MSS: MATLAB SOFTWARE FOR L-BFGS TRUST-REGION SUBPROBLEMS FOR LARGE-SCALE OPTIMIZATION

JENNIFER B. ERWAY AND ROUMMEL F. MARCIA

Abstract. A MATLAB implementation of the Moré-Sorensen sequential (MSS) method is presented. The MSS method computes the minimizer of a quadratic function defined by a limited-memory BFGS matrix subject to a two-norm trust-region constraint. This solver is an adaptation of the Moré-Sorensen direct method into an L-BFGS setting for large-scale optimization. The MSS method makes use of a recently proposed stable fast direct method for solving large shifted BFGS systems of equations [13, 12] and is able to compute solutions to any user-defined accuracy. This MATLAB implementation is a matrix-free iterative method for large-scale optimization. Numerical experiments on the CUTEr [3, 16]) suggest that using the MSS method as a trust-region subproblem solver can require significantly fewer function and gradient evaluations needed by a trust-region method as compared with the Steihaug-Toint method.

1. Introduction

In this paper we describe a MATLAB implementation for minimizing a quadratic function defined by a limited-memory BFGS (L-BFGS) matrix subject to a two-norm constraint, i.e., for a given x_k,

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

where $g \triangleq \nabla f(x_k)$, B is an L-BFGS approximation to $\nabla^2 f(x_k)$, and δ is a given positive constant. Approximately solving (1) is of interest to the optimization community, as it is the computational bottleneck of trust-region methods for large-scale optimization.

Generally speaking, there is a trade-off in computational cost per subproblem and the number of overall trust-region iterations (i.e., function and gradient evaluations): The more accurate the subproblem solver, the fewer overall iterations required. Solvers that reduce the overall number of function and gradient evaluations are of particular interest when function (or gradient) evaluations are time-consuming, e.g., simulation-based optimization. Some solvers such as the "dogleg" method (see [23, 22]) or the "double-dogleg" strategy (see [8])

Date: July 15, 2013.

Key words and phrases. Large-scale unconstrained optimization, trust-region methods, limited-memory quasi-Newton methods, L-BFGS.

J. B. Erway is supported in part by National Science Foundation grant CMMI-1334042. R. F. Marcia is supported in part by National Science Foundation grant CMMI-1333326.

compute an *approximate* solution to (1) by taking convex combinations of the steepest descent direction and the Newton step. Other solvers seek an approximate solution to the trust-region subproblem using an iterative approach. In particular, the Steihaug-Toint method computes an approximate solution to (1) that is guaranteed to achieve at least half the optimal reduction in the quadratic function when the model is convex [25, 15], but does not specifically seek to solve the minimization problem to high accuracy when the solution lies on the boundary. This paper presents an algorithm to solve (1) to any user-defined accuracy.

Methods to solve the trust-region subproblem to high accuracy are often based on optimality conditions given in the following theorem (see, e.g., Gay [14], Sorensen [24], Moré and Sorensen [19] or Conn, Gould and Toint [6]):

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

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

Moreover, if $B + \sigma^* I$ is positive definite, then the global minimizer is unique.

The Moré-Sorensen algorithm [19] seeks (p^*, σ^*) that satisfy the optimality conditions (2) by trading off between updating p and σ. That is, each iteration, the method updates p (fixing σ) by solving the linear system $(B + \sigma I)p = -g$ using the Cholesky factorization of the $B + \sigma I$; then, σ is updated using a safeguarded Newton method to find a root of

$$\phi(\sigma) \triangleq \frac{1}{\|p(\sigma)\|_2} - \frac{1}{\delta}. \tag{3}$$

The Moré-Sorensen direct method is arguably the best direct method for solving the trust-region subproblem; in fact, the accuracy of each solve can be specified by the user. While this method is practical for smaller-sized problems, in large-scale optimization it is too computationally expensive to compute and store Cholesky factorizations for unstructured Hessians.

Several researchers have proposed adaptations of the Moré-Sorensen direct method into the limited-memory BFGS setting. Burke et al. [4] derive a method via the Sherman-Morrison-Woodbury formula that uses two $M \times M$ Cholesky factorizations, where M is the number of limited-memory updates. While this technique is able to exploit properties of L-BFGS updates, there are potential instability issues related to their proposed use of the Sherman-Morrison-Woodbury that are not addressed. Lu and Monteiro [18] also explore a Moré-Sorensen method implementation when B has special structure; namely, $B = D + VEV^T$, where D and E are positive diagonal matrices, and V has a small number of columns. Their approach uses the Sherman-Morrison-Woodbury formula to replace solves with $(B + \sigma I)$ with solves with an $M \times M$ system composed of a diagonal plus a low rank matrix, and thus, avoid computing Choleksy factorizations. Like with [4], there are potential stability issues that are not addressed regarding inverting the $M \times M$ matrix.

