ON SOLVING LARGE-SCALE LIMITED-MEMORY QUASI-NEWTON EQUATIONS

JENNIFER B. ERWAY AND ROUMMEL F. MARCIA

Abstract. We consider the problem of solving linear systems of equations arising with limited-memory members of the restricted Broyden class of updates and the symmetric rank-one (SR1) update. In this paper, we propose a new approach based on a practical implementation of the compact representation for the inverse of these limited-memory matrices. Numerical results suggest that the proposed method compares favorably in speed and accuracy to other algorithms and is competitive with several update-specific methods available to only a few members of the Broyden class of updates. Using the proposed approach has an additional benefit: The condition number of the system matrix can be computed efficiently.

Limited-memory quasi-Newton methods, compact representation, restricted Broyden class of updates, symmetric rank-one update, Broyden-Fletcher-Goldfarb-Shanno update, Davidon-Fletcher-Powell update, Sherman-Morrison-Woodbury formula

1. Introduction

We consider linear systems of the following form:

\[ B_{k+1} r = z, \]

where \( r, z \in \mathbb{R}^n \) and \( B_{k+1} \in \mathbb{R}^{n \times n} \) is a limited-memory quasi-Newton matrix obtained from applying \( k+1 \) Broyden class updates to an initial matrix \( B_0 \). We assume \( n \) is large, and thus, explicitly forming and storing \( B_{k+1} \) is impractical or impossible; moreover, in this setting, we assume limited-memory quasi-Newton matrices so that only the most recently-computed \( k \) updates are stored and used to update \( B_0 \). In practice, the value of \( k \) is small, i.e., less than 10 (see e.g., [5]), making \( k \ll n \). Problems such as (1) arise in quasi-Newton line-search and trust-region methods for large-scale optimization (see, e.g., [6,7,12,18]), as well as in preconditioning iterative solvers (see, e.g., [16,19]). In this paper, we propose a compact formulation of \( B_{k+1}^{-1} \) that can be used to efficiently solve (1).

Traditional quasi-Newton methods for minimizing a continuously differentiable function \( f : \mathbb{R}^n \to \mathbb{R} \) generate a sequence of iterates \( \{x_k\} \) such that \( f \) is strictly decreasing on this sequence. Moreover, at each iteration, the most recently-computed iterate \( x_{k+1} \) is used to update the quasi-Newton matrix by defining a new quasi-Newton pair \( (s_k, y_k) \) given by

\[ s_k \triangleq x_{k+1} - x_k \quad \text{and} \quad y_k \triangleq \nabla f(x_{k+1}) - \nabla f(x_k). \]

Research supported in part by NSF grants CMMI-1334042 and CMMI-1333326.
In the case of the Broyden class updates, $B_{k+1}$ is updated as follows:

$$B_{k+1} = B_k - \frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k y_k^T + \phi(s_k^T B_k s_k)w_k w_k^T,$$

with

$$w_k = \frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k},$$

where $\phi \in \mathbb{R}$. The most well-known quasi-Newton update is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update, which is obtained by setting $\phi = 0$. While this is the most widely-used update, research has suggested that other values of $\phi$ may lead to faster convergence [4,14,20]. Because of this interest in other values of $\phi$, the entire Broyden class (i.e., $\phi \in \mathbb{R}$) of quasi-Newton methods has been generalized to solve minimization problems over Riemannian manifolds [13]. For these reasons, in this paper, we consider solving (1) for other values of $\phi$ other than $\phi = 0$ (the BFGS update). In particular, we consider the restricted Broyden class of updates, where $\phi \in [0,1]$. Under mild assumptions, quasi-Newton matrices in this class are positive definite, a property that is important for computing search directions in optimization. We also consider the special case of the symmetric rank-one (SR1) update, an update that has been of recent interest in trust-region methods for unconstrained optimization [1,2]. (For more details on limited-memory quasi-Newton matrices and the Broyden class of matrices see, e.g., [5,12,18].)

Solving systems of the form (1) can be done efficiently in the case of the BFGS update ($\phi = 0$) using the well-known two-loop recursion [17] (see A for details). However, there is no known corresponding recursion method for other updates of the Broyden class (see [18, p.190]); in fact, for this reason many researchers prefer the BFGS update. In this paper we present various methods for solving linear systems of equations with limited-memory quasi-Newton matrices. The main contribution of this paper is a new approach by formulating a compact representation for the inverse of any member of the restricted Broyden class. This representation allows us to efficiently solve linear systems of the form (1). The compact formulation for the inverse of these matrices is based on ideas found in [10], where a compact formulation for the restricted Broyden class of matrices is presented. An additional benefit of our proposed approach is the ability to calculate the eigenvalues of the limited-memory matrix, and hence, the condition number of the linear system.

This paper is organized in seven main sections. In Section 2, we present current methods for solving linear systems with a limited-memory quasi-Newton matrix. In Section 3, we review the compact formulation for matrices obtained using the restricted Broyden class of updates. The compact formulation for the inverse of these matrices is presented in Section 4. In Section 5, we provide a practical implementation for solving systems of the form (1) when the system matrix is either a member of the restricted Broyden class or an SR1 matrix. In addition, we discuss how to compute the condition number of the linear system being solved and how to obtain additional computational savings when a new quasi-Newton pair is computed. We present numerical experiments in Section 6 that demonstrate the competitiveness of our proposed approach. Finally, concluding remarks are in Section 7.
2. Current methods

One approach to solve linear equations with members of the Broyden class is to use the Sherman-Morrison-Woodbury (SMW) formula to update the inverse after each rank-one change (see, e.g., [11]). Thus, to compute the inverse after a rank-two update, the SMW formula may be applied twice. (For algorithmic details on this approach, see B.) Alternatively, linear solves with a general member of the Broyden class can be performed using the following recursion formula found in [7] for $H_{k+1}$, the inverse of $B_{k+1}$. Specifically, the inverse of $B_{k+1}$ in (2) is given by the following rank-two update:

$$H_{k+1} = H_k + \frac{1}{s_k y_k} s_k s_k^T - \frac{1}{y_k^T H_k y_k} H_k y_k y_k^T H_k + \Phi_k (y_k^T H_k y_k) v_k v_k^T,$$

where

$$v_k = s_k - y_k^T H_k y_k,$$

$$\Phi_k = \frac{(1-\phi)(y_k^T s_k)^2}{(1-\phi)(y_k^T s_k)^2 + \phi(y_k^T H_k y_k)(s_k^T B_k s_k)},$$

and $H_k \triangleq B_k^{-1}$. (Algorithmic details associated with this approach can be found in [C].)

The method proposed in this paper makes use of the so-called compact formulation of quasi-Newton matrices that not only appears to be competitive in terms of speed but also yields additional information about the system matrix.

3. Compact representation

In this section, we briefly review the compact representation of members of the Broyden class of quasi-Newton matrices.

Given $k+1$ update pairs of the form $\{(s_i, y_i)\}$, the compact formulation of $B_{k+1}$ is given by

$$B_{k+1} = B_0 + \Psi_k M_k \Psi_k^T,$$

where $M_k$ is a square matrix and $B_0 \in \mathbb{R}^{n \times n}$ is an initial matrix, which is often taken to be a scalar multiple of the identity. (Note that the size of $\Psi$ is not specified.) The compact formulation of members of the Broyden class of updates are defined in terms of

$$S_k = \begin{bmatrix} s_0 & s_1 & s_2 & \cdots & s_k \end{bmatrix} \in \mathbb{R}^{n \times (k+1)},$$

$$Y_k = \begin{bmatrix} y_0 & y_1 & y_2 & \cdots & y_k \end{bmatrix} \in \mathbb{R}^{n \times (k+1)},$$

and the following decomposition of $S_k^T Y_k \in \mathbb{R}^{(k+1) \times (k+1)}$:

$$S_k^T Y_k = L_k + D_k + R_k,$$

where $L_k$ is strictly lower triangular, $D_k$ is diagonal, and $R_k$ is strictly upper triangular.

Assuming all updates are well defined, Erway and Marcia [10] derive the compact formulation for all members of the Broyden class for $\phi \in [0, 1]$. In particular, for any $\phi \in [0, 1]$, the compact formulation for $B_{k+1}$ is given by (5) where

$$\Psi_k = [B_0 S_k \ Y_k] \quad \text{and} \quad M_k = -\left[ S_k^T B_0 S_k - \phi \Lambda_k \begin{bmatrix} L_k - \phi \Lambda_k \\ (L_k - \phi \Lambda_k)^T \\ -(D_k + \phi \Lambda_k) \end{bmatrix} \right]^{-1},$$
where
\[ \Lambda_k = \text{diag} \left( \lambda_i \right) \quad \text{where} \quad \lambda_i = \frac{1}{1 - \frac{\phi}{s_i^T B_i s_i} - \frac{\phi}{s_i^T y_i}} \quad \text{for} \quad 0 \leq i \leq k. \]

When \( \phi = 0 \), this representation simplifies to the BFGS compact representation found in [5]. In the limited-memory case, \( k \ll n \), and thus, \( M_k \) is easily inverted.

4. Inverses

In this section, we present the compact formulation for the inverse of any member of the restricted Broyden class of matrices and for SR1 matrices. We then demonstrate how this compact formulation can be used to solve linear systems with these limited-memory matrices. Finally, we discuss computing the condition number of the linear system and potential computational savings when a new quasi-Newton pair is computed.

The main contribution of this paper is formulating a compact representation for the inverse for any member of the restricted Broyden class. It should be noted that the compact formulation for the inverse of a BFGS matrix is already known (see [5]). We now derive a general expression for the compact representation of any member of the restricted Broyden class. To do this, we apply the Sherman-Morrison-Woodbury formula (see, e.g., [11]) to the compact representation of \( B_{k+1} \) found in [5]:

\[
B_{k+1}^{-1} = B_0^{-1} + B_0^{-1} \Psi_k \left( -M_k^{-1} - \Psi_k^T B_0^{-1} \Psi_k \right)^{-1} \Psi_k^T B_0^{-1}.
\]

For quasi-Newton matrices it is conventional to let \( H_i \) denote the inverse of \( B_i \) for each \( i \); using this notation, the inverse of \( B_{k+1}^{-1} \) is given by

\[
H_{k+1} = H_0 + H_0 \Psi_k \left( -M_k^{-1} - \Psi_k^T H_0 \Psi_k \right)^{-1} \Psi_k^T H_0.
\]

Since \( H_0 \Psi_k = H_0 [B_0 S_k \ Y_k] = [S_k \ H_0 Y_k] \), (8) can be written as

\[
H_{k+1} = H_0 + [S_k \ H_0 Y_k] \left( -M_k^{-1} - \Psi_k^T H_0 \Psi_k \right)^{-1} \left[ \begin{array}{c} S_k^T \\ Y_k^T H_0 \end{array} \right].
\]

