Algorithm 1030: SC-SR1: MATLAB Software for Limited-memory SR1 Trust-region Methods

JOHANNES BRUST, Argonne National Laboratory
OLEG BURDAKOV, Linköping University
JENNIFER ERWAY, Wake Forest University
ROUMMEL MARCIA, University of California, Merced

We present a MATLAB implementation of the symmetric rank-one (SC-SR1) method that solves trust-region subproblems when a limited-memory symmetric rank-one (L-SR1) matrix is used in place of the true Hessian matrix, which can be used for large-scale optimization. The method takes advantage of two shape-changing norms [Burdakov et al. 2017; Burdakov and Yuan 2002] to decompose the trust-region subproblem into two separate problems. Using one of the proposed norms, the resulting subproblems have closed-form solutions. Meanwhile, using the other proposed norm, one of the resulting subproblems has a closed-form solution while the other is easily solvable using techniques that exploit the structure of L-SR1 matrices. Numerical results suggest that the SC-SR1 method is able to solve trust-region subproblems to high accuracy even in the so-called "hard case." When integrated into a trust-region algorithm, extensive numerical experiments suggest that the proposed algorithms perform well, when compared with widely used solvers, such as truncated conjugate-gradients.

CCS Concepts: • **Mathematics of computing → Solvers;**

Additional Key Words and Phrases: Large-scale unconstrained optimization, trust-region methods, limited-memory quasi-Newton methods, symmetric rank-one update, shape-changing norm

This research is support in part by National Science Foundation grants CMMI-1333326, CMMI-1334042, IIS-1741264, and IIS-1741490 and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) under Contract DE-AC02-06CH11347.

Authors' addresses: J. Brust, Department of Mathematics, University of California, San Diego, 9500 Gilman Dr., La Jolla, California 92093 (formerly at Argonne National Laboratory); email: jjbrust@ucsd.edu; O. Burdakov, Division of Optimization, Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden; email: oleg.burdakov@liu.se; J. Erway, Department of Mathematics, Wake Forest University, P. O. Box 7388, Winston-Salem, North Carolina 27109; email: erwayjb@wfu.edu; R. Marcia, Applied Mathematics, University of California, Merced, 5200 N. Lake Road, Merced, California 95343; email: rmarcia@ucmerced.edu.

0098-3500/2022/12-ART48 $15.00
https://doi.org/10.1145/3550269

ACM Reference format:
Johannes Brust, Oleg Burdakov, Jennifer Erway, and Roummel Marcia. 2022. Algorithm 1030: SC-SR1: MATLAB Software for Limited-memory SR1 Trust-region Methods. *ACM Trans. Math. Softw.* 48, 4, Article 48 (December 2022), 33 pages.
https://doi.org/10.1145/3550269

1 INTRODUCTION

At each iteration of a trust-region method for minimizing a general nonconvex function $f(\mathbf{x})$, the so-called *trust-region subproblem* must be solved to obtain a step direction:

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

where $\mathbf{g} \triangleq \nabla f(\mathbf{x}_k)$, $\mathbf{B}$ is an approximation to $\nabla^2 f(\mathbf{x}_k)$, δ is a positive constant, and $\|\cdot\|$ is a given norm. In this article, we describe a MATLAB implementation for solving the trust-region subproblem (1) when $\mathbf{B}$ is a **limited-memory symmetric rank-one (L-SR1)** matrix approximation of $\nabla^2 f(\mathbf{x}_k)$. In large-scale optimization, solving (1) represents the bulk of the computational effort in trust-region methods (besides function and derivative evaluations). The norm used in Equation (1) not only defines the trust region shape but also determines the difficulty of solving each subproblem.

The most widely used norm chosen to define the trust-region subproblem is the two-norm. One reason for this choice of norm is that the necessary and sufficient conditions for a global solution to the subproblem defined by the two-norm are well known [Gay 1981; Moré and Sorensen 1983; Sorensen 1982]; many methods exploit these conditions to compute high-accuracy solutions to the trust-region subproblem (see, e.g., Erway and Gill [2010], Erway et al. [2009], Erway and Marcia [2014], Gould et al. [2010], Brust et al. [2017], and Moré and Sorensen [1983]). The infinity-norm is sometimes used to define the subproblem; however, when $\mathbf{B}$ is indefinite, as can be the case when $\mathbf{B}$ is a L-SR1 matrix, the subproblem is NP-hard [Murty and Kabadi 1987; Vavasis 1992]. For more discussion on norms other than the infinity-norm we refer the reader to Conn et al. [2000].

In this article, we consider the trust-region subproblems defined by *shape-changing* norms originally proposed in Burdakov and Yuan [2002]. Generally speaking, shape-changing norms are norms that depend on $\mathbf{B}$; thus, in the quasi-Newton setting where the quasi-Newton matrix $\mathbf{B}$ is updated each iteration, the shape of the trust region changes each iteration. One of the earliest references to shape-changing norms is found in Goldfarb [1980] where a norm is implicitly defined by the product of a permutation matrix and a unit lower triangular matrix that arise from a symmetric indefinite factorization of $\mathbf{B}$. Perhaps the most widely-used shape-changing norm is the "elliptic norm" given by $\|\mathbf{x}\|_A \triangleq \mathbf{x}^T\mathbf{A}\mathbf{x}$, where $\mathbf{A}$ is a positive-definite matrix (see, e.g., Conn et al. [2000]). A well-known use of this norm is found in the Steihaug method [Steihaug 1983] and, more generally, truncated preconditioned **conjugate-gradients (CG)** [Conn et al. 2000]; these methods reformulate a two-norm trust-region subproblem using an elliptic norm to maintain the property that the iterates from preconditioned CG are increasing in norm. Other examples of shape-changing norms include those defined by vectors in the span of $\mathbf{B}$ (see, e.g., Conn et al. [2000]).

The shape-changing norms proposed in Burdakov and Yuan [2002] and Burdakov et al. [2017] have the advantage to allow one to break the trust-region subproblem into two separate subproblems. Using one of the proposed shape-changing norms, the solution of the subproblem then has a closed-form solution. In the other proposed norm, one of the subproblems has a closed-form solution while the other is easily solvable. The publicly available LMTR codes [Burdakov et al. 2018] solve trust-region subproblems (1) defined by these shape-changing norms and the **limited-memory broyden-fletcher-goldfarb-shanno (L-BFGS)** updates of $\mathbf{B}$. To our

knowledge, there are no other implementations for solving trust-region subproblems defined by these shape-changing norms.

1.1 Overview of the Proposed Method

In this article, we develop a MATLAB implementation for solving **trust-region (TR)** subproblems defined by the two shape-changing norms described in Burdakov and Yuan [2002] when L-SR1 approximations to the Hessian are used instead of L-BFGS approximations. For limited-memory algorithms a re-scaling strategy (i.e., effectively re-initializing the Hessian approximation at each iteration) is often important for the practical performance of the method. Yet, because the structure of L-SR1 matrices can be exploited to reduce the memory usage even further when a constant initialization is used (i.e., no re-scaling), we provide an option to chose between such strategies. Moreover, our implementation enables the testing and addition of new solvers by swapping out the respective TR subproblem algorithm. In this way, we conduct numerical experiments on large-scale CUTEst problems [Gould et al. 2003], comparing the shape-changing methods to truncated CG and an ℓ_2-norm based algorithm. The proposed method, called the **shape-changing SR1 method (SC-SR1)**, enables high-accuracy subproblem solutions by exploiting the structure of L-SR1 matrices.

This article is organized as follows: In Section 2, we review L-SR1 matrices, including the compact representation for these matrices and a method to efficiently compute their eigenvalues and a partial eigenbasis. In Section 3, we demonstrate how the shape-changing norms decouple the original trust-region subproblem into two problems and describe the proposed solver for each subproblem. Finally, for each shape-changing norm, we show how to construct a global solution to Equation (1) from the solutions of the two decoupled subproblems. Optimality conditions are presented for each of these decoupled subproblems in Section 4. In Section 5, we demonstrate the accuracy of the proposed solvers and compare them on a collection of large-scale optimization problems. Concluding remarks can be found in Section 6.

1.2 Notation

In this article, the identity matrix of dimension d is denoted by $\mathbf{I}_d = [\mathbf{e}_1|\cdots|\mathbf{e}_d]$, and depending on the context the subscript d may be suppressed. Integer $k \geq 0$ represents the iteration index. Bold uppercase symbols represent matrices while lowercase symbols represent vectors. Lowercase Greek letters represent scalars.

2 L-SR1 MATRICES

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth objective function and $\{\mathbf{x}_i\}$, $i = 0, \ldots k$, is a sequence of iterates, then the **symmetric rank-one (SR1)** matrix is defined using pairs $(\mathbf{s}_i, \mathbf{y}_i)$, where

$$\mathbf{s}_i \triangleq \mathbf{x}_{i+1} - \mathbf{x}_i \quad \text{and} \quad \mathbf{y}_i \triangleq \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i),$$

and ∇f denotes the gradient of f. Specifically, given an initial matrix $\mathbf{B}_0$, $\mathbf{B}_{k+1}$ is defined recursively as

$$\mathbf{B}_{k+1} \triangleq \mathbf{B}_k + \frac{(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)^T}{(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)^T\mathbf{s}_k}, \tag{2}$$

provided $(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)^T\mathbf{s}_k \neq 0$. In practice, $\mathbf{B}_0 = \mathbf{B}_0^{(k)}$ is often taken to be a scalar multiple of the identity matrix that re-scales $\mathbf{B}_k$ each iteration; for the duration of this article we assume that $\mathbf{B}_0 = \gamma_k\mathbf{I}$, $\gamma_k \in \mathbb{R}$. L-SR1matrices store and make use of only the m most-recently computed pairs $\{(\mathbf{s}_i, \mathbf{y}_i)\}$, where $m \ll n$ (for example, Byrd et al. [1994] suggest $m \in [3, 7]$). For simplicity of notation, let us consider the case when $k \geq m + 1$ (so that enough pairs have been stored to perform m updates) and when the update (2) is well defined for all k (i.e., no division by zero).

Otherwise, all of our discussion is still accurate, though the number of pairs involved in computing the approximation would be less than m.

The SR1 update is a member of the Broyden class of updates (see, e.g., Nocedal and Wright [2006]). Similarly to widely used updates such as the **broyden-fletcher-goldfarb-shanno (BFGS)** and the **davidon-fletcher-powell (DFP)** updates, this update can yield indefinite matrices. In practice it is common to impose the curvature condition on BFGS and DFP updates, ensuring positive definite matrices [Nocedal and Wright 2006], even though this is not absolutely necessary in a trust-region algorithm. For SR1 matrices there does not appear to exist a similar condition that guarantees positive definiteness, which means that the updates are typically defined so that they can incorporate negative curvature information without restrictions. In fact, the SR1 update has convergence properties superior to other widely used positive-definite quasi-Newton matrices such as BFGS; in particular, Conn et al. [1991] give conditions under which the SR1 update formula generates a sequence of matrices that converge to the true Hessian. (For more background on the SR1 update formula, see, e.g., Griva et al. [2009], Kelley and Sachs [1998], Khalfan et al. [1993], Nocedal and Wright [2006], Sun and Yuan [2006], and Wolkowicz [1994].)

2.1 Compact Representation

The compact representation of SR1 matrices can be used to compute the eigenvalues and a partial eigenbasis of these matrices. In this section, we review the compact formulation of SR1 matrices.

To begin, we define the following matrices:

$$\begin{aligned} \mathbf{S}_k &\triangleq [\, \mathbf{s}_0 \;\; \mathbf{s}_1 \;\; \mathbf{s}_2 \;\; \cdots \;\; \mathbf{s}_{k-1} \,] \in \mathbb{R}^{n\times k}, \\ \mathbf{Y}_k &\triangleq [\, \mathbf{y}_0 \;\; \mathbf{y}_1 \;\; \mathbf{y}_2 \;\; \cdots \;\; \mathbf{y}_{k-1} \,] \in \mathbb{R}^{n\times k}. \end{aligned}$$

The matrix $\mathbf{S}_k^T\mathbf{Y}_k \in \mathbb{R}^{k\times k}$ can be written as the sum of the following three matrices:

$$\mathbf{S}_k^T\mathbf{Y}_k = \mathbf{L}_k + \mathbf{D}_k + \mathbf{R}_k,$$

where $\mathbf{L}_k$ is strictly lower triangular, $\mathbf{D}_k$ is diagonal, and $\mathbf{R}_k$ is strictly upper triangular. Then, $\mathbf{B}_k$ can be written as

$$\mathbf{B}_k = \gamma_k\mathbf{I} + \mathbf{\Psi}_k\mathbf{M}_k\mathbf{\Psi}_k^T, \tag{3}$$

where $\mathbf{\Psi}_k \in \mathbb{R}^{n\times k}$ and $\mathbf{M}_k \in \mathbb{R}^{k\times k}$. In particular, $\mathbf{\Psi}_k$ and $\mathbf{M}_k$ are given by

$$\mathbf{\Psi}_k = \mathbf{Y}_k - \gamma_k\mathbf{S}_k \quad \text{and} \quad \mathbf{M}_k = \left(\mathbf{D}_k + \mathbf{L}_k + \mathbf{L}_k^T - \gamma_k\mathbf{S}_k^T\mathbf{S}_k\right)^{-1}. \tag{4}$$

The right side of Equation (3) is the *compact representation* of $\mathbf{B}_k$; this representation is due to Byrd et al. [1994, Theorem 5.1]. Since we assume that updates are made when the SR1 matrix $\mathbf{B}_k$ is well defined, $\mathbf{M}_k$ exists [Byrd et al. 1994, Theorem 5.1]. For notational simplicity, we assume $\mathbf{\Psi}_k$ has full column rank; when $\mathbf{\Psi}_k$ does not have full column rank, we refer to Burdakov et al. [2017] for the modifications needed for computing the eigenvalues, which we also review in Section 2.2. Notice that the computation of $\mathbf{M}_k$ is computationally admissible, since it is a very small symmetric square matrix.

2.2 Limited-Memory Updating

For large optimization problems, limited-memory approaches store only a small number of vectors to define the L-SR1 representations. Depending on the initialization strategy, specifically whether γ_k varies between iterations or is constant ($\gamma_k = \bar{\gamma}$) the matrices in Reference (4) can be effectively stored and updated. We will describe these techniques in subsequent sections. By setting the parameter $m \ll n$ limited-memory techniques enable inexpensive computations and replace or insert one column at each iteration in $\mathbf{Y}_k$ and $\mathbf{S}_k$. Let an underline below a matrix represent the

matrix with its first column removed. That is, $\underline{S}_k$ represents S_k without its first column. With this notation, a column update of a matrix, say, S_k, by a vector $\mathbf{s}_k$ is defined as follows:

$$\text{colUpdate}\,(S_k, \mathbf{s}_k) \triangleq \begin{cases} [\, S_k \; \mathbf{s}_k \,] & \text{if } k < m \\ [\, \underline{S}_k \; \mathbf{s}_k \,] & \text{if } k \geq m \end{cases}.$$

This column update can be implemented efficiently, without copying large amounts of memory, by appropriately updating a vector that stores index information ("mIdx"). A function to do so is described in Procedure 1:

PROCEDURE 1: Limited-memory column updating of S_k by the vector $\mathbf{s}_k$

Function: [S_k,mIdx]=`colUpdate`(S_k, $\mathbf{s}_k$, mIdx, m, k);

1: **if** $k = 0$ **then**
2: mIdx $\leftarrow$ zeros(m, 1);
3: **end if**
4: **if** $k < m$ **then**
5: mIdx($k+1$) $\leftarrow$ $k+1$;
6: $S_k(:, \text{mIdx}(k+1)) \leftarrow \mathbf{s}_k$;
7: **else if** $m \leq k$ **then**
8: $k_m \leftarrow$ mIdx(1);
9: mIdx($1 : (m-1)$) $\leftarrow$ mIdx($2 : m$);
10: mIdx(m) $\leftarrow k_m$;
11: $S_k(:, \text{mIdx}(m)) \leftarrow \mathbf{s}_k$;
12: **end if**
13: **return** S_k, mIdx;

Note that this procedure does not copy (or overwrite) large blocks of memory as would commands such as $\{S_k(:, 1 : (m-1)) \leftarrow S_k(:, 2 : m); S_k(:, m) \leftarrow \mathbf{s}_k\}$ but instead accesses the relevant locations using a stored vector of indices. Certain matrix products can also be efficiently updated. As such, the product $S_k^T Y_k$ does not have to be re-computed from scratch. To describe the matrix product updating mechanism, let an overline above a matrix represent the matrix with its first row removed. That is, $\overline{S_k^T Y_k}$ represents $S_k^T Y_k$ without its first row. With this notation, a product update of, say, $S_k^T Y_k$, by matrices S_k, Y_k and vectors $\mathbf{s}_k$, $\mathbf{y}_k$ is defined as

$$\text{prodUpdate}\left(S_k^T Y_k, S_k, Y_k, \mathbf{s}_k, \mathbf{y}_k\right) \triangleq \begin{cases} \begin{bmatrix} S_k^T Y_k & S_k^T \mathbf{y}_k \\ \mathbf{s}_k^T Y_k & \mathbf{s}_k^T \mathbf{y}_k \end{bmatrix} & \text{if } k < m \\ \begin{bmatrix} \left(\overline{S_k^T Y_k}\right) & \underline{S}_k^T \mathbf{y}_k \\ \mathbf{s}_k^T \underline{Y}_k & \mathbf{s}_k^T \mathbf{y}_k \end{bmatrix} & \text{if } k \geq m \end{cases}.$$

This product update can be implemented without recomputing potentially large multiplications by storing previous products and information about the column order in S_k and Y_k. In particular, updating the matrix product is based on storing $S_k^T Y_k$, S_k, Y_k and the vector "mIdx." Although a different order is possible, we apply the product update after column updates of S_k, Y_k have been done previously. In such a situation the vector, which stores the appropriate index information ("mIdx") is defined at such a point.

Note that such a product update is computationally much more efficient than recomputing the product from scratch. Specifically, when $m \leq k$, the direct product $S_k^T Y_k$ is done at $O(m^2 n)$ multiplications. However, Procedure 2 does this update with $O(2mn)$ multiplications in lines 6 and 7, by reusing previous values from line 5. Moreover, when the product is symmetric, e.g., Procedure 2 is

PROCEDURE 2: Limited-memory product update $S_k^T Y_k$ ($\mathbf{s}_k$, $\mathbf{y}_k$ are column updates to S_k, Y_k)

Function: $[S_k^T Y_k]$=prodUpdate($S_k^T Y_k$, S_k, Y_k, $\mathbf{s}_k$, $\mathbf{y}_k$, mIdx, m, k);
1: **if** $k < m$ **then**
2: $S_k^T Y_k(1:(k+1), k+1) \leftarrow S_k(:, \text{mIdx}(1:(k+1)))^T \mathbf{y}_k$;
3: $S_k^T Y_k(k+1, 1:k) \leftarrow \mathbf{s}_k^T Y_k(:, \text{mIdx}(1:k))$;
4: **else if** $m \leq k$ **then**
5: $S_k^T Y_k(1:(m-1), 1:(m-1)) \leftarrow S_k^T Y_k(2:m, 2:m)$;
6: $S_k^T Y_k(1:m, m) \leftarrow S_k(:, \text{mIdx}(1:m))^T \mathbf{y}_k$;
7: $S_k^T Y_k(m, 1:(m-1)) \leftarrow \mathbf{s}_k^T Y_k(:, \text{mIdx}(1:(m-1)))$;
8: **end if**
9: **return** $S_k^T Y_k$;

invoked by prodUpdate($S_k^T S_k$, S_k, S_k, $\mathbf{s}_k$, $\mathbf{s}_k$, mIdx, m, k), then $S_k(:, \text{mIdx}(1:m))^T \mathbf{s}_k$ can be stored in line 6 and reused in line 7 (thus only one matrix-vector product is needed, instead of two). Since limited-memory updating of the L-SR1 matrices varies for the chosen initialization strategy, we describe the cases of non-constant initializations γ_k and constant $\gamma_k = \bar{\gamma}$ next.