Finally, Apostolopoulou et al. [2, 1] derive a closed-form expression for $(B + \sigma I)^{-1}$ to solve the first equation in (2). The authors are able to explicitly compute the eigenvalues of

B, provided $M = 1$ [2, 1] or $M = 2$ [1]. While their formula avoids potential instabilities associated the Sherman-Morrison-Woodbury formula, their formula is restricted to the case when the number of updates is at most two.

1.1. **Overview of the proposed methods.** In this paper, we describe a new adaptation of the Moré-Sorensen solver into a large-scale L-BFGS setting. The proposed method, called the *Moré-Sorensen sequential (MSS) method*, is able to exploit the structure of BFGS matrices to solve the shifted L-BFGS system in (2) using a fast direct recursion method that the authors originally proposed in [13]. (This recursion was later proven to be stable in [12].) The MSS method is able to solve (1) to any prescribed accuracy. A practical trust-region implementation is given in the numerical results section; less stringent implementations will be addressed in future work.

The paper is organized in five sections. In Section 2 we review L-BFGS quasi-Newton matrices and introduce notation that will be used for the duration of this paper. Section 4 includes numerical results comparing the Moré-Sorensen method and the MSS method. Finally, Section 5 includes some concluding remarks and observations.

1.2. **Notation and Glossary.** Unless explicitly indicated, $\|\cdot\|$ denotes the vector two-norm or its subordinate matrix norm. In this paper, all methods use L-BFGS updates and we assume they are selected to ensure the quasi-Newton matrices remain sufficiently positive definite (see, e.g., [4]).

2. Background

In this section, we begin with an overview of the L-BFGS quasi-Newton matrices described by Nocedal [20], defining notation that will be used throughout the paper.

The L-BFGS quasi-Newton method generates a sequence of positive-definite matrices $\{B_j\}$ from a sequence of vectors $\{y_j\}$ and $\{s_j\}$ defined as

$$y_j = \nabla f(x_{j+1}) - \nabla f(x_j) \quad \text{and} \quad s_j = x_{j+1} - x_j,$$

where $j = 0, \ldots m-1$ where $m \leq M$, and M is the maximum number of allowed stored pairs (y_j, s_j). This method can be viewed as the BFGS quasi-Newton method where no more than the M most recently computed updates are stored and used to update an initial matrix B_0. The L-BFGS quasi-Newton approximation to the Hessian of f is implicitly updated as follows:

$$B_m = B_0 - \sum_{i=0}^{m-1} a_i {a_i}^T + \sum_{i=0}^{m-1} b_i b_i^T, \tag{4}$$

where

$$a_i = \frac{B_i s_i}{\sqrt{s_i^T B_i s_i}}, \quad b_i = \frac{y_i}{\sqrt{y_i^T s_i}}, \quad B_0 = \gamma_m^{-1} I, \tag{5}$$

and $\gamma_m > 0$ is a constant. In practice, γ_m is often defined to be $\gamma_m \triangleq s_{m-1}^T y_{m-1} / \|y_{m-1}\|^2$ (see, e.g., [17] or [20]). In order to maintain that the sequence $\{B_i\}$ is positive definite for $i = 1, \ldots m$, each of the accepted pairs must satisfy $y_i^T s_i > 0$ for $i = 0, \ldots, m-1$.

One of the advantages of using an L-BFGS quasi-Newton is that there is an efficient recursion relation to compute products with B_m^{-1}. Given a vector z, the following algorithm [20, 21] terminates with $r \triangleq B_m^{-1}z$:

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

$q \leftarrow z$;
for $k = m-1, \ldots, 0$
  $\rho_k \leftarrow 1/(y_k^T s_k)$;
  $\alpha_k \leftarrow \rho_k s_k^T q$;
  $q \leftarrow q - \alpha_k y_k$;
end
$r \leftarrow B_0^{-1} q$;
for $k = 0, \ldots, m-1$
  $\beta \leftarrow \rho_k y_k^T r$;
  $r \leftarrow r + (\alpha_k - \beta) s_k$:
end

The two-term recursion formula requires at most $\mathcal{O}(Mn)$ multiplications and additions. To compute products with the L-BFGS quasi-Newton matrix, one may use the so-called "unrolling" formula, which requires $\mathcal{O}(M^2 n)$ multiplications, or one may use a compact matrix representation of the L-BFGS that can be used to compute products with the L-BFGS quasi-Newton matrix, which requires $\mathcal{O}(Mn)$ multiplications (see, e.g., [21]). Further details on L-BFGS updates can found in [21]; further background on the BFGS updates can be found in [7].

