Linear Algebra and its Applications 437 (2012) 333–344

Contents lists available at SciVerse ScienceDirect

Linear Algebra and its Applications

journal homepage: www.elsevier.com/locate/laa

Limited-memory BFGS systems with diagonal updates

Jennifer B. Erway [a,*,1], Roummel F. Marcia [b,2]

[a] Department of Mathematics, Wake Forest University, Winston-Salem, NC 27109, United States
[b] Department of Applied Mathematics, University of California, Merced, Merced, CA 95343, United States

ARTICLE INFO

Article history:
Received 28 December 2011
Accepted 8 February 2012
Available online 7 March 2012

Submitted by H. Schneider

Keywords:
L-BFGS
Quasi-Newton methods
Limited-memory methods
Inverses
Sherman–Morrison–Woodbury
Diagonal updates

ABSTRACT

In this paper, we investigate a formula to solve systems of the form $(B + \sigma I)x = y$, where B is a limited-memory BFGS quasi-Newton matrix and σ is a positive constant. These types of systems arise naturally in large-scale optimization such as trust-region methods as well as doubly-augmented Lagrangian methods. We show that provided a simple condition holds on B_0 and σ, the system $(B + \sigma I)x = y$ can be solved via a recursion formula that requires 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.

1. Introduction

In this paper we develop a recursion formula for solving systems of the form $(B_k + \sigma I)x = y$, where B_k is the kth step $n \times n$ limited-memory (L-BFGS) quasi-Newton Hessian (see e.g. [1,18,20,24]), σ is a positive constant and $x, y \in \Re^n$. Linear systems of this form appear in both large-scale unconstrained and constrained optimization. For example, these equations arise in the optimality conditions for the two-norm trust-region subproblem and the so-called *doubly-augmented* system and their applications [8,9,11]. Additionally, matrices of the form $B_k + \sigma I$ can be used as preconditioners for $H + \sigma I$, where H is often the Hessian of a twice-continuously differentiable function f.

* Corresponding author. Tel.: +1 336 758 5356.
E-mail addresses: erwayjb@wfu.edu (J.B. Erway), rmarcia@ucmerced.edu (R.F. Marcia).
[1] Supported in part by NSF Grant DMS-08-11106.
[2] Supported in part by NSF Grant DMS-0965711.

0024-3795/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.laa.2012.02.005

The algorithm proposed in this paper is an extension of the Sherman–Morrison–Woodbury (SMW) formula to compute the inverse of $B_k + \sigma I$. The algorithm begins by pairing the initial L-BFGS matrix B_0 with the initial diagonal update σI; then, the algorithm uses the SMW formula to compute the inverse of $B_k + \sigma I$ after updating with a sequence of the L-BFGS updates. In this paper, we derive a recursion formula for efficiently computing matrix-vector products with this inverse. The proposed algorithm requires $\mathcal{O}(M^2 n)$ multiplications, where M is the maximum number of L-BFGS updates.

The structure of the remainder of the paper is as follows. In Section 2, we motivate in detail why solving systems of the form $(B_k + \sigma I)x = y$ is crucial in several optimization settings. Specifically, we consider its use in solving the trust-region subproblem and in the preconditioning of doubly-augmented systems in barrier methods. In Section 3, we provide an overview of the L-BFGS quasi-Newton matrix, including operation counts of the well-known recursive formulas for operations with the quasi-Newton matrix. In Section 4, we consider the formula for solving systems of the form $(B_k + \sigma I)x = y$ and show how it compares computationally with direct and indirect methods in Section 5. We offer potential extensions of this formula in Section 6 and draw some conclusions in Section 7.

In this paper, we assume that updates are chosen that ensure all L-BFGS quasi-Newton matrices are positive definite.

2. Motivations from large-scale optimization

Systems of the form $(B + \sigma I)x = y$, where B is a quasi-Newton Hessian appear throughout large-scale, nonlinear optimization. In this section, we motivate our research by presenting two instances in optimization that would benefit from a recursion formula to directly solve systems with a system matrix of the form $B + \sigma I$. The first motivation is in trust-region methods for unconstrained optimization; the second motivation comes from barrier methods for constrained optimization.