Finally, (9) can be simplified using

\[
\Psi_k^T H_0 \Psi_k = \left[ \begin{array}{cc} S_k^T B_0 S_k & S_k^T Y_k \\ Y_k^T S_k & Y_k^T H_0 Y_k \end{array} \right]
\]

to obtain

\[
\left( -M_k^{-1} - \Psi_k^T H_0 \Psi_k \right)^{-1} = \left[ \begin{array}{cc} S_k^T B_0 S_k - \phi \Lambda_k & (L_k - \phi \Lambda_k)^T \\ (L_k - \phi \Lambda_k) & \left( -D_k - \phi \Lambda_k \right) \left( Y_k^T S_k \ Y_k^T H_0 Y_k \right) \end{array} \right]^{-1}
\]

\[
= \left[ \begin{array}{cc} -\phi \Lambda_k & L_k - \phi \Lambda_k - S_k^T Y_k \\ L_k - \phi \Lambda_k - Y_k^T S_k & -D_k - \phi \Lambda_k - Y_k^T H_0 Y_k \end{array} \right]^{-1}
\]

\[
= \left[ \begin{array}{cc} -\phi \Lambda_k & -R_k - D_k - \phi \Lambda_k \\ -R_k^T - D_k - \phi \Lambda_k & -D_k - \phi \Lambda_k - Y_k^T H_0 Y_k \end{array} \right]^{-1}.
\]
In other words, the compact formulation for the inverse of a member of the restricted Broyden class is given by
\begin{equation}
H_{k+1} = H_0 + \tilde{\Psi}_k \tilde{M}_k \tilde{\Psi}_k^T
\end{equation}
where \( \tilde{\Psi}_k \equiv [S_k \ H_0 Y_k] \),
and
\begin{equation}
\tilde{M}_k \equiv \begin{bmatrix}
-\phi \Lambda_k & -R_k - D_k - \phi \Lambda_k \\
-R_k^T - D_k - \phi \Lambda_k & -D_k - \phi \Lambda_k - Y_k^T H_0 Y_k
\end{bmatrix}^{-1}.
\end{equation}

In the following section, we discuss a practical method to compute \( \tilde{M}_k \). In this section, we present a practical method to compute \( \tilde{M}_k \).

Lemma 1. Suppose \( H_1 = H_0 + \tilde{\Psi}_0 \tilde{M}_0 \tilde{\Psi}_0^T \), is the inverse of a member of the restricted Broyden class of updates after performing one update. Then,
\begin{equation}
\tilde{M}_0 = \begin{bmatrix}
\tilde{\alpha}_0 \\
\tilde{\beta}_0 \\
\tilde{\delta}_0
\end{bmatrix},
\end{equation}
where
\begin{equation}
\tilde{\alpha}_0 = \frac{1}{s_0^T y_0} + \Phi_0 \frac{y_0^T H_0 y_0}{(s_0^T y_0)^2}, \quad \tilde{\beta}_0 = -\frac{\Phi_0}{y_0^T s_0}, \quad \tilde{\delta}_0 = -\frac{1 - \Phi_0}{y_0^T H_0 y_0},
\end{equation}
and \( \Phi_0 \) is given in (4).

Proof. Expanding (5), yields
\begin{align*}
H_1 &= H_0 + \left( \frac{1}{s_0^T y_0} + \Phi_0 \frac{y_0^T H_0 y_0}{(s_0^T y_0)^2} \right) s_0 s_0^T - \frac{\Phi_0}{y_0^T s_0} H_0 y_0 y_0^T H_0 s_0 \\
&\quad - \frac{\Phi_0}{y_0^T s_0} s_0 y_0^T H_0 - \frac{1 - \Phi_0}{y_0^T H_0 y_0} y_0^T H_0,
\end{align*}
which simplifies to
\begin{equation}
H_1 = H_0 + \begin{bmatrix}
s_0 & H_0 y_0
\end{bmatrix} \begin{bmatrix}
\tilde{\alpha}_0 & \tilde{\beta}_0 \\
\tilde{\beta}_0 & \tilde{\delta}_0
\end{bmatrix} \begin{bmatrix}
s_0^T \\
y_0^T H_0
\end{bmatrix},
\end{equation}
where \( \tilde{\alpha}_0, \tilde{\beta}_0, \) and \( \tilde{\delta}_0 \) are defined as in (13) and \( \Phi_0 \) is given by (4). Note the (14) is of the form \( H_1 = H_0 + \tilde{\Psi}_0 \tilde{M}_0 \tilde{\Psi}_0^T \).

We now show that \( \tilde{M}_0 \) defined by (12) and (13) is equivalent to (11) with \( k = 0 \). To see this, we simplify the entries of (12). First, define
\begin{align*}
\Delta_0 \equiv (1 - \phi)(y_0^T s_0)^2 + \phi(y_0^T H_0 y_0)(s_0^T B_0 s_0).
\end{align*}
Note that \( \Delta_0 \neq 0 \) since \( H_0 \) and \( B_0 \) are positive definite and \( \phi \in [0, 1] \). Then, \( \Phi_0 = (1 - \phi)(s_0^T y_0)^2/\Delta_0 \), and thus,
\begin{align*}
\tilde{\alpha}_0 &= \frac{1}{s_0^T y_0} + \frac{(1 - \phi)y_0^T H_0 y_0}{\Delta_0}, \quad \tilde{\beta}_0 = -\frac{(1 - \phi)(s_0^T y_0)}{\Delta_0}, \quad \text{and} \quad \tilde{\delta}_0 = -\frac{(1 - \phi)(s_0^T B_0 s_0)}{\Delta_0}.
\end{align*}
Consider the inverse of (12), which is given by
\[
\begin{bmatrix}
\tilde{\alpha}_0 & \tilde{\beta}_0 \\
\tilde{\beta}_0 & \tilde{\delta}_0
\end{bmatrix}^{-1} = \frac{1}{\tilde{\alpha}_0 \tilde{\delta}_0 - \tilde{\beta}_0^2} \begin{bmatrix}
\tilde{\delta}_0 & -\tilde{\beta}_0 \\
-\tilde{\beta}_0 & \tilde{\alpha}_0
\end{bmatrix}.
\]

We now simplify the entries on the right side of (15). The determinant of (12) is given by
\[
(16) \tilde{\lambda}_0 - \tilde{\beta}_0^2 = -\frac{(1 - \Phi_0)}{(s_0^T y_0)(y_0^T H_0 y_0)} - \frac{\Phi_0(1 - \Phi_0)}{(s_0^T y_0)^2} - \frac{\Phi_0^2}{(s_0^T y_0)^2} = \frac{1}{\Delta_0} \left( s_0^T B_0 s_0 \right),
\]
where \( \lambda_0 \) in (16) is defined in (7). Thus, the first entry of (15) is given by
\[
\frac{\tilde{\beta}_0}{\tilde{\alpha}_0 \tilde{\delta}_0 - \tilde{\beta}_0^2} = -\frac{\phi(s_0^T B_0 s_0)}{\Delta_0} \Delta_0 \lambda_0 = -\phi \lambda_0.
\]

The off-diagonal elements of the right-hand side of (15) simplify as follows:
\[
(18) \frac{\tilde{\beta}_0}{\tilde{\alpha}_0 \tilde{\delta}_0 - \tilde{\beta}_0^2} = -\left( (1 - \phi)(s_0^T y_0) \right) \left( \Delta_0 \left( \frac{\lambda_0}{s_0^T B_0 s_0} \right) \right) = s_0^T y_0 + \phi \lambda_0.
\]
Finally, the last entry of (15) can be simplified as follows:
\[
\frac{\tilde{\alpha}_0}{\tilde{\alpha}_0 \tilde{\delta}_0 - \tilde{\beta}_0^2} = \frac{1}{s_0^T y_0 s_0^T B_0 s_0} + (1 - \phi)y_0^T H_0 y_0 \frac{\lambda_0}{s_0^T B_0 s_0} = -s_0^T y_0 - \phi \lambda_0 - y_0^T H_0 y_0.
\]

(For details in the calculations of (16), (18), (19), see [F]) Thus, (15) together with (17), (18), (19) gives
\[
\begin{bmatrix}
\tilde{\alpha}_0 & \tilde{\beta}_0 \\
\tilde{\beta}_0 & \tilde{\delta}_0
\end{bmatrix}^{-1} = \begin{bmatrix}
-\phi \lambda_0 & -s_0^T y_0 - \phi \lambda_0 \\
-s_0^T y_0 - \phi \lambda_0 & -s_0^T y_0 - \phi \lambda_0 - y_0^T H_0 y_0
\end{bmatrix},
\]
showing that \( \tilde{M}_0 \) defined by (12) and (13) is equivalent to (11) with \( k = 0 \).

Together with Lemma 1, the following theorem shows how \( \tilde{M}_k \) is related to \( \tilde{M}_{k-1} \); from this, we present an algorithm to compute \( \tilde{M}_k \) using recursion. This computation avoids explicitly forming \( B_k \), which is used in the definition of \( \Delta_k \) and \( \Phi_k \).

**Theorem 1.** Suppose \( H_{j+1} = H_0 + \tilde{\Psi}_j \tilde{M}_j \tilde{\Psi}_j^T \) as in (16) for all \( j \in \{0, \ldots, k\} \), is the inverse of a member of the Broyden class of updates for a fixed \( \phi \in [0, 1] \). If \( \tilde{M}_0 \) is defined by (12), then for all \( j \in \{1, \ldots, k\} \), \( \tilde{M}_j \) satisfies the recursion relation
\[
(20) \quad \tilde{M}_j = \Pi_j^T \begin{bmatrix}
\tilde{M}_{j-1} + \tilde{\beta}_j \tilde{u}_j^T \tilde{\beta}_j & \tilde{\beta}_j \tilde{u}_j^T \\
\tilde{\beta}_j \tilde{u}_j & \tilde{\delta}_j \tilde{u}_j^T
\end{bmatrix} \Pi_j, \text{ where } \Pi_j = \begin{bmatrix}
I_j & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & I_j & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\]
is such that \[ \tilde{\Psi}_{j-1} s_j H_0 y_j \Pi_j = \tilde{\Psi}_j, \]
and
\[
\tilde{\alpha}_j = \frac{1}{s_j^T y_j + \Phi_j y_j^T H_j y_j} \tilde{\beta}_j = -\frac{\Phi_j}{y_j^T s_j}, \quad \tilde{\delta}_j = -\frac{1 - \Phi_j}{y_j^T H_j y_j},
\]
and \( \tilde{u}_j = \tilde{M}_{j-1} \tilde{\Psi}_{j-1}^T y_j \).
Proof. This proof is by induction on \( j \). For the base case, we show that equation (20) holds for \( j = 1 \). Setting \( k = 1 \) in (3) yields

\[
H_2 = H_1 + \left[ s_1 \ H_1 y_1 \right] \begin{bmatrix} \tilde{\alpha}_1 & \tilde{\beta}_1 \\ \tilde{\beta}_1 & \tilde{\delta}_1 \end{bmatrix} \begin{bmatrix} s_1^T \\ y_1^T H_1 \end{bmatrix},
\]

where

\[
\tilde{\alpha}_1 = \frac{1}{s_1^T y_1} + \Phi_1 \frac{y_1^T H_1 y_1}{(s_1^T y_1)^2}, \quad \tilde{\beta}_1 = -\frac{\Phi_1}{y_1^T s_1}, \quad \text{and} \quad \tilde{\delta}_1 = -\frac{1 - \Phi_1}{y_1^T H_1 y_1}.
\]