Without loss of generality, for the duration of the paper we assume that B is a symmetric positive-definite quasi-Newton matrix formed using m ($m \leq M$) L-BFGS updates.

3. The Moré-Sorensen Sequential (MSS) Method

In this section, we present the MSS method to solve the constrained optimization problem (1). We begin by considering the Moré-Sorensen direct method proposed in [19].

The Moré-Sorensen direct method seeks a pair (p, σ) that satisfy the optimality conditions (2) by alternating between updating p and σ. In the case that the B is positive definite (as in L-BFGS matrices), the method simplifies to Algorithm 2 [19].

Algorithm 2: Moré-Sorensen Method.

$\sigma \leftarrow 0$; $p \leftarrow -B^{-1}g$;
if $\|p\| \leq \delta$
  return;

else
  while *not* converged **do**
    Factor $B + \sigma I = R^T R$;
    Solve $R^T R p = -g$;
    Solve $R^T q = p$;
    $\sigma \leftarrow \sigma + \frac{\|p\|^2}{\|q\|^2} \frac{\|p\| - \delta}{\delta}$;
  end do
end

The update to σ in Algorithm 2 can be shown to be Newton's method applied (3). (Since B is positive definite, a safeguarded Newton's method is unnecessary.) Convergence is predicated on solving the optimality conditions (2) to a prescribed accuracy. The only difficulty in implementing the Moré-Sorensen method in a large-scale setting is the shifted solve $(B + \sigma I)p = -g$.

One method to directly solve systems of the form $(B + \sigma I)x = y$ is to view the system matrix as the sum of σI and rank-one L-BFGS updates to an initial diagonal matrix B_0. It is important to distinguish between *applying* rank-one L-BFGS updates to $B_0 + \sigma I$ and *viewing* the system matrix as the sum of rank-one updates to $B_0 + \sigma I$. To compute $(B + \sigma I)^{-1} y$, one cannot simply substitute $B_0 + \sigma I$ in for B_0 in Algorithm 1. (For a discussion on this, see [13]). In [13], Erway and Marcia present a stable fast direct method for solving L-BFGS systems that are shifted by a constant diagonal matrix (stability is shown in [12]). Specifically, it is shown that products with $(B + \sigma I)^{-1}$ can be computed provided $\gamma\sigma$ is bounded away from zero, where $B_0 \triangleq \gamma^{-1} I$. The following theorem is found in [13]:

Theorem 2. *Let $\gamma > 0$ and $\sigma \geq 0$. Suppose $G = (\gamma^{-1} + \sigma)I$, and let $H = \sum_{i=0}^{2m-1} E_i$, where*

$$E_0 = -a_0 a_0^T, \;\; E_1 = b_0 b_0^T, \;\; \ldots, \;\; E_{2m-2} = -a_{m-1} a_{m-1}^T, \;\; E_{2m-1} = b_{m-1} b_{m-1}^T,$$

with $\{a_i\}$ and $\{b_i\}$ are defined as in (5). Further, define $C_{m+1} = G + E_0 + \cdots + E_m$. If there exists some $\epsilon > 0$ such that $\gamma\sigma > \epsilon$, then $B + \sigma I = G + H$, and $(B + \sigma I)^{-1}$ is given by

$$(B + \sigma I)^{-1} = C_{2m-1}^{-1} - v_{2m-1} C_{2m-1}^{-1} E_{2m-1} C_{2m-1}^{-1}$$

where

$$C_{i+1}^{-1} = C_i^{-1} - v_i C_i^{-1} E_i C_i^{-1}, \;\; i = 0, \ldots, 2m-1,$$

and

$$v_i = \frac{1}{1 + \operatorname{trace}\left(C_i^{-1} E_i\right)}.$$

Proof. See [13]. □

This theorem is the basis for the following algorithm derived in [13] to compute products with $(B + \sigma I)^{-1}$. The algorithm was shown to be stable in [12].

Algorithm 3: Recursion to compute $x = (B + \sigma I)^{-1} y$.