2.1. Motivation 1: Trust-region methods

Trust-region methods are one of the most popular types of methods for unconstrained optimization. Trust-region methods have been extended into the quasi-Newton setting by using BFGS or L-BFGS updates (see, for example [3,5,7,14,17,23,26,27,30]). The bulk of the work for a trust-region method occurs when solving the trust-region subproblem. Given the current trust-region iterate $\bar{x}$, the two-norm trust-region subproblem for minimizing a function f is given by

$$\underset{s\in\Re^n}{\text{minimize}} \;\; q(s) \triangleq g^T s + \frac{1}{2} s^T B s \quad \text{subject to} \;\; \|s\|_2 \leq \delta, \tag{1}$$

where $g \triangleq \nabla f(\bar{x})$, B is an L-BFGS quasi-Newton approximation to the Hessian of f at $\bar{x}$, and δ is a given positive trust-region radius.

Optimality conditions for the L-BFGS quasi-Newton trust-region subproblem are summarized in the following result (adapted from [2,13,22,28]).

Theorem 1. *Let δ be a given positive constant. A vector s^* is a global solution of the trust-region problem (1) if and only if $\|s^*\|_2 \leq \delta$ and there exists a unique $\sigma^* \geq 0$ such that*

$$(B + \sigma^* I)s^* = -g \quad \text{and} \quad \sigma^*(\delta - \|s^*\|_2) = 0. \tag{2}$$

Moreover, the global minimizer is unique.

The Moré–Sorensen method [21] is the preferred *direct* solver for the general trust-region subproblem. The method seeks a solution (x^*, σ^*) that satisfies the optimality conditions for the trust-region subproblem (in this case (2)) to any prescribed accuracy. The algorithm below summarizes the Moré–Sorensen method [21] for the *general* trust-region subproblem:

Algorithm 2.1: Moré–Sorensen method.
Let $\sigma \geq 0$ with $B + \sigma I$ positive definite and $\delta > 0$
while not *converged* **do**
Factor $B + \sigma I = R^T R$;
Solve $R^T R p = -g$;
Solve $R^T q = p$;
$\sigma \leftarrow \sigma + \left(\frac{\|p\|_2}{\|q\|_2}\right)^2 \left(\frac{\|p\|_2 - \delta}{\delta}\right)$;
end

Notice that the Moré–Sorensen method solves systems of the form $(B+\sigma I)s = -g$ at each iteration by computing the Cholesky factorization of $B+\sigma I$. For general large-scale optimization, where B is not a quasi-Newton Hessian, this method is often prohibitively expensive if one cannot exploit structure in the system matrix B. However, in the quasi-Newton setting, it is possible to compute the Cholesky factorization of a BFGS matrix of B_{k+1} by updating the factorization for B_k (see, for example [15]).

In this paper, we develop a method for solving the system $(B + \sigma I)s = -g$ using a recursion relation in place of using Cholesky factorizations. It is then possible to continue on with the Moré–Sorensen method by updating σ in accordance with the ideas proposed in [21]. (Note: The source of the formula for updating σ in Algorithm 2.1 is based on applying Newton's method to the second optimality condition in (2).) The recursion formula proposed in this paper extends the Moré–Sorensen method into the quasi-Newton setting without having to update Cholesky factorizations.

2.2. Motivation 2: Barrier methods

The second motivating example comes from barrier methods for constrained optimization (see, e.g. [10,29]). Consider the following problem:

$$\underset{x\in\Re^n}{\text{minimize}} \;\; f(x) \quad \text{subject to} \;\; c(x) \geq 0, \tag{3}$$

where $f(x) : \Re^n \to \Re$ is a real-valued function and $c(x) : \Re^n \to \Re$ is a quadratic constraint of the form $c(x) = \frac{1}{2}\delta^2 - \frac{1}{2}x^T x$, i.e., a two-norm constraint on the size of x where δ is a positive constant. (Note this can be considered a generalization of the trust-region subproblem.) A sufficient condition for a point x^* to be the minimizer of (3) is the existence of the Lagrange multiplier λ^* satisfying the following:

$$\begin{aligned} \nabla c(x^*)\lambda^* &= g(x^*), \qquad && \text{with} \;\; H(x^*) + \lambda^* I \;\; \text{positive definite,} \\ c(x^*)\lambda^* &= 0, && \text{with} \;\; \lambda^* > 0 \;\; \text{if} \;\; c(x^*) = 0 \;\; \text{and} \;\; \lambda^*, c(x^*) \geq 0, \end{aligned} \tag{4}$$

