive definite. Therefore $C_{j+1}$ must be nonsingular. If $j$ is even, i.e., $j = 2i$ for some $i$, then $C_j = B_i + D$, and

$$C_{j+1} = C_j - a_ia_i^T = B_i - \frac{1}{s_i^TB_is_i}B_is_is_i^TB_i^T + D.$$

We will demonstrate that $C_{j+1}$ is nonsingular by showing that it is positive definite. Consider $z \in \mathbb{R}^n$ with $z \neq 0$. Then

$$\begin{aligned}
&z^TC_{j+1}z \\
&= z^T\left(B_i - \frac{1}{s_i^TB_is_i}B_is_is_i^TB_i^T\right)z + z^TDz \\
&= z^TB_iz - \frac{(z^TB_is_i)^2}{s_i^TB_is_i} + z^TDz \\
&= \|B_i^{1/2}z\|_2^2 - \frac{\left((B_i^{1/2}z)^T(B_i^{1/2}s_i)\right)^2}{\|B_i^{1/2}s_i\|_2^2} + z^TDz \\
&= \|B_i^{1/2}z\|_2^2 - \|B_i^{1/2}z\|_2^2\cos^2\left(\angle(B_i^{1/2}z, B_i^{1/2}s_i)\right) \\
&\qquad + z^TDz \\
&= \|B_i^{1/2}z\|_2^2\left(1 - \cos^2\left(\angle(B_i^{1/2}z, B_i^{1/2}s_i)\right)\right) + z^TDz \\
&\geq \sigma\|z\|_2^2 \\
&> 0,
\end{aligned} \tag{10}$$

since each $d_{i,i} \geq \sigma > 0$. This demonstrates that $C_{j+1}$ is positive definite, and therefore, it is nonsingular.

We have now thus satisfied all the assumptions of Theorem 1. Therefore, $(B_k + D)^{-1}$, which is equal to $C_{2k}^{-1}$, can be computed recursively and is specifically given using (6) and (8). ■

Now, we show that computing $C_{k+1}^{-1}z$ can be done efficiently as is done in [11]. We note that using (6), we have

$$\begin{aligned}
C_{k+1}^{-1}z &= C_k^{-1}z - \tau_kC_k^{-1}E_kC_k^{-1}z \\
&= \begin{cases} C_k^{-1}z + \tau_kC_k^{-1}a_{\frac{k}{2}}a_{\frac{k}{2}}^TC_k^{-1}z & \text{if } k \text{ is even} \\ C_k^{-1}z - \tau_kC_k^{-1}b_{\frac{k-1}{2}}b_{\frac{k-1}{2}}^TC_k^{-1}z & \text{if } k \text{ is odd.} \end{cases}
\end{aligned} \tag{11}$$

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

We define $p_k$ according to the following rules:

$$p_k = \begin{cases} C_k^{-1} a_{\frac{k}{2}} & \text{if } k \text{ is even} \\ C_k^{-1} b_{\frac{k-1}{2}} & \text{if } k \text{ is odd.} \end{cases} \quad (12)$$

Thus, (11) simplifies to

$$C_{k+1}^{-1} z = C_k^{-1} z + (-1)^k \tau_k (p_k^T z) p_k.$$

Applying this recursively to $C_k^{-1} z$ yields the following formula:

$$C_{k+1}^{-1} z = C_0^{-1} z + \sum_{i=0}^{k} (-1)^i \tau_i (p_i^T z) p_i, \quad (13)$$

with $C_0^{-1} z = (\gamma_k^{-1} I + D)^{-1} z$. Thus, the main computational effort in forming $C_k^{-1} z$ involves the inner product of $z$ with the vectors $p_i$ for $i = 0, 1, \ldots, k$.

What remains to be shown is how to compute $\tau_k$ in (7) and $p_k$ in (12) efficiently. The quantity $\tau_k$ is obtained by computing $\mathrm{trace}(C_k^{-1} E_k)$, which after substituting in the definition of $E_k$, is given by

$$\mathrm{trace}(C_k^{-1} E_k) = \begin{cases} -a_{\frac{k}{2}}^T C_k^{-1} a_{\frac{k}{2}} & \text{if } k \text{ is even} \\ b_{\frac{k-1}{2}}^T C_k^{-1} b_{\frac{k-1}{2}} & \text{if } k \text{ is odd.} \end{cases}$$

$$= \begin{cases} -a_{\frac{k}{2}}^T p_k & \text{if } k \text{ is even} \\ b_{\frac{k-1}{2}}^T p_k & \text{if } k \text{ is odd.} \end{cases}$$

using (12). Thus, $\tau_k$ simplifies to

$$\tau_k = \begin{cases} \dfrac{1}{1 - a_{\frac{k}{2}}^T p_k} & \text{if } k \text{ is even} \\ \dfrac{1}{1 + b_{\frac{k-1}{2}}^T p_k} & \text{if } k \text{ is odd.} \end{cases}$$