2.2.1 Limited-memory Updating of Equation (4) Using Non-constant γ_k. When γ_k varies for every iteration, Ψ_k is best implicitly represented by storing S_k and Y_k instead of explicitly forming it (forming Ψ_k explicitly incurs additional $O(mn)$ memory locations in $\Psi_k = Y_k - \gamma_k S_k$). By storing the previous m pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m}^{k-1}$ in the limited-memory matrices $S_k = [\, \mathbf{s}_{k-m} \; \cdots \; \mathbf{s}_{k-1} \,] \in \mathbb{R}^{n \times m}$ and $Y_k = [\, \mathbf{y}_{k-m} \; \cdots \; \mathbf{y}_{k-1} \,] \in \mathbb{R}^{n \times m}$ the matrix-vector product $\Psi_k^T \mathbf{g}$ (for a vector $\mathbf{g}$) is done as

$$\Psi_k^T \mathbf{g} = Y_k^T \mathbf{g} - \gamma_k \left(S_k^T \mathbf{g} \right).$$

2.2.2 Limited-memory Updating of Equation (4) Using Constant $\gamma_k = \bar{\gamma}$. When $\gamma_k = \bar{\gamma}$ is constant, then Y_k and S_k do not have to be stored separately. Instead the limited-memory method stores m previous vectors $\{\boldsymbol{\psi}_i = \mathbf{y}_i - \bar{\gamma} \mathbf{s}_i\}_{i=k-m}^{k-1}$, concatenated in the matrix

$$\Psi_k = \left[\, \boldsymbol{\psi}_{k-m} \; \cdots \; \boldsymbol{\psi}_{k-1} \, \right] \in \mathbb{R}^{n \times m}.$$

Matrix vector products are directly computed as $\Psi_k^T \mathbf{g}_k$. Subsequently, $\mathbf{M}_k$ from Equation (4) can be updated efficiently by noting that

$$\mathbf{M}_k^{-1} \mathbf{e}_k = \left(\mathbf{D}_k + \mathbf{L}_k + \mathbf{L}_k^T - \bar{\gamma} S_k^T S_k \right) \mathbf{e}_k = \Psi_k^T \mathbf{s}_k.$$

Because of these simplifications an L-SR1 algorithm with constant initialization strategy can be implemented with about half the memory footprint (storing only Ψ_k as opposed to Y_k, S_k (and previous small products)). However, often the ability to rescale the computations via a non-constant γ_k parameter can be advantageous in solving large-scale optimization problems. We provide an option to choose between constant or non-constant initialization strategies in our implementations.

2.3 Eigenvalues

In this subsection, we demonstrate how the eigenvalues and a partial eigenbasis can be computed for SR1 matrices. In general, this derivation can be done for any limited-memory quasi-Newton matrix that admits a compact representation; in particular, it can be done for any member of the Broyden convex class [Brust 2018; Brust et al. 2019; Erway and Marcia 2015]. This discussion is based on Burdakov et al. [2017].

Consider the problem of computing the eigenvalues of $\mathbf{B}_k$, which is assumed to be an L-SR1 matrix, obtained from performing m rank-one updates to $\mathbf{B}_0 = \gamma\mathbf{I}$. For notational simplicity, we drop subscripts and consider the compact representation of $\mathbf{B}$:

$$\mathbf{B} = \gamma\mathbf{I} + \mathbf{\Psi}\mathbf{M}\mathbf{\Psi}^T. \tag{5}$$

The "thin" QR factorization of $\mathbf{\Psi}$ can be written as $\mathbf{\Psi} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q} \in \mathbb{R}^{n\times m}$ and $\mathbf{R} \in \mathbb{R}^{m\times m}$ is invertible, because, as it was assumed above, $\mathbf{\Psi}$ has full column rank. Then

$$\mathbf{B} = \gamma\mathbf{I} + \mathbf{Q}\mathbf{R}\mathbf{M}\mathbf{R}^T\mathbf{Q}^T. \tag{6}$$

The matrix $\mathbf{R}\mathbf{M}\mathbf{R}^T \in \mathbb{R}^{m\times m}$ is of a relatively small size, and thus, it is computationally inexpensive to compute its spectral decomposition. We define the spectral decomposition of $\mathbf{R}\mathbf{M}\mathbf{R}^T$ as $\mathbf{U}\hat{\Lambda}\mathbf{U}^T$, where $\mathbf{U} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix whose columns are made up of eigenvectors of $\mathbf{R}\mathbf{M}\mathbf{R}^T$ and $\hat{\Lambda} = \text{diag}(\hat{\lambda}_1, \ldots, \hat{\lambda}_m)$ is a diagonal matrix whose entries are the associated eigenvalues.

Thus,

$$\mathbf{B} = \gamma\mathbf{I} + \mathbf{Q}\mathbf{U}\hat{\Lambda}\mathbf{U}^T\mathbf{Q}^T. \tag{7}$$

Since both $\mathbf{Q}$ and $\mathbf{U}$ have orthonormal columns, $\mathbf{P}_{\|} \triangleq \mathbf{Q}\mathbf{U} \in \mathbb{R}^{n\times m}$ also has orthonormal columns. Let $\mathbf{P}_{\perp}$ denote the matrix whose columns form an orthonormal basis for $(\mathbf{P}_{\|})^{\perp}$. Thus, the spectral decomposition of $\mathbf{B}$ is defined as $\mathbf{B} = \mathbf{P}\Lambda_\gamma\mathbf{P}^T$, where

$$\mathbf{P} \triangleq \begin{bmatrix}\mathbf{P}_{\|} & \mathbf{P}_{\perp}\end{bmatrix} \quad \text{and} \quad \Lambda_\gamma \triangleq \begin{bmatrix}\Lambda & 0\\ 0 & \gamma\mathbf{I}_{n-m}\end{bmatrix}, \tag{8}$$

with $\Lambda_\gamma = \text{diag}(\lambda_1, \ldots, \lambda_n)$ and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_m) = \hat{\Lambda} + \gamma\mathbf{I} \in \mathbb{R}^{m\times m}$.

We emphasize three important properties of the eigendecomposition. First, all eigenvalues of $\mathbf{B}$ are explicitly obtained and represented by Λ_γ. Second, if desired, then one need only compute the m eigenvectors of $\mathbf{B}$ represented by $\mathbf{P}_{\|}$. In particular, since $\mathbf{\Psi} = \mathbf{Q}\mathbf{R}$, then

$$\mathbf{P}_{\|} = \mathbf{Q}\mathbf{U} = \mathbf{\Psi}\mathbf{R}^{-1}\mathbf{U}. \tag{9}$$

If $\mathbf{P}_{\|}$ needs to only be available to compute matrix-vector products, then one can avoid explicitly forming $\mathbf{P}_{\|}$ by storing $\mathbf{\Psi}$, $\mathbf{R}$, and $\mathbf{U}$. Third, the eigenvalues given by the parameter γ can be interpreted as an estimate of the curvature of f in the space spanned by the columns of $\mathbf{P}_{\perp}$.

While there is no reason to assume the function f has negative curvature throughout the entire subspace $\mathbf{P}_{\perp}$, in this article, we consider the case $\gamma \leq 0$ for the sake of completeness.

For the duration of this article, we assume the first m eigenvalues in Λ_γ are ordered in increasing values, i.e., $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_m)$ where $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_m$ and that r is the multiplicity of λ_1, i.e., $\lambda_1 = \lambda_2 = \cdots = \lambda_r < \lambda_{r+1}$. For details on updating this partial spectral decomposition when a new quasi-Newton pair is computed, see Erway and Marcia [2015].

2.4 Implementation

In the above presentation, the QR factorization was used for ease of readability to find a partial spectral decomposition of $\mathbf{B}$. However, there are other approaches that may be better suited for different applications. An alternative approach to computing the eigenvalues of $\mathbf{B}$ is presented in Lu [1996] that replaces the QR factorization of $\mathbf{\Psi}$ with the SVD and an eigendecomposition of a $m \times m$ matrix and $t \times t$ matrix, respectively, where $t \leq m$. (For more details, see Lu [1996].) However, experiments in Brust et al. [2019] indicate that the QR version of this computation outperforms the SVD approach. When $\mathbf{\Psi}^T\mathbf{\Psi}$ is positive definite (i.e., $\mathbf{\Psi}$ is full rank), the Cholesky factorization of $\mathbf{\Psi}^T\mathbf{\Psi} = \mathbf{R}^T\mathbf{R}$ provides the same $\mathbf{R}$ needed to form $\mathbf{P}_{\|}$ in Equation (9) [Burdakov

et al. 2017]. Since $\mathbf{\Psi}$ is not explicitly formed when a non-constant initialization $\gamma = \gamma_k$ is used (in this case $\mathbf{\Psi}$ is defined by storing $\mathbf{Y}, \mathbf{S}$) the product matrix $\mathbf{\Psi}^T\mathbf{\Psi}$ is represented by

$$\mathbf{\Psi}^T\mathbf{\Psi} = \mathbf{Y}^T\mathbf{Y} - 2\gamma\mathbf{Y}^T\mathbf{S} + \gamma^2\mathbf{S}^T\mathbf{S} \tag{10}$$

(in Equation (10) the matrices $\mathbf{Y}^T\mathbf{Y}, \mathbf{Y}^T\mathbf{S}$ and $\mathbf{S}^T\mathbf{S}$ are stored and updated). In contrast, with a constant initialization $\gamma_k = \bar{\gamma}$ the product $\mathbf{\Psi}^T\mathbf{\Psi}$ can be directly updated.

For the algorithm proposed in this article, it is necessary to be able to compute the eigenvalues of $\mathbf{B}$ and to be able to compute products with $\mathbf{P}_\parallel$. However, in our application, it could be the case that $\mathbf{\Psi}$ is not full rank; in this case, it is preferable to use the LDL^T decomposition [Golub and Van Loan 1996] of $\mathbf{\Psi}^T\mathbf{\Psi}$ as proposed in Burdakov et al. [2017]. Specifically,

$$\mathbf{\Pi}^T\mathbf{\Psi}^T\mathbf{\Psi}\mathbf{\Pi} = \mathbf{L}\mathbf{D}\mathbf{L}^T,$$

where $\mathbf{\Pi}$ is a permutation matrix. If $\mathbf{\Psi}$ is rank deficient, i.e., $\text{rank}(\mathbf{\Psi}) = r < m$, then at least one diagonal entry of $\mathbf{D}$ is zero. (In computer arithmetic, it will be relatively small.) In the proposed algorithm, we use the following criteria to determine whether entries in $\mathbf{D}$ are sufficiently large: The ith entry of $\mathbf{D}$, i.e., d_i, is sufficiently large provided that

$$d_i > 10^{-8} \times [\mathbf{\Pi}^T\mathbf{\Psi}^T\mathbf{\Psi}\mathbf{\Pi}]_{ii}. \tag{11}$$

Now, let J to be the set of indices that satisfy Equation (11), i.e., $r = |J|$. Furthermore, define $\mathbf{D}_\dagger$ to be the matrix $\mathbf{D}$ having removed any rows and columns indexed by an element not in J and $\mathbf{L}_\dagger$ to be the matrix $\mathbf{L}$ having removed columns indexed by an element not in J. Then,

$$\mathbf{\Psi}^T\mathbf{\Psi} \approx \mathbf{\Pi}\mathbf{L}_\dagger\mathbf{D}_\dagger\mathbf{L}_\dagger^T\mathbf{\Pi}^T = \mathbf{\Pi}\mathbf{R}_\dagger^T\mathbf{R}_\dagger\mathbf{\Pi}^T,$$

where $\mathbf{R}_\dagger \triangleq \sqrt{\mathbf{D}_\dagger}\mathbf{L}_\dagger^T \in \mathbb{R}^{r\times m}$. Furthermore,

$$\mathbf{B} \approx \gamma\mathbf{I} + \mathbf{Q}_\dagger\mathbf{R}_\dagger\mathbf{\Pi}^T\mathbf{M}\mathbf{\Pi}\mathbf{R}_\dagger^T\mathbf{Q}_\dagger^T \quad \text{with} \quad \mathbf{Q}_\dagger \triangleq (\mathbf{\Psi}\mathbf{\Pi})_\dagger\,\mathbf{R}_\ddagger^{-1} \in \mathbb{R}^{n\times r}, \tag{12}$$

where $(\mathbf{\Psi}\mathbf{\Pi})_\dagger$ is the matrix $\mathbf{\Psi}\mathbf{\Pi}$ having deleted any columns indexed by an element not in J, and $\mathbf{R}_\ddagger \in \mathbb{R}^{r\times r}$ is the matrix $\mathbf{R}_\dagger$ having removed columns indexed by elements not in J. Notice that the matrix $\mathbf{R}_\dagger\mathbf{\Pi}^T\mathbf{M}\mathbf{\Pi}\mathbf{R}_\dagger^T \in \mathbb{R}^{r\times r}$ is full rank.

Thus, the eigenvalue decomposition $\mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}^T$ is now computed not for $\mathbf{R}\mathbf{M}\mathbf{R}^T$ as in Section 2.3, but for $\mathbf{R}_\dagger\mathbf{\Pi}^T\mathbf{M}\mathbf{\Pi}\mathbf{R}_\dagger{}^T$. Furthermore, $\mathbf{P}_\parallel$ in Equation (9) is computed as

$$\mathbf{P}_\parallel = \mathbf{Q}_\dagger\mathbf{U} = (\mathbf{\Psi}\mathbf{\Pi})_\dagger\,\mathbf{R}_\ddagger^{-1}\mathbf{U} \tag{13}$$

when a constant initialization is used (since $\mathbf{\Psi}$ is explicitly formed), and as

$$\mathbf{P}_\parallel = \mathbf{Q}_\dagger\mathbf{U} = (\mathbf{Y}\mathbf{\Pi})_\dagger\,\mathbf{R}_\ddagger^{-1}\mathbf{U} - \gamma\,(\mathbf{S}\mathbf{\Pi})_\dagger\,\mathbf{R}_\ddagger^{-1}\mathbf{U} \tag{14}$$

when a non-constant initialization is used.

Algorithm 1 details the computation of the elements needed to form $\mathbf{P}_\parallel$, using the LDL^T decomposition. It produces $\mathbf{\Lambda}$, $\mathbf{R}_\ddagger$, $\mathbf{U}$, and $\mathbf{\Pi}$. There are several pre-processing and post-processing steps in this algorithm. Namely, lines 7 and 9 are used to remove any spurious complex round-off error, line 10 is to order the eigenvalues and associated eigenvectors, and line 12 sets any small eigenvalue (in absolute value) to zero. An alternative to forming and storing $\mathbf{R}_\ddagger$ is to maintain $\mathbf{R}_\dagger$ and the index set J. Moreover, since it is typically more efficient to update the product $\mathbf{\Psi}^T\mathbf{\Psi}$ instead of forming it from scratch, the argument "$\mathbf{\Psi} \vee \mathbf{\Psi}^T\mathbf{\Psi}$" is used to enable passing either of the two inputs, depending on the context.

ALGORITHM 1: Computing $\mathbf{R}_{\ddagger}$, $\mathbf{\Lambda}$, $\mathbf{U}$, and $\mathbf{\Pi}$ using the LDL^T decomposition

Function: $[\mathbf{R}_{\ddagger}, \mathbf{\Lambda}, \mathbf{U}, \mathbf{\Pi}, J]$=`ComputeSpectral`$(\Psi \vee \Psi^T\Psi, \mathbf{M}^{-1}, \gamma, \tau)$;

1: Compute the LDL^T decomposition of $\mathbf{\Psi}^T\mathbf{\Psi}$ and store the factors $\mathbf{L}$ and $\mathbf{D}$ matrices, and store $\mathbf{\Pi}$ (as a vector with the permutation information);
2: Find the indices of elements of $\mathbf{D}$ that are sufficiently large using (11) and store as J;
3: Form $\mathbf{D}_\dagger$ by storing the rows and columns of $\mathbf{D}$ corresponding to indices of J;
4: Form $\mathbf{L}_\dagger$ by storing the columns of $\mathbf{L}$ corresponding the indices of J;
5: $\mathbf{R}_\dagger \leftarrow \sqrt{\mathbf{D}_\dagger}\mathbf{L}_\dagger^T$;
6: $\mathbf{T} \leftarrow \mathbf{R}_\dagger\mathbf{\Pi}^T\mathbf{M}\mathbf{\Pi}\mathbf{R}_\dagger^T$;
7: Compute the spectral decomposition $\mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}^T$ of $(\mathbf{T} + \mathbf{T}^T)/2$;
8: Form $\mathbf{R}_{\ddagger}$ by storing the columns of $\mathbf{R}_\dagger$ corresponding to columns of J;
9: $\hat{\mathbf{\Lambda}} \leftarrow \text{real}(\hat{\mathbf{\Lambda}})$
10: Order the entries in $\hat{\mathbf{\Lambda}}$ from low to high and rearrange the columns of $\mathbf{U}$ accordingly to maintain the spectral decomposition of $(\mathbf{T} + \mathbf{T}^T)/2$;
11: $\mathbf{\Lambda} \leftarrow \hat{\mathbf{\Lambda}} + \gamma\mathbf{I}$;
12: **if** $|\mathbf{\Lambda}_{ii}| < \tau$ for any i **then**
13: $\quad \mathbf{\Lambda}_{ii} \leftarrow 0$;
14: **end if**
15: **return** $\mathbf{R}_{\ddagger}$, $\mathbf{\Lambda}$, $\mathbf{U}$, $\mathbf{\Pi}$;

The output of Algorithm 1 includes the factors of $\mathbf{P}_{\|}$ (see Equation (13)), i.e., $\mathbf{R}_{\ddagger}$, $\mathbf{U}$, and $\mathbf{\Pi}$, as well as J. For the method proposed in Section 3, products with $\mathbf{P}_{\|}$ are computed as a sequence of explicit matrix-vector products with the factors of $\mathbf{P}_{\|}$. In practice, the permutation matrix $\mathbf{\Pi}$ is not stored explicitly; instead, the permutation is applied implicitly using a vector that maintains the order of the columns after the permutation matrix is applied. Thus, products with $\mathbf{P}_{\|}$ are computed using only matrix-vector products together with a rearranging of columns.

3 PROPOSED METHOD

The proposed method is able to solve the L-SR1 trust-region subproblem to high accuracy, even when $\mathbf{B}$ is indefinite. The method makes use of the eigenvalues of $\mathbf{B}$ and the factors of $\mathbf{P}_{\|}$. To describe the method, we first transform the trust-region subproblem (1) so that the quadratic objective function becomes separable. Then, we describe the shape-changing norms proposed in Burdakov and Yuan [2002] and Burdakov et al. [2017] that decouples the separable problem into two minimization problems, one of which has a closed-form solution while the other can be solved very efficiently. Finally, we show how these solutions can be used to construct a solution to the original trust-region subproblem.