where $g(x) = \nabla f(x)$ is the gradient and $H(x) = \nabla^2 f(x)$ is the Hessian of $f(x)$. Primal-dual methods [4,12,25] solve this type of problem by solving a sequence of *perturbed* problems. Specifically, we define the function $F_\mu : \Re^{n+1} \to \Re^{n+1}$ with

$$F_\mu(x, \lambda) = \begin{pmatrix} g(x) + \lambda x \\ c(x)\lambda - \mu \end{pmatrix},$$

for some fixed perturbation parameter $\mu > 0$. Note that $\nabla c(x) = -x$ and as $\mu \to 0$, the root of $F_\mu(x, \lambda)$ converges to a point that satisfies the equations in (4).

The Newton equations associated with finding a root of $F_\mu(x, \lambda)$ are given by

$$\begin{pmatrix} H(x) + \lambda I & x \\ -\lambda x^T & c(x) \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = - \begin{pmatrix} g(x) + \lambda x \\ c(x)\lambda - \mu \end{pmatrix}. \tag{5}$$

By dividing the second row by $-\lambda$ and letting $d \triangleq c(x)/\lambda$, we get the symmetric system

$$\begin{pmatrix} H(x)+\lambda I & x \\ x^T & -d \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = -\begin{pmatrix} g(x)+\lambda x \\ (\mu - d)/\lambda \end{pmatrix}. \tag{6}$$

Unfortunately, the system matrix in (6) is indefinite even when $H+\lambda I$ is positive definite; however, after rearranging terms, it can be shown that this system is equivalent to the *doubly-augmented* system [11]:

$$\begin{pmatrix} H(x)+\lambda I+2xx^T & -x \\ -x^T & d \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = -\begin{pmatrix} g(x)+\lambda x+(2/d)(\mu - d)x \\ (d-\mu)/\lambda \end{pmatrix}, \tag{7}$$

which is a positive-definite system when $H + \lambda I$ is positive definite [11].

The linear system (7) can be preconditioned by the matrix

$$P = \begin{pmatrix} B+\lambda I+2xx^T & -x \\ -x^T & d \end{pmatrix}, \tag{8}$$

where B is an L-BFGS quasi-Newton approximation to $\nabla^2 f(x)$. Solves with the preconditioner P for a general system of the form

$$\begin{pmatrix} B+\lambda I+2xx^T & -x \\ -x^T & d \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

can be performed by solving the following equivalent system:

$$\begin{pmatrix} B+\lambda I & x \\ x^T & -d \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1+2y_2x \\ -y_2 \end{pmatrix} \tag{9}$$

(see [8,11]). Note that the inverse of the system matrix in (9) is given by

$$\begin{pmatrix} B+\lambda I & x \\ x^T & -1 \end{pmatrix}^{-1} = \begin{pmatrix} I & -w \\ 0 & 1 \end{pmatrix} \begin{pmatrix} (B+\lambda I)^{-1} & 0 \\ 0 & -(1+w^Tx)^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -w^T & 1 \end{pmatrix},$$

where $w = (B+\lambda I)^{-1}x$ (for details, see [8,11]). Thus, provided that solves with $B + \lambda I$ are efficient, solves with P are efficient.

In this paper, we develop a recursion formula that solves $(B+\lambda I)x = y$. Thus, this recursion formula allows the use of preconditioners the form (8) where B is a L-BFGS quasi-Newton approximation of $\nabla^2 f$.

3. The limited-memory BFGS method

In this section, we review the limited-memory BFGS (L-BFGS) method and state important and well-known recursion formulas for computing products with an L-BFGS quasi-Newton Hessian and its inverse.

The BFGS quasi-Newton method generates a sequence of positive-definite matrices $\{B_k\}$ 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. The L-BFGS quasi-Newton method can be viewed as the BFGS quasi-Newton method where only at most M recently computed updates are stored and used to update the initial matrix B_0.

Here M is a positive constant with $M \ll n$. The L-BFGS quasi-Newton approximation to the Hessian of f is implicitly updated as follows:

$$B_k = B_0 - \sum_{i=0}^{k-1} a_i a_i{}^T + \sum_{i=0}^{k-1} b_i b_i^T, \tag{10}$$

where

$$a_i = \frac{B_i s_i}{\sqrt{s_i^T B_i s_i}}, \quad b_i = \frac{y_i}{\sqrt{y_i^T s_i}}, \quad B_0 = \gamma_k^{-1} I, \tag{11}$$

and $\gamma_k > 0$ is a constant. In practice, γ_k is often taken to be $\gamma_k \triangleq s_{k-1}^T y_{k-1} / \|y_{k-1}\|_2^2$ (see, e.g. [18] or [24]).