By Lemma 1, (21) can be written as

\[
H_2 = H_0 + \tilde{\Psi}_0 \tilde{M}_0 \tilde{\Psi}_0^T + \left[ s_1 \ H_1 y_1 \right] \begin{bmatrix} \tilde{\alpha}_1 & \tilde{\beta}_1 \\ \tilde{\beta}_1 & \tilde{\delta}_1 \end{bmatrix} \begin{bmatrix} s_1^T \\ y_1^T H_1 \end{bmatrix},
\]

where \( \tilde{\Psi}_0 = [S_0 \ H_0 y_0] \) and \( \tilde{M}_0 \) is given by (12). Letting \( \tilde{u}_1 = \tilde{M}_0 \tilde{\Psi}_0^T y_1 \), we have that

\[
H_1 y_1 = \left( H_0 + \tilde{\Psi}_0 \tilde{M}_0 \tilde{\Psi}_0^T \right) y_1 = H_0 y_1 + \tilde{\Psi}_0 \tilde{u}_1.
\]

Substituting (23) into the last quantity on the right side of (22) yields

\[
\begin{bmatrix} s_1 & H_1 y_1 \end{bmatrix} \begin{bmatrix} \tilde{\alpha}_1 \\ \tilde{\beta}_1 \\ \tilde{\delta}_1 \end{bmatrix} \begin{bmatrix} s_1^T \\ y_1^T H_1 \end{bmatrix} = \begin{bmatrix} s_1 & H_0 y_1 + \tilde{\Psi}_0 \tilde{u}_1 \end{bmatrix} \begin{bmatrix} \tilde{\alpha}_1 \\ \tilde{\beta}_1 \\ \tilde{\delta}_1 \end{bmatrix} \begin{bmatrix} s_1^T \\ y_1^T H_1 + \tilde{u}_1^T \tilde{\Psi}_0^T \end{bmatrix}.
\]

Letting \( \Pi_1 \) be defined as in (20) with \( j = 1 \), gives that \( [\tilde{\Psi}_0 \ s_1 \ H_0 y_1] \Pi_1 = \tilde{\Psi}_1 \), and thus,

\[
H_2 = H_0 + \tilde{\Psi}_1 \Pi_1^T \tilde{M}_1 \Pi_1 \tilde{\Psi}_1^T,
\]

where

\[
\tilde{M}_1 \triangleq \begin{bmatrix} \tilde{M}_0 + \tilde{\delta}_1 \tilde{u}_1 \tilde{u}_1^T & \tilde{\beta}_1 \tilde{u}_1 & \tilde{\delta}_1 \tilde{u}_1 \\ \tilde{\beta}_1 \tilde{u}_1^T & \tilde{\alpha}_1 & \tilde{\beta}_1 \\ \tilde{\delta}_1 \tilde{u}_1^T & \tilde{\beta}_1 & \tilde{\delta}_1 \end{bmatrix}.
\]

It remains to show that \( \tilde{M}_1 = \Pi_1^T \tilde{M}_1 \Pi_1 \); however, for simplicity, we instead show \( \tilde{M}_1^{-1} = \Pi_1^T \tilde{M}_1^{-1} \Pi_1 \).

Notice that \( \tilde{M}_1 \) can be written as the product of three matrices:

\[
\tilde{M}_1 = \begin{bmatrix} I_2 & 0 & \tilde{u}_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{M}_0 & 0 \\ 0 & \tilde{N}_0 \end{bmatrix} \begin{bmatrix} I_2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{where} \quad \tilde{N}_0 = \begin{bmatrix} \tilde{\alpha}_1 & \tilde{\beta}_1 \\ \tilde{\beta}_1 & \tilde{\delta}_1 \end{bmatrix}.
\]

Then, \( \tilde{M}_1^{-1} \) is given by

\[
\tilde{M}_1^{-1} = \begin{bmatrix} I & 0 & 0 \\ 0 & 1 & 0 \\ -\tilde{u}_1^T & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{M}_0^{-1} & 0 \\ 0 & \tilde{N}_0^{-1} \end{bmatrix} \begin{bmatrix} I & 0 & -\tilde{u}_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\]

It can be shown, using similar computations as in the proof of Lemma 1, that

\[
\tilde{N}_0^{-1} = \frac{1}{\tilde{\alpha}_1 \tilde{\delta}_1 - \tilde{\beta}_1^2} \begin{bmatrix} \tilde{\delta}_1 & -\tilde{\beta}_1 \\ -\tilde{\beta}_1 & \tilde{\alpha}_1 \end{bmatrix} = \begin{bmatrix} -\phi \lambda_1 & -s_1^T y_1 - \phi \lambda_1 \\ -s_1^T y_1 - \phi \lambda_1 & -s_1^T y_1 - \phi \lambda_1 - y_1^T H_1 y_1 \end{bmatrix}.
\]
Substituting \( \tilde{N}_0^{-1} \) into (24) and simplifying yields

\[
(25) \tilde{M}_1^{-1} = \begin{bmatrix}
\tilde{M}_0^{-1} & 0 & -\tilde{\Psi}_0^Ty_1 \\
0 & -\phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} \\
-y_1^T \tilde{\Psi}_0 & -s_1^T y_1 - \phi_{\lambda_1} & \tilde{u}_1^T \tilde{M}_0^{-1} \tilde{u}_1 - s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_1 y_1
\end{bmatrix}.
\]

Terms in the (3,3)-entry in the right side of (25) can be simplified using that \( \tilde{M}_0^{-1} \tilde{u}_1 = \tilde{\Psi}_0^T y_1 \) and that \( y_1^T H_1 y_1 = y_1^T H_0 y_1 + y_1^T \tilde{\Psi}_0 \tilde{M}_0 \tilde{\Psi}_0^T y_1 \) as follows:

\[
\tilde{u}_1^T \tilde{M}_0^{-1} \tilde{u}_1 - y_1^T H_1 y_1 = y_1^T \tilde{\Psi}_0 \tilde{M}_0 \tilde{\Psi}_0^T y_1 - y_1^T H_1 y_1 = -y_1^T H_0 y_1 + y_1^T H_1 y_1 - y_1^T H_1 y_1 = -y_1^T H_0 y_1.
\]

Thus,

\[
(26) \tilde{M}_1^{-1} = \begin{bmatrix}
\tilde{M}_0^{-1} & 0 & -\tilde{\Psi}_0^Ty_1 \\
0 & -\phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} \\
-y_1^T \tilde{\Psi}_0 & -s_1^T y_1 - \phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1
\end{bmatrix}.
\]

Lemma 1, together with the substitution \( \tilde{\Psi}_0^T y_1 = \begin{bmatrix} S_0^T y_1 \\ Y_0^T H_0 y_1 \end{bmatrix} \), gives that

\[
\Pi_j^T \tilde{M}_1^{-1} \Pi_j = \Pi_j^T \begin{bmatrix}
\tilde{M}_0^{-1} & 0 & -\tilde{\Psi}_0^Ty_1 \\
0 & -\phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} \\
-y_1^T \tilde{\Psi}_0 & -s_1^T y_1 - \phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1
\end{bmatrix} \Pi_j
\]

\[
= \begin{bmatrix}
-\phi_{\lambda_1} & 0 & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1 \\
0 & -\phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1 \\
-y_1^T \tilde{\Psi}_0 & -s_1^T y_1 - \phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
-\phi_{\lambda_1} & 0 & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1 \\
0 & -\phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1 \\
-y_1^T \tilde{\Psi}_0 & -s_1^T y_1 - \phi_{\lambda_1} & -s_1^T y_1 - \phi_{\lambda_1} - y_1^T H_0 y_1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
-\phi_{\lambda_1} & -R_1 - D_1 - \phi_{\lambda_1} \\
-R_1 - D_1 - \phi_{\lambda_1} & -D_1 - \phi_{\lambda_1} - Y_1^T H_0 Y_1
\end{bmatrix}
\]

proving the base case.

The inductive step is similar to the base case. For the inductive step, we assume the theorem holds for \( k = 1, \ldots, j - 1 \) and now show it holds for \( k = j \). We begin, as before, setting \( k = j \) in (25) to obtain

\[
(27) H_{j+1} = H_j + [s_j \ H_j y_j] \begin{bmatrix}
\tilde{\alpha}_j \\ \tilde{\beta}_j \\ \tilde{\delta}_j
\end{bmatrix} \begin{bmatrix}
\Phi_j y_j^T H_j y_j \\
\Phi_j y_j^T H_j y_j \\
\Phi_j y_j^T H_j y_j
\end{bmatrix} \begin{bmatrix}
s_j^T y_j^T H_j
\end{bmatrix},
\]

where

\[
\tilde{\alpha}_j = \frac{1}{s_j^T y_j} + \Phi_j y_j^T H_j y_j, \quad \tilde{\beta}_j = -\frac{\Phi_j}{y_j^T s_j}, \quad \text{and} \quad \tilde{\delta}_j = -\frac{1 - \Phi_j}{y_j^T H_j y_j}.
\]
Using the induction hypothesis, (27) becomes

\[ H_{j+1} = H_0 + \check{\Psi}_{j-1} \hat{M}_{j-1} \check{\Psi}_{j-1} + [s_j \ H_j y_j] \left[ \begin{array}{c} \check{\alpha}_j \\ \check{\beta}_j \\ \check{\delta}_j \end{array} \right] \left[ \begin{array}{c} s_j^T \\ y_j^T H_j \end{array} \right]. \]

Similar to the base case,

\[
\left[ s_j \ H_j y_j \right] \left[ \begin{array}{c} \check{\alpha}_j \\ \check{\beta}_j \\ \check{\delta}_j \end{array} \right] \left[ \begin{array}{c} s_j^T \\ y_j^T H_j \end{array} \right] = \left[ s_j \ H_0 y_j + \check{\Psi}_{j-1} \hat{u}_j \check{\beta}_j \check{\delta}_j \check{u}_j \right] \left[ \begin{array}{c} \check{\alpha}_j \\ \check{\beta}_j \\ \check{\delta}_j \end{array} \right] \left[ \begin{array}{c} y_j^T H_0 + \hat{u}_j^T \check{\Psi}_{j-1} \\ \check{\Psi}_{j-1} \check{\Pi}_j \end{array} \right],
\]

where \( \hat{u}_j = \hat{M}_{j-1} \check{\Psi}_{j-1} y_j \). With \( \Pi_j \) defined as in (20), then \[ \left[ \check{\Psi}_{j-1} \ s_j \ H_0 y_j \right] \Pi_j = \check{\Psi}_j \] and so \( H_j = H_0 + \check{\Psi}_j \check{\Pi}_j \check{M}_j \check{\Psi}_j \), where

\[
\check{M}_j = \left[ \begin{array}{ccc} \check{M}_{j-1} & 0 & -\check{\Psi}_{j-1} y_j \\ 0 & -\phi \lambda_j & -s_j^T y_j - \phi \lambda_j \\ -y_j^T \check{\Psi}_{j-1} & -s_j^T y_j - \phi \lambda_j & -s_j^T y_j - \phi \lambda_j - y_j^T H_0 y_j \end{array} \right] .
\]