3.1 Transforming the Trust-region Subproblem

Let $\mathbf{B} = \mathbf{P}\Lambda_\gamma\mathbf{P}^T$ be the eigendecomposition of $\mathbf{B}$ described in Section 2.2. Letting $\mathbf{v} = \mathbf{P}^T\mathbf{p}$ and $\mathbf{g}_\mathbf{P} = \mathbf{P}^T\mathbf{g}$, the objective function $Q(\mathbf{p})$ in Equation (1) can be written as a function of $\mathbf{v}$:

$$Q(\mathbf{p}) = \mathbf{g}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{B}\mathbf{p} = \mathbf{g}_\mathbf{P}^T\mathbf{v} + \frac{1}{2}\mathbf{v}^T\Lambda_\gamma\mathbf{v} \triangleq q(\mathbf{v}).$$

With $\mathbf{P} = \begin{bmatrix}\mathbf{P}_{\|} & \mathbf{P}_{\perp}\end{bmatrix}$, we partition $\mathbf{v}$ and $\mathbf{g}_\mathbf{P}$ as follows:

$$\mathbf{v} = \mathbf{P}^T\mathbf{p} = \begin{bmatrix}\mathbf{P}_{\|}^T\mathbf{p} \\ \mathbf{P}_{\perp}^T\mathbf{p}\end{bmatrix} = \begin{bmatrix}\mathbf{v}_{\|} \\ \mathbf{v}_{\perp}\end{bmatrix} \quad \text{and} \quad \mathbf{g}_\mathbf{P} = \begin{bmatrix}\mathbf{P}_{\|}^T\mathbf{g} \\ \mathbf{P}_{\perp}^T\mathbf{g}\end{bmatrix} = \begin{bmatrix}\mathbf{g}_{\|} \\ \mathbf{g}_{\perp}\end{bmatrix},$$

where $\mathbf{v}_{\|}, \mathbf{g}_{\|} \in \mathbb{R}^m$ and $\mathbf{v}_{\perp}, \mathbf{g}_{\perp} \in \mathbb{R}^{n-m}$. Then

$$\begin{aligned} q(\mathbf{v}) &= \begin{bmatrix} \mathbf{g}_{\|}^T & \mathbf{g}_{\perp}^T \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\|} \\ \mathbf{v}_{\perp} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{v}_{\|}^T & \mathbf{v}_{\perp}^T \end{bmatrix} \begin{bmatrix} \Lambda & \\ & \gamma \mathbf{I}_{n-m} \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\|} \\ \mathbf{v}_{\perp} \end{bmatrix} \\ &= \mathbf{g}_{\|}^T \mathbf{v}_{\|} + \mathbf{g}_{\perp}^T \mathbf{v}_{\perp} + \frac{1}{2} \left(\mathbf{v}_{\|}^T \Lambda \mathbf{v}_{\|} + \gamma \left\| \mathbf{v}_{\perp} \right\|^2 \right) \\ &= q_{\|}\left(\mathbf{v}_{\|}\right) + q_{\perp}\left(\mathbf{v}_{\perp}\right), \end{aligned} \tag{15}$$

where

$$q_{\|}\left(\mathbf{v}_{\|}\right) \triangleq \mathbf{g}_{\|}^T \mathbf{v}_{\|} + \frac{1}{2} \mathbf{v}_{\|}^T \Lambda \mathbf{v}_{\|} \quad \text{and} \quad q_{\perp}\left(\mathbf{v}_{\perp}\right) \triangleq \mathbf{g}_{\perp}^T \mathbf{v}_{\perp} + \frac{\gamma}{2} \left\| \mathbf{v}_{\perp} \right\|^2.$$

Thus, the trust-region subproblem (1) can be expressed as

$$\underset{\|\mathbf{P}\mathbf{v}\| \leq \delta}{\text{minimize}} \; q(\mathbf{v}) = \left\{ q_{\|}\left(\mathbf{v}_{\|}\right) + q_{\perp}\left(\mathbf{v}_{\perp}\right) \right\}. \tag{16}$$

Note that the function $q(\mathbf{v})$ is now separable in $\mathbf{v}_{\|}$ and $\mathbf{v}_{\perp}$. To completely decouple Equation (16) into two minimization problems, we use a shape-changing norm so that the norm constraint $\|\mathbf{P}\mathbf{v}\| \leq \delta$ decouples into separate constraints, one involving $\mathbf{v}_{\|}$ and the other involving $\mathbf{v}_{\perp}$.

3.2 Shape-changing Norms

Consider the following shape-changing norms proposed in Burdakov and Yuan [2002] and Burdakov et al. [2017]:

$$\|\mathbf{p}\|_{\mathbf{P},2} \triangleq \max\left(\|\mathbf{P}_{\|}^T \mathbf{p}\|_2, \|\mathbf{P}_{\perp}^T \mathbf{p}\|_2 \right) = \max\left(\|\mathbf{v}_{\|}\|_2, \|\mathbf{v}_{\perp}\|_2 \right), \tag{17}$$

$$\|\mathbf{p}\|_{\mathbf{P},\infty} \triangleq \max\left(\|\mathbf{P}_{\|}^T \mathbf{p}\|_{\infty}, \|\mathbf{P}_{\perp}^T \mathbf{p}\|_2 \right) = \max\left(\|\mathbf{v}_{\|}\|_{\infty}, \|\mathbf{v}_{\perp}\|_2 \right). \tag{18}$$

We refer to them as the $(\mathbf{P}, 2)$ and the $(\mathbf{P}, \infty)$ norms, respectively. Since $\mathbf{p} = \mathbf{P}\mathbf{v}$, the trust-region constraint in Equation (16) can be expressed in these norms as

$$\begin{aligned} \|\mathbf{P}\mathbf{v}\|_{\mathbf{P},2} &\leq \delta \qquad \text{if and only if} \qquad \|\mathbf{v}_{\|}\|_2 \leq \delta \text{ and } \|\mathbf{v}_{\perp}\|_2 \leq \delta, \\ \|\mathbf{P}\mathbf{v}\|_{\mathbf{P},\infty} &\leq \delta \qquad \text{if and only if} \qquad \|\mathbf{v}_{\|}\|_{\infty} \leq \delta \text{ and } \|\mathbf{v}_{\perp}\|_2 \leq \delta. \end{aligned}$$

Thus, from Equation (16), the trust-region subproblem is given for the $(\mathbf{P}, 2)$ norm by

$$\underset{\|\mathbf{P}\mathbf{v}\|_{\mathbf{P},2} \leq \delta}{\text{minimize}} \; q(\mathbf{v}) = \underset{\|\mathbf{v}_{\|}\|_2 \leq \delta}{\text{minimize}} \; q_{\|}\left(\mathbf{v}_{\|}\right) + \underset{\|\mathbf{v}_{\perp}\|_2 \leq \delta}{\text{minimize}} \; q_{\perp}\left(\mathbf{v}_{\perp}\right), \tag{19}$$

and using the $(\mathbf{P}, \infty)$ norm it is given by

$$\underset{\|\mathbf{P}\mathbf{v}\|_{\mathbf{P},\infty} \leq \delta}{\text{minimize}} \; q(\mathbf{v}) = \underset{\|\mathbf{v}_{\|}\|_{\infty} \leq \delta}{\text{minimize}} \; q_{\|}\left(\mathbf{v}_{\|}\right) + \underset{\|\mathbf{v}_{\perp}\|_2 \leq \delta}{\text{minimize}} \; q_{\perp}\left(\mathbf{v}_{\perp}\right). \tag{20}$$

As shown in Burdakov et al. [2017], these norms are equivalent to the two-norm, i.e.,

$$\begin{aligned} \frac{1}{\sqrt{2}} \|\mathbf{p}\|_2 &\leq \|\mathbf{p}\|_{\mathbf{P},2} \leq \|\mathbf{p}\|_2 \\ \frac{1}{\sqrt{m}} \|\mathbf{p}\|_2 &\leq \|\mathbf{p}\|_{\mathbf{P},\infty} \leq \|\mathbf{p}\|_2. \end{aligned}$$

Note that the latter equivalence factor depends on the number of stored quasi-Newton pairs m and not on the number of variables (n).

Notice that the shape-changing norms do not place equal value on the two subspaces, since the region defined by the subspaces is of different size and shape in each of them. However, because of norm equivalence, the shape-changing region insignificantly differs from the region defined by the two-norm, the most commonly-used choice of norm.

We now show how to solve the decoupled subproblems.

3.3 Solving for the Optimal $\mathbf{v}_\perp^*$

The subproblem

$$\underset{\|\mathbf{v}_\perp\|_2 \le \delta}{\text{minimize}} \quad q_\perp(\mathbf{v}_\perp) \equiv \mathbf{g}_\perp^T \mathbf{v}_\perp + \frac{\gamma}{2}\|\mathbf{v}_\perp\|_2^2 \tag{21}$$

appears in both Equations (19) and (20); its optimal solution can be computed by formula. For the quadratic subproblem (21) the solution $\mathbf{v}_\perp^*$ must satisfy the following optimality conditions found in Gay [1981], Moré and Sorensen [1983], and Sorensen [1982] associated with Equation (21): For some $\sigma_\perp^* \in \mathbb{R}^+$,

$$\left(\gamma + \sigma_\perp^*\right)\mathbf{v}_\perp^* = -\mathbf{g}_\perp, \tag{22a}$$

$$\sigma_\perp^*\left(\|\mathbf{v}_\perp^*\|_2 - \delta\right) = 0, \tag{22b}$$

$$\|\mathbf{v}_\perp^*\|_2 \le \delta, \tag{22c}$$

$$\gamma + \sigma_\perp^* \ge 0. \tag{22d}$$

Note that the optimality conditions are satisfied by $(\mathbf{v}_\perp^*, \sigma_\perp^*)$ given by

$$\mathbf{v}_\perp^* = \begin{cases} -\frac{1}{\gamma}\mathbf{g}_\perp & \text{if } \gamma > 0 \text{ and } \|\mathbf{g}_\perp\|_2 \le \delta|\gamma|, \\ \delta\mathbf{u} & \text{if } \gamma \le 0 \text{ and } \|\mathbf{g}_\perp\|_2 = 0, \\ -\frac{\delta}{\|\mathbf{g}_\perp\|_2}\mathbf{g}_\perp & \text{otherwise,} \end{cases} \tag{23}$$

and

$$\sigma_\perp^* = \begin{cases} 0 & \text{if } \gamma > 0 \text{ and } \|\mathbf{g}_\perp\|_2 \le \delta|\gamma|, \\ \frac{\|\mathbf{g}_\perp\|_2}{\delta} - \gamma & \text{otherwise,} \end{cases} \tag{24}$$

where $\mathbf{u} \in \mathbb{R}^{n-m}$ is any unit vector with respect to the two-norm.

3.4 Solving for the Optimal $\mathbf{v}_\parallel^*$

In this section, we detail how to solve for the optimal $\mathbf{v}_\parallel^*$ when either the $(\mathbf{P}, \infty)$-norm or the $(\mathbf{P}, 2)$-norm is used to define the trust-region subproblem.

$(\mathbf{P}, \infty)$-norm solution. If the shape-changing $(\mathbf{P}, \infty)$-norm is used in Equation (16), then the subproblem in $\mathbf{v}_\parallel$ is

$$\underset{\|\mathbf{v}_\parallel\|_\infty \le \delta}{\text{minimize}} \quad q_\parallel\left(\mathbf{v}_\parallel\right) = \mathbf{g}_\parallel^T \mathbf{v}_\parallel + \frac{1}{2}\mathbf{v}_\parallel^T \Lambda \mathbf{v}_\parallel. \tag{25}$$

The solution to this problem is computed by separately minimizing m scalar quadratic problems of the form

$$\underset{|[\mathbf{v}_\parallel]_i| \le \delta}{\text{minimize}} \quad q_{\parallel,i}([\mathbf{v}_\parallel]_i) = \left[\mathbf{g}_\parallel\right]_i \left[\mathbf{v}_\parallel\right]_i + \frac{\lambda_i}{2}\left(\left[\mathbf{v}_\parallel\right]_i\right)^2, \qquad 1 \le i \le m. \tag{26}$$

The minimizer depends on the convexity of $q_{\parallel,i}$, i.e., the sign of λ_i. The solution to Equation (26) is given as follows:

$$[\mathbf{v}_{||}^*]_i = \begin{cases} -\frac{[\mathbf{g}_{||}]_i}{\lambda_i} & \text{if } \left|\frac{[\mathbf{g}_{||}]_i}{\lambda_i}\right| \le \delta \text{ and } \lambda_i > 0, \\ c & \text{if } \left[\mathbf{g}_\parallel\right]_i = 0,\ \lambda_i = 0, \\ -\text{sgn}(\left[\mathbf{g}_\parallel\right]_i)\delta & \text{if } \left[\mathbf{g}_\parallel\right]_i \ne 0,\ \lambda_i = 0, \\ \pm\delta & \text{if } \left[\mathbf{g}_\parallel\right]_i = 0,\ \lambda_i < 0, \\ -\frac{\delta}{|[\mathbf{g}_{||}]_i|}\left[\mathbf{g}_{||}\right]_i & \text{otherwise,} \end{cases} \tag{27}$$

where c is any real number in $[-\delta, \delta]$ and "sgn" denotes the signum function (see Burdakov et al. [2017] for details).

$(\mathbf{P}, 2)$-norm solution: If the shape-changing $(\mathbf{P}, 2)$-norm is used in Equation (16), then the subproblem in $\mathbf{v}_\parallel$ is

$$\underset{\|\mathbf{v}_\parallel\|_2 \leq \delta}{\text{minimize}} \quad q_\parallel\left(\mathbf{v}_\parallel\right) = \mathbf{g}_\parallel^T \mathbf{v}_\parallel + \frac{1}{2}\mathbf{v}_\parallel^T \Lambda \mathbf{v}_\parallel. \tag{28}$$

The solution $\mathbf{v}_\parallel^*$ must satisfy the following optimality conditions [Gay 1981; Moré and Sorensen 1983; Sorensen 1982] associated with Equation (28): For some $\sigma_\parallel^* \in \mathbb{R}^+$,

$$(\Lambda + \sigma_\parallel^* \mathbf{I})\mathbf{v}_\parallel^* = -\mathbf{g}_\parallel, \tag{29a}$$

$$\sigma_\parallel^* \left(\|\mathbf{v}_\parallel^*\|_2 - \delta\right) = 0, \tag{29b}$$

$$\|\mathbf{v}_\parallel^*\|_2 \leq \delta, \tag{29c}$$

$$\lambda_i + \sigma_\parallel^* \geq 0 \quad \text{for } 1 \leq i \leq m. \tag{29d}$$

A solution to the optimality conditions (29a)–(29d) can be computed using the method found in Brust et al. [2017]. For completeness, we outline the method here; this method depends on the sign of λ_1. Throughout these cases, we make use of the expression of $\mathbf{v}_\parallel$ as a function of $\sigma_\parallel$. That is, from the first optimality condition (29a), we write

$$\mathbf{v}_\parallel\left(\sigma_\parallel\right) = -\left(\Lambda + \sigma_\parallel \mathbf{I}\right)^{-1} \mathbf{g}_\parallel, \tag{30}$$

with $\sigma_\parallel \neq -\lambda_i$ for $1 \leq i \leq m$.

Case 1 ($\lambda_1 > 0$). When $\lambda_1 > 0$, the unconstrained minimizer is computed (setting $\sigma_\parallel^* = 0$):

$$\mathbf{v}_\parallel(0) = -\Lambda^{-1}\mathbf{g}_\parallel. \tag{31}$$

If $\mathbf{v}_\parallel(0)$ is feasible, i.e., $\|\mathbf{v}_\parallel(0)\|_2 \leq \delta$, then $\mathbf{v}_\parallel^* = \mathbf{v}_\parallel(0)$ is the global minimizer; otherwise, $\sigma_\parallel^*$ is the solution to the secular equation (35) (discussed below). The minimizer to the problem (28) is then given by

$$\mathbf{v}_\parallel^* = -\left(\Lambda + \sigma_\parallel^* \mathbf{I}\right)^{-1} \mathbf{g}_\parallel. \tag{32}$$

Case 2 ($\lambda_1 = 0$). If $\mathbf{g}_\parallel$ is in the range of Λ, i.e., $[\mathbf{g}_\parallel]_i = 0$ for $1 \leq i \leq r$, then set $\sigma_\parallel = 0$ and let

$$\mathbf{v}_\parallel(0) = -\Lambda^\dagger \mathbf{g}_\parallel,$$

where $\dagger$ denotes the pseudo-inverse. If $\|\mathbf{v}_\parallel(0)\|_2 \leq \delta$, then

$$\mathbf{v}_\parallel^* = \mathbf{v}_\parallel(0) = -\Lambda^\dagger \mathbf{g}_\parallel$$

satisfies all optimality conditions (with $\sigma_\parallel^* = 0$). Otherwise, i.e., if either $[\mathbf{g}_\parallel]_i \neq 0$ for some $1 \leq i \leq r$ or $\|\Lambda^\dagger \mathbf{g}_\parallel\|_2 > \delta$, then $\mathbf{v}_\parallel^*$ is computed using Equation (32), where $\sigma_\parallel^*$ solves the secular equation in Equation (35) (discussed below).

Case 3 ($\lambda_1 < 0$): If $\mathbf{g}_\parallel$ is in the range of $\Lambda - \lambda_1 \mathbf{I}$, i.e., $[\mathbf{g}_\parallel]_i = 0$ for $1 \leq i \leq r$, then we set $\sigma_\parallel = -\lambda_1$ and

$$\mathbf{v}_\parallel(-\lambda_1) = -\left(\Lambda - \lambda_1 \mathbf{I}\right)^\dagger \mathbf{g}_\parallel.$$

If $\|\mathbf{v}_\parallel(-\lambda_1)\|_2 \leq \delta$, then the solution is given by

$$\mathbf{v}_\parallel^* = \mathbf{v}_\parallel(-\lambda_1) + \alpha \mathbf{e}_1, \tag{33}$$

where $\alpha = \sqrt{\delta^2 - \|\mathbf{v}_{\|}(-\lambda_1)\|_2^2}$. (This case is referred to as the "hard case" [Conn et al. 2000; Moré and Sorensen 1983].) Note that $\mathbf{v}_{\|}^*$ satisfies the first optimality condition (29a):

$$(\Lambda - \lambda_1 \mathbf{I})\, \mathbf{v}_{\|}^* = (\Lambda - \lambda_1 \mathbf{I}) \left(\mathbf{v}_{\|}(-\lambda_1) + \alpha \mathbf{e}_1\right) = -\mathbf{g}_{\|}.$$

The second optimality condition (29b) is satisfied by observing that

$$\|\mathbf{v}_{\|}^*\|_2^2 = \|\mathbf{v}_{\|}(-\lambda_1)\|_2^2 + \alpha^2 = \delta^2.$$

Finally, since $\sigma_{\|}^* = -\lambda_1 > 0$ the other optimality conditions are also satisfied.

However, if $[\mathbf{g}_{\|}]_i \neq 0$ for some $1 \leq i \leq r$ or $\|(\Lambda - \lambda_1 \mathbf{I})^{\dagger} \mathbf{g}_{\|}\|_2 > \delta$, then $\mathbf{v}_{\|}^*$ is computed using Equation (32), where $\sigma_{\|}^*$ solves the secular equation (35).

The secular equation. We now summarize how to find a solution of the so-called *secular equation*. Note that from Equation (30),

$$\|\mathbf{v}_{\|}(\sigma_{\|})\|_2^2 = \sum_{i=1}^{m} \frac{(\mathbf{g}_{\|})_i^2}{(\lambda_i + \sigma_{\|})^2}.$$

If we combine the terms above that correspond to the same eigenvalues and remove the terms with zero numerators, then for $\sigma_{\|} \neq -\lambda_i$, we have

$$\|\mathbf{v}_{\|}(\sigma_{\|})\|_2^2 = \sum_{i=1}^{\ell} \frac{\bar{a}_i^2}{(\bar{\lambda}_i + \sigma_{\|})^2},$$

where $\bar{a}_i \neq 0$ for $i = 1, \ldots, \ell$ and $\bar{\lambda}_i$ are *distinct* eigenvalues of $\mathbf{B}$ with $\bar{\lambda}_1 < \bar{\lambda}_2 < \cdots < \bar{\lambda}_\ell$. Next, we define the function

$$\phi_{\|}(\sigma_{\|}) = \begin{cases} \dfrac{1}{\sqrt{\displaystyle\sum_{i=1}^{\ell} \frac{\bar{a}_i^2}{(\bar{\lambda}_i + \sigma_{\|})^2}}} - \dfrac{1}{\delta} & \text{if } \sigma_{\|} \neq -\bar{\lambda}_i \text{ where } 1 \leq i \leq \ell \\ -\dfrac{1}{\delta} & \text{otherwise.} \end{cases} \tag{34}$$

From the optimality conditions (29b) and (29d), if $\sigma_{\|}^* \neq 0$, then $\sigma_{\|}^*$ solves the *secular equation*

$$\phi_{\|}(\sigma_{\|}) = 0, \tag{35}$$

with $\sigma_{\|} \geq \max\{0, -\lambda_1\}$. Note that $\phi_{\|}$ is monotonically increasing and concave on the interval $[-\lambda_1, \infty)$; thus, with a judicious choice of initial $\sigma_{\|}^0$, Newton's method can be used to efficiently compute $\sigma_{\|}^*$ in Equation (35) (see Brust et al. [2017]).