$x \leftarrow (\gamma^{-1} + \sigma)^{-1} y$;
for $k = 0, \ldots, 2m-1$
  if k even
    $c \leftarrow a_{k/2}$;
  else
    $c \leftarrow b_{(k-1)/2}$;
  end
  $r_k \leftarrow (\gamma^{-1} + \sigma)^{-1} c$;
  for $i = 0, \ldots, k-1$
    $r_k \leftarrow r_k + (-1)^i v_i (r_i^T c) r_i$;
  end
  $v_k \leftarrow 1/(1 + (-1)^{k+1} r_k^T c)$;
  $x \leftarrow x + (-1)^k v_k (r_k^T y) r_k$;
end

It is important to note that $\{a_i\}$ and $\{b_i\}$ need to be precomputed. Algorithm 3 requires at most $\mathcal{O}(M^2 n)$. Operations with $C_0 = B_0 + \sigma I$ and C_1 can be easily computed with minimal extra expense since C_0^{-1} is a diagonal matrix. It is generally known that M may be kept small (for example, Byrd et al. [5] suggest $M \in [3, 7]$). When $M^2 \ll n$, the extra storage requirements and computations are affordable.

3.1. **Handling small σ.** ε In this section we discuss the case of solving $(B + \sigma I)p = -g$ for small σ. For stability, it is important to maintain $\gamma\sigma > \varepsilon$ for a small positive ε. Thus, in order to use Algorithm 3, we require that both $\gamma > \sqrt{\varepsilon}$ and $\sigma > \sqrt{\varepsilon}$. The first requirement is easily met by thresholding γ, i.e., $\gamma \leftarrow \max\{\sqrt{\varepsilon}, \gamma\}$. (Recall that $y_i^T s_i > 0$ for each $i = 0, \ldots m-1$, and thus, $\gamma > 0$.) When $\sigma \leq \sqrt{\varepsilon}$, we set $\sigma = 0$ and use the two-loop recursion (Algorithm 1) to solve an unshifted L-BFGS system.

3.2. **The algorithm.** The MSS method adapts the Moré-Sorensen method into an L-BFGS setting by solving $(B + \sigma I)p = -g$ using a recursion method instead of a Cholesky factorization. When σ is sufficiently large, the recently proposed recursion algorithm (Algorithm 3) is used to update s; otherwise, $\sigma \approx 0$ and the two-loop recursion (Algorithm 1) is used. Also, note that the Moré-Sorensen method updates σ using Newton's method applied to (3), i.e.,

$$\sigma \leftarrow \sigma - \phi(p)/\phi'(p). \tag{6}$$

In Algorithm 2 this update is written in terms of the Cholesky factors. In the MSS algorithm, Cholesky factors are unavailable, and thus, the update to σ is performed by explicitly computing the Newton step (6). For details on this update see [6].

The MSS method is summarized in Algorithm 4:

Algorithm 4: Moré-Sorensen Sequential (MSS) Method.
Input: $\gamma > \sqrt{\varepsilon}$ where $0 < \varepsilon \ll 1$

Output: (p, σ)
$\sigma \leftarrow 0$; $p \leftarrow -B^{-1}g$;
if $\|p\| \leq \delta$
  return;
else
  while *not* converged **do**
    $\phi(p) \leftarrow 1/\|p\| - 1/\delta$;
    if $\sigma > \sqrt{\varepsilon}$
      Compute $\hat{p}$ such that $(B + \sigma)\hat{p} = -p$ using Algorithm 3;
    else
      Compute $\hat{p}$ such that $B\hat{p} = -p$ using Algorithm 1;
      $\sigma \leftarrow 0$;
    end
    $\phi'(p) \leftarrow -(p^T\hat{p})/\|p\|^3$;
    $\sigma \leftarrow \sigma - \phi(p)/\phi'(p)$;
    if $\sigma > \sqrt{\varepsilon}$
      Compute p such that $(B + \sigma)p = -g$ using Algorithm 3;
    else
      Compute p such that $Bp = -g$ using Algorithm 1;
      $\sigma \leftarrow 0$;
    end
  end
end

3.3. **Stopping criteria.** If the initial solution $p = -B^{-1}g$ is within the trust-region, i.e., $\|p\| \leq \delta$, the MSS method terminates. Otherwise, the MSS method iterates until the solution to (1) is sufficiently close to the constraint boundary, i.e., $|\|p\| - \delta| \leq \tau\delta$, for a fixed $\tau \ll 1$.