It remains to show that \( \check{M}_j = \Pi_j \check{M}_j \Pi_j \); however, for simplicity, we instead show \( \check{M}_j^{-1} = \Pi_j^T \check{M}_j^{-1} \Pi_j \) using a similar argument as in the base case. In particular, similar to (25), it can be shown that

\[
\check{M}_j^{-1} = \left[ \begin{array}{ccc} -\phi \Lambda_{j-1} & -R_{j-1} - D_{j-1} - \phi \Lambda_{j-1} & -Y_{j-1}^T H_0 Y_{j-1} \\ -R_{j-1} - D_{j-1} - \phi \Lambda_{j-1} & -D_{j-1} - \phi \Lambda_k - Y_{j-1}^T H_0 Y_{j-1} \end{array} \right] .
\]

By the inductive hypothesis that (11) holds for \( k = j - 1 \),

\[
\check{M}_j^{-1} = \left[ \begin{array}{ccc} -\phi \Lambda_{j-1} & -R_{j-1} - D_{j-1} - \phi \Lambda_{j-1} & -Y_{j-1}^T H_0 Y_{j-1} \\ -R_{j-1} - D_{j-1} - \phi \Lambda_{j-1} & -D_{j-1} - \phi \Lambda_k - Y_{j-1}^T H_0 Y_{j-1} \end{array} \right] .
\]

and so,

\[
\Pi_j^T \check{M}_j^{-1} \Pi_j = \check{M}_j^{-1},
\]

proving the induction step.
Using Theorem 1, Algorithm 1 computes \( \tilde{M}_k \) and then uses the compact formulation for the inverse to solve linear systems of the form (1). There are two parts to the for loop in Algorithm 1: The first half of the loop is used to build products of the form \( s_j^T B_j s_j \); the second half of the loop computes \( \tilde{M}_k \), making use of \( s_j^T B_j s_j \) to compute \( \tilde{\Phi}_j \). In the special case when \( \phi = 0 \) or \( \phi = 1 \), \( \tilde{\Phi}_j \) can be quickly computed as \( \Phi_j = I \) and \( \tilde{\Phi}_j = 0 \), respectively.

Define \( \phi \), \( B_0 \), and \( H_0 \);
Form \( M_0 \) using (9) and \( \tilde{M}_0 \) using (12);
Let \( \Psi_0 = (B_0 s_0 \ y_0) \) and \( \tilde{\Psi}_0 = (s_0 \ H_0 y_0) \);
for \( j = 1 : k \)

\[
\text{% Part 1: Compute } s_j^T B_j s_j \\
\begin{align*}
&u_j \leftarrow M_{j-1} (\Psi_{j-1}^T s_j); \\
&s_j^T B_j s_j \leftarrow s_j^T B_0 s_j + (s_j^T \Psi_{j-1}) u_j; \\
&\alpha_j \leftarrow -1/(s_j^T B_j s_j); \\
&\beta_j \leftarrow -\phi/(y_j^T s_j); \\
&\delta_j \leftarrow (1 + \phi(s_j^T B_j s_j))/(y_j^T s_j);
\end{align*}
\]

\[
M_j \leftarrow \Pi_j \left( \begin{array}{ccc}
\alpha_j u_j^T & \alpha_j & \beta_j \\
\beta_j u_j^T & \beta_j & \delta_j
\end{array} \right) \Pi_j^T, \text{ where } \Pi_j \text{ is as in (20)};
\]

\[
\text{% Part 2: Compute } \tilde{M}_j \\
\begin{align*}
&\tilde{u}_j \leftarrow M_{j-1} (\tilde{\Psi}_{j-1} y_j); \\
&y_j^T H_j y_j \leftarrow y_j^T H_0 y_j + (y_j^T \tilde{\Psi}_{j-1}) \tilde{u}_j; \\
&\tilde{\Phi}_j = (1 - \phi)(y_j^T s_j)^2 / ((1 - \phi)(y_j^T s_j)^2 + \phi(y_j^T H_j y_j)(s_j^T B_j s_j)); \\
&\tilde{\alpha}_j \leftarrow (1 + \tilde{\Phi}_j(y_j^T H_j y_j))/(y_j^T s_j); \\
&\tilde{\beta}_j \leftarrow -\tilde{\Phi}_j/(y_j^T s_j); \\
&\tilde{\delta}_j \leftarrow -1/(\tilde{\Phi}_j + y_j^T H_j y_j); \\
&\tilde{M}_j \leftarrow \Pi_j^T \left( \begin{array}{ccc}
\tilde{\alpha}_j \tilde{u}_j^T & \tilde{\beta}_j \tilde{u}_j^T & \tilde{\delta}_j \tilde{u}_j^T \\
\tilde{\beta}_j \tilde{u}_j^T & \tilde{\alpha}_j & \tilde{\beta}_j \\
\tilde{\delta}_j \tilde{u}_j^T & \tilde{\beta}_j & \tilde{\delta}_j
\end{array} \right) \Pi_j, \text{ where } \Pi_j \text{ is as in (20)};
\end{align*}
\]
end

\[
r = H_0 z + \tilde{\Psi}_k \tilde{M}_k \tilde{\Psi}_k^T z;
\]

**ALGORITHM 1:** Computing \( r = B_{k+1}^{-1} z \) using (10)

In Algorithm 1, the computations for \( u_j \) and \( \tilde{u}_j \) in lines 7 and 15, respectively, can be simplified by noting that

\[
\Psi_{j-1}^T s_j = \begin{bmatrix} S_{j-1}^T B_0 s_j \\ Y_{j-1}^T s_j \end{bmatrix} \quad \text{and} \quad \tilde{\Psi}_{j-1}^T y_j = \begin{bmatrix} S_{j-1}^T y_j \\ Y_{j-1}^T H_0 y_j \end{bmatrix}.
\]

In other words, \( \Psi_{j-1}^T s_j \) is a 2\( j \) vector whose first \( j \) entries are the first \( j \) entries in the \((j+1)\)th column of \( S_{j}^T B_0 s_k \) and whose last \( j \) entries are the first \( j \) entries in the \((j+1)\)th row of \( S_{j}^T Y_k \); similarly, \( \tilde{\Psi}_{j-1}^T y_j \) is a vector whose first \( j \) entries are the first \( j \) entries in the \((j+1)\)th column of \( S_{j}^T Y_k \) and whose last \( j \) entries are the first \( j \)th entries in the \((j+1)\)th column of \( Y_{j}^T H_0 Y_k \). Thus, computing \( u_j \) and \( \tilde{u}_j \)
only requires \(2j(4j - 1)\) flops additional flops after precomputing and storing the quantities \(S_k^T B_0 S_k\), \(Y_k^T H_0 Y_k\), and \(S_k^T Y_k\). Note that provided \(B_0\) is a scalar multiple of the identity matrix and \(H_0 = B_0^{-1}\), then forming \(S_k^T B_0 S_k\) and \(Y_k^T H_0 Y_k\) requires \((k + 1)(k + 2)n\) flops each, and forming \(S_k^T Y_k\) requires \((k + 1)^2(2n - 1)\) flops.

Computing the scalar \(s_j^T B_j s_j\) in line 8 of Algorithm 1 requires performing one inner product between \(\Psi_{j-1} s_j\) and \(u_j\) and then summing this with \(s_j^T B_0 s_j\), resulting in an operation cost of \(4j\). This is the same cost is associated with forming \(y_j^T H_j y_j\) in line 16. The scalar \(\Phi_j\) requires ten flops to compute. The other scalars \((\alpha_j, \beta_j, \delta_j, \tilde{\alpha}_j, \tilde{\beta}_j, \text{ and } \tilde{\delta}_j)\) require a total of 14 flops. Forming the matrices \(M_j\) and \(\tilde{M}_j\) requires \(24j^2 + 16j\) flops each. Thus, excluding the cost of forming \(S_k^T B_0 S_k\), \(S_k^T Y_k\), and \(Y_k^T H_0 Y_k\), the operation count for computing \(M_k\) is given by

\[
\sum_{j=1}^{k} 4j(4j - 1) + 8j + 24 + 24j^2 + 16j = \frac{1}{3} (40k^3 + 90k^2 + 122k).
\]

Finally, forming \(r\) in line 23 requires \(k + 1\) vector inner products and \((2k + 1)(n + k + 1) + 2n\) additional flops. Thus, the overall flop count for Algorithm 1 is \((2k + 1)(n + k + 1) + 2n + (40k^3 + 90k^2 + 122k)/3 + (2n - 1)(k + 1)\), which includes the cost of \(k + 1\) vector inner products.

Table 1 presents the computational complexity of Algorithm 1 and the algorithms in Appendices A-C for linear solves where the system matrix is from the restricted Broyden class of updates. The operational costs do not include the computation for \(S_k^T Y_k, S_k^T B_0 S_k\), and \(Y_k^T H_0 Y_k\), which all can be efficiently updated as new quasi-Newton pairs are computed.

| Alg. | Flop count (including vector inner products) |
|------|---------------------------------------------|
| 1    | \((2k + 1)(n + k + 1) + 2n + \frac{1}{3}(40k^3 + 90k^2 + 122k) + (2n - 1)(k + 1)\) |
| 3    | \(4nk + 3k + n + 2k(2n - 1)\) |
| 4    | \(\frac{1}{2}(k + 1)(23kn + 52n + 12k + 42) + (2n - 1)(6(k + 1)^2 + 7(k + 1))\) |
| 6    | \((5n + 3)(k + 1)k + (10n + 14)(k + 1) + (2n - 1)(3k + 2)(k + 1)\) |

Table 1. Computational complexity comparison of Algorithms 1, 3, 4, and 6 when the system matrix is a member of the restricted Broyden class of quasi-Newton updates.

5.1. Efficient solves after updates. To compute the factors in (8), it is necessary to form the matrices \(L_k\), \(D_k\) and \(R_k\). After a new quasi-Newton pair has been computed, updating (8) can be done efficiently by storing \(S_k^T S_k, Y_k^T Y_k\) and \(S_k^T Y_k\). In this case, to update \(H_{k+1}\) we add a column to (and possibly delete a column from) \(S_k\) and \(Y_k\); the corresponding changes can then be made in \(S_k^T S_k, Y_k^T Y_k\) and \(S_k^T Y_k\) (see [5] for more details).

In the practical implementation given in Algorithm 1, it is necessary to compute \(\Psi_{j-1} s_j\) and \(\Psi_{j-1} y_j\) given by (29), which can be formed using the stored quantities \(S_k^T S_k, Y_k^T Y_k\), and \(S_k^T Y_k\). To compute these quantities when a new quasi-Newton pair of updates is obtained we apply the same strategy as described above by adding (and possibly deleting) columns in \(S_k\) and \(Y_k\), enabling \(\Psi_{j-1} s_j\) and \(\Psi_{j-1} y_j\) to be updated efficiently. With these updates, the work required for Algorithm 1 is significantly reduced.
5.2. **Compact representation of the inverse of an SR1 matrix.** In this subsection, we demonstrate that this same strategy can be applied to the SR1 update. The symmetric rank-one (SR1) update is obtained by setting $\phi = y_k^T s_k / (y_k^T s_k - s_k^T B_k s_k)$; in this case,

$$B_{k+1} = B_k + \frac{1}{s_k^T (y_k - B_k s_k)} (y_k - B_k s_k)(y_k - B_k s_k)^T.$$  \hfill (30)

The SR1 update is a member of the Broyden class but not the restricted Broyden class. This update exhibits hereditary symmetric but not positive definiteness. This update is remarkable in that it is self-dual: Initializing $B_0^{-1}$ in place of $B_0$ and interchanging $y_k$ and $s_k$ everywhere in (30) yields a recursion relation for $B_{k+1}^{-1}$. Thus, like the BFGS update, solving linear systems with SR1 matrices can be performed using vector inner products (see [E] for details).