More details on the solution method for subproblem (28) are given in Brust et al. [2017].

3.5 Computing $\mathbf{p}^*$

Given $\mathbf{v}^* = [\mathbf{v}_{\|}^* \ \mathbf{v}_{\perp}^*]^T$, the solution to the trust-region subproblem (1) using either the $(\mathbf{P}, 2)$ or the $(\mathbf{P}, \infty)$ norms is

$$\mathbf{p}^* = \mathbf{P}\mathbf{v}^* = \mathbf{P}_{\|}\mathbf{v}_{\|}^* + \mathbf{P}_{\perp}\mathbf{v}_{\perp}^*. \tag{36}$$

(Recall that using either of the two norms generates the same $\mathbf{v}_{\perp}^*$ but different $\mathbf{v}_{\|}^*$.) It remains to show how to form $\mathbf{p}^*$ in Equation (36). Matrix-vector products involving $\mathbf{P}_{\|}$ are possible using Equation (13) or Equation (14), and thus $\mathbf{P}_{\|}\mathbf{v}_{\|}^*$ can be computed; however, an explicit formula to compute products $\mathbf{P}_{\perp}$ is not available. To compute the second term, $\mathbf{P}_{\perp}\mathbf{v}_{\perp}^*$, we observe that

$\mathbf{v}_\perp^*$, as given in Equation (23), is a multiple of either $\mathbf{g}_\perp = \mathbf{P}_\perp^T\mathbf{g}$ or a vector $\mathbf{u}$ with unit length, depending on the sign of γ and the magnitude of $\mathbf{g}_\perp$. In the latter case, define $\mathbf{u} = \frac{\mathbf{P}_\perp^T\mathbf{e}_i}{\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2}$, where $i \in \{1, 2, \ldots, k+2\}$ is the first index such that $\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2 \neq 0$. (Such an $\mathbf{e}_i$ exists, since $\text{rank}(\mathbf{P}_\perp) = n - m$.) Thus, we obtain

$$\mathbf{p}^* = \mathbf{P}_\parallel\left(\mathbf{v}_\parallel^* - \mathbf{P}_\parallel^T\mathbf{w}^*\right) + \mathbf{w}^*, \tag{37}$$

where

$$\mathbf{w}^* = \begin{cases} -\frac{1}{\gamma}\mathbf{g} & \text{if } \gamma > 0 \text{ and } \|\mathbf{g}_\perp\|_2 \leq \delta|\gamma|, \\ \frac{\delta}{\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2}\mathbf{e}_i & \text{if } \gamma \leq 0 \text{ and } \|\mathbf{g}_\perp\|_2 = 0, \\ -\frac{\delta}{\|\mathbf{g}_\perp\|_2}\mathbf{g} & \text{otherwise.} \end{cases} \tag{38}$$

Algorithm 2 summarizes the computation of $\mathbf{w}^*$.

ALGORITHM 2: Computing $\mathbf{w}^*$

Function: $[\mathbf{w}^*, \beta, \text{hasBeta}]$=ComputeW$(\mathbf{g}, \delta, \gamma, \|\mathbf{g}_\perp\|_2, \mathbf{\Pi}, \mathbf{\Psi}, \mathbf{R}_\ddagger, \mathbf{U}, \tau, [\texttt{varargin} = \{\mathbf{S}, \mathbf{Y}\}])$;

1: **if** $\gamma > 0$ **and** $\|\mathbf{g}_\perp\|_2 \leq \delta\gamma$ **then**
2: $\quad\beta \leftarrow -(1/\gamma)$, hasBeta $\leftarrow 1$;
3: $\quad\mathbf{w}^* \leftarrow \beta\mathbf{g}$;
4: **else if** $\gamma \leq 0$ **and** $\|\mathbf{g}_\perp\|_2 < \tau$ **then**
5: $\quad$Find the first index i such that $\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2 \neq 0$;
6: $\quad\beta \leftarrow 0$, hasBeta $\leftarrow 0$;
7: $\quad\mathbf{w}^* \leftarrow (\delta/\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2)\mathbf{e}_i$;
8: **else**
9: $\quad\beta \leftarrow -(\delta/\|\mathbf{g}_\perp\|_2)$, hasBeta $\leftarrow 1$;
10: $\quad\mathbf{w}^* \leftarrow \beta\mathbf{g}$;
11: **end if**
12: **return** $\mathbf{w}^*$;

The quantities $\|\mathbf{g}_\perp\|_2$ and $\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2$ are computed using the orthogonality of $\mathbf{P}$, which implies

$$\left\|\mathbf{g}_\parallel\right\|_2^2 + \|\mathbf{g}_\perp\|_2^2 = \|\mathbf{g}\|_2^2, \quad \text{and} \quad \|\mathbf{P}_\parallel^T\mathbf{e}_i\|_2^2 + \|\mathbf{P}_\perp^T\mathbf{e}_i\|_2^2 = 1. \tag{39}$$

Then $\|\mathbf{g}_\perp\|_2 = \sqrt{\|\mathbf{g}\|_2^2 - \|\mathbf{g}_\parallel\|_2^2}$ and $\|\mathbf{P}_\perp^T\mathbf{e}_i\|_2 = \sqrt{1 - \|\mathbf{P}_\parallel^T\mathbf{e}_i\|_2^2}$. Note that $\mathbf{v}_\perp^*$ is never explicitly computed. Since $\mathbf{\Psi}$ is either explicitly computed when a constant initialization is used or represented through $\mathbf{S}$ and $\mathbf{Y}$, the optional input $[\texttt{varargin} = \{\mathbf{S}, \mathbf{Y}\}]$ can be used to pass $\mathbf{S}, \mathbf{Y}$ if $\mathbf{\Psi}$ is represented implicitly.

4 THE PROPOSED ALGORITHMS

In this section, we summarize Section 3 in two algorithms that solve the trust-region subproblem using the $(\mathbf{P}, \infty)$ and the $(\mathbf{P}, 2)$ norms. The required inputs depend on the initialization strategy and often include $\mathbf{g}$, $\mathbf{S}$, $\mathbf{Y}$, γ, and δ, which define the trust-region subproblem (including the L-SR1 matrix). The input τ is a small positive number used as a tolerance. The output of each algorithm is $\mathbf{p}^*$, the solution to the trust-region subproblem in the given shape-changing norm. In Algorithm 3, we detail the algorithm for solving Equation (1) using the $(\mathbf{P}, \infty)$ norm to define the subproblem; Algorithm 4 solves the subproblem using the $(\mathbf{P}, 2)$ norm.

Both algorithms accept either the matrices that hold the quasi-Newton pairs, $\mathbf{S}$ and $\mathbf{Y}$, or factors for the compact formulation $\mathbf{\Psi}$ and $\mathbf{M}^{-1}$. To reflect this option, the second and third input parameters are labeled "$S \vee \Psi$" and "$Y \vee M^{-1}$" in both algorithms. If the inputs are the quasi-Newton pairs $\mathbf{S}, \mathbf{Y}$, then the input parameter `flag` must be "0," and then factors for the compact formulation are

computed; if the inputs are factors for the compact formulation, `flag` must be "1," and then Ψ and $\mathbf{M}^{-1}$ are set to the second and third inputs. Another option is to pass the product $\Psi^T\Psi$ and the matrix $\mathbf{M}^{-1}$ along with $\mathbf{S}$ and $\mathbf{Y}$. This can be particularly advantageous when the matrix $\Psi^T\Psi$ is updated, instead of recomputed (by using, e.g., Procedure 2).

ALGORITHM 3: The SC-SR1 algorithm for the shape-changing $(\mathbf{P}, \infty)$ norm

Function: $[\mathbf{p}^*]$=`sc_sr1_infty`$(\mathbf{g}, S \vee \Psi, Y \vee M^{-1}, \gamma, \delta,$ flag, [`varargin` = $\{\Psi^T\Psi, M^{-1}\}$])

Pick τ such that $0 < \tau \ll 1$;
if flag= 0 **then**
  $\mathbf{S} \leftarrow S \vee \Psi$ and $\mathbf{Y} \leftarrow Y \vee M^{-1}$;
  if `isempty(varargin)` **then**
    Compute Ψ and $\mathbf{M}^{-1}$ as in (4)
  else
    $\Psi \vee \Psi^T\Psi \leftarrow$ `varargin`$\{1\}$ and $\mathbf{M}^{-1} \leftarrow$ `varargin`$\{2\}$;
  end if
else
  $\Psi \leftarrow S \vee \Psi$; $\mathbf{M}^{-1} \leftarrow Y \vee M^{-1}$;
end if
$[\mathbf{R}_\ddagger, \mathbf{\Lambda}, \mathbf{U}, \mathbf{\Pi}, J]$=`ComputeSpectral`$(\Psi \vee \Psi^T\Psi, \mathbf{M}^{-1}, \gamma, \tau)$;
$m \leftarrow |J|$;
$\mathbf{g}_\| \leftarrow \mathbf{P}_\|^T\mathbf{g}$ using (13);
$\|\mathbf{g}_\perp\| \leftarrow \sqrt{\|\mathbf{g}\|_2^2 - \|\mathbf{g}_\|\|^2}$;
if $\|\mathbf{g}_\perp\| < \tau$ **then**
  $\|\mathbf{g}_\perp\| \leftarrow 0$;
end if
for $i = 1$ **to** m **do**

  if $|[\mathbf{g}_\|]_i| < \delta\,|[\mathbf{\Lambda}]_{ii}|$ **and** $[\mathbf{\Lambda}]_{ii} > \tau$ **then**
    $[\mathbf{v}_\|]_i \leftarrow -[\mathbf{g}_\|]_i/[\mathbf{\Lambda}]_{ii}$;
  else if $|[\mathbf{g}_\|]_i| < \tau$ **and** $|[\mathbf{\Lambda}]_{ii}| < \tau$ **then**
    $[\mathbf{v}_\|]_i \leftarrow \delta/2$;
  else if $|[\mathbf{g}_\|]_i| > \tau$ **and** $|[\mathbf{\Lambda}]_{ii}| < \tau$ **then**
    $[\mathbf{v}_\|]_i \leftarrow -\mathrm{sgn}\left([\mathbf{g}_\|]_i\right)\delta$;
  else if $|[\mathbf{g}_\|]_i| < \tau$ **and** $[\mathbf{\Lambda}]_{ii} < -\tau$ **then**
    $[\mathbf{v}_\|]_i \leftarrow \delta$;
  else
    $[\mathbf{v}_\|]_i \leftarrow -\left(\delta/\left|[\mathbf{g}_\|]_i\right|\right)[\mathbf{g}_\|]_i$;
  end if
end for
$[\mathbf{w}^*, \beta,$hasBeta$]$=`ComputeW`$(\mathbf{g}, \delta, \gamma, \|\mathbf{g}_\perp\|_2, \mathbf{\Pi}, \mathbf{\Psi}, \mathbf{R}_\ddagger, \mathbf{U}, \tau,$ [`varargin` = $\{\mathbf{S}, \mathbf{Y}\}$]);
if hasBeta = 1 **then**
  $\mathbf{p}^* \leftarrow \mathbf{P}_\|(\mathbf{v}_\| - \beta\mathbf{g}_\|) + \mathbf{w}^*$
else
  $\mathbf{p}^* \leftarrow \mathbf{P}_\|(\mathbf{v}_\| - \mathbf{P}_\|^T\mathbf{w}^*) + \mathbf{w}^*$
end if
return $\mathbf{p}^*$

The computation of $\mathbf{p}^*$ in both Algorithms 3 and 4 is performed as in Equation (37) using two matrix-vector products with $\mathbf{P}_\|^T$ and $\mathbf{P}_\|$, respectively, to avoid matrix-matrix products. Products with $\mathbf{P}_\|$ are done using the factors $\mathbf{\Psi}$, $\mathbf{R}$, and $\mathbf{U}$ (see Section 2.3).

ALGORITHM 4: The SC-SR1 algorithm for the shape-changing $(\mathbf{P}, 2)$ norm

Function: $[\mathbf{p}^*]$=sc_sr1_2($\mathbf{g}$, $S \vee \Psi$, $Y \vee M^{-1}$, γ, δ, flag, [varargin = $\{\Psi^T\Psi, M^{-1}\}$])

Pick τ such that $0 < \tau \ll 1$;
if flag= 0 **then**
  $\mathbf{S} \leftarrow S \vee \Psi$ and $\mathbf{Y} \leftarrow Y \vee M^{-1}$;
  if isempty(varargin) **then**
    Compute $\mathbf{\Psi}$ and $\mathbf{M}^{-1}$ as in (4)
  else
    $\Psi \vee \Psi^T\Psi \leftarrow$ varargin{1} and $\mathbf{M}^{-1} \leftarrow$ varargin{2};
  end if
else
  $\mathbf{\Psi} \leftarrow S \vee \Psi$; $\mathbf{M}^{-1} \leftarrow Y \vee M^{-1}$;
end if
$[\mathbf{R}_\ddagger, \mathbf{\Lambda}, \mathbf{U}, \mathbf{\Pi}, J]$=ComputeSpectral($\Psi \vee \Psi^T\Psi$, $\mathbf{M}^{-1}$, γ, τ);
$\mathbf{g}_\parallel \leftarrow \mathbf{P}_\parallel^T \mathbf{g}$ using (13), and $\|\mathbf{g}_\perp\| \leftarrow \sqrt{\|\mathbf{g}\|_2^2 - \|\mathbf{g}_\parallel\|^2}$;
if $\|\mathbf{g}_\perp\| < \tau$ **then**
  $\|\mathbf{g}_\perp\| \leftarrow 0$;
end if
if $[\mathbf{\Lambda}]_{11} > \tau$ **then**
  if $\|\mathbf{\Lambda}^{-1}\mathbf{g}_\parallel\| < \delta$ **then**
    $\sigma_\parallel \leftarrow 0$;
  else
    Use Newton's method with $\sigma_0 = 0$ to find $\sigma_\parallel$, a solution to (30);
  end if
  $\mathbf{v}_\parallel \leftarrow -\left(\Sigma + \sigma_\parallel \mathbf{I}\right)^{-1} \mathbf{g}_\parallel$;
else if $|[\mathbf{\Lambda}]_{11}| < \tau$ **then**
  Define r to be the first i such that $|\Lambda_{ii}| > \tau$;
  if $|\mathbf{g}_{ii}| < \tau$ for $1 \le i \le r$ **and** $\|\mathbf{\Lambda}^\dagger \mathbf{g}_\parallel\| < \delta$ **then**
    $\sigma_\parallel \leftarrow 0$;
    $\mathbf{v}_\parallel \leftarrow -\mathbf{\Lambda}^\dagger \mathbf{g}_\parallel$;
  else
    $\hat{\sigma} = \max_i([\mathbf{g}_\parallel]_i/\delta - \mathbf{\Lambda}_{ii})$;
    Use Newton's method with $\sigma_0 = \hat{\sigma}$ to find $\sigma_\parallel$, a solution to (30);
    $\mathbf{v}_\parallel \leftarrow -\left(\mathbf{\Lambda} + \sigma_\parallel \mathbf{I}\right)^{-1} \mathbf{g}_\parallel$;
  end if
else
  Define r to be the first i such that $|[\Lambda]_{ii}| > \tau$;
  if $|\mathbf{g}_{ii}| < \tau$ for $1 \le i \le r$ **then**
    $\sigma_\parallel = -[\mathbf{\Lambda}]_{11}$, $\mathbf{v} \leftarrow (\mathbf{\Lambda} - [\mathbf{\Lambda}]_{11}\mathbf{I})^\dagger \mathbf{g}_\parallel$;
    if $\|\mathbf{v}\| < \delta$ **then**
      $\alpha \leftarrow \sqrt{\delta^2 - \|\mathbf{v}\|^2}$;
      $\mathbf{v}_\parallel = \mathbf{v} + \alpha \mathbf{e}_1$, where $\mathbf{e}_1$ is the first standard basis vector;
    else
      $\hat{\sigma} \leftarrow \max_i([\mathbf{g}_\parallel]_i/\delta - [\mathbf{\Lambda}]_{ii})$;
      Use Newton's method with $\sigma_0 = \max(\hat{\sigma}, 0)$ to find $\sigma_\parallel$, a solution to (30);
      $\mathbf{v}_\parallel \leftarrow -\left(\mathbf{\Lambda} + \sigma_\parallel \mathbf{I}\right)^{-1} \mathbf{g}_\parallel$;
    end if
  else
    $\hat{\sigma} \leftarrow \max_i([\mathbf{g}_\parallel]_i/\delta - [\mathbf{\Lambda}]_{ii})$;
    Use Newton's method with $\sigma_0 = \max(\hat{\sigma}, 0)$ to find $\sigma_\parallel$, a solution to (30);
    $\mathbf{v}_\parallel \leftarrow -(\mathbf{\Lambda} + \sigma_\parallel \mathbf{I})^{-1} \mathbf{g}_\parallel$;
  end if
end if
$[\mathbf{w}^*, \beta$,hasBeta]=ComputeW($\mathbf{g}$, δ, γ, $\|\mathbf{g}_\perp\|_2$, $\mathbf{\Pi}$, $\mathbf{\Psi}$, $\mathbf{R}_\ddagger$, $\mathbf{U}$, τ, [varargin = $\{\mathbf{S}, \mathbf{Y}\}$]);
if hasBeta = 1 **then**
  $\mathbf{p}^* \leftarrow \mathbf{P}_\parallel(\mathbf{v}_\parallel - \beta \mathbf{g}_\parallel) + \mathbf{w}^*$
else
  $\mathbf{p}^* \leftarrow \mathbf{P}_\parallel(\mathbf{v}_\parallel - \mathbf{P}_\parallel^T \mathbf{w}^*) + \mathbf{w}^*$
end if

Besides the optional arguments, the MATLAB implementation of both algorithms have an additional input and output variable. The additional input variable is a verbosity setting; the additional output variable is a flag that reveals whether there were any detected runtime errors.

As described in Section 2.2 the limited-memory updating techniques vary with the choice of initialization strategy $\mathbf{B}_0^{(k)} = \gamma_k \mathbf{I}$. A commonly used value is $\gamma_k = \frac{\|\mathbf{y}_k\|^2}{\mathbf{s}_k^T \mathbf{y}_k}$ [Barzilai and Borwein 1988]. Recall from Section 2.3 that γ_k is the eigenvalue for the large $n - m$-dimensional subspace, spanned by the eigenvector in $\mathbf{P}_\perp$. At a local minimum all eigenvalues of $\nabla^2 f(\mathbf{x})$ will be non-negative, motivating non-negative values of γ_k. For our implementation we tested three different strategies, one of which uses a constant initialization (C Init.)

$$\gamma_k = \begin{cases} \max\left(\min\left(\frac{\|\mathbf{y}_0\|^2}{\mathbf{s}_0^T \mathbf{y}_0}, \gamma_{\max}\right), 1\right) & \text{C Init.} \\ \frac{\|\mathbf{y}_k\|^2}{\mathbf{s}_k^T \mathbf{y}_k} \ (\text{if } \mathbf{s}_k^T \mathbf{y}_k > 0) & \text{Init. 1} \\ \max\left(\frac{\|\mathbf{y}_{k-q}\|^2}{\mathbf{s}_{k-q}^T \mathbf{y}_{k-q}}, \ldots, \frac{\|\mathbf{y}_k\|^2}{\mathbf{y}_k^T \mathbf{s}_k}\right) & \text{Init. 2.} \end{cases} \tag{40}$$

Observe that Init. 2 includes the additional parameter $q > 0$, which determines the number of pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$ to use. For C Init., the parameter $\gamma_{\max}$ ensures that the constant initialization, which uses $\mathbf{s}_0, \mathbf{y}_0$ does not exceed this threshold value. In the experiments the parameter is set as $\gamma_{\max} = 1 \times 10^4$.