4. Numerical Results

We test the efficiency of the MSS method by solving large problems from the CUTEr test collection (see [3, 16]). The test set was constructed using the CUTEr interactive `select` tool, which allows the identification of groups of problems with certain characteristics. In our case, the `select` tool was used to identify the twice-continuously differentiable unconstrained problems for which the number of variables can be varied. This process selected 67 problems: `arwhead`, `bdqrtic`, `broydn7d`, `brybnd`, `chainwoo`, `cosine`, `cragglvy`, `curly10`, `curly20`, `curly30`, `dixmaana`, `dixmaanb`, `dixmaanc`, `dixmaand`, `dixmaane`, `dixmaanf`, `dixmaang`, `dixmaanh`, `dixmaani`, `dixmaanj`, `dixmaank`, `dixmaanl`, `dixon3dq`, `dqdrtic`, `dqrtic`, `edensch`, `eg2`, `engval1`, `extrosnb`, `fletchcr`, `fletcbv2`, `fminsrf2`, `fminsurf`, `freuroth`, `genhumps`, `genrose`, `liarwhd`, `morebv`, `ncb20`, `ncb20b`, `noncvxu2`, `noncvxun`, `nondia`, `nondquar`, `penalty1`, `penalty2`, `powellsg`, `power`, `quartc`, `sbrybnd`, `schmvett`, `scosine`, `scurly10`, `scurly20`, `scurly30`, `sinquad`, `sparsine`, `sparsqur`, `spmsrtls`, `srosenbr`, `testquad`, `tointgss`, `tquartic`, `tridia`,

`vardim`, `vareigvl` and `woods`. The dimensions were selected so that $n \geq 1000$, with a default of $n = 1000$ unless otherwise recommended in the CUTEr documentation.

The MATLAB implementation of the MSS method was tested against the Steihaug-Toint method. A basic trust-region algorithm (Algorithm 5) was implemented based on ([15, Algorithm 6.1]) with some extra precautions based on ([8]) to prevent the trust-region radius from becoming too large.

Algorithm 5: Basic Trust-Region Algorithm.
Input: x_0; $M > 0$ (an integer); $\hat{\delta} \gg 0$; $\gamma_1 > 1$; $\gamma_2 \in (0,1)$; $\delta_0 > 0$ and $\eta_1, \eta_2 < 1$, such that $0 < \eta_1 < \eta_2 < 1$
Output: x
$\delta \leftarrow \delta_0$; $x \leftarrow x_0$;
$B \leftarrow I$; (implicit)
while *not* converged **do**
  Compute p to (approximately) solve (1);
  Evaluate $f(x+p)$ and $g(x+p)$;
  $\rho \leftarrow (f(x) - f(x+p))/(-g(x)^T p - \frac{1}{2} p^T B p)$;
  if $\rho \geq \eta_1$
    $x \leftarrow x + p$; $f(x) \leftarrow f(x+p)$ $g(x) \leftarrow g(x+p)$;
    if $\rho \geq \eta_2$
      $\delta \leftarrow \min\{\gamma_1 \|p\|, \hat{\delta}\}$;
    else
      $\delta \leftarrow \|p\|$;
    end
  else
    (reject the update)
    $\delta \leftarrow \gamma_2 \delta$;
  end
  Possibly update the pairs $\{(s_i, y_i)\}$, $i = 1 \ldots M$, to implicitly update B;
end

The following termination criteria was used for the basic trust-region algorithm:

$$\|g(x)\| < \max\{\tau \|f(x_0)\|, \tau \|g(x_0)\|, 1 \times 10^{-5}\}, \tag{7}$$

where $\tau \triangleq 1 \times 10^{-6}$. The trust-region algorithm terminated unsuccessful whenever the trust-region radius becomes too small or the number of function evaluations exceeded $\max\{1000, n\}$, where n is the problem size.

For these numerical experiments, the L-BFGS pairs were updated whenever the updates satisfied the following inequalities:

$$s_+^T y_+ < 1/\sqrt{\epsilon} \quad \text{and} \quad s_+^T y_+ > \sqrt{\epsilon}, \tag{8}$$

where $s_+ \triangleq p$, $y_+ \triangleq g(x+p) - g(x)$ and ϵ is machine precision in MATLAB (i.e., `eps`). The update conditions (8) were tested each iteration, regardless of whether the update to x was accepted.

Finally, the following constants were used for Algorithm 5: $M \triangleq 5$, $\gamma_1 \triangleq 2.0$, $\gamma_2 \triangleq 0.5$, $\delta_0 \triangleq 1$, $\eta_1 \triangleq 0.01$, $\eta_2 \triangleq 0.95$, and $\hat{\delta} \triangleq 1/(100 \times \epsilon)$ where ϵ is machine precision in MATLAB (i.e., `eps`). The initial guess x_0 was the default starting point associated with each CUTEr problem.