The compact formulation for the SR1 update is given by $B_{k+1} = B_0 + \tilde{\Psi}_k \tilde{M}_k \tilde{\Psi}_k^T$, where

$$\tilde{\Psi}_k = Y_k - B_0 S_k \quad \text{and} \quad \tilde{M}_k = (D_k + L_k + L_k^T - S_k^T B_0 S_k)^{-1},$$  \hfill (31)

where $\Psi_k \in \mathbb{R}^{n \times (k+1)}$ and $M_k \in \mathbb{R}^{(k+1) \times (k+1)}$ (see [F]). The SR1 update has the distinction of being the only rank-one update in the Broyden class of updates. It is for this reason that $\Psi_k$ has only $k + 1$ columns and $M_k$ is a $(k + 1) \times (k + 1)$ matrix. (In contrast, the compact representation for rank-two updates have twice as many columns and $M_k$ is twice the size as in the SR1 case.)

The compact formulation for the inverse of an SR1 matrix can be derived using the same strategy as for the restricted Broyden class. The SMW formula applied to the compact formulation of an SR1 with $\Psi_k$ and $M_k$ defined as in (31) yields

$$H_{k+1} = H_0 + H_0 (Y_k - B_0 S_k) (-M_k - \Psi_k^T H_0 \Psi_k)^{-1} (Y_k - B_0 S_k)^T H_0.$$  \hfill (32)

Substituting in for $M_k$ using (31) together with the identity

$$\Psi_k^T H_0 \Psi_k = Y_k^T H_0 Y_k - S_k^T Y_k - Y_k^T S_k + S_k^T B_0 S_k,$$

yields

$$H_{k+1} = H_0 + (H_0 Y_k - S_k)(- (D_k + L_k + L_k^T - S_k^T B_0 S_k))$$

$$- (Y_k^T H_0 Y_k - S_k^T Y_k - Y_k^T S_k + S_k^T B_0 S_k))^{-1} (H_0 Y_k - S_k)^T$$

$$= H_0 + (S_k - H_0 Y_k)^T (D_k + R_k + R_k^T - Y_k^T H_0 Y_k)^{-1} (S_k - H_0 Y_k)^T.$$  \hfill (32)

In other words, the compact formulation for the inverse of an SR1 matrix is given by

$$H_{k+1} = H_0 + \tilde{\Psi}_k \tilde{M}_k \tilde{\Psi}_k^T \quad \text{where} \quad \tilde{\Psi}_k \triangleq [S_k - H_0 Y_k],$$

and $\tilde{M}_k \triangleq (D_k + R_k + R_k^T - Y_k^T H_0 Y_k)^{-1}$. Algorithm 2 details how to solve a linear system defined by an L-SR1 matrix via the compact representation for its inverse.

Assuming $S_k^T Y_k$ is precomputed, Algorithm 2 requires $(2k + 1)(n + k + 1) + 2n + (2n - 1)(k + 1)$ flops, which includes $(k + 1)$ vector inner products to compute $\Psi_k^T z$ in line 5. Note that this count does not include the cost of inverting the $(k + 1) \times (k + 1)$ matrix, $\tilde{M}_k$, in line 4 since when $k \ll n$, this cost is significantly smaller than the dominant costs for Algorithm 2.
Define $H_0, S_k$ and $Y_k$;
Form $S_k^T Y_k = L_k + D_k + R_k$;
Form $Ψ_k = S_k - H_0 Y_k$;
$\tilde{M}_k \leftarrow (D_k + R_k + R_k^T - Y_k^T H_0 Y_k)^{-1}$;
$r = H_0 z + \tilde{Ψ}_k \tilde{M}_k \tilde{Ψ}_k^T z$;

**ALGORITHM 2:** Computing $r = B^{-1}_{k+1} z$ using [32] when $B_{k+1}$ is an SR1 matrix

Table 2 presents the computational complexity of Algorithm 2 and Algorithm 8 in [5] for solving SR1 linear systems. The operational costs do not include the computation for $S_k^T Y_k, S_k^T B_0 S_k, \text{ and } Y_k^T H_0 Y_k$, which all can be efficiently updated as new quasi-Newton pairs are computed.

| Algorithm | Flop count (including vector inner products) |
|-----------|---------------------------------------------|
| 2         | $(2k + 1)(n + k + 1) + 2n + (2n - 1)(k + 1)$ |
| 8         | $\frac{1}{2}(k + 1)(k(2n + 1) + 2(3n + 1)) + 2n + \frac{1}{2}(2n - 1)(k + 2)(k + 1)$ |

**Table 2.** Computational complexity comparison of Algorithms 2 and 8 when the system matrix is an SR1 matrix. Note that the computational cost for Algorithm 2 does not include the cost of inverting a $(k + 1) \times (k + 1)$ matrix, where $k \ll n$.

5.3. **Computing condition numbers.** Provided the initial approximate Hessian $B_0$ is taken to be a scalar multiple of the identity, (i.e., $B_0 = \gamma I$), it is possible to compute the condition number of $B_{k+1}$ in [11]. To do this, we consider the condition number of $H_{k+1} = B_{k+1}^{-1}$. The eigenvalues of $H_{k+1}$ can be obtained from the compact representation of $H_{k+1}$ together with the techniques from [3, 10]. For completeness, we review this approach below.

Suppose $B_{k+1}$ is a member of the restricted Broyden class. The eigenvalues of $H_{k+1}$ can be computed using the compact formulation for $H_{k+1}$ given in [10]:

$$H_{k+1} = H_0 + \tilde{Ψ}_k \tilde{M}_k \tilde{Ψ}_k^T.$$  

Letting $\tilde{Ψ}_k = QR$ be the QR decomposition of $\tilde{Ψ}_k$ where $Q \in \mathbb{R}^{n \times n}$ is an orthogonal matrix and $R \in \mathbb{R}^{n \times 2(k+1)}$ is an upper triangular matrix. Then,

$$H_{k+1} = H_0 + \tilde{Ψ}_k \tilde{M}_k \tilde{Ψ}_k^T = H_0 + Q R \tilde{M}_k R^T Q^T.$$  

The matrix $R \tilde{M}_k R^T$ is a real symmetric $n \times n$ matrix. However, since $\tilde{Ψ}_k \in \mathbb{R}^{n \times 2(k+1)}$ has at most rank $2(k + 1)$, then $R$ can be written in the form

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix},$$  

where $R_1 \in 2(k + 1) \times 2(k + 1)$. Thus,

$$R \tilde{M}_k R^T = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \tilde{M}_k \begin{pmatrix} R_1^T \\ 0 \end{pmatrix} = \begin{pmatrix} R_1 \tilde{M}_k R_1^T \\ 0 \end{pmatrix}.$$
Since $R_1 \tilde{M}_k R_1^T \in \mathbb{R}^{2(k+1) \times 2(k+1)}$, its spectral decomposition can be computed explicitly. Letting $V_1 D_1 V_1^T$ is the spectral decomposition of $R_1 \tilde{M}_k R_1^T$ and substituting into (34) yields:

$$H_{k+1} = H_0 + Q V D V^T Q^T$$

where

$$V \triangleq \begin{pmatrix} V_1 & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n \times n} \quad \text{and} \quad D \triangleq \begin{pmatrix} D_1 & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n \times n}.$$ 

Thus,

$$H_{k+1} = H_0 + Q V D V^T Q^T = Q V (\gamma^{-1} I + D) V^T Q^T,$$

giving the spectral decomposition of $H_{k+1}$. In particular, the matrix $H_{k+1}$ has an eigenvalue of $\gamma^{-1} - 1$ with multiplicity $n - 2(k + 1)$ and $2(k + 1)$ eigenvalues given by $\gamma^{-1} + d_i$ for $1 \leq i \leq 2(k + 1)$ where $d_i$ denotes the $i$th diagonal element of $D_1$.

Thus, the condition number of $H_{k+1}$ can be computed as the ratio of the largest eigenvalue to the smallest eigenvalue (in magnitude). The condition number of $B_{k+1}$ is the reciprocal of this value.

When $B_{k+1}$ a quasi-Newton matrix generated by SR1 updates, the procedure is similar except $\tilde{Ψ}_k$ has half as many columns resulting in $(k + 1)$ eigenvalues given by $\gamma^{-1} + d_i$ for $1 \leq i \leq (k + 1)$ and $n - (k + 1)$ eigenvalues of $\gamma^{-1}$. After a new quasi-Newton pair is computed it is possible to update the QR factorization (for details, see [10]).

6. Numerical Experiments

In this section, we demonstrate the accuracy of the proposed method for solving quasi-Newton equations. We solve the following linear system:

$$B_{k+1} p_{k+1} = -g_{k+1},$$

where $B_{k+1}$ is a member of the restricted Broyden class of updates or an SR1 matrix. For members of the restricted Broyden class of updates, we considered three values of $\phi$: 0.00, 0.50, and 0.99 and compare the following methods:

1. “Two-loop recursion” for the BFGS case ($\phi = 0.0$) (Algorithm 3)
2. Recursive SMW approach (Algorithm 4)
3. Recursion to compute products with $B_{k+1}^{-1} = H_{k+1}$ (Algorithm 6)
4. Compact inverses formulation (Algorithm 1)

For SR1 matrices, we compare the following methods:

1. The self-duality method (Algorithm 8)
2. Compact inverses formation (Algorithm 2)

We consider problem sizes ($n$) ranging from 10,000 to 1,000,000. We present the norms of the relative residuals:

$$\frac{\|B_{k+1} p_{k+1} - (-g_{k+1})\|_2}{\|g_{k+1}\|_2},$$

and the time (in seconds) needed to compute each solution. The number of limited-memory updates was set to five.

We simulate the first five iterations of an unconstrained line-search method. We generate the initial point $x_0$ and the gradients $g_j = \nabla f(x_j)$, for $0 \leq j \leq 5$, randomly and compute the subsequent iterations

$$x_{j+1} = x_j - \alpha_j H_j g_j, \quad \text{for} \ 0 \leq j \leq 5,$$
where \( H_j = B_j^{-1} \) is the inverse of the quasi-Newton matrix.

**Results.** We ran each algorithm ten times for each value of \( \phi (0.00, 0.50, \text{and } 0.99) \) and for each \( n (10,000, 50,000, 100,000 \text{ and } 1,000,000) \). The computational time results are shown in semi-log plots in Figure 1. Representative relative error results are presented in Tables 3–6. All algorithms were able to solve the linear systems to high accuracy. For the restricted Broyden class of matrices, the proposed compact inverse algorithm (Algorithm 1) outperformed all methods for all sizes of systems except for the L-BFGS case. In this case, Algorithm 1 is comparable to the two-loop recursion (Algorithm 3), which is specific to the L-BFGS update. (When \( n = 1,000,000 \), Algorithm 1 was slightly more efficient than Algorithm 3.) For the SR1 update, the compact inverse formulation outperformed the algorithm based on self-duality (Algorithm 8).