4.1 Computational Complexity

We estimate the cost of one iteration using the proposed method to solve the trust-region subproblem defined by shape-changing norms (17) and (18). We make the practical assumption that $\gamma > 0$. Computational savings can be achieved by reusing previously computed matrices and not forming certain matrices explicitly. We begin by highlighting the case when a non-constant initialization strategy is used. First, we do not form $\mathbf{\Psi} = \mathbf{Y} - \gamma \mathbf{S}$ explicitly. Rather, we compute matrix-vector products with $\mathbf{\Psi}$ by computing matrix-vector products with $\mathbf{Y}$ and $\mathbf{S}$. Second, to form $\mathbf{\Psi}^T\mathbf{\Psi}$, we only store and update the small $m \times m$ matrices $\mathbf{Y}^T\mathbf{Y}$, $\mathbf{S}^T\mathbf{Y}$, and $\mathbf{S}^T\mathbf{S}$. This update involves only $3m$ vector inner products. Third, assuming we have already obtained the Cholesky factorization of $\mathbf{\Psi}^T\mathbf{\Psi}$ associated with the previously stored limited-memory pairs, it is possible to update the Cholesky factorization of the new $\mathbf{\Psi}^T\mathbf{\Psi}$ at a cost of $O(m^2)$ [Bennett 1965; Gill et al. 1974].

We now consider the dominant cost for a single subproblem solve. The eigendecomposition $\mathbf{R}_\dagger \mathbf{\Pi}^T \mathbf{M} \mathbf{\Pi} \mathbf{R}_\dagger = \mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}^T$ costs $O(m^3) = (\frac{m^2}{n})O(mn)$, where $m \ll n$. To compute $\mathbf{p}^*$ in Equation (37), one needs to compute $\mathbf{v}^*$ from Section 3.4 and $\mathbf{w}^*$ from Equation (38). The dominant cost for computing $\mathbf{v}^*$ and $\mathbf{w}^*$ is forming $\mathbf{\Psi}^T\mathbf{g}$, which requires $2mn$ operations. Note that both $\mathbf{v}_\parallel^*$ and $\mathbf{P}_\parallel^T\mathbf{w}^*$ are typically computed from $\mathbf{P}_\parallel^T\mathbf{g}$, whose main operation is $\mathbf{\Psi}^T\mathbf{g}$. Subsequently, computing $\mathbf{P}_\parallel(\mathbf{v}_\parallel^* - \mathbf{P}_\parallel^T\mathbf{w}^*)$ incurs $O(2mn)$ additional multiplications, as this operation reduces to $\mathbf{\Psi}\mathbf{f}$ for a vector $\mathbf{f}$. Thus, the dominant complexity is $O(2mn + 2mn) = O(4mn)$. The following theorem summarizes the dominant computational costs.

Theorem 4.1. *The dominant computational cost of solving one trust-region subproblem for the proposed method is $4mn$ floating point operations.*

We note that the floating point operation count of $O(4mn)$ is the same cost as for L-BFGS [Nocedal 1980].

If a constant initialization is used, then the complexity can essentially be halved, because the mat-vec applies $\mathbf{\Psi}^T\mathbf{g}$ and $\mathbf{\Psi}\mathbf{f}$ (for some vector $\mathbf{f}$) each take $O(mn)$ multiplications for a total of $O(2mn)$.

4.2 Characterization of Global Solutions

It is possible to characterize global solutions to the trust-region subproblem defined by shape-changing norm $(\mathbf{P}, 2)$-norm. The following theorem is based on well-known optimality conditions for the two-norm trust-region subproblem [Gay 1981; Moré and Sorensen 1983].

THEOREM 4.2. *A vector* $\mathbf{p}^* \in \mathbb{R}^n$ *such that* $\|\mathbf{P}_{\|}^T\mathbf{p}^*\|_2 \leq \delta$ *and* $\|\mathbf{P}_{\perp}^T\mathbf{p}^*\|_2 \leq \delta$, *is a global solution of (1) defined by the* $(\mathbf{P}, 2)$*-norm if and only if there exists unique* $\sigma_{\|}^* \geq 0$ *and* $\sigma_{\perp}^* \geq 0$ *such that*

$$\left(\mathbf{B} + \mathbf{C}_{\|}\right)\mathbf{p}^* + \mathbf{g} = 0, \quad \sigma_{\|}^*\left(\left\|\mathbf{P}_{\|}^T\mathbf{p}^*\right\|_2 - \delta\right) = 0, \quad \sigma_{\perp}^*\left(\left\|\mathbf{P}_{\perp}^T\mathbf{p}^*\right\|_2 - \delta\right) = 0, \tag{41}$$

where $\mathbf{C}_{\|} \triangleq \sigma_{\perp}^*\mathbf{I} + (\sigma_{\|}^* - \sigma_{\perp}^*)\mathbf{P}_{\|}\mathbf{P}_{\|}^T$, *the matrix* $\mathbf{B} + \mathbf{C}_{\|}$ *is positive semi-definite, and* $\mathbf{P} = [\mathbf{P}_{\|}\ \mathbf{P}_{\perp}]$ *and* $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_m) = \hat{\Lambda} + \gamma\mathbf{I}$ *are as in Equation (8).*

When run in the "verbose" mode, `sc_sr1_2.m` returns values needed to establish the optimality of $\mathbf{p}^*$ using this theorem. In particular, the code computes $\sigma_{\|}^*$, which, depending on the case, is either 0, the absolute value of the most negative eigenvalue, or obtained from Newton's method. The code also computes $\sigma_{\perp}^*$ using Equation (24), and $\|\mathbf{P}_{\perp}^T\mathbf{p}^*\|$ is computed by noting that $\|\mathbf{P}_{\perp}^T\mathbf{p}^*\|_2^2 = \|\mathbf{p}^*\|_2^2 - \|\mathbf{P}_{\|}^T\mathbf{p}^*\|_2^2$. The variables `opt1`, `opt2`, and `opt3` contain the errors in each of the equations in Equation (41); `spd_check` finds the minimum eigenvalue of $(\mathbf{B} + \mathbf{C}_{\|})$ in Equation (41), enabling one to ensure $(\mathbf{B} + \mathbf{C}_{\|})$ is positive definite; and $\sigma_{\|}^*$ and $\sigma_{\perp}^*$ are displayed to verify that they are nonnegative.

5 NUMERICAL EXPERIMENTS

In this section, we report on numerical experiments with the proposed SC-SR1 algorithm implemented in MATLAB to solve limited-memory SR1 trust-region subproblems. The experiments are divided into solving the TR subproblems with Algorithms 3 and 4, and general unconstrained minimization problems, which use the TR subproblem solvers, using 62 large-scale CUTEst problems [Gould et al. 2003].

5.1 $(\mathbf{P}, 2)$-norm Results

The SC-SR1 algorithm was tested on randomly generated problems of size $n = 10^3$ to $n = 10^7$, organized as five experiments when there is no closed-form solution to the shape-changing trust-region subproblem and one experiment designed to test the SC-SR1 method in the "hard case." These six cases only occur using the $(\mathbf{P}, 2)$-norm trust region. (In the case of the $(\mathbf{P}, \infty)$ norm, $\mathbf{v}_{\|}^*$ has the closed-form solution given by Equation (27).) The six experiments are outlined as follows:

(E1) $\mathbf{B}$ is positive definite with $\|\mathbf{v}_{\|}(0)\|_2 \geq \delta$.
(E2) $\mathbf{B}$ is positive semidefinite and singular with $[\mathbf{g}_{\|}]_i \neq 0$ for some $1 \leq i \leq r$.
(E3) $\mathbf{B}$ is positive semidefinite and singular with $[\mathbf{g}_{\|}]_i = 0$ for $1 \leq i \leq r$ and $\|\Lambda^{\dagger}\mathbf{g}_{\|}\|_2 > \delta$.
(E4) $\mathbf{B}$ is indefinite and $[\mathbf{g}_{\|}]_i = 0$ for $1 \leq i \leq r$ with $\|(\Lambda - \lambda_1\mathbf{I})^{\dagger}\mathbf{g}_{\|}\|_2 > \delta$.
(E5) $\mathbf{B}$ is indefinite and $[\mathbf{g}_{\|}]_i \neq 0$ for some $1 \leq i \leq r$.
(E6) $\mathbf{B}$ is indefinite and $[\mathbf{g}_{\|}]_i = 0$ for $1 \leq i \leq r$ with $\|\mathbf{v}_{\|}(-\lambda_1)\|_2 \leq \delta$ (the "hard case").

For these experiments, $\mathbf{S}$, $\mathbf{Y}$, and $\mathbf{g}$ were randomly generated and then altered to satisfy the requirements described above by each experiment. In experiments (E2) and (E5), δ was chosen as a random number. (In the other experiments, δ was set in accordance with the experiments' rules.) All randomly generated vectors and matrices were formed using the MATLAB `randn` command, which draws from the standard normal distribution. The initial SR1 matrix was set to $\mathbf{B}_0 = \gamma\mathbf{I}$, where $\gamma = |10 * \texttt{randn(1)}|$. Finally, the number of limited-memory updates m was set to 5, and r

Table 1. Experiment 1: $\mathbf{B}$ is Positive Definite with $\|\mathbf{v}_{\|}(0)\|_2 \geq \delta$

n	opt 1	opt 2	opt 3	$\min(\lambda(B+C_{\|}))$	$\sigma^*_{\|}$	$\sigma^*_{\perp}$	γ	time
1×10^3	2.45e-14	0.00e+00	2.45e-14	4.33e+01	1.09e+01	5.89e+02	1.63e+01	9.97e-03
1×10^4	1.21e-13	2.82e-16	4.26e-13	3.25e+01	8.14e+00	1.98e+03	1.22e+01	1.55e-03
1×10^5	5.32e-13	2.28e-16	1.40e-13	2.19e+01	5.47e+00	5.05e+03	8.14e+00	4.49e-03
1×10^6	3.56e-12	5.51e-16	2.05e-11	1.44e+01	3.61e+00	9.57e+03	5.32e+00	8.03e-02
1×10^7	1.46e-11	1.16e-11	3.64e-11	4.07e+01	1.02e+01	5.52e+04	1.52e+01	9.66e-01

Table 2. Experiment 2: $\mathbf{B}$ is Positive Semidefinite and Singular and $[\mathbf{g}_{\|}]_i \neq 0$ for Some $1 \leq i \leq r$

n	opt 1	opt 2	opt 3	$\min(\lambda(B+C_{\|}))$	$\sigma^*_{\|}$	$\sigma^*_{\perp}$	γ	time
1×10^3	1.14e-14	0.00e+00	0.00e+00	9.19e+00	9.19e+00	5.45e+02	1.82e+01	3.24e-03
1×10^4	4.24e-14	1.39e-11	1.29e-13	6.55e+00	6.55e+00	3.86e+02	5.33e-01	2.81e-03
1×10^5	4.02e-13	9.37e-14	2.04e-12	2.81e+00	2.81e+00	8.56e+02	1.16e+01	1.80e-02
1×10^6	2.53e-12	3.54e-15	3.55e-11	2.65e+00	2.65e+00	2.01e+03	1.86e+01	8.18e-02
1×10^7	1.77e-11	1.61e-11	2.44e-10	4.90e+00	4.90e+00	6.29e+03	9.44e+00	9.51e-01

Table 3. Experiment 3: $\mathbf{B}$ is Positive Semidefinite and Singular with $[\mathbf{g}_{\|}]_i = 0$ for $1 \leq i \leq r$ and $\|\Lambda^{\dagger}\mathbf{g}_{\|}\|_2 > \delta$

n	opt 1	opt 2	opt 3	$\min(\lambda(B+C_{\|}))$	$\sigma^*_{\|}$	$\sigma^*_{\perp}$	γ	time
1×10^3	1.38e-14	1.35e-09	1.21e-14	1.99e+00	1.99e+00	1.45e+02	2.80e+00	3.84e-03
1×10^4	7.38e-14	2.98e-17	4.35e-13	8.60e+00	8.60e+00	3.80e+03	1.29e+01	2.03e-03
1×10^5	1.73e-13	8.84e-17	4.17e-12	3.19e+00	3.19e+00	3.19e+03	4.67e+00	6.31e-03
1×10^6	2.04e-12	1.22e-11	4.25e-11	8.57e+00	8.57e+00	2.97e+04	1.28e+01	7.37e-02
1×10^7	3.98e-11	7.53e-11	2.42e-10	4.47e+00	4.47e+00	2.25e+04	6.63e+00	9.42e-01

Table 4. Experiment 4: $\mathbf{B}$ is Indefinite and $[\mathbf{g}_{\|}]_i = 0$ for $1 \leq i \leq r$ with $\|(\Lambda - \lambda_1\mathbf{I})^{\dagger}\mathbf{g}_{\|}\|_2 > \delta$

n	opt 1	opt 2	opt 3	$\min(\lambda(B+C_{\|}))$	$\sigma^*_{\|}$	$\sigma^*_{\perp}$	γ	time
1×10^3	1.95e-14	2.57e-16	0.00e+00	2.34e+00	3.09e+00	2.38e+02	3.04e+00	3.03e-03
1×10^4	8.69e-14	2.16e-16	0.00e+00	2.18e+00	2.59e+00	4.63e+02	2.91e+00	6.16e-03
1×10^5	2.52e-13	4.65e-17	1.72e-12	1.33e+01	1.34e+01	2.15e+04	1.98e+01	6.44e-03
1×10^6	4.45e-12	1.24e-12	1.91e-11	7.02e+00	7.21e+00	2.58e+04	1.04e+01	6.93e-02
1×10^7	2.52e-11	5.27e-10	7.46e-11	1.02e+00	1.21e+00	1.71e+04	8.35e-01	9.23e-01

was set to 2. In the five cases when there is no closed-form solution, SC-SR1 uses Newton's method to find a root of $\phi_{\|}$. We use the same procedure as in Brust et al. [2017, Algorithm 2] to initialize Newton's method, since it guarantees monotonic and quadratic convergence to σ^*. The Newton iteration was terminated when the ith iterate satisfied $\|\phi_{\|}(\sigma^i)\| \leq \texttt{eps} \cdot \|\phi_{\|}(\sigma^0)\| + \sqrt{\texttt{eps}}$, where σ^0 denotes the initial iterate for Newton's method and eps is machine precision. This stopping criteria is both a relative and absolute criteria, and it is the only stopping criteria used by SC-SR1.

To report on the accuracy of the subproblem solves, we make use of the optimality conditions found in Theorem 4.2. For each experiment, we report the following: (i) the norm of the residual of the first optimality condition, opt 1 $\triangleq \|(\mathbf{B}+\mathbf{C}_{\|})\mathbf{p}^* + \mathbf{g}\|_2$; (ii) the first complementarity condition, opt 2 $\triangleq |\sigma^*_{\|}(\|\mathbf{P}_{\|}^T\mathbf{p}^*\|_2 - \delta)|$; (iii) the second complementarity condition, opt 3 $\triangleq \|\sigma^*_{\perp}(\|\mathbf{P}_{\perp}^T\mathbf{p}^*\|_2 - \delta)|$; (iv) the minimum eigenvalue of $\mathbf{B}+\mathbf{C}_{\|}$; (v) $\sigma^*_{\|}$; (vi) $\sigma^*_{\perp}$; (vii) γ; and (viii) time. The quantities (i)–(vi) are reported to check the optimality conditions given in Theorem 4.2. Finally, we ran each experiment five times and report one representative result for each experiment.

Tables 1–6 show the results of the experiments. In all tables, the residual of the two optimality conditions opt 1, opt 2, and opt 3 are on the order of 1×10^{-10} or smaller. Columns 4 in all

Table 5. Experiment 5: $\mathbf{B}$ is Indefinite and $[\mathbf{g}_\parallel]_i \neq 0$ for Some $1 \leq i \leq r$

n	opt 1	opt 2	opt 3	$\min(\lambda(B+C_\parallel))$	$\sigma_\parallel^*$	$\sigma_\perp^*$	γ	time
1×10^3	9.11e-15	5.14e-16	4.35e-15	7.54e-01	1.16e+00	1.31e+01	2.27e+01	5.60e-03
1×10^4	6.04e-14	8.71e-12	1.25e-13	1.88e+00	2.23e+00	1.41e+02	4.15e+00	1.75e-03
1×10^5	3.16e-13	3.27e-11	2.36e-12	6.23e-01	1.24e+00	3.86e+02	4.89e+00	7.91e-03
1×10^6	1.19e-12	0.00e+00	2.82e-11	3.01e+00	3.59e+00	1.89e+03	1.77e+01	7.00e-02
1×10^7	5.25e-11	1.02e-14	1.30e-10	7.37e-01	1.43e+00	4.32e+03	1.48e+01	9.40e-01

Table 6. Experiment 6: $\mathbf{B}$ is Indefinite and $[\mathbf{g}_\parallel]_i = 0$ for $1 \leq i \leq r$ with $\|\mathbf{v}_\parallel(-\lambda_1)\|_2 \leq \delta$ (the "Hard Case")

n	opt 1	opt 2	opt 3	$\min(\lambda(B+C_\parallel))$	$\sigma_\parallel^*$	$\sigma_\perp^*$	γ	time
1×10^3	1.58e-14	1.21e-17	2.83e-14	0.00e+00	1.09e-01	1.45e+02	1.19e+00	2.06e-03
1×10^4	9.07e-14	2.65e-17	2.62e-13	0.00e+00	3.19e-01	4.49e+02	9.14e+00	1.31e-03
1×10^5	8.34e-13	8.80e-17	1.86e-12	0.00e+00	1.67e-01	1.45e+03	5.04e+00	4.45e-03
1×10^6	3.87e-12	7.21e-17	5.46e-12	0.00e+00	1.30e-01	3.51e+03	3.31e+00	6.77e-02
1×10^7	4.19e-11	1.30e-17	3.05e-10	0.00e+00	2.68e-02	2.81e+04	1.19e+01	9.45e-01

tables show that $(\mathbf{B}+\mathbf{C}_\parallel)$ are postiive semidefinite. Columns 6 and 7 in all the tables show that $\sigma_\parallel^*$ and $\sigma_\perp^*$ are nonnegative. Thus, the solutions obtained by SC-SR1 for these experiments satisfy the optimality conditions to high accuracy.

Also reported in each table are the number of Newton iterations. In the first five experiments no more than four Newton iterations were required to obtain $\sigma_\parallel$ to high accuracy (Column 8). In the hard case, no Newton iterations are required, since $\sigma_\parallel^* = -\lambda_1$. This is reflected in Table 6, where Column 4 shows that $\sigma_\parallel^* = -\lambda_1$ and Column 8 reports no Newton iterations.)

The final column reports the time required by SC-SR1 to solve each subproblem. Consistent with the best limited-memory methods, as n gets large, the time required to solve each subproblem appears to grow linearly with n, as predicted in Section 4.1.

Additional experiments were run with $\mathbf{g}_\parallel \to 0$. In particular, the experiments were rerun with $\mathbf{g}$ scaled by factors of 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, and 10^{-10}. All experiments resulted in tables similar to those in Tables 1–6: The optimality conditions were satisfied to high accuracy, no more than three Newton iterations were required in any experiment to find $\sigma_\parallel^*$, and the CPU times are similar to those found in the tables.