Suppose that we have computed k updates ($k \leqslant M-1$) and have the following updates stored in S and Y:

$$S = [s_0, \ldots, s_{k-1}] \quad \text{and} \quad Y = [y_0, \ldots, y_{k-1}].$$

We update S and Y with the most recently computed vector pair (s_k, y_k) as follows:

Algorithm 3.1: Update S and Y.

if $k < M-1$,
  $S \leftarrow [S \;\; s_k]$; $Y \leftarrow [Y \;\; y_k]$; $k \leftarrow k+1$;
else
  for $i = 0, \ldots, k-1$
    $s_i \leftarrow s_{i+1}$; $y_i \leftarrow y_{i+1}$;
  end
  $S \leftarrow [s_0, \ldots s_{k-1}]$; $Y \leftarrow [y_0, \ldots y_{k-1}]$;
end

Thus, at all times we have exactly k stored vectors with $k \leqslant M-1$.

For the L-BFGS method, there is an efficient recursion relation to compute products with B_k^{-1}. Given a vector z, the following algorithm [24,25] terminates with $r \triangleq B_k^{-1} z$:

Algorithm 3.2: Two-loop recursion to compute $r = B_k^{-1} z$.

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

Pre-computing and storing $1/y_i^T s_i$ for $0 \leqslant i \leqslant k-1$, makes Algorithm 3.2 even more efficient. Further details on the L-BFGS method can found in [25]; further background on the BFGS can be found in [4]. The two-term recursion formula requires at most $\mathcal{O}(Mn)$ multiplications and additions. There is a compact matrix representation of the L-BFGS that can be used to compute products with the L-BFGS quasi-Newton matrix (see, e.g. [25]). The computational complexity at most $\mathcal{O}(Mn)$ multiplications.

There is an alternative representation of B_k^{-1} from which the two-term recursion can be more easily understood:

$$\begin{aligned} B_k^{-1} &= (V_{k-1}^T \cdots V_0^T) B_0^{-1} (V_0 \cdots V_{k-1}) + \frac{1}{y_0^T s_0} (V_{k-1}^T \cdots V_1^T) s_0 s_0^T (V_1 \cdots V_{k-1}) \\ &\quad + \frac{1}{y_1^T s_1} (V_{k-1}^T \cdots V_2^T) s_1 s_1^T (V_2 \cdots V_{k-1}) + \cdots + \frac{1}{y_{k-1}^T s_{k-1}} s_{k-1} s_{k-1}^T, \end{aligned} \tag{12}$$

where $V_i = I - \frac{1}{y_i^T s_i} y_i s_i^T$ (see, e.g. [25]). The first loop in the two-term recursion for $B_k^{-1} z$ computes and stores the products $(V_j \cdots V_{k-1})z$ for $j = 0, \ldots, k-1$; in between the first and second loop, B_0^{-1} is applied; and finally, the second loop computes the remainder of (12). Computing the inverse of $B_k + \sigma I$ is not equivalent to simply replacing B_0^{-1} in the two-loop recursion with $(B_0 + \sigma I)^{-1}$. To see this, notice that replacing B_0^{-1} in (12) with $(B_0 + \sigma I)^{-1}$ would apply the updates V_i to the full quantity $(B_0 + \sigma I)^{-1}$ instead of only B_0^{-1}. The main contribution of this paper is a recursion formula that computes $(B_k + \sigma I)^{-1} z$ in an efficient manner using only vector inner products.