**Table 3.** Relative error for Broyden class matrices with \( \phi = 0.00 \) (BFGS).

| \( n \)   | Two-Loop Recursion (Alg. 3) | Recursive SMW (Alg. 4) | Recursive \( H_{k+1} \) (Alg. 6) | Compact Inverses (Alg. 1) |
|-------|-----------------------------|------------------------|-------------------------------|-------------------------|
| 10,000| 4.88e-16                    | 4.63e-16               | 5.37e-16                      | 3.59e-16                |
| 50,000| 2.93e-16                    | 4.44e-16               | 3.61e-16                      | 4.20e-16                |
| 100,000| 6.64e-16                   | 3.74e-16               | 6.16e-16                      | 3.81e-16                |
| 1,000,000| 1.47e-15                | 1.46e-15               | 1.45e-15                      | 1.51e-15                |

**Table 4.** Relative error for Broyden class matrices with \( \phi = 0.50 \).

| \( n \)   | Recursive SMW (Alg. 4) | Recursive \( H_{k+1} \) (Alg. 6) | Compact Inverses (Alg. 1) |
|-------|------------------------|-------------------------------|-------------------------|
| 10,000| 9.90e-16               | 8.37e-16                      | 8.15e-16                |
| 50,000| 4.25e-16               | 6.80e-16                      | 5.82e-15                |
| 100,000| 7.00e-16              | 6.31e-16                      | 9.14e-16                |
| 1,000,000| 2.77e-16           | 2.40e-16                      | 3.56e-16                |

**Table 5.** Relative error for Broyden class matrices with \( \phi = 0.99 \).

| \( n \)   | Recursive SMW (Alg. 4) | Recursive \( H_{k+1} \) (Alg. 6) | Compact Inverses (Alg. 1) |
|-------|------------------------|-------------------------------|-------------------------|
| 10,000| 8.33e-16               | 1.59e-15                      | 1.63e-15                |
| 50,000| 8.45e-15               | 1.21e-14                      | 3.88e-15                |
| 100,000| 7.28e-14              | 6.67e-14                      | 2.67e-14                |
| 1,000,000| 2.59e-15           | 1.80e-15                      | 3.29e-15                |
Table 6. Relative error for SR1 matrices.

| n      | Self-Duality (Alg. 8) | Compact Inverses (Alg. 2) |
|--------|-----------------------|---------------------------|
| 10,000 | 1.98e-15              | 6.10e-15                  |
| 50,000 | 2.24e-14              | 7.57e-14                  |
| 100,000| 5.07e-14              | 6.44e-14                  |
| 1,000,000| 8.67e-13            | 2.26e-12                  |

7. Concluding Remarks

We derived the compact formulation for members of the restricted Broyden class and the SR1 update. With this compact formulation, we showed how to solve linear systems defined by limited-memory quasi-Newton matrices. Numerical results suggest that this proposed approach is efficient and accurate. This approach has two distinct advantages over existing procedures for solving limited-memory quasi-Newton systems. First, there is a natural way to use the compact formulation for the inverse to obtain the condition number of the linear system. Second, when a new quasi-Newton pair is computed, computational savings can be achieved by simple updates to the matrix factors in the compact formulation.

Future work includes integrating this linear solver inside line-search and trust-region methods for large-scale optimization.

8. Acknowledgements

The authors would like to thank Tammy Kolda for initial conversations on this subject.

Appendix A. Two-loop recursion

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) update is obtained by setting \( \phi = 0 \) in (2). In this case, \( B_{k+1} \) simplifies to

\[
B_{k+1} = B_k - \frac{1}{s_k y_k^T} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k^T y_k.
\]

The inverse of the BFGS matrix \( B_{k+1} \) is given by

\[
B_{k+1}^{-1} = \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) B_k^{-1} \left( I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k},
\]

which can be written recursively as

\[
B_{k+1}^{-1} = (V_k^T \cdots V_0^T) B_0^{-1} (V_0 \cdots V_k) + \frac{1}{y_0^T s_0} (V_k^T \cdots V_1^T) s_0 s_0^T (V_1 \cdots V_k)
\]

\[
+ \frac{1}{y_k^T s_k} (V_k^T \cdots V_2^T) s_1 s_1^T (V_2 \cdots V_k) + \cdots + \frac{1}{y_k^T s_k} s_k s_k^T,
\]

where \( V_i = I - \frac{1}{y_i^T s_i} y_i s_i^T \) (see, e.g., [18, p.177–178]). Solving (1) can be done efficiently using the well-known two-loop recursion [17].

Assuming that the inner products \( y_i^T s_i \) can be precomputed and \( B_0 \) is a scalar multiple of the identity matrix, then the total operation count for Algorithm 3 is \( 4nk + 3k + n + 2k(2n - 1) \) flops, which includes \( 2k \) vector inner products.
ON SOLVING LARGE-SCALE LIMITED-MEMORY QUASI-NEWTON EQUATIONS

Matrix size

| n  | 10,000 | 50,000 | 100,000 | 1,000,000 |
|----|--------|--------|---------|-----------|
| Time (seconds) | $10^{-3}$ | $10^{-2}$ | $10^{-1}$ | $10^{0}$ |

Alg. 4 (SMW recursion)
Alg. 6 (Dual recursion)
Alg. 3 (Two-loop recursion)
Alg. 1 (Compact inverses)

(a) $\phi = 0$ (BFGS)
(b) $\phi = 0.5$
(c) $\phi = 0.99$
(d) SR1

Figure 1. Semi-log plots of the computational times (in seconds) for ten runs of each of the algorithms discussed in this paper. In (a), (b), and (c), the system matrix is a member of the restricted Broyden class of updates with $\phi = 0, 0.5,$ and 0.99, respectively. In (d), the system matrix is an SR1 matrix. The proposed method using compact inverses generally outperforms the other methods. Note that when $\phi = 0$, our proposed method is competitive with the “two-loop recursion”, which is specific to the BFGS update.

$q \leftarrow z$;
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

Algorithm 3: Two-loop recursion to compute $r = B_k^{-1} z$ when $B_k$ is a BFGS matrix.
Appendix B. SMW Recursion

In the symmetric case, the SMW formula for a rank-one change is given by
\[ (A + \alpha uu^T)^{-1} = A^{-1} - \frac{\alpha}{1 + \alpha u^T A u} A^{-1} uu^T A^{-1}. \]

We now show how (37) can be used to compute the inverse of \( B_{k+1} \). First, for \( 0 \leq j \leq k \), define
\[
\begin{align*}
\alpha_{3j} &= (y_j^T s_j)^{-1}, \\
\alpha_{3j+1} &= \phi(s_j^T B_j s_j), \\
\alpha_{3j+2} &= -(s_j^T B_j s_j)^{-1},
\end{align*}
\]
for \( 0 \leq i \leq 3k \), we define the matrices
\[ C_0 = B_0 \quad \text{and} \quad U_i = \alpha_i u_i u_i^T \]
By construction, \( C_{3(k+1)} = B_{k+1} \). It can be shown that each \( C_i \) is positive definite and so \( u_i^T C_i^{-1} u_i > 0 \) for each \( i \). For the restricted Broyden class, \( \alpha_{3j} \) and \( \alpha_{3j+1} \), \( j = 0, \ldots, k \) are both strictly positive. Thus, the denominator in (37) is strictly positive for the rank-one changes associated the 3\( j \) and 3\( j+1 \), \( j = 0, \ldots, k \), and thus, the SMW formula is well-defined for these rank-one updates. Since \( C_{3j} = B_j \) and \( B_j \) is positive definite, the rank-one update associated with \( j + 2, j = 0, \ldots, k \), must result in a positive-definite matrix; in other words, (37) is well defined. Thus, solving the system \( B_{k+1} r = z \) is equivalent to computing \( r = C_{3(k+1)}^{-1} z \). Apply the SMW formula in (37) to obtain the inverse of \( C_{i+1} \) from \( C_i^{-1} \), we obtain
\[ C_{i+1}^{-1} z = C_i^{-1} z - \frac{\alpha_i}{1 + \alpha_i u_i^T C_i^{-1} u_i} C_i^{-1} u_i u_i^T C_i^{-1} z, \]
for \( 0 \leq i < 3(k+1) \). Recursively applying (40) to \( C_i^{-1} z \), we obtain that
\[ C_{i+1}^{-1} z = C_i^{-1} z - \sum_{j=0}^{i} \frac{\alpha_j}{1 + \alpha_j u_j^T C_j^{-1} u_j} C_j^{-1} u_j u_j^T C_j^{-1} z, \]
and more importantly,
\[ C_{3(k+1)}^{-1} z = C_0^{-1} z - \sum_{j=0}^{3(k+1)-1} \frac{\alpha_j}{1 + \alpha_j u_j^T C_j^{-1} u_j} C_j^{-1} u_j u_j^T C_j^{-1} z. \]
Computing (42) can be simplified by defining the following two quantities:
\[ \tau_j = \frac{\alpha_j}{1 + \alpha_j u_j^T C_j^{-1} u_j} \quad \text{and} \quad p_j = C_j^{-1} u_j, \]
Substituting (43) into (42) yields
\[ C_{3(k+1)}^{-1} z = C_0^{-1} z - \sum_{j=0}^{3(k+1)-1} \tau_j (p_j^T z) p_j, \]
where \( \tau_j = \alpha_j(1 + \alpha_j p_j^T u_j)^{-1} \). The advantage of representation is that computing \( C_{3(k+1)}^{-1} z \) requires only vector inner products. Moreover, the vectors \( p_j \) can be computed by evaluating (41) at \( z = u_j \) for \( i = j - 1 \); that is,

\[
p_j = C_j^{-1} u_j = C_0^{-1} u_j - \sum_{i=0}^{j-1} \tau_i (p_i^T u_j) p_i.
\]

In Algorithm 4, we present the recursion to solve the linear system \( B_{k+1} r = z \) using the SMW formula (see related methods in [9,15]). We assume \( B_0 \) is an easily-invertible initial matrix.

\[
r \leftarrow C_0^{-1} z;
\]

for \( j = 0, \ldots, 3(k+1) - 1 \)

if \( \text{mod} (j, 3) = 0 \)

\( i \leftarrow j/3; \quad u \leftarrow y_i; \quad \alpha \leftarrow y_i^T s_i; \)

else if \( \text{mod} (j, 3) = 1 \)

\( i \leftarrow (j-1)/3; \quad u \leftarrow (y_i)/(y_i^T s_i) - (B_i s_i)/(s_i^T B_i s_i); \quad \alpha \leftarrow \phi(s_i^T B_i s_i); \)

else

\( i \leftarrow (j-2)/3; \quad u \leftarrow B_i s_i; \quad \alpha \leftarrow -(s_i^T B_i s_i)^{-1}; \)

end

\( p_j \leftarrow C_0^{-1} u; \)

for \( l = 0, \ldots, j \)

\( p_j \leftarrow p_j - \tau_l (p_l^T u) p_l; \)

end

\( \tau_j \leftarrow \alpha/(1 + \alpha p_j^T u); \)

\( r \leftarrow r - \tau_j (p_j^T z) p_j; \)

end

**ALGORITHM 4:** Computing \( r = B_{k+1}^{-1} z \)

At each iteration, Algorithm 4 computes \( \alpha \) and \( u \), which are defined in (38) and require matrix-vector products involving the matrices \( B_i \) for \( 0 \leq i \leq k \). The main difficulty in computing \( \alpha \) and \( u \) is the computation of matrix-vector products with the matrices \( B_i \) for each \( i \). Note that if we are able to form \( u_{3j+2} = B_j s_j \), then we are able to compute all other terms that use \( B_j s_j \) in (38). In what follows, we show how to compute \( u_{3j+2} \) without storing \( B_i \) for \( 0 \leq i \leq k \). This idea is based on [18, Procedure 7.6].