5.2 $(\mathbf{P},\infty)$-norm Results

The SC-SR1 method was tested on randomly generated problems of size $n = 10^3$ to $n = 10^7$, organized as five experiments that test the cases enumerated in Algorithm 3. Since Algorithm 3 proceeds componentwise (i.e., the components of $\mathbf{g}_\parallel$ and $\mathbf{\Lambda}$ determine how the algorithm proceeds), the experiments were designed to ensure at least one randomly chosen component satisfied the conditions of the given experiment. The five experiments are below:

(E1) $\left|[\mathbf{g}_\parallel]_i\right| < \delta\,|[\mathbf{\Lambda}]_{ii}|$ and $[\mathbf{\Lambda}]_{ii} > \tau$.
(E2) $\left|[\mathbf{g}_\parallel]_i\right| < \tau$ and $|[\mathbf{\Lambda}]_{ii}| < \tau$.
(E3) $\left|[\mathbf{g}_\parallel]_i\right| > \tau$ and $|[\mathbf{\Lambda}]_{ii}| < \tau$.
(E4) $\left|[\mathbf{g}_\parallel]_i\right| < \tau$ and $[\mathbf{\Lambda}]_{ii} < -\tau$.
(E5) $\left|[\mathbf{g}_\parallel]_i\right| > \delta\,|[\mathbf{\Lambda}]_{ii}|$ and $\|[\mathbf{\Lambda}]_{ii}\| > \tau$.

For these experiments, $\mathbf{S}$, $\mathbf{Y}$, and $\mathbf{g}$ were randomly generated and then altered to satisfy the requirements described above by each experiment. In (E2)–(E4), δ was chosen as a random number (in the other experiments, it was set in accordance with the experiments' rules). All

Table 7. Results Using the $(\mathbf{P}, \infty)$ Norm

	n	γ	time
Experiment 1	1×10^3	8.76e+00	1.34e-03
	1×10^4	1.80e-01	1.21e-03
	1×10^5	7.39e+00	6.71e-03
	1×10^6	2.13e-02	1.12e-01
	1×10^7	1.11e+01	1.51e+00
Experiment 2	1×10^3	4.47e+00	1.05e-03
	1×10^4	6.38e+00	8.74e-04
	1×10^5	1.10e+00	7.37e-03
	1×10^6	2.74e+00	7.94e-02
	1×10^7	8.30e-01	1.39e+00
Experiment 3	1×10^3	2.09e+01	1.07e-03
	1×10^4	4.67e+00	9.63e-04
	1×10^5	1.39e+01	6.63e-03
	1×10^6	1.76e+01	7.38e-02
	1×10^7	1.51e+01	1.45e+00
Experiment 4	1×10^3	1.08e+01	1.43e-03
	1×10^4	1.34e+01	1.06e-03
	1×10^5	7.43e+00	1.23e-02
	1×10^6	3.16e+00	9.00e-02
	1×10^7	2.22e+00	1.41e+00
Experiment 5	1×10^3	1.04e+01	1.15e-03
	1×10^4	1.74e+01	9.40e-04
	1×10^5	4.38e+00	1.15e-02
	1×10^6	5.21e+00	9.05e-02
	1×10^7	2.01e+00	1.40e+00

randomly generated vectors and matrices were formed using the MATLAB `randn` command, which draws from the standard normal distribution. The initial SR1 matrix was set to $\mathbf{B}_0 = \gamma \mathbf{I}$, where $\gamma = |10 * \texttt{randn(1)}|$. Finally, the number of limited-memory updates m was set to 5, and for simplicity, the randomly chosen i (that defines [E1]–[E5]) was chosen to be an integer in the range [1 5].

Table 7 displays the results of the five experiments. Each experiment was run five times; the results of the third iteration are stored in Table 7. In all cases, the results of the third iteration were representative of all the iterations. The first column of the table denotes the experiment, the second column displays the size of the problem, and the third column reports the value of γ. Finally, the forth column reports the time taken to obtain the solution.

5.3 Trust-region Algorithm

In this experiment, we embed the TR subproblem solvers in a trust-region algorithm to solve unconstrained optimization problems. In particular, we implemented our subproblem solvers in an algorithm that is based on Nocedal and Wright [2006, Algorithm 6.2]. A feature of this algorithm is that the L-SR1 matrix is updated by every pair $\{(\mathbf{s}_i, \mathbf{y}_i)\}_{i=k-m+1}^{k}$ as long as $|\mathbf{s}_i^T(\mathbf{y}_i - \mathbf{B}_0\mathbf{s}_i)| \geq \|\mathbf{s}_i\|_2 \|\mathbf{y}_i - \mathbf{B}_i\mathbf{s}_i\|_2 \varepsilon_{\text{SR1}}$ (updates are skipped if this condition is not met). In case a full memory

strategy is used (i.e., $m = \infty$) then a SR-1 matrix is updated by almost every pair $\{(\mathbf{s}_i, \mathbf{y}_i)\}$ to help achieve the superlinear convergence rate of quasi-Newton methods, in contrast to updating the matrix only when a step is accepted. An outline of our trust-region method implementation (Algorithm 5) is included in the Appendix. In our comparisons we use the following four algorithms to solve the TR subproblems:

TR:SC-INF	Algorithm 3
TR:SC-L2	Algorithm 4
TR:L2	ℓ_2-norm [Brust et al. 2017, Algorithm 1]
tr:CG	truncated CG [Conn et al. 2000, Algorithm 7.5.1]

(tr:CG was implemented in MATLAB by the authors, while TR:L2 is a minor modification of `obs.m` publicly available at https://github.com/johannesbrust/OBS). Initially, we included a fifth algorithm, LSTRS [Rojas et al. 2008], which performed markedly inferior to any of the above solvers and is thus not reported as part of the outcomes in this section. We also found that Init. 2 performed significantly better than Init. 1 and therefore report the outcomes with Init. 2 below. Because the limited-memory updating mechanism is different whether a constant or non-constant initialization strategy is used, we describe our results separately for C Init. and Init. 2. As part of our comparisons we first select the best algorithm using only C Init. and only using Init. 2. Subsequently, we compare the best algorithms to each other. To find default parameters for our best algorithms, Figures 6, 7, and 8 report results for a considerable range of m and q values.

All remaining experiments are for the general unconstrained minimization problem,

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\mathbf{x}), \tag{42}$$

where $f : \mathbb{R}^n \to \mathbb{R}$. We consider this problem solved once $\|\nabla f(\mathbf{x}_k)\|_\infty \leq \varepsilon$. Our convergence tolerance is set to be $\varepsilon = 5 \times 10^{-4}$. With γ fixed, a L-SR1 algorithm can be implemented by only storing the matrices $\mathbf{\Psi}_k$ and $\mathbf{M}_k^{-1}$. In particular, with a fixed $\gamma = \gamma_k$ in Equation (4) then $\mathbf{M}_k^{-1}\mathbf{e}_k = \mathbf{\Psi}_k^T\mathbf{s}_k$, so that updating the symmetric matrix $\mathbf{M}_k^{-1}$ only uses $O(nm)$ multiplications. In this way, the overall computational complexity and memory requirements of the L-SR1 method are reduced as compared to non-constant initializations. However, using a non-constant initialization strategy can adaptively incorporate additional information, which can be advantageous. Therefore, we compare the best algorithms for constant and non-constant initialization strategies in Sections 5.4, 5.5, and 5.6. Parameters in Algorithm 5 are set as follows: $c_1 = 9 \times 10^{-4}$, $c_2 = 0.75$, $c_3 = 0.8$, $c_4 = 2$, $c_5 = 0.1$, $c_6 = 0.75$, $c_7 = 0.5$, and $\varepsilon_{\text{SR1}} = 1 \times 10^{-8}$.

Extended performance profiles as in Mahajan et al. [2012] are provided. These profiles are an extension of the well known profiles of Dolan and Moré [2002]. We compare total computational time for each solver on the test set of problems. The performance metric $\rho_s(\tau)$ with a given number of test problems n_p is

$$\rho_s(\tau) = \frac{\text{card}\left\{p : \pi_{p,s} \leq \tau\right\}}{n_p} \quad \text{and} \quad \pi_{p,s} = \frac{t_{p,s}}{\min\limits_{1\leq i\leq S, i\neq s} t_{p,i}},$$

where $t_{p,s}$ is the "output" (i.e., time) of "solver" s on problem p. Here S denotes the total number of solvers for a given comparison. This metric measures the proportion of how close a given solver is to the best result. The extended performance profiles are the same as the classical ones for $\tau \geq 1$. In the profiles, we include a dashed vertical grey line to indicate $\tau = 1$. The solvers are compared on 62 large-scale CUTEst problems, which are the same problems as in Burdakov et al. [2017]. Prior to detailed experiments we include a reference comparison of all solvers on number of function

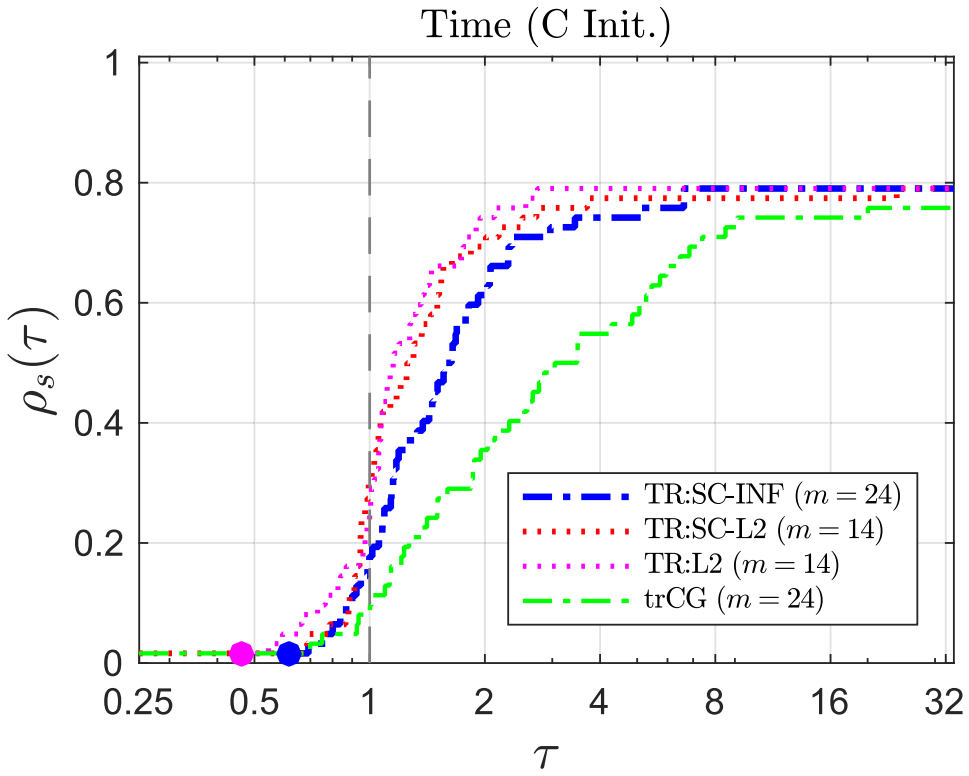

Fig. 1. Comparison of best algorithms with C Init. (constant initialization), which are selected from Figure 6.

calls and time when the same memory parameter is used across all solvers $m = 7$ (note that this setting may not correspond to the best value for a specific solver)

Additionally, Appendix A.3 includes supplementary comparisons on quadratics and the Rosenbrock objectives.

5.4 Comparisons with Constant Initialization Strategy (C Init.)

This experiment compares the algorithms when the constant initialization C Init. from Equation (40) is used. Because the memory allocation is essentially halved (relative to a non-constant initialization) the memory parameter m includes larger values, too (such as $m = 24$). For each individual solver, we first determine its optimal m parameter in Figure 6. After selecting the best parameters, these best solvers are then compared in Figure 1. Observe that TR:L2 obtains the best results in this comparison. The limited-memory parameter m is relatively large for all solvers, however, since a constant initialization is used larger memory values are permissible.

5.5 Comparisons with Non-constant Initialization Strategy (Init. 2)

Since Init. 2 depends on the parameter q, Figures 7 and 8 test each algorithm on a combination of m and q values. A comparison of the best values for each algorithm is in Figure 2. Observe that TR:SC-INF and TR:SC-L2 obtain the overall best results. All algorithms use a small memory

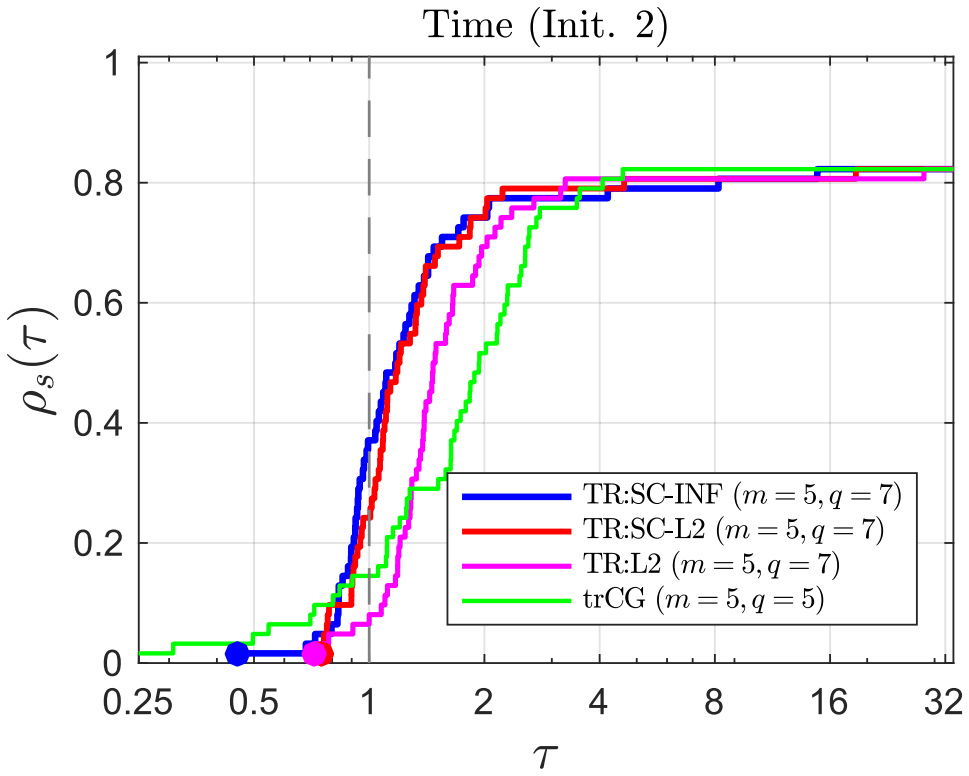

Fig. 2. Comparison of best algorithms with Init.2 (non-constant initialization), which are selected as the best ones from Figures 7 and 8.

parameter $m = 5$. Since Init. 2 is a non-constant initialization these algorithms store $\mathbf{S}_k, \mathbf{Y}_k$ to implicitly represent $\mathbf{\Psi}_k$, and thus the memory allocations scale with $2m$.

5.6 Comparisons of Best Outcomes

The overall best algorithms from Figures 1 and 2 are compared in Figures 3 and 4. This declares that the best performing algorithm over the sequence of experiments is TR:SC-INF with the indicated parameter values.

5.7 Comparison with L-BFGS-B

In a final comparison, we include the best algorithm from the preceeding section and a widely distributed method. Specifically, we use L-BFGS-B [Zhu et al. 1997] and a translation of the software into C with an interface to MATLAB. L-BFGS-B is a limited-memory quasi-Newton algorithm based on a compact representation, like the methods described in this article. We consider L-BFGS-B a state-of-the-art unconstrained algorithm when the problem is unbounded. For both algorithms we set the memory parameter $m = 5$, the convergence tolerance $\varepsilon = 5 \times 10^{-4}$ and the total number of permissible iterations 25,000. The number of function evaluations and computational times for the same 62 large-scale CUTEst problems as in the preceding sections (i.e., the problems in Burdakov et al. [2018]) are reported in Figure 5. We observe that the proposed shape-changing L-SR1 algorithm converged on 93% (i.e., 58/62) problems when compared to 90% (i.e., 56/62) for

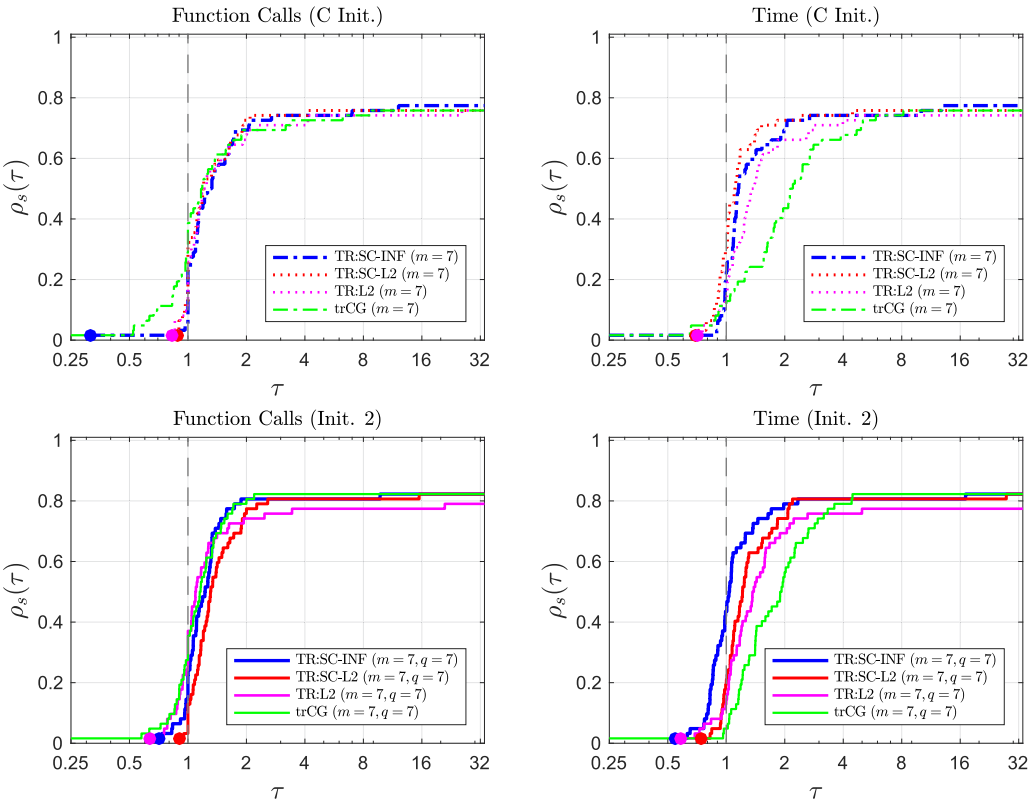

Fig. 3. Comparison of the number of function calls and computational time for the four algorithms {TR:SC-INF, TR:SC-L2, TR:L2, trCG} when a constant initialization (C Init.) or non-constant initialization (Init. 2) is used. The memory parameter $m = 7$ is the same across all solvers and $q = 7$ for Init. 2.

L-BFGS-B and the given parameters. Overall, the shape-changing algorithm used fewer function calls yet often higher computational times than L-BFGS-B. An implementation of the proposed algorithms in a lower-level language could yield further reductions in computational cost. However, the proposed method obtains robust results on the tested large-scale problems and may provide further advantages when function calls are expensive. Detailed comparisons of L-SR1 and L-BFGS algorithms are directions for future research.