4. The recursion formula

Consider the problem of finding the inverse of $B + \sigma I$, where B is an L-BFGS quasi-Newton matrix. The Sherman–Morrison–Woodbury (SMW) formula gives the following formula for computing the inverse of $A + UV^T$, assuming A is invertible (see [16]):

$$(A + UV^T)^{-1} = A^{-1} - A^{-1} U (I + V^T A^{-1} U)^{-1} V^T A^{-1}.$$

In the special case when UV^T is a rank-one update to A, this formula becomes

$$(A + uv^T)^{-1} = A^{-1} - A^{-1} u (I + v^T A^{-1} u)^{-1} v^T A^{-1},$$

where u and v are both n-vectors. For simplicity, first consider computing the inverse of an L-BFGS quasi-Newton matrix after only one update, i.e., the inverse of $B_1 + \sigma I$. Recall that

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

To compute the inverse of this, we apply SMW twice. To see this clearly, let

$$C_0 = (\gamma_1^{-1} + \sigma) I, \quad C_1 = (\gamma_1^{-1} + \sigma) I - a_0 a_0^T, \quad C_2 = (\gamma_1^{-1} + \sigma) I - a_0 a_0^T + b_0 b_0^T.$$

Applying SMW to C_1 yields

$$C_1^{-1} = C_0^{-1} + C_0^{-1} a_0 (1 - a_0^T C_0^{-1} a_0)^{-1} a_0^T C_0^{-1} \tag{13}$$

$$= C_0^{-1} + \frac{1}{(1 - a_0^T C_0^{-1} a_0)} C_0^{-1} a_0 a_0^T C_0^{-1}. \tag{14}$$

Applying SMW once more we obtain C_2^{-1} from C_1^{-1}:

$$C_2^{-1} = C_1^{-1} - \frac{1}{(1 + b_0^T C_1^{-1} b_0)} C_1^{-1} b_0 b_0^T C_1^{-1}$$

giving an expression for $(B_1+\sigma I)^{-1}$. This is the basis for the following recursion method that appears in [19]:

Theorem 2. *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 $C_0 = G$,*

$$C_{k+1}^{-1} = C_k^{-1} - v_k C_k^{-1} E_k C_k^{-1}, \quad k = 0,\ldots,M-1, \tag{15}$$

where

$$v_k = \frac{1}{1+\mathit{trace}\left(C_k^{-1}E_k\right)}. \tag{16}$$

In particular,

$$(G+H)^{-1} = C_{M-1}^{-1} - v_{M-1}C_{M-1}^{-1}E_{M-1}C_{M-1}^{-1}. \tag{17}$$

Proof. See [19]. □

We now show that applying the above recursion method to $B_k+\sigma I$, the product $(B_k+\sigma I)^{-1}z$ can be computed recursively, assuming $\gamma_k\sigma$ is bounded away from zero.

Theorem 3. *Let $\gamma_k > 0$ and $\sigma > 0$ with $\gamma_k\sigma > \epsilon$ for some $\epsilon > 0$. Let $G = B_0+\sigma I = (\gamma_k^{-1}+\sigma)I$, and let $H = \sum_{i=0}^{2k-1} E_i$, where*

$$E_0 = -a_0a_0^T, \quad E_1 = b_0b_0^T, \quad \ldots, \quad E_{2k-2} = -a_{k-1}a_{k-1}^T, \quad E_{2k-1} = b_{k-1}b_{k-1}^T.$$

Then $(B_k+\sigma I)^{-1} = (G+H)^{-1}$ is given by (17) together with (15).

Proof. Notice that this theorem follows from Theorem 2, provided we satisfy its assumptions. By construction, $B_k+\sigma I = G+H$. Both B_k and $B_k+\sigma I$ are nonsingular since B_k is positive definite and $\sigma \geq 0$. It remains to show that C_j, which is given by

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

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

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 [6]:

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

In other words, C_1 is invertible if C_0 is invertible and $a_0C_0^{-1}a_0 \neq 1$. We already established that C_0 is invertible; to show the latter condition, we use the definition of $a_0 = B_0s_0/\sqrt{s_0^TB_0s_0}$ together with $C_0^{-1} = (\gamma_k^{-1}+\sigma)^{-1}I$ to obtain the following:

$$a_0^TC_0^{-1}a_0 = \frac{\gamma_k^{-2}(\gamma_k^{-1}+\sigma)^{-1}}{\gamma_k^{-1}s_0^Ts_0}s_0^Ts_0 = \frac{1}{\gamma_k(\gamma_k^{-1}+\sigma)} = \frac{1}{1+\gamma_k\sigma}. \tag{18}$$

By hypothesis, $\gamma_k\sigma > \epsilon$, which implies that $\det(C_1) \neq 0$; thus, C_1 is 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 $C_{j+1} = B_i+\sigma I$, which is positive definite and therefore nonsingular. If j is even, i.e., $j = 2i$ for some i, then $C_j = B_i+\sigma I$, and