We begin by writing \( C_{3(k+1)} \) in (39) as follows:

\[
C_{3(k+1)} = B_0 + \sum_{j=0}^{k} \alpha_{3j} u_{3j}^T u_{3j} + \alpha_{3j+1} u_{3j+1}^T u_{3j+1} + \alpha_{3j+2} u_{3j+2}^T u_{3j+2},
\]

where \( \alpha_i \) and \( u_i \) are defined by (38). Since \( B_j = C_{3j} \), then

\[
u_{3j+2} = B_j s_j
\]

\[
= \left( B_0 + \sum_{i=0}^{j-1} \alpha_{3i} u_{3i}^T u_{3i} + \alpha_{3i+1} u_{3i+1}^T u_{3i+1} + \alpha_{3i+2} u_{3i+2}^T u_{3i+2} \right) s_j
\]

\[
= B_0 s_j + \sum_{i=0}^{j-1} \alpha_{3i} (u_{3i}^T s_j) u_{3i} + \alpha_{3i+1} (u_{3i+1}^T s_j) u_{3i+1} + \alpha_{3i+2} (u_{3i+2}^T s_j) u_{3i+2}.
\]
All the terms in the above summation only involve vectors that have been previously computed. Having computed $u_{3j+2}$, it is then possible to compute $\alpha_{3j+1}, \alpha_{3j+2},$ and $u_{3j+1}$. (The other terms $\alpha_{3j}$ and $u_{3j}$ do not depend on $B_js_j$.) The following algorithm computes the terms $\alpha$ and $u$ that are used in Algorithm 4.

\begin{algorithm}
for $j = 0, \ldots, k$
    $u_{3j} \leftarrow y_j$
    $\alpha_{3j} \leftarrow 1/(s_j^T y_j)$
    $u_{3j+2} \leftarrow B_0 s_j + \sum_{i=0}^{j-1} \left[ \alpha_{3i} (u_{3i}^T s_j) u_{3i} + \alpha_{3i+1} (u_{3i+1}^T s_j) u_{3i+1} + \alpha_{3i+2} (u_{3i+2}^T s_j) u_{3i+2} \right]$
    $\alpha_{3j+2} \leftarrow -1/(s_j^T u_{3j+2})$
    $u_{3j+1} \leftarrow \alpha_{3j} y_j + \alpha_{3j+2} u_{3j+2}$
    $\alpha_{3j+1} = -\phi/\alpha_{3j+2}$
end
\end{algorithm}

Algorithm 5: Unrolling the limited-memory Broyden convex class formula

Assuming $s_j^T y_j$ is precomputed at each step, Algorithm 5 requires

$$\sum_{j=0}^{k} \{3j+1\} = \frac{3(k+1)k}{2} + k + 1 = \frac{(k+1)(3k+2)}{2}$$

vector inner products and

$$\sum_{j=0}^{k} \{1 + 2n + (5n + 3)j + 1 + 3n + 1\} = \frac{(5n + 3)(k + 2)(k + 1)}{2}$$

additional flops. With the $\alpha$'s and the $u$'s computed in Algorithm 5, the rest of Algorithm 4 requires

$$\sum_{j=0}^{3(k+1)-1} \{j + 1\} = \frac{3(k+1)(3k+1+5)}{2}$$

vector inner products and

$$\sum_{j=0}^{3(k+1)-1} \{n + (2n + 1)\ell + 3 + (2n + 1)\} = \frac{3(k+1)(3k+1+1)(2n+1)}{2} + 3(k+1)(3n+4)$$

additional flops. Thus, the total number of vector inner products for Algorithm 4 is $6(k+1)^2 + 7(k+1)$ and the total flop count is

$$\frac{1}{2} (k+1)(23kn + 52n + 12k + 42) + (2n - 1)(6(k+1)^2 + 7(k+1)),$$

which includes $6(k+1)^2 + 7(k+1)$ vector inner products.

Appendix C. Computing products with $H_{k+1}$ recursively

The inverse of $B_{k+1}$ in (2) is given by

$$H_{k+1} = H_k + \frac{1}{y_k^T y_k} s_k s_k^T - \frac{1}{y_k^T H_k y_k} H_k y_k y_k^T H_k + \Phi_k (y_k^T H_k y_k) v_k v_k^T,$$
where

\[ v_k = \frac{s_k}{y_k^T s_k} - \frac{H_k y_k}{y_k^T H_k y_k} \quad \text{and} \quad \Phi_k = \frac{(1 - \phi)(y_k^T s_k)^2}{(1 - \phi)(y_k^T s_k)^2 + \phi(y_k^T H_k y_k)(s_k^T B_k s_k)}. \]

and \( H_k \triangleq B_k^{-1} \). Thus, the solution to \( B_{k+1} r = z \) can be obtained from

\[
B_{k+1}^{-1} z = \left( H_0 z + \sum_{i=0}^{k} \frac{1}{s_i^T y_i} s_i^T - \frac{1}{y_i^T H_i y_i} H_i y_i y_i^T H_i + \Phi_i(y_i^T H_i y_i) v_i^T v_i \right) z
\]

\[ = H_0 z + \sum_{i=0}^{k} \frac{s_i^T z}{s_i^T y_i} s_i - \frac{y_i^T H_i z}{y_i^T H_i y_i} H_i y_i + \Phi_i(y_i^T H_i y_i)(u_i^T z)v_i. \]

In order to avoid storing \( H_0, \ldots, H_k \) for matrix-vector products, we make use of the following variables:

\[
\begin{align*}
\alpha_{3i} &= (s_i^T y_i)^{-1}, & u_{3i} &= s_i, \\
\alpha_{3i+2} &= -(y_i^T H_i y_i)^{-1}, & u_{3i+2} &= H_i y_i, \\
\alpha_{3i+1} &= \Phi_i(y_i^T H_i y_i), & u_{3i+1} &= \frac{s_i}{y_i^T s_i} - \frac{H_i y_i}{y_i^T H_i s_i}.
\end{align*}
\]

With these definitions, we have that

\[
B_{k+1}^{-1} z = H_0 z + \sum_{i=0}^{k} \alpha_{3i} u_{3i} + \alpha_{3i} u_{3i+1} + \alpha_{3i} u_{3i+2}. 
\]

and, thus, a recursion relation can be used for matrix-vector products with \( H_0, \ldots, H_k \).

Algorithm 6 details how to solve for \( r \) in (1) using the expression for \( B_{k+1}^{-1} \) in (46) without explicitly storing \( H_0, \ldots, H_k \).

\[ r \leftarrow H_0 z; \]

\[ \text{for } j = 0, \ldots, 3(k + 1) - 1 \]

\[ \begin{cases} 
\text{if } \mod(j, 3) = 0 \\
\quad i \leftarrow j/3; \ u \leftarrow s_i; \ \alpha \leftarrow (y_i^T s_i)^{-1};\\
\text{else if } \mod(j, 3) = 1 \\
\quad i \leftarrow (j-1)/3; \ u \leftarrow H_i y_i; \ \alpha \leftarrow -(y_i^T H_i y_i)^{-1};\\
\text{else} \\
\quad i \leftarrow (j-2)/3; \ u \leftarrow (s_i)/(y_i^T s_i) - (H_i y_i)/(y_i^T H_i y_i); \\
\quad \Phi \leftarrow (1 - \phi)(y_i^T s_i)^2 / ((1 - \phi)(y_i^T s_i)^2 + \phi(y_i^T H_i y_i)(s_i^T B_i s_i)); \\
\quad \alpha \leftarrow \Phi(y_i^T H_i y_i); \\
\end{cases} \]

\[ r \leftarrow r + \alpha(u^T z)u; \]

\textbf{ALGORITHM 6: Computing } \( r = H_{k+1} z \)

The following algorithm computes products involving the matrices \( H_i \) for \( 0 \leq i \leq k \) for use in Algorithm 6. The algorithm avoids storing any matrices, and
matrix-vector products are computed recursively. The derivation of this algorithm is similar to that for Algorithm 5.

For $j = 0, \ldots, k$
\begin{align*}
\hat{u}_{3j} &\leftarrow s_j; \\
\hat{\alpha}_{3j} &\leftarrow 1/(s_j^T y_j); \\
\bar{u}_{3j+1} &\leftarrow H_0 y_j + \sum_{i=0}^{j-1} \left[ \hat{\alpha}_{3i} (\hat{u}_{3i}^T y_j) \hat{u}_{3i} + \hat{\alpha}_{3i+1} (\hat{u}_{3i+1}^T y_j) \hat{u}_{3i+1} + \hat{\alpha}_{3i+2} (\hat{u}_{3i+2}^T y_j) \hat{u}_{3i+2} \right]; \\
\bar{\alpha}_{3j+1} &\leftarrow -1/(y_j^T \bar{u}_{3j+2}); \\
\tilde{u}_{3j+2} &\leftarrow \bar{\alpha}_{3j+2} \tilde{u}_{3j+2}; \\
\text{Compute } \alpha_{3j+2} \text{ using Algorithm 5}; \\
\Phi_j &\leftarrow (1 - \phi)/(1 - \phi \hat{\alpha}_{3j}/(\hat{\alpha}_{3j+1} \alpha_{3j+2})); \\
\hat{\alpha}_{3j+2} &\leftarrow -\Phi_j/\hat{\alpha}_{3j+1}; \\
\end{align*}

**ALGORITHM 7:** Unrolling the limited-memory Broyden convex class formula

Assuming $s_j^T y_j$ is precomputed at each step, Algorithm 7 requires
\[ \sum_{j=0}^{k} (3j + 1) + \frac{(k + 1)(3k + 2)}{2} = (3k + 2)(k + 1) \]
vector inner products and
\[ \sum_{j=0}^{k} (1 + (5n + 3)j + 2n + 1 + 3n + 8 + 1) + \frac{3(k + 1)(3k + 1 + 5)}{2} = (5n + 3)(k + 1)k + (10n + 14)(k + 1) \]
additional flops. With the $\alpha$'s and the $u$'s computed in Algorithm 7, the rest of Algorithm 6 requires $3(k + 1)$ vector inner products and $3(k + 1)(2n + 1) + n$ additional flops. Thus, the total flop count for Algorithm 6 is
\[ (5n + 3)(k + 1)k + (k + 1)(16n + 17) + n + (2n - 1)(3k + 5)(k + 1), \]
which includes $(3k + 5)(k + 1)$ vector inner products.