4.1. **Termination criteria for the subproblem solvers.** The Steihaug-Toint truncated conjugate-gradient method as found in [6] was implemented in MATLAB. The maximum number of iterations per subproblem was limited to the minimum of either n, the size of each CUTEr problem, or 100. The subproblem solver was said to converge at the ith iteration when the residual of the linear solve r_i satisfied

$$\|r_i\| \leq \|g(x)\| \min\{0.1, \|g(x)\|^{0.1}\}$$

(see, e.g., [6]).

The MSS method terminates whenever either of the following conditions hold [19]:

$$\|B^{-1}g\| \leq \delta \quad \text{or} \quad |\|p\| - \delta| \leq \tau_{ms}\delta,$$

where τ_{ms} is a small non-negative constant. In the numerical experiments, $\tau_{ms} \triangleq \sqrt{\epsilon}$, where ϵ denotes MATLAB machine precision. Furthermore the value for ϵ in Algorithm 4 was also taken to be MATLAB machine precision. The maximum number of iterations per subproblem was limited to the minimum of n, the size of each CUTEr problem, or 100.

4.2. **Results.** Of the 67 CUTEr problems the basic trust-region method combined with the two subproblems solvers failed to solve the following problems: `curly10`, `curly20`, `curly30`, `dixon3dq`, `fletchcr`, `genhumps`, `genrose`, `sbrybnd`, `scosine`, `scurly10`, `scurly20`, and `scurly30`. The problems `fletcbv2` and `penalty2` satisfied the convergence criteria (7) at the initial point x_0 and thus, were removed from the test set. Tables I and II contain the results of the remaining 53 problems.

Each table reports the total time spent in each subproblem solver for each problem, the total number of function evaluations required using each subproblem solver, and the total number of iterations performed by each subproblem solver required to obtain a solution to the unconstrained problem. On the problems `ncb20` and `tquartic`, the Steihaug-Toint method was unable to converge; this is denoted in the table with asterisks. On the problem `sinquad`, the basic-trust region algorithm terminated early using the Steihaug-Toint method due to the trust-region radius becoming too small (3.78×10^{-14}); this is denoted in the table with a dagger. It should be noted that it takes one matrix-vector product per inner iteration for Steihaug's method; for brevity's sake, only the number of inner iterations are reported. It is also worth noting that neither the MSS method nor the Steihaug-Toint method reached the limit on the maximum number of iterations per subproblem; thus, each subproblem was solved before reaching the maximum number of inner iterations.

TABLE 1. MSS method and Steihaug-Toint method on CUTEr problems A–E.

Problem		MSS Method			Steihaug-Toint Method		
name	n	Time	FEs	Inner Itns	Time	FEs	Inner Itns
ARWHEAD	5000	1.11e-01	15	29	1.93e-02	15	24
BDQRTIC	5000	3.12e-01	40	69	1.06e-01	40	122
BROYDN7D	5000	5.98e+00	1566	1014	7.67e+00	1618	8859
BRYBND	5000	2.56e-01	59	52	5.56e-02	24	66
CHAINWOO	4000	4.18e-01	54	99	1.33e-01	46	167
COSINE	10000	8.45e-03	14	14	1.02e-02	14	13
CRAGGLVY	5000	2.78e-01	36	64	1.04e-01	39	122
DIXMAANA	3000	2.32e-02	13	14	1.12e-02	13	17
DIXMAANB	3000	2.25e-02	13	14	7.47e-03	13	12
DIXMAANC	3000	2.32e-02	14	14	8.02e-03	14	13
DIXMAAND	3000	2.54e-02	15	14	8.61e-03	15	14
DIXMAANE	3000	1.76e-01	54	46	1.43e-01	52	213
DIXMAANF	3000	4.61e-02	24	18	3.66e-02	24	55
DIXMAANG	3000	2.97e-02	21	14	2.12e-02	21	33
DIXMAANH	3000	2.96e-02	20	14	1.81e-02	20	27
DIXMAANI	3000	4.99e-01	84	129	2.27e-01	75	340
DIXMAANJ	3000	9.93e-02	28	30	4.82e-02	27	72
DIXMAANK	3000	7.77e-02	24	27	3.70e-02	25	56
DIXMAANL	3000	4.47e-02	22	18	1.97e-02	21	31
DQDRTIC	5000	5.36e-02	11	20	7.12e-03	11	10
DQRTIC	5000	6.54e-03	29	56	1.76e-02	29	28
EDENSCH	2000	3.86e-02	22	23	1.74e-02	24	39
EG2	1000	8.52e-04	5	2	9.72e-04	5	4
ENGVAL1	5000	4.54e-02	17	17	1.63e-02	17	21
EXTROSNB	1000	1.13e-01	39	67	5.78e-02	57	162