Finally, notice that we can compute $p_k$ in (12) by evaluating (13) at $z = a_{\frac{k}{2}}$ or $z = b_{\frac{k-1}{2}}$:

$$p_k = \begin{cases} C_0^{-1} a_{\frac{k}{2}} + \sum_{i=0}^{k-1} (-1)^i \tau_i (p_i^T a_{\frac{k}{2}}) p_i & \text{if } k \text{ is even} \\ C_0^{-1} b_{\frac{k-1}{2}} + \sum_{i=0}^{k-1} (-1)^i \tau_i (p_i^T b_{\frac{k-1}{2}}) p_i & \text{if } k \text{ is odd.} \end{cases}$$

Thus, by computing and storing $a_{\frac{k}{2}}^T p_k$ and $b_{\frac{k-1}{2}}^T p_k$, we can form $\tau_k$ and $p_k$ rather easily. The following pseudocode summarizes the algorithm for computing $C_{k+1}^{-1} z$:

**Algorithm 3.1:** Proposed recursion to compute $q = C_{k+1}^{-1} z$.

$q \leftarrow (\gamma_{k+1}^{-1} I + D)^{-1} z$;
**for** $j = 0, \ldots, k$
  **if** $j$ even
    $c \leftarrow a_{j/2}$;
  **else**
    $c \leftarrow b_{(j-1)/2}$;
  **end**
  $p_j \leftarrow (\gamma_{k+1}^{-1} I + D)^{-1} c$;
  **for** $i = 0, \ldots, j-1$
    $p_j \leftarrow p_j + (-1)^i v_i (p_i^T c) p_i$;
  **end**
  $v_j \leftarrow 1/(1 + (-1)^j p_j^T c)$;
  $q \leftarrow q + (-1)^i v_i (p_i^T z) p_i$;
**end**

Algorithm 3.1 requires $\mathcal{O}(k^2)$ vector inner products. Operations with $C_0$ and $C_1$ can be hard-coded since $C_0$ is a scalar-multiple of the identity. Since $k$ is kept small the extra storage requirements and computations are affordable.

TABLE I

| $n$ | Relative Residual | | |
|---|---|---|---|
| | Direct | CG | Recursion |
| 1,000 | 1.27e-15 | 8.78e-16 | 7.21e-16 |
| 2,000 | 1.91e-15 | 1.21e-15 | 1.20e-15 |
| 5,000 | 2.83e-15 | 1.61e-15 | 1.41e-15 |
| 10,000 | 3.47e-15 | 1.29e-15 | 8.98e-16 |
| 20,000 | 5.03e-15 | 2.12e-15 | 1.51e-15 |

A SAMPLE RUN COMPARING THE RELATIVE RESIDUALS OF THE SOLUTIONS USING THE MATLAB "BACKSLASH" COMMAND, CONJUGATE GRADIENT METHOD, AND THE PROPOSED RECURSION FORMULA. THE RELATIVE RESIDUALS FOR THE RECURSION FORMULA ARE USED AS THE CRITERIA FOR CONVERGENCE FOR CG.

TABLE II

| $n$ | Time (sec) | | |
|---|---|---|---|
| | Direct | CG | Recursion |
| 1,000 | 0.0249 | 0.0152 | 0.0053 |
| 2,000 | 0.2015 | 0.0188 | 0.0102 |
| 5,000 | 1.5276 | 0.0473 | 0.0158 |
| 10,000 | 8.0836 | 0.1549 | 0.0170 |
| 20,000 | 51.8179 | 0.2519 | 0.0198 |

THE COMPUTATIONAL TIMES TO ACHIEVE THE RESULTS IN TABLE I.

## IV. Numerical Experiments

We demonstrate the effectiveness of the proposed recursion formula by solving linear systems of the form (1) where the matrices are of the form (2) and of various sizes. Specifically, we let the number of updates $M = 5$ and the size of the matrix range from $n = 10^3$ up to $10^7$. We implemented the proposed method in Matlab on a Two 2.4 GHz Quad-Core Intel Xeon "Westmere" Apple Mac Pro and compared it to a direct method using the Matlab "backslash" command and the built-in conjugate-gradient (CG) method (pcg.m). Because of limitations in memory, we were only able to use the direct method for problems where $n \leq 20,000$. For our numerical experiments, we let the entries in $D$ to be evenly distributed between 1 and $n/10$. The relative residuals for the recursion formula are used as the criteria for convergence for CG. In other words, the time reported in this table reflects how long it takes for CG to achieve the same accuracy as the proposed recursion method. Often, CG terminated without converging to the desired tolerance because the method stagnated (`flag = 3` in Matlab).