**APPENDIX D. RELATIONSHIPS BETWEEN UPDATES**

We note that in the case of the BFGS update ($\phi = 0$), the compact representation of the BFGS matrix is consistent with its known compact representation derived in [5]. In particular, $\dot{M}_k$ in (11) simplifies to
\[ \dot{M}_k = (-M_k^{-1} - \Psi_k^T H_0 \Psi_k)^{-1} \]
\[ = \begin{bmatrix} 0 & -R_k - D_k \\ -R_k^T - D_k & -D_k - Y_k^T H_0 Y_k \end{bmatrix}^{-1} \]
\[ = \begin{bmatrix} \tilde{R}_k^{-T} (D_k + Y_k^T H_0 Y_k) \tilde{R}_k^{-1} & -\tilde{R}_k^{-T} \\ -\tilde{R}_k^{-1} & 0 \end{bmatrix}, \]
where $\tilde{R}_k = R_k + D_k$. Thus, the compact representation for the inverse of a BFGS matrix is given by
\[ H_{k+1} = H_0 + \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} \begin{bmatrix} \tilde{R}_k^{-T} (D_k + Y_k^T H_0 Y_k) \tilde{R}_k^{-1} & -\tilde{R}_k^{-T} \\ -\tilde{R}_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T Y_k^T \end{bmatrix}, \]
which is equivalent to that found in \cite{5} Equation (2.6)].

When \( \phi = 1 \), then \( B_{k+1} \) in (2) is known as the Davidon-Fletcher-Powell (DFP) update, which preceded the BFGS update. The BFGS and DFP formulas are known to be duals of each other, meaning one update can be obtained from the other by interchanging \( s_k \) with \( y_k \) and \( B_k \) with \( H_k \). Here, we demonstrate explicitly that the compact representation of the BFGS and DFP updates are also duals of each other. In addition, we show that the compact representation of the SR1 matrix is self-dual.

Consider the compact formulation for the inverse of a BFGS matrix (i.e., \( \phi = 0 \)) in (47). This is equivalent to

\[
H_{k+1} = H_0 + \begin{bmatrix} H_0y_k & s_k \end{bmatrix} \begin{bmatrix} 0 & -\bar{R}_k^{-T} \\ -\bar{R}_k & H_k^{-1} \end{bmatrix} \begin{bmatrix} y_k^T H_0 \\ s_k^T \end{bmatrix}.
\]

Now we consider interchanging \( B_0 \) with \( H_0 \) and \( Y_k \) with \( S_k \). Notice that if \( S_k \) and \( Y_k \) are interchanged then the upper triangular part of \( S_k^T Y_k \) corresponds to the lower triangular part of \( Y_k^T S_k \), implying that \( L_k \) and \( R_k \) must also be interchanged. Putting this all together yields:

\[
B_{k+1} = B_0 - \begin{bmatrix} B_0s_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0s_k + D_k & (L_k + D_k)^T \\ L_k + D_k & 0 \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix},
\]

which is the compact formulation for a DFP matrix (see \cite{8} Theorem 1). In other words, the compact representations of BFGS and DFP are complementary updates of each other.

For the DFP update, \( \lambda_k = -D_k \) in (7) since \( \phi = 1 \). Thus, the compact formulation for the inverse of a DFP matrix is given by

\[
H_{k+1} = H_0 + \begin{bmatrix} S_k & H_0y_k \end{bmatrix} \begin{bmatrix} D_k & -\bar{R}_k \\ -\bar{R}_k^T & -y_k^T H_0y_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T \\ y_k^T \end{bmatrix}.
\]

Interchanging \( H_0 \) with \( B_0 \) and \( S_k \) with \( Y_k \) (and \( R_k \) with \( L_k \)) yields

\[
B_{k+1} = B_0 - \begin{bmatrix} B_0s_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0s_k & L_k^T \\ L_k & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix},
\]

which is the compact formulation of the BFGS matrix (see Eq. (2.17) in \cite{5}).

Finally, it is worth noting that the inverse of compact formulation for an SR1 matrix is self-dual in the sense of Section 5.2. That is, replacing \( H_0 \) with \( B_0 \) and \( S_k \) with \( Y_k \) (and, thus, \( R_k \) with \( L_k \)) in (32) yields the compact formulation for the SR1 matrix given by (31).

**Appendix E. Solves with SR1 Matrices.**

The SR1 update is remarkable in that it is self-dual: initializing with \( B_0^{-1} \) instead of \( B_0 \), replacing \( s_k \) with \( y_k \), and replacing \( y_k \) with \( s_k \) for all \( k \) in (30) results in \( B_{k+1}^{-1} \) (see e.g., \cite{18}). Thus, \( r = B_{k+1}^{-1}z \) can be computed using recursion (see Algorithm
E.1). The first loop of Algorithm E.1 computes $p_i = s_i - H_i y_i$ for $i = 0, \ldots, k$; the final line of Algorithm E.1 computes $r = B_{k+1}^{-1} z = B_0^{-1} z + \sum_{i=0}^{k} (p_i^T z)/(p_i^T y_i) p_i$.

\begin{algorithm}
\textbf{Algorithm 8: Computing} $r = B_{k+1}^{-1} z$, when $B_{k+1}$ is an SR1 matrix

Thus, in the case of SR1 updates, solving linear systems with SR1 matrices can be performed using vector inner products; the cost for solving linear systems with an SR1 matrix is the same as the cost of computing products with an SR1 matrix, which can be done with

$$\left\{ k \sum_{i=0}^{k} i \right\} + k + 1 = \frac{(k + 2)(k + 1)}{2}$$

vector inner products and

$$\left\{ k \sum_{i=0}^{k} 2n + (2n + 1)i \right\} + 2n + (k + 1)(1 + n) = \frac{(k + 1)}{2} \left( k(2n + 1) + 2(3n + 1) \right) + 2n$$

additional flops. It should be noted that unlike members of the restricted Broyden class, SR1 matrices can be indefinite, and in particular, numerically singular. Methods found in [3,10] can be used to compute the eigenvalues (and thus, the condition number) of SR1 matrices before performing linear solves to help avoid solving ill-conditioned systems.
where \( \Delta_0 \) is defined in (7). Thus, the first entry of (15) is given by
\[
\tilde{\beta}_0 = \frac{1}{\Delta_0} \left( \frac{\lambda_0}{s_0^T B_0 s_0} \right)
\]
and the off-diagonal elements of the right-hand side of (15) simplify as follows:
\[
\tilde{\alpha}_0 \tilde{\delta}_0 - \tilde{\beta}_0^2 = -\frac{1 - \Phi_0}{s_0^T y_0(y_0 y_0)^T H_0 y_0} \left( \Phi_0 (1 - \Phi_0) - \frac{\Phi_0^2}{(s_0^T y_0)^2} \right)
\]
\[
= -\frac{\phi s_0^T B_0 s_0}{\Delta_0} \frac{1 - \phi}{\Delta_0} + \frac{\phi (s_0^T B_0 s_0)(s_0^T y_0)(1 - \phi)(y_0 y_0)^T H_0 y_0)}{\Delta_0}
\]
\[
= \frac{1}{(1 - \phi)(y_0 y_0)^T H_0 y_0) (s_0^T y_0)} \left( \Delta_0 (1 - \phi)(y_0 y_0)^T H_0 y_0) (s_0^T y_0) \right)
\]
Finally, the last entry of (15) can be simplified as follows:
\[
\tilde{\alpha}_0 = \frac{1}{s_0^T y_0 (s_0^T B_0 s_0)} \left( \Delta_0 \right) \frac{\lambda_0}{s_0^T B_0 s_0}
\]
\[
= \frac{1}{(1 - \phi)(y_0 y_0)^T H_0 y_0)(s_0^T y_0)} \left( \Delta_0 \right) \frac{\lambda_0}{s_0^T B_0 s_0}
\]
\[
= \frac{1}{(1 - \phi)(y_0 y_0)^T H_0 y_0)(s_0^T y_0)} \left( \Delta_0 \right) \frac{\lambda_0}{s_0^T B_0 s_0}
\]
\[
= \frac{1}{(1 - \phi)(y_0 y_0)^T H_0 y_0)(s_0^T y_0)} \left( \Delta_0 \right) \frac{\lambda_0}{s_0^T B_0 s_0}
\]
\[
= \frac{1}{(1 - \phi)(y_0 y_0)^T H_0 y_0)(s_0^T y_0)} \left( \Delta_0 \right) \frac{\lambda_0}{s_0^T B_0 s_0}
\]

References

[1] J. Brust, O. Burdakov, J. B. Erway, R. F. Marcia, and Y.-X. Yuan, Shape-changing L-SR1 trust-region methods. Submitted.
J. Brust, J. B. Erway, and R. F. Marcia, On solving L-SR1 trust-region subproblems.

O. Burdakov, L. Gong, Y.-X. Yuan, and S. Zikrin, On efficiently combining limited memory and trust-region techniques, Tech. Rep. 2013:13, Linköping University, The Institute of Technology, 2013.

R. H. Byrd, D. C. Liu, and J. Nocedal, On the behavior of broyden’s class of quasi-newton methods, SIAM Journal on Optimization, 2 (1992), pp. 533–557.

R. H. Byrd, J. Nocedal, and R. B. Schnabel, Representations of quasi-Newton matrices and their use in limited-memory methods, Math. Program., 63 (1994), pp. 129–156.

A. R. Conn, N. I. M. Gould, and P. L. Toint, Trust-Region Methods, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

J. E. Dennis, Jr and J. J. Moré, Quasi-newton methods, motivation and theory, SIAM Review, 19 (1977), pp. 46–89.

J. B. Erway, V. Jain, and R. F. Marcia, Shifted limited-memory DFP systems, in 2013 Asilomar Conference on Signals, Systems and Computers, Nov 2013, pp. 1033–1037.

J. B. Erway and R. F. Marcia, Limited-memory BFGS systems with diagonal updates, Linear Algebra and its Applications, 437 (2012), pp. 333–344.

J. B. Erway and R. F. Marcia, On efficiently computing the eigenvalues of limited-memory quasi-newton matrices, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 1338–1359.

G. H. Golub and C. F. Van Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, Maryland, third ed., 1996.

I. Griva, S. G. Nash, and A. Sofer, Linear and nonlinear programming, Society for Industrial and Applied Mathematics, Philadelphia, 2009.

W. Huang, K. A. Gallivan, and P.-A. Absil, A broyden class of quasi-newton methods for riemannian optimization, SIAM Journal on Optimization, 25 (2015), pp. 1660–1685.

C. Liu and S. A. Vander Wiel, Statistical quasi-newton: A new look at least change, SIAM Journal on Optimization, 18 (2007), pp. 1266–1285.

K. S. Miller, On the Inverse of the Sum of Matrices, Mathematics Magazine, 54 (1981), pp. 67–72.

J. L. Morales and J. Nocedal, Automatic preconditioning by limited memory quasi-newton updating, SIAM Journal on Optimization, 10 (2000), pp. 1079–1096.

J. Nocedal, Updating quasi-Newton matrices with limited storage, Mathematics of Computation, 35 (1980), pp. 773–782.

J. Nocedal and S. J. Wright, Numerical Optimization, Springer-Verlag, New York, second ed., 2006.

D. P. O’Leary and A. Yeremin, The linear algebra of block quasi-newton algorithms, Linear Algebra and Its Applications, 212 (1994), pp. 153–168.

Y. Zhang and R. Tewarson, Quasi-newton algorithms with updates from the preconvex part of broyden’s family, IMA Journal of Numerical Analysis, 8 (1988), pp. 487–509.

E-mail address: erwayjb@wfu.edu

Department of Mathematics, PO Box 7388, Wake Forest University, Winston-Salem, NC 27109

E-mail address: rmarcia@ucmerced.edu

School of Natural Sciences, University of California, Merced, 5200 N. Lake Road, Merced, CA 95343