Comparing the number of function evaluations and inner iterations between Tables I and II, it appears that the more difficult problems occur in Table II. While the MSS method often took slightly more time than the Steihaug-Toint method, it often required fewer function evaluations on these problems. The results of Tables I and II are summarized in Table III. (The three problems on which the Steihaug-Toint method did not converge were removed when generating Table III.) The results suggest that solving the trust-region subproblem more accurately using the MSS method leads to fewer overall function evaluations for the overall trust-region method. This is consistent with other subproblem solvers that solve trust-region subproblems to high accuracy [11, 10]. Table III also suggests that while the MSS method took more time overall, it solved the subproblems within a comparable time frame.

TABLE 2. MSS method and Steihaug-Toint method on CUTEr problems F–W.

Problem		MSS Method			Steihaug-Toint Method		
name	n	Time	FEs	Inner Itns	Time	FEs	Inner Itns
FMINSRF2	5625	1.91e+00	569	268	3.26e+00	841	3513
FMINSURF	1024	4.88e-01	241	210	7.68e-01	460	2182
FREUROTH	5000	2.28e-01	36	61	6.68e-02	39	80
LIARWHD	5000	1.95e-01	30	50	1.01e-01	69	116
MOREBV	5000	1.70e-01	261	4	4.45e-01	262	703
NCB20	1010	2.64e+00	772	1251	*	*	*
NCB20B	2000	9.42e-02	49	38	1.34e-01	69	307
NONCVXU2	5000	1.94e-01	19	53	1.45e-02	19	18
NONCVXUN	5000	2.20e-01	19	55	1.60e-02	19	19
NONDIA	5000	1.34e-03	4	2	2.09e-03	4	3
NONDQUAR	5000	3.93e-01	47	85	1.41e-01	49	167
PENALTY1	1000	2.89e-03	25	48	4.83e-03	25	24
POWELLSG	5000	9.90e-02	29	28	5.33e-02	25	65
POWER	1000	8.49e-03	37	56	1.15e-02	37	45
QUARTC	5000	6.38e-03	29	56	9.98e-03	29	28
SCHMVETT	5000	2.17e-01	46	48	6.68e-02	29	80
SINQUAD	5000	2.73e-01	36	88	3.23e-01†	171†	376†
SPARSINE	5000	6.72e-01	115	131	5.30e-01	126	620
SPARSQUR	10000	4.77e-02	17	14	3.86e-02	18	30
SPMSRTLS	4999	5.37e-01	114	107	5.05e-01	128	589
SROSENBR	5000	1.51e-01	22	43	3.59e-02	27	41
TESTQUAD	5000	2.77e-01	72	53	3.27e-01	78	387
TOINTGSS	5000	5.06e-02	11	20	8.98e-03	12	12
TQUARTIC	5000	2.34e+00	47	370	*	*	*
TRIDIA	5000	2.19e+00	263	426	1.35e+00	291	1507
VARDIM	200	1.13e-03	12	22	2.01e-03	12	11
VAREIGVL	1000	1.18e-01	31	70	2.65e-02	25	77
WOODS	4000	1.25e-01	22	37	3.17e-02	22	39

TABLE 3. Summary of results on CUTEr problems.

	MSS method	Steihaug-Toint method
Problems solved	53	50
Function evals	4359	4974
Total time (subproblem)	1.71e+01	1.68e+01

The results of Tables I and II are also summarized using a performance profile (see Dolan and Moré [9]). Let card($\mathcal{S}$) denote the number of elements in a finite set $\mathcal{S}$. Let $\mathcal{P}$ denote the set of problems used for a given numerical experiment. For each method s we define the function $\pi_s : [0, r_M] \mapsto \Re^+$ such that

$$\pi_s(\tau) = \frac{1}{\text{card}(\mathcal{P})} \text{card}(\{p \in \mathcal{P} : \log_2(r_{p,s}) \leq \tau\}),$$

where $r_{p,s}$ denotes the ratio of the number of function evaluations needed to solve problem p with method s and the least number of function evaluations needed to solve problem p. The number r_M is the maximum value of $\log_2(r_{p,s})$. Figure 1 depicts the functions π_s for each of the methods tested. All performance profiles are plotted on a $\log_2$-scale. Based on the performance profile, the MSS method appears more competitive in terms of function evaluations than the Steihaug-Toint method for the L-BFGS application.