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

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

$$\begin{aligned} z^T C_{j+1} z &= z^T \left(B_i - \frac{1}{s_i^T B_i s_i} B_i s_i s_i^T B_i^T \right) z + \sigma \|z\|_2^2 \\ &= z^T B_i z - \frac{(z^T B_i s_i)^2}{s_i^T B_i s_i} + \sigma \|z\|_2^2 \\ &= \|B_i^{1/2} z\|_2^2 - \frac{((B_i^{1/2} z)^T (B_i^{1/2} s_i))^2}{\|B_i^{1/2} s_i\|_2^2} + \sigma \|z\|_2^2 \\ &= \|B_i^{1/2} z\|_2^2 - \|B_i^{1/2} z\|_2^2 \cos^2(\angle(B_i^{1/2} z, B_i^{1/2} s_i)) + \sigma \|z\|_2^2 \\ &> 0. \end{aligned} \tag{19}$$

We have now satisfied all the assumptions of Theorem 2. Therefore, $(B_j + \sigma I)^{-1}$ is given by (17) together with (15). □

Now, we show $r = C_{k+1}^{-1} z$ in (15) can be computed efficiently using recursion. We note that using (15), we have

$$C_{k+1}^{-1} z = C_k^{-1} z - v_k C_k^{-1} E_k C_k^{-1} z = \begin{cases} C_k^{-1} z + v_k C_k^{-1} a_{\frac{k}{2}} a_{\frac{k}{2}}^T C_k^{-1} z & \text{if } k \text{ is even} \\ C_k^{-1} z - v_k C_k^{-1} b_{\frac{k-1}{2}} b_{\frac{k-1}{2}}^T C_k^{-1} z & \text{if } k \text{ is odd.} \end{cases}$$

The quantity v_k is obtained using (16) and computing $\operatorname{trace}(C_k^{-1} E_k)$, which after substituting in the definition of E_k and computing the trace, is given by

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

If we define p_k according to the following rules

$$p_k = \begin{cases} C_k^{-1} a_{k/2} & \text{if } k \text{ is even} \\ C_k^{-1} b_{(k-1)/2} & \text{if } k \text{ is odd,} \end{cases} \tag{20}$$

then

$$v_k = \begin{cases} \dfrac{1}{1 - p_k^T a_{k/2}} & \text{if } k \text{ is even} \\ \dfrac{1}{1 + p_k^T b_{(k-1)/2}} & \text{if } k \text{ is odd,} \end{cases}$$

and thus,

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

Applying this recursively yields the following formula:

$$C_{k+1}^{-1} z = C_0^{-1} z + \sum_{i=0}^{k} (-1)^i v_i (p_i^T z) p_i, \tag{21}$$

for $k \geq 0$ and with $C_0^{-1} z = (\gamma_k^{-1} + \sigma)^{-1} z$.

Finally, it remains to demonstrate how to compute p_k in (20). Notice that we can compute p_k using (21):

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

The following pseudocode summarizes the algorithm for computing $r = C_{k+1}^{-1} z$:

Algorithm 4.1: Proposed recursion to compute $r = C_{k+1}^{-1} z$.

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

Algorithm 4.1 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. Experience has shown that k may be kept small (for example, Byrd et al. [1] suggest $k \in [2, 6]$), making the extra storage requirements and computations affordable.

5. Numerical experiments

We demonstrate the effectiveness of the proposed recursion formula by solving linear systems of the form (10) with 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 $n \leq 20,000$. In Tables 1–3, we show the time and the relative residuals for each method. 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, which is why the CG relative residuals are always less than those for the proposed recursion formula (except for one instance where the CG method stagnated.)

Analysis. The three methods were run on numerous problems with various problem sizes, and we note that all methods achieve very small relative residual errors for each of the problems we consid-

Table 1
A sample run comparing the relative residuals of the solutions using the Matlab "backslash" command, conjugate gradient (CG) method, and the proposed recursion formula. The relative residuals for the recursion formula are used as the criteria for convergence for CG.

n	Relative residual		
	Direct	CG	Recursion
1,000	2.84e-14	2.55e-14	3.62e-14
2,000	6.59e-14	5.57e-14	2.95e-13
5,000	1.02e-13	7.83e-14	8.83e-14
10,000	1.39e-13	1.09e-13	1.11e-13
20,000	2.63e-14	2.10e-14	2.14e-14

Table 2
The computational times to achieve the results in Table 1.