6 CONCLUDING REMARKS

In this article, we presented a high-accuracy trust-region subproblem solver for when the Hessian is approximated by L-SR1 matrices. The method makes use of special shape-changing norms that decouple the original subproblem into two separate subproblems. Numerical experiments using the $(\mathbf{P}, 2)$ norm verify that solutions are computed to high accuracy in cases when there are no closed-form solutions and also in the "hard case." Experiments on large-scale unconstrained optimization problems demonstrate that the proposed algorithms perform well when compared to widely used methods, such as truncated CG or an ℓ_2 TR subproblem algorithm.

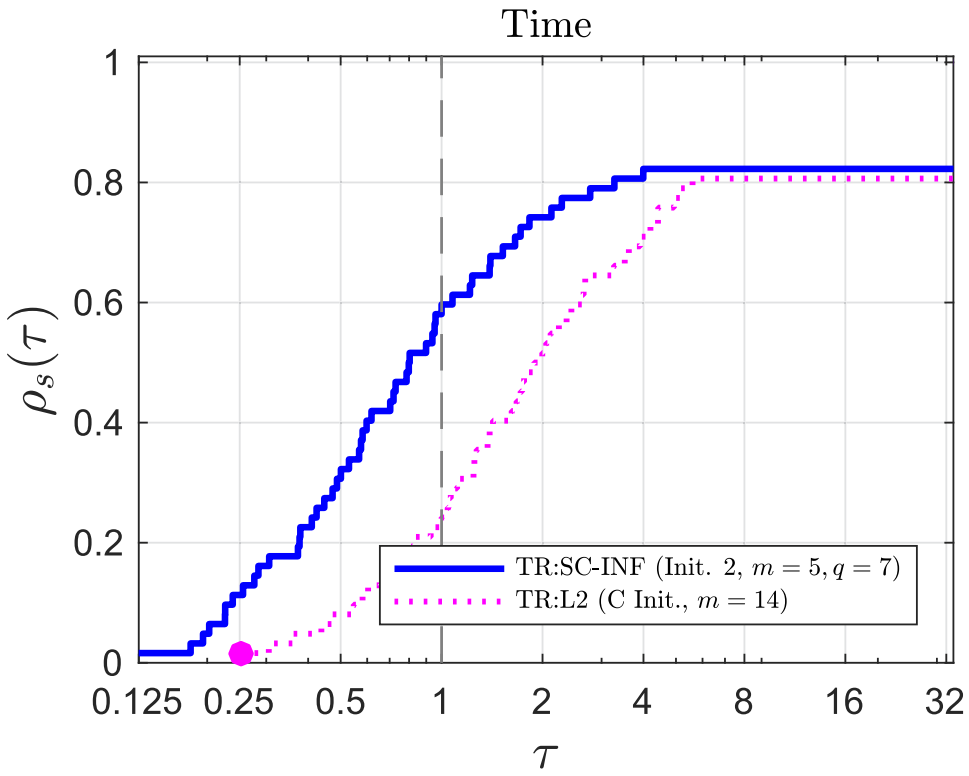

Fig. 4. Overall comparison of best algorithms by selecting winners in Figures 1 (C Init.) and 2 (Init. 2). Observe that TR:SC-INF with non-constant initialization strategy outperforms the best algorithm with a constant initialization (TR:L2). In sum, the trust-region algorithm with the proposed shape-changing infinity subproblem solver (TR:SC-INF) obtains the best results among the comparisons on 62 large-scale CUTEst problems.

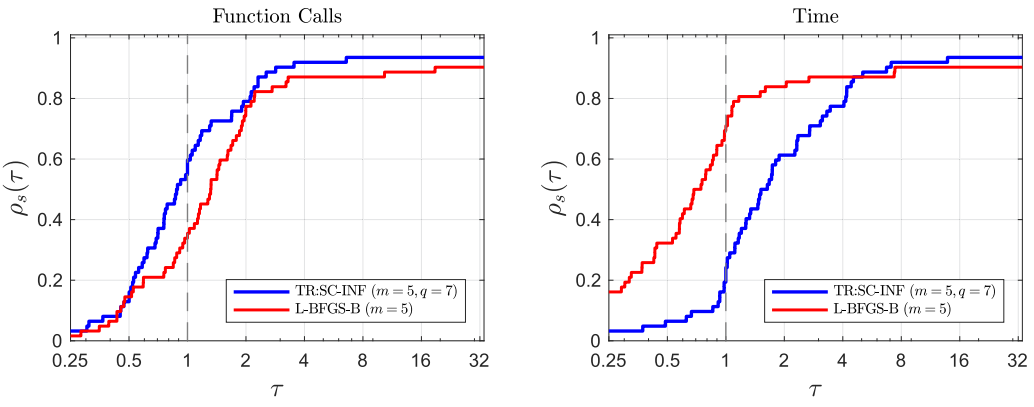

Fig. 5. Comparison of the number of function calls and computational time for the best algorithm from Section 5.6 and L-BFGS-B with memory parameter $m = 5$.

APPENDIX

This appendix lists our implementation of the L-SR1 trust-region algorithm from the numerical experiments in Section 5.3. This trust-region algorithm uses the trust-region radius adjustments from Nocedal and Wright [2006, Algorithm 6.2] and the subproblem solvers in Algorithms 3 and 4, as well as the orthonormal basis method from Brust et al. [2017].

A.1 Experiments to Determine Default Parameters with Constant Initialization (C Init.)

ALGORITHM 5: L-SR1 Shape-Changing Trust-region Algorithms (LSR1_SC)

Function: $[\mathbf{x}_k, \mathbf{g}_k, f_k, \text{out}]$=LSR1_SC($\mathbf{x}, f(\mathbf{x}), \nabla f(\mathbf{x})$, pars)

Set constants from pars: $0 < c_1 < 1 \times 10^{-3}$, $0 < c_2, 0 < c_3 < 1$, $1 < c_4$, $0 < c_5 \leq c_2$, $0 < c_6 < 1$, $0 < c_7 < 1$,$0 < \varepsilon$, $0 < m$, $0 < q$, $0 < \varepsilon_{\text{SR1}}$, ALG $\leftarrow$ pars.whichSub, INIT $\leftarrow$ pars.whichInit, SAVE $\leftarrow$ pars.storePsiPsi, ;
Initialize $k \leftarrow 0$, $k_m \leftarrow 0$, $\mathbf{x}_k \leftarrow \mathbf{x}$, $0 < \gamma_k$, $0 < \gamma_{\max}$, $\text{invM}_k \leftarrow []$, mIdx $\leftarrow 1 : m$, iEx $\leftarrow 0$;
$f_k \leftarrow f(\mathbf{x}_k)$, $\mathbf{g}_k \leftarrow \nabla f(\mathbf{x}_k)$;
$[\mathbf{x}_{k+1}, \mathbf{g}_{k+1}, f_{k+1}] \leftarrow \text{lineSearch}(\mathbf{x}_k, \mathbf{g}_k, f_k)$;
$\mathbf{s}_k \leftarrow \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{y}_k \leftarrow \mathbf{g}_{k+1} - \mathbf{g}_k$;
if INIT = C.Init. **then**
    `% Constant initialization`
    $\gamma_k \leftarrow \max(\min(\|\mathbf{y}_0\|^2/\mathbf{s}_0^T\mathbf{y}_0, \gamma_{\max}), 1)$
    $\Psi_k \leftarrow []$;
else
    `% Non-constant initialization.`
    $\gamma_k \leftarrow \|\mathbf{y}_k\|^2/\mathbf{s}_k^T\mathbf{y}_k$;
    $\mathbf{S}_k \leftarrow []$, $\mathbf{Y}_k \leftarrow []$, $\mathbf{D}_k \leftarrow []$, $\mathbf{L}_k \leftarrow []$, $\mathbf{T}_k \leftarrow []$, $\text{SS}_k \leftarrow []$, $\text{YY}_k \leftarrow []$;
end if
if SAVE = 1 **then**
    $\Psi\Psi_k \leftarrow []$;
end if
$b_k \leftarrow \mathbf{s}_k^T(\mathbf{y}_k - \gamma_k\mathbf{s}_k)$;
if $\varepsilon_{\text{SR1}}\|\mathbf{s}_k\|_2\|\mathbf{y}_k - \gamma_k\mathbf{s}_k\|_2 < \text{abs}(b_k)$ **then**
    $k_m \leftarrow k_m + 1$;
    $\text{invM}_k(k_m, k_m) \leftarrow b_k$;
    if INIT = C.Init. **then**
        $[\Psi_k, \text{mIdx}]$ = `colUpdate`$(\Psi_k, \mathbf{y}_k - \gamma_k\mathbf{s}_k, \text{mIdx}, m, k)$ `% From Procedure 1`
    else
        $[\mathbf{Y}_k, \sim]$ = `colUpdate`$(\mathbf{Y}_k, \mathbf{y}_k, \text{mIdx}, m, k)$;
        $[\mathbf{S}_k, \text{mIdx}]$ = `colUpdate`$(\mathbf{S}_k, \mathbf{s}_k, \text{mIdx}, m, k)$;
        $\mathbf{D}_k(k_m, k_m) = \mathbf{s}_k^T\mathbf{y}_k$;
        $\mathbf{L}_k(k_m, k_m) = \mathbf{s}_k^T\mathbf{y}_k$;
        $\mathbf{T}_k(k_m, k_m) = \mathbf{s}_k^T\mathbf{y}_k$;
        $\text{SS}_k(k_m, k_m) = \mathbf{s}_k^T\mathbf{s}_k$;
        $\text{YY}_k(k_m, k_m) = \mathbf{y}_k^T\mathbf{y}_k$;
    end if
end if
$\delta_k \leftarrow 2\|\mathbf{s}_k\|$;
$k \leftarrow k + 1$;
while $(\varepsilon \leq \|\mathbf{g}_k\|_\infty)$ and $(k \leq \text{maxIt})$ **do**
    Choose TR subproblem solver to compute $\mathbf{s}_k$ (e.g., Algorithms 3 and 4, ℓ_2-norm, truncated CG);
    `% For example: sc_sr1_infty with` $\Psi^T\Psi$ `updating`
    $\mathbf{s}_k \leftarrow$ `sc_sr1_infty`$(\mathbf{g}_k, \mathbf{S}_k(:, \text{mIdx}(1 : k_m)), \mathbf{Y}_k(:, \text{mIdx}(1 : k_m)), \gamma_k, \delta_k, 1, 0, \ldots$
    $\Psi\Psi_k(1 : k_m, 1 : k_m), \text{invM}_k(1 : k_m, 1 : k_m))$;
    $\widehat{\mathbf{x}}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{s}_k$, $\widehat{f}_{k+1} \leftarrow f(\widehat{\mathbf{x}}_{k+1})$, $\widehat{\mathbf{g}}_{k+1} \leftarrow \nabla f(\widehat{\mathbf{x}}_{k+1})$;
    if INIT = C.Init **then**
        $\mathbf{b}_k(1 : k_m) \leftarrow \Psi_k(:, \text{mIdx}(1 : k_m))^T\mathbf{s}_k$;
    else
        `% Non-constant initialization, stores additionally` $\mathbf{b1}_k, \mathbf{b2}_k$
        $\mathbf{b1}_k(1 : k_m) \leftarrow \mathbf{Y}_k(:, \text{mIdx}(1 : k_m))^T\mathbf{s}_k$;

$\mathbf{b2}_k(1:k_m) \leftarrow \mathrm{S}_k(:, \mathrm{mIdx}(1:k_m))^T \mathbf{s}_k$;
$\mathbf{b}_k(1:k_m) \leftarrow \mathbf{b1}_k(1:k_m) - \gamma_k \mathbf{b2}_k(1:k_m)$;
end if
$(sBs)_k \leftarrow \gamma_k \mathbf{s}_k^T \mathbf{s}_k + \frac{1}{2}\mathbf{b}_k(1:k_m)^T(\mathrm{invM}_k(1:k_m, 1:k_m) \backslash \mathbf{b}_k(1:k_m))$;
if INIT = Init. 2 **then**
% Other non-constant initialization strategies can be implemented here
$\gamma_k \leftarrow \max(\|\mathbf{y}_{k-q}\|^2 / \mathbf{s}_{k-q}^T \mathbf{y}_{k-q}, \ldots, \|\mathbf{y}_k\|^2 / \mathbf{y}_k^T \mathbf{s}_k)$
end if
$\rho_k \leftarrow \frac{\widehat{f}_{k+1} - f_k}{\mathbf{s}_k^T \mathbf{g}_k + (sBs)_k}$;
if $c_1 < \rho_k$ **then**
$\mathbf{x}_{k+1} \leftarrow \widehat{\mathbf{x}}_{k+1}$;
$\mathbf{g}_{k+1} \leftarrow \widehat{\mathbf{g}}_{k+1}$;
$f_{k+1} \leftarrow \widehat{f}_{k+1}$;
else
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$;
end if
if $c_2 < \rho_k$ **then**
if $\|\mathbf{s}_k\|_2 \leq c_3 \delta_k$ **then**
$\delta_k \leftarrow \delta_k$;
else
$\delta_k \leftarrow c_4 \delta_k$;
end if
else if $c_5 \leq \rho_k \leq c_6$ **then**
$\delta_k \leftarrow \delta_k$;
else
$\delta_k \leftarrow c_7 \delta_k$;
end if
$\mathbf{y}_k \leftarrow \widehat{\mathbf{g}}_{k+1} - \mathbf{g}_k$;
$b_k \leftarrow \mathbf{s}_k^T \mathbf{y}_k + (sBs)_k$;
if $\varepsilon_{\mathrm{SR1}} \|\mathbf{s}_k\|_2 \|\mathbf{y}_k - \gamma_k \mathbf{s}_k\|_2 \leq \mathrm{abs}(b_k)$ **then**
if INIT = C.Init. **then**
[Ψ_k, mIdx] = colUpdate(Ψ_k, $\mathbf{y}_k - \gamma_k \mathbf{s}_k$, mIdx, m, k);
if $(k_m < m)$ **then**
$k_m \leftarrow k_m + 1$;
end if
$\mathrm{invM}_k(1:(k_m-1), k_m) \leftarrow \mathbf{b}_k(1:(k_m-1))$;
$\mathrm{invM}_k(k_m, 1:(k_m-1)) \leftarrow \mathbf{b}_k(1:(k_m-1))$;
$\mathrm{invM}_k(k_m, k_m) \leftarrow b_k$;
if SAVE = 1 **then**
% Update and store the product $\Psi_k^T \Psi_k$
$\Psi\Psi_k(1:k_m, 1:k_m)$ = prodUpdate($\Psi\Psi_k$, Ψ_k, Ψ_k, $\mathbf{y}_k - \gamma_k \mathbf{s}_k$, $\mathbf{y}_k - \gamma_k \mathbf{s}_k$, mIdx, m, k);
end if
else
% Non-constant initialization
[Y_k, ~] = colUpdate(Y_k, $\mathbf{y}_k$, mIdx, m, k);
[S_k, mIdx] = colUpdate(S_k, $\mathbf{s}_k$, mIdx, m, k);
$\mathbf{T}_k$ = prodUpdate($\mathbf{T}_k$, S_k, 0, $\mathbf{s}_k$, $\mathbf{y}_k$, mIdx, m, k);
YY_k = prodUpdate(YY_k, Y_k, Y_k, $\mathbf{y}_k$, $\mathbf{y}_k$, mIdx, m, k);
if $(k_m < m)$ **then**
$k_m \leftarrow k_m + 1$;
end if
$\mathbf{D}_k(k_m, k_m) \leftarrow \mathbf{s}_k^T \mathbf{y}_k$;
$\mathbf{L}_k(k_m, 1:(k_m-1)) \leftarrow \mathbf{b1}_k(1:(k_m-1))$;
$\mathrm{SS}_k(1:(k_m-1), k_m) \leftarrow \mathbf{b2}_k(1:(k_m-1))$;
$\mathrm{SS}_k(k_m, 1:(k_m-1)) \leftarrow \mathbf{b2}_k(1:(k_m-1))$;
$\mathrm{SS}_k(k_m, k_m) \leftarrow \mathbf{s}_k^T \mathbf{s}_k$;
$\mathrm{invM}_k(1:k_m, 1:k_m) \leftarrow \mathbf{D}_k(1:k_m, 1:k_m) + \mathbf{L}_k(1:k_m, 1:k_m) + \mathbf{L}_k(1:k_m, 1:k_m)^T - \gamma_k \mathrm{SS}_k(1:k_m, 1:k_m)$;
if SAVE = 1 **then**
% Update and store the product $\Psi_k^T \Psi_k$ with non-constant initialization

105: $\quad\quad\quad\quad \Psi\Psi_k(1:k_m, 1:k_m) = \mathrm{YY}_k(1:k_m, 1:k_m) - \gamma_k(\mathbf{T}_k(1:k_m, 1:k_m) + \mathbf{T}_k(1:k_m, 1:k_m)^T$
$\quad\quad\quad\quad + \mathbf{L}_k(1:k_m, 1:k_m) + \mathbf{L}_k(1:k_m, 1:k_m)^T) + \gamma_k^2 \mathrm{SS}_k(1:k_m, 1:k_m)$;
106: **end if**
107: **end if**
108: **end if**
109: $\mathbf{x}_k \leftarrow \mathbf{x}_{k+1}, \mathbf{g}_k \leftarrow \mathbf{g}_{k+1}, f_k \leftarrow f_{k+1}, k \leftarrow k+1$;
110: **end while**
111: out.numiter $\leftarrow k$, out.ng $\leftarrow \|\mathbf{g}_k\|$;
112: **return** $\mathbf{x}_k, \mathbf{g}_k, f_k$, out

A.2 Experiments to Determine Default Parameters with Non-constant Initialization (Init. 2)

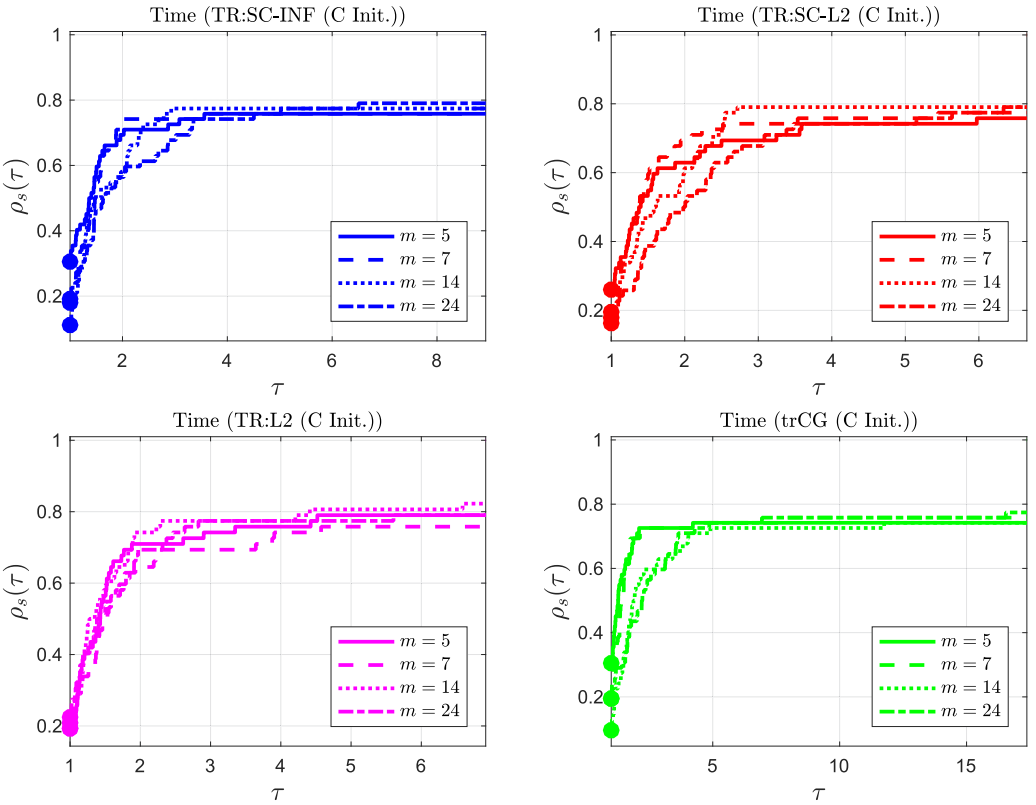

Fig. 6. Comparison of the computational times for the four algorithms {TR:SC-INF, TR:SC-L2, TR:L2, trCG} when a constant initialization (C Init.) is used, and the limited memory parameter is $m = [5, 7, 14, 24]$.