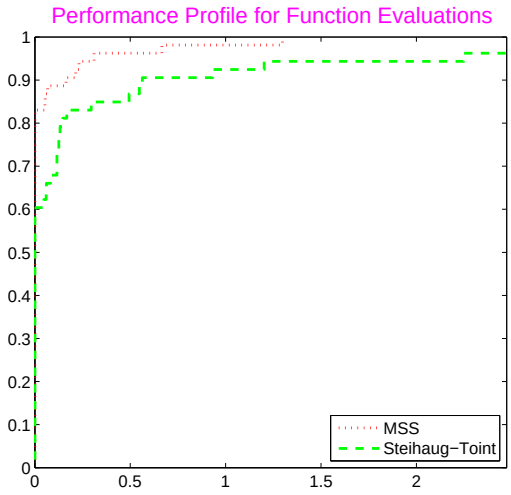

FIGURE 1. Function evaluations for the MSS and Steihaug-Toint methods

5. Concluding Remarks

In this paper, we have presented a MATLAB implementation of the MSS algorithm that is able to solve problems of the form (1). This solver is stable and numerical results confirm that the method can compute solutions to any prescribed accuracy. Future research directions include practical trust-region schemes that include less stringent requirements on accepting updates to the L-BFGS matrix and improvements for cases when L-BFGS pairs become linearly dependent, i.e., either the vectors $\{s_i\}$ or the vectors $\{y_i\}$, $i = 1 \dots M$, become linearly dependent.

References

[1] M. S. Apostolopoulou, D. G. Sotiropoulos, C. A. Botsaris, and P. E. Pintelas. A practical method for solving large-scale TRS. *Optimization Letters*, 5:207–227, 2011.

[2] M. S. Apostolopoulou, D. G. Sotiropoulos, and P. Pintelas. Solving the quadratic trust-region subproblem in a low-memory BFGS framework. *Optimization Methods Software*, 23(5):651–674, Oct. 2008.

[3] I. Bongartz, A. R. Conn, N. I. M. Gould, and P. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Trans. Math. Software*, 21(1):123–160, 1995.

[4] J. V. Burke, A. Wiegmann, and L. Xu. Limited memory BFGS updating in a trust-region framework. Technical report, University of Washington, 1996.

[5] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited-memory methods. *Math. Program.*, 63:129–156, 1994.

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

[7] J. E. Dennis, Jr. and 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.

[8] J. E. Dennis Jr. and H. H. Mei. Two new unconstrained optimization algorithms which use function and gradient values. *J. Optim. Theory Appl.*, 28:453–482, 1979.

[9] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002.

[10] J. B. Erway and P. E. Gill. A subspace minimization method for the trust-region step. *SIAM Journal on Optimization*, 20(3):1439–1461, 2009.

[11] J. B. Erway, P. E. Gill, and J. D. Griffin. Iterative methods for finding a trust-region step. *SIAM J. Optim.*, 20(2):1110–1131, 2009.

[12] J. B. Erway, V. Jain, and R. F. Marcia. Shifted L-BFGS systems. Technical Report 2012-6, Wake Forest University, 2012.

[13] J. B. Erway and R. F. Marcia. Limited-memory BFGS systems with diagonal updates. *Linear Algebra and its Applications*, 437(1):333 – 344, 2012.

[14] D. M. Gay. Computing optimal locally constrained steps. *SIAM J. Sci. Statist. Comput.*, 2(2):186–197, 1981.

[15] N. I. M. Gould, S. Lucidi, M. Roma, and P. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM J. Optim.*, 9(2):504–525, 1999.

[16] N. I. M. Gould, D. Orban, and P. L. Toint. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Software*, 29(4):373–394, 2003.

[17] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45:503–528, 1989.

[18] Z. Lu and R. D. C. Monteiro. A modified nearly exact method for solving low-rank trust region subproblem. *Math. Program.*, 109(2):385–411, Jan. 2007.

[19] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. and Statist. Comput.*, 4:553–572, 1983.

[20] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, 35:773–782, 1980.

[21] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, second edition, 2006.

[22] M. J. D. Powell. A fortran subroutine for solving systems of nonlinear algebraic equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*. Gordon and Breach, 1970.

[23] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114. Gordon and Breach, 1970.

[24] D. C. Sorensen. Newton's method with a model trust region modification. *SIAM J. Numer. Anal.*, 19(2):409–426, 1982.

[25] Y.-X. Yuan. On the truncated conjugate gradient method. *Math. Program.*, 87(3, Ser. A):561–573, 2000.

E-mail address: erwayjb@wfu.edu

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

E-mail address: rmarcia@ucmerced.edu

Appied Mathematics, University of California, Merced, Merced, CA 95343