n	Time (s)		
	Direct	CG	Recursion
1,000	0.0311	0.0078	0.0015
2,000	0.2068	0.0099	0.0019
5,000	1.3692	0.0211	0.0048
10,000	8.0280	0.0306	0.0083
20,000	51.7772	0.0862	0.0160

Table 3
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 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.

n	Relative residual		Time (s)	
	CG	Recursion	CG	Recursion
100,000	8.71e-14	1.50e-13	0.2339	0.0584
200,000	1.47e-14	3.27e-14	0.4686	0.1155
500,000	4.90e-14[a]	3.55e-14	1.6996	0.3587
1,000,000	9.99e-15	1.03e-14	6.0068	1.0914
2,000,000	1.10e-13	6.54e-13	15.9798	2.5653
5,000,000	3.08e-14	4.84e-14	30.2865	6.2738
10,000,000	1.33e-14	3.97e-14	67.3049	11.5946

[a] In this case, CG terminated without converging to the desired tolerance because "the method stagnated."

ered. Besides from memory issues, the direct method suffers from significantly longer computational time, especially for the larger problems. Generally, the recursion algorithm takes about one-fourth the amount of time as the CG method. The CG method requires $2M + 2$ vector–vector products per iteration ($2M$ for the matrix-vector product and 2 for other vector–vector products) and in exact arithmetic will converge in $2M + 1$ iterations (because the matrix B_M in (10) is the sum of a scaled multiple of the identity with $2M$ rank-1 matrices, which means that B_M has at most $2M + 1$ unique eigenvalues). However, from our computational experience, the number of iterations is closer to $4M$, which brings the total vector–vector multiplications for CG to around $8M^2$. Meanwhile, the number of vector–vector multiplications for the recursion formula in Algorithm 4.1 is $(2M + 1)(2M + 2)/2 = 2M^2 + 3M + 1$, which explains why the CG algorithm takes roughly four times as long to achieve the same accuracy as the proposed recursion algorithm.

6. Extensions

The proposed recursion algorithm also computes products of the form $(B_k + D)^{-1}z$, where D is any positive-definite diagonal matrix. In this case, we must assume that each diagonal entry in D satisfies $d_{ii} \geq \sigma$ for some $\sigma > 0$. Provided $\gamma_k\sigma > \epsilon$, a theorem similar to Theorem 3 will hold true *mutatis mutandis*: the only steps in the proof that need changing are (18), which becomes

$$a_0^T C_0^{-1} a_0 = \frac{\gamma_k^{-2}}{\gamma_k^{-1} s_0^T s_0} s_0^T (B_0 + D)^{-1} s_0 = \frac{1}{\gamma_k s_0^T s_0} \sum_{i=1}^{n} \frac{1}{\gamma_k^{-1} + d_{ii}} (s_0)_i^2 \leq \frac{1}{1 + \gamma_k \sigma},$$

and the cascading equations in (19), whose $\sigma\|z\|_2^2$ terms become $z^T Dz$, which is greater than or equal to $\sigma\|z\|_2^2$. Additionally, the recursion formula for diagonal updates need not be limited to L-BFGS systems. In particular, it is also applicable to other quasi-Newton systems where a recursion formula exists and the quasi-Newton matrices are guaranteed to be positive definite. For example, the proposed recursion will work with quasi-Newton matrices using the Davidon–Fletcher–Powell (DFP) updating formula updating formula, but it will not work with quasi-Newton matrices based on the Symmetric Rank 1 (SR1) formula, which are not guaranteed to be positive definite (for more information on the DFP and SR1 method see, for example [25]).

7. Concluding remarks

In this paper, we proposed an algorithm based on the SMW formula to solve systems of the form $B + \sigma I$, where B is an $n \times n$ L-BFGS quasi-Newton matrix. We showed that as long as $\gamma\sigma > \epsilon$, 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.) While the algorithm is designed to handle constant diagonal updates of a quasi-Newton matrix, it can be extended to handle general diagonal updates of a quasi-Newton matrix. Furthermore, this algorithm can be extended to handle any quasi-Newton updating that ensures the quasi-Newton matrices are all positive definite. The algorithm proposed in this paper can be found at http://www.wfu.edu/∼erwayjb/software.