**Results.** The three methods were run on numerous problems with varying problems sizes. Tables I–IV show the time and the relative residuals for each method. In Tables I and II, $n$ ranges from 1,000 to 20,000; Tables III and IV contain results for $n \geq 20,000$. We note that all methods achieve very small relative residual errors for each of the problems we considered. Besides from memory issues, the direct method suffers from significantly longer computational

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE III

| $n$ | Relative residual | |
|---|---|---|
| | CG | Recursion |
| $100,000$ | 5.68e-16 | 2.31e-16 |
| $200,000$ | 6.66e-16 | 2.34e-16 |
| $500,000$ | 5.35e-16 | 2.32e-16 |
| $1,000,000$ | 6.32e-16 | 2.29e-16 |
| $2,000,000$ | 7.24e-16 | 2.30e-16 |
| $5,000,000$ | 8.21e-16 | 2.28e-16 |
| $10,000,000$ | 7.85e-16 | 2.33e-16 |

COMPARISON BETWEEN THE PROPOSED RECURSION METHOD AND THE BUILT-IN CONJUGATE GRADIENT (CG) METHOD IN MATLAB. THE RELATIVE RESIDUALS FOR THE RECURSION FORMULA ARE USED AS THE CRITERIA FOR CONVERGENCE FOR CG. IN ALL CASES, CG TERMINATED WITHOUT CONVERGING TO THE DESIRED TOLERANCE BECAUSE "THE METHOD STAGNATED."

TABLE IV

| $n$ | Computational Time (sec) | |
|---|---|---|
| | CG | Recursion |
| $100,000$ | 1.006 | 0.066 |
| $200,000$ | 3.069 | 0.123 |
| $500,000$ | 4.568 | 0.471 |
| $1,000,000$ | 28.573 | 1.404 |
| $2,000,000$ | 113.832 | 2.934 |
| $5,000,000$ | 362.982 | 6.801 |
| $10,000,000$ | 540.158 | 12.208 |

COMPARISON OF THE COMPUTATIONAL TIMES BETWEEN THE PROPOSED RECURSION METHOD AND THE BUILT-IN CONJUGATE GRADIENT (CG) METHOD IN MATLAB.

time, especially for the larger problems. Compared to CG for the larger problems, not only does the proposed method yield more accurate solutions (its relative residuals are smaller than those from CG), but it is also significantly faster.

## V. CONCLUSION

In this paper, we proposed an algorithm based on the SMW formula to solve systems of the form $B_k + D$, where $B_k$ is an $n \times n$ L-BFGS quasi-Newton matrix. We showed that as long as the diagonal elements of $D$ are bounded away from zero, the algorithm is well-defined. The algorithm requires at most $M^2$ vector inner products. (Note: We assume that $M \ll n$, and thus, $M^2$ is also significantly smaller than $n$.) Numerical results demonstrate that the proposed method is not only accurate (the resulting relative residuals are comparable to the Matlab direct solver and the conjugate gradient method), but it is also efficient with respect to computational time (often up to about $50\times$ faster than CG). The algorithm proposed in this paper can be found at `http://www.wfu.edu/~erwayjb/software`.

## REFERENCES

[1] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-Newton matrices and their use in limited-memory methods," *Math. Program.*, vol. 63, pp. 129–156, 1994.
[2] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, pp. 503–528, 1989.
[3] J. Morales, "A numerical study of limited memory BFGS methods," *Applied Mathematics Letters*, vol. 15, no. 4, pp. 481–487, 2002.
[4] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Math. Comput.*, vol. 35, pp. 773–782, 1980.
[5] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer-Verlag, 2006.
[6] A. Fiacco and G. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques*, ser. Classics in applied mathematics. Society for Industrial and Applied Mathematics, 1990.
[7] P. E. Gill and D. P. Robinson, "A primal-dual augmented Lagrangian," *Computational Optimization and Applications*, pp. 1–25, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10589-010-9339-1
[8] N. I. M. Gould, "On the accurate determination of search directions for simple differentiable penalty functions," *IMA J. Numer. Anal.*, vol. 6, pp. 357–372, 1986.
[9] M. H. Wright, "The interior-point revolution in optimization: history, recent developments, and lasting consequences," *Bull. Amer. Math. Soc. (N.S*, vol. 42, pp. 39–56, 2005.
[10] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," in *Acta Numerica, 2005*, ser. Acta Numer. Cambridge: Cambridge Univ. Press, 2005, vol. 14, pp. 1–137.
[11] J. B. Erway and R. F. Marcia, "Limited-memory BFGS systems with diagonal updates," *Linear Algebra and its Applications*, 2012, accepted for publication.
[12] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1996, corrected reprint of the 1983 original.
[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, Maryland: The Johns Hopkins University Press, 1996.
[14] K. S. Miller, "On the Inverse of the Sum of Matrices," *Mathematics Magazine*, vol. 54, no. 2, pp. 67–72, 1981.
[15] J. E. Dennis Jr. and J. J. Moré, "Quasi-Newton methods, motivation and theory," *SIAM Review*, vol. 19, pp. 46–89, 1977.

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)