References

[1] R.H. Byrd, J. Nocedal, R.B. Schnabel, Representations of quasi-Newton matrices and their use in limited-memory methods, Math. Program. 63 (1994) 129–156.
[2] A.R. Conn, N.I.M. Gould, P.L. Toint, Trust-Region Methods, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.
[3] W.C. Davidon, New optimization algorithms with line searches, Math. Program. 9 (1976) 1–30.
[4] J.E. Dennis Jr., R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996., Corrected reprint of the 1983 original.
[5] J.E. Dennis Jr., H.H. Mei, Two new unconstrained optimization algorithms which use function and gradient values, J. Optim. Theory Appl. 28 (1979) 453–482.
[6] J.E. Dennis Jr., J.J. Moré, Quasi-Newton methods, motivation and theory, SIAM Rev. 19 (1977) 46–89.
[7] J.B. Erway, Quasi-Newton methods for the trust-region step, Technical report, Wake Forest University, 2011.
[8] J.B. Erway, P.E. Gill, A subspace minimization method for the trust-region step, SIAM J. Optim. 20 (2009) 1439–1461.
[9] J.B. Erway, P.E. Gill, J.D. Griffin, Iterative methods for finding a trust-region step, SIAM J. Optim. 20 (2009) 1110–1131.
[10] A. Fiacco, G. McCormick, Nonlinear programming: sequential unconstrained minimization techniques, in: Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 1990
[11] A. Forsgren, P.E. Gill, J.D. Griffin, Iterative solution of augmented systems arising in interior methods, SIAM J. Optim. 18 (2007) 666–690.
[12] A. Forsgren, P.E. Gill, M.H. Wright, Interior methods for nonlinear optimization, SIAM Rev. 44 (2002) 525–597.
[13] D.M. Gay, Computing optimal locally constrained steps, SIAM J. Sci. Statist. Comput. 2 (1981) 186–197.
[14] E.M. Gertz, A quasi-Newton trust-region method, Math. Program. 100 (2004) 447–470.
[15] P.E. Gill, G.H. Golub, W. Murray, M.A. Saunders, Methods for modifying matrix factorizations, in: Milestones in Matrix Computation: The Selected Works of Gene H. Golub with Commentaries, Oxford University Press, Walton Street, Oxford OX2 6DP, UK, 2007, pp. 309–344.
[16] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[17] L. Kaufman, Reduced storage, quasi-Newton trust region approaches to function optimization, SIAM J. Optim. 10 (1999) 56–69.
[18] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1989) 503–528.
[19] K.S. Miller, On the inverse of the sum of matrices, Math. Mag. 54 (1981) 67–72.
[20] J. Morales, A numerical study of limited memory BFGS methods, Appl. Math. Lett. 15 (2002) 481–487.
[21] J.J. Moré, D.C. Sorensen, Computing a trust region step, SIAM J. Sci. Statist. Comput. 4 (1983) 553–572.
[22] J.J. Moré, D.C. Sorensen, Newton's method, in: G.H. Golub (Ed.), Studies in Mathematics, MAA Studies in Numerical Analysis, vol. 24, Math. Assoc. America, Washington, DC, 1984, pp. 29–82.
[23] Q. Ni, Y.X. Yuan, A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization, Math. Comput. 66 (1997) 1509–1520.
[24] J. Nocedal, Updating quasi-Newton matrices with limited storage, Math. Comput. 35 (1980) 773–782.
[25] J. Nocedal, S.J. Wright, Numerical Optimization, second ed., Springer-Verlag, New York, 2006.
[26] M.J.D. Powell, A fortran subroutine for solving systems of nonlinear algebraic equations, in: P. Rabinowitz (Ed.), Numerical Methods for Nonlinear Algebraic Equations, Gordon and Breach, 1970, pp. 115–161.
[27] M.J.D. Powell, A hybrid method for nonlinear equations, in: P. Rabinowitz (Ed.), Numerical Methods for Nonlinear Algebraic Equations, Gordon and Breach, 1970, pp. 87–114.

[28] D.C. Sorensen, Newton's method with a model trust region modification, SIAM J. Numer. Anal. 19 (1982) 409–426.
[29] M.H. Wright, The interior-point revolution in optimization: history, recent developments, and lasting consequences, Bull. Amer. Math. Soc. (N.S.) 42 (2005) 39–56.
[30] Y.X. Yuan, Z.H. Wang, A subspace implementation of quasi-newton trust region methods for unconstrained optimization, Numer. Math. 104 (2006) 241–269.