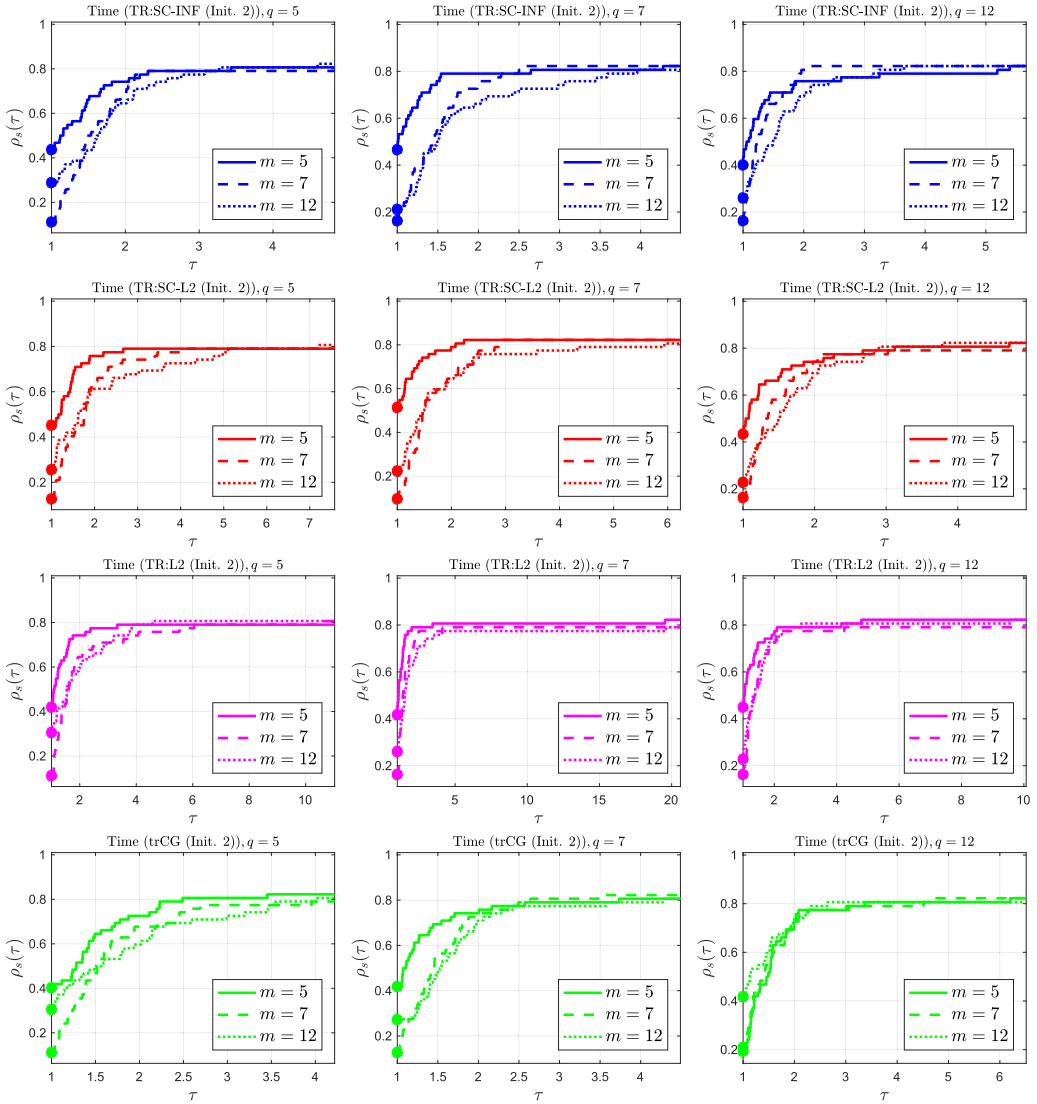

Fig. 7. Comparison of the computational times for the four algorithms {TR:SC-INF, TR:SC-L2, TR:L2, trCG} when the non-constant initialization (Init. 2) is used, and the parameters are $q = [5, 7, 12]$ and $m = [5, 7, 12]$.

A.3 Experiments on Quadratics and the Rosenbrock Functions

In this set of experiments we vary the problem dimension as $n = [5 \times 10^2, 1 \times 10^3, 5 \times 10^3, 1 \times 10^4, 5 \times 10^4, 1 \times 10^5, 3 \times 10^5]$, set the memory parameter $m = 5$, use Init. 2 for all solvers and set the maximum iterations as maxIt = 500. In Table 8, we let $f(\mathbf{x})$ be the Rosenbrock function defined by $f(\mathbf{x}) = \sum_{i=1}^{n}(\mathbf{x}_{2i} - \mathbf{x}_{2i-1}^2)^2 + (1 - \mathbf{x}_{2i-1}^2)^2$. We initialize the trust-region algorithm (Algorithm 5) from the starting point $[\mathbf{x}_0]_1 = 30, [\mathbf{x}_0]_{2:n} = 0$ (with this initial point the gradient norm $\|\nabla f(\mathbf{x}_0)\|_2 \approx 10^5$). Table 8 reports the outcomes of using the trust-region algorithm with these three different subproblem solvers.

In Table 9, we let $f(\mathbf{x})$ be quadratic functions defined by $f(\mathbf{x}) = \mathbf{g}^T\mathbf{x} + \frac{1}{2}(\mathbf{x}^T(\phi\mathbf{I} + \mathbf{Q}\mathbf{D}\mathbf{Q}^T)\mathbf{x})$. In particular, we let $\mathbf{Q} \in \mathbb{R}^{n\times r}$ be a rectangular matrix and $\mathbf{D} \in \mathbb{R}^{r\times r}$ be a diagonal matrix. We

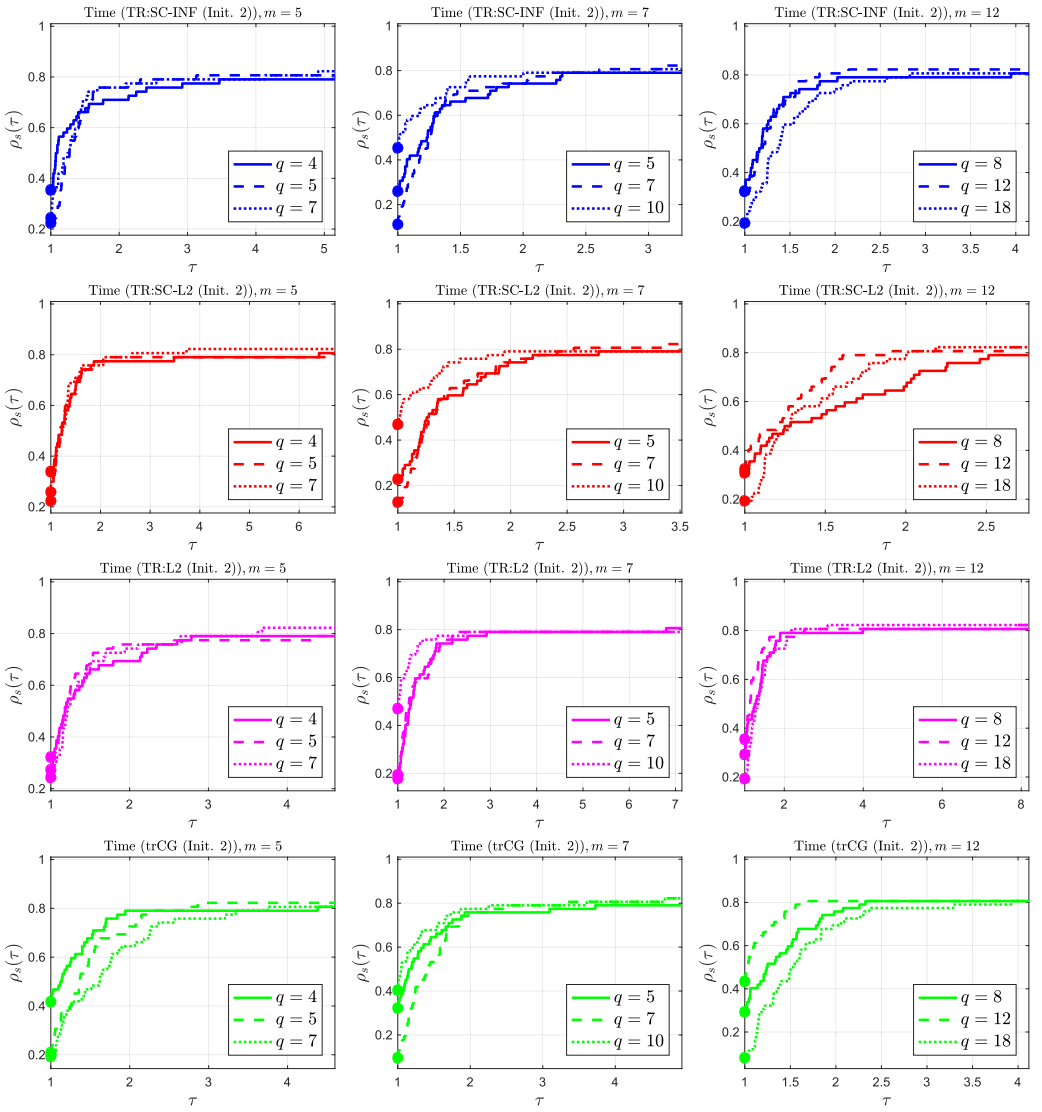

Fig. 8. Comparison of the computational times for the four algorithms {TR:SC-INF, TR:SC-L2, TR:L2, trCG} when the non-constant initialization (Init. 2) is used, and the parameters are $m = [5, 7, 12]$ and $q = [\texttt{ceil}(2/3 \cdot m_i), m_i, \texttt{floor}(3/2 \cdot m_i)], 1 \leq i \leq 3$.

initialize the trust-region algorithm (Algorithm 5) from the starting point $\mathbf{x}_0 = \mathbf{0}$. We generate $\mathbf{Q} = \texttt{rand}(n, r)$, $\mathbf{D} = \texttt{diag}(\texttt{rand}(r, 1))$, and $\mathbf{g} = \texttt{randn}(n, 1)$, after initializing the random number generator by the command `rng('default')`. Moreover, we set $r = 10$, $\phi = 100$ and the maximum number of iterations as maxIt = 500. All other parameters of the method are as before. Table 9 reports the outcomes of using the trust-region algorithm with the three different subproblem solvers.

Remarkably, observe in the outcomes of Tables 8 and 9 that a limited memory trust-region algorithm using our subproblem solvers is able to solve large optimization problems, with $n \approx 1 \times 10^5$, within seconds. Moreover, we observe that the proposed algorithms (Algorithms 3 and 4) may require fewer iterations on some problems than a ℓ_2-norm method and use less computational

Table 8. Results of Solving Problem (42) with the Rosenbrock Objective Function

n	TR:SC-INF (Alg. 3)				TR:SC-L2 (Alg. 4)				TR-L2			
	k	nA	Time	$\|\nabla f(\mathbf{x}_k)\|$	k	nA	Time	$\|\nabla f(\mathbf{x}_k)\|$	k	nA	Time	$\|\nabla f(\mathbf{x}_k)\|$
5×10^2	40	26	2.00e-02	7.18e-05	46	29	1.59e-02	3.11e-05	36	24	1.34e-02	2.89e-06
1×10^3	38	24	1.13e-02	2.23e-05	41	24	1.28e-02	3.83e-05	32	22	1.14e-02	2.02e-05
5×10^3	42	31	3.02e-02	1.17e-05	38	29	2.75e-02	6.38e-05	43	26	4.26e-02	5.03e-05
1×10^4	46	30	5.22e-02	4.57e-07	40	28	4.20e-02	5.80e-05	48	29	6.30e-02	8.87e-05
5×10^4	47	33	2.14e-01	1.01e-06	39	28	1.73e-01	1.22e-05	54	35	2.85e-01	6.92e-05
1×10^5	40	31	3.94e-01	6.82e-05	58	39	4.81e-01	1.06e-05	44	27	4.97e-01	1.57e-08
3×10^5	60	39	2.74e+00	1.63e-06	53	33	2.49e+00	3.52e-06	68	43	3.53e+00	1.70e-05

The maximum number of iterations are maxIt = 500 and the convergence tolerance is $\|\nabla f(\mathbf{x}_k)\|_\infty \leq 1 \times 10^{-4}$. The memory parameter is $m = 5$. The column nA denotes the number of "accepted" search directions, which corresponds to line 55 in Algorithm 5 being true. Observe that all algorithms converged to the prescribed tolerances on all problem instances.

Table 9. Results of Solving Problem (42) with Quadratic Objective Functions

n	TR:SC-INF (Alg. 3)				TR:SC-L2 (Alg. 4)				TR:L2 (ℓ_2 [Brust et al. 2017])			
	k	nA	Time	$\|\nabla f(\mathbf{x}_k)\|$	k	nA	Time	$\|\nabla f(\mathbf{x}_k)\|$	k	nA	Time	$\|\nabla f(\mathbf{x}_k)\|$
5×10^2	8	6	5.75e-02	6.56e-06	8	6	2.66e-02	6.56e-06	6	4	2.89e-02	8.03e-06
1×10^3	8	6	7.25e-03	4.06e-05	8	6	7.51e-03	4.06e-05	6	4	6.67e-03	5.11e-05
5×10^3	21	15	2.47e-02	8.96e-05	21	15	2.83e-02	9.14e-05	16	10	2.79e-02	3.71e-05
1×10^4	23	18	3.86e-02	7.79e-05	23	18	3.65e-02	5.21e-05	19	14	3.65e-02	4.28e-05
5×10^4	45	33	2.16e-01	1.58e-05	60	46	2.28e-01	9.71e-05	27	21	1.20e-01	9.13e-05
1×10^5	62	49	5.04e-01	9.80e-05	79	64	5.86e-01	9.72e-05	500	494	4.05e+00	4.09e-04
3×10^5	20	15	8.99e-01	3.49e-05	22	17	8.37e-01	3.86e-05	26	17	1.11e+00	9.97e-05

The maximum number of iterations are set as maxIt = 500 and the convergence tolerance $\|\nabla f(\mathbf{x}_k)\|_\infty \leq 1 \times 10^{-4}$. The memory parameter is $m = 5$. The column nA denotes the number of "accepted" search directions (line 55 in Algorithm 5 is true). Observe that Algorithms 3 and 4 converged on all problems. Moreover, Algorithms 3 and 4 were fastest on the on the largest two problem instances.

time. Future research, can investigate the effectiveness of a L-SR1 trust-region algorithm for non-convex objective functions and improve on the efficiency of the implementation.

REFERENCES

J. Barzilai and J. Borwein. 1988. Two-point step size gradient methods. *IMA J. Numer. Anal.* 8, 1 (01 1988), 141–148. https://doi.org/10.1093/imanum/8.1.141

J. M. Bennett. 1965. Triangular factors of modified matrices. *Numer. Math.* 7, 3 (1965), 217–221.

J. J. Brust, O. Burdakov, J. B. Erway, and R. F. Marcia. 2019. A dense initialization for limited-memory quasi-newton methods. *Comput. Optim. Appl.* 74 (2019), 121–142.

J. J. Brust. 2018. *Large-Scale Quasi-Newton Trust-Region Methods: High-Accuracy Solvers, Dense Initializations, and Extensions.* Ph.D. Dissertation. University of California, Merced.

J. J. Brust, J. B. Erway, and R. F. Marcia. 2017. On solving L-SR1 trust-region subproblems. *Comput. Optim. Appl.* 66, 2 (2017), 245–266.

J. J. Brust, R. F. Marcia, and C. G. Petra. 2019. Large-scale quasi-newton trust-region methods with low dimensional linear equality constraints. *Comput. Optim. Appl.* 74 (2019), 669–701.

Oleg Burdakov, Lujin Gong, Spartak Zikrin, and Ya-xiang Yuan. 2017. On efficiently combining limited-memory and trust-region techniques. *Math. Program. Comput.* 9, 1 (2017), 101–134.

O. Burdakov, S. Gratton, Y.-X. Yuan, and S. Zikrin. 2018. LMTR Suite for Unconstrained Optimization. Retrieved from http://gratton.perso.enseeiht.fr/LBFGS/index.html.

O. Burdakov and Y.-X. Yuan. 2002. On limited-memory methods with shape changing trust region. In *Proceedings of the 1st International Conference on Optimization Methods and Software.* p. 21.

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

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. 1991. Convergence of quasi-newton matrices generated by the symmetric rank one update. *Math. Program.* 50, 2 (1991), 177–195. https://doi.org/10.1007/BF01594934

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

E. Dolan and J. J. Moré. 2002. Benchmarking optimization software with performance profiles. *Math. Program.* 91 (2002), 201–213.

J. B. Erway and P. E. Gill. 2010. A subspace minimization method for the trust-region step. *SIAM J. Optim.* 20, 3 (2010), 1439–1461. https://doi.org/10.1137/08072440X

J. B. Erway, P. E. Gill, and J. D. Griffin. 2009. Iterative methods for finding a trust-region step. *SIAM J. Optim.* 20, 2 (2009), 1110–1131. https://doi.org/10.1137/070708494

J. B. Erway and R. F. Marcia. 2014. Algorithm 943: MSS: MATLAB software for L-BFGS trust-region subproblems for large-scale optimization. *ACM Trans. Math. Softw.* 40, 4 (June 2014), 28:1–28:12. http://doi.acm.org/10.1145/2616588

J. B. Erway and R. F. Marcia. 2015. On efficiently computing the eigenvalues of limited-memory quasi-newton matrices. *SIAM J. Matrix Anal. Appl.* 36, 3 (2015), 1338–1359. https://doi.org/10.1137/140997737

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

P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. 1974. Methods for modifying matrix factorizations. *Math. Comp.* 28, 126 (1974), 505–535.

D. Goldfarb. 1980. *The Use of Negative Curvature in Minimization Algorithms*. Technical Report 80-412. Cornell University.

G. H. Golub and C. F. Van Loan. 1996. *Matrix Computations* (3 ed.). The Johns Hopkins University Press, Baltimore, MD.

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

N. I. M. Gould, D. P. Robinson, and H. S. Thorne. 2010. On solving trust-region and other regularised subproblems in optimization. *Math. Program. Comput.* 2, 1 (2010), 21–57.

I. Griva, S. G. Nash, and A. Sofer. 2009. *Linear and Nonlinear Programming*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

C. T. Kelley and E. W. Sachs. 1998. Local convergence of the symmetric rank-one iteration. *Comput. Optim. Appl.* 9, 1 (1998), 43–63. https://doi.org/10.1023/A:1018330119731

H. Fayez Khalfan, R. H. Byrd, and R. B. Schnabel. 1993. A theoretical and experimental study of the symmetric rank-one update. *SIAM J. Optim.* 3, 1 (1993), 1–24. https://doi.org/10.1137/0803001

X. Lu. 1996. *A Study of the Limited Memory SR1 Method in Practice*. Ph.D. Dissertation. Department of Computer Science, University of Colorado at Boulder.

A. Mahajan, S. Leyffer, and C. Kirches. 2012. *Solving Mixed-Integer Nonlinear Programs by QP Diving*. Technical Report ANL/MCS-P2071-0312. Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL.

J. J. Moré and D. C. Sorensen. 1983. Computing a trust region step. *SIAM J. Sci. Statist. Comput.* 4 (1983), 553–572.

K. G. Murty and S. N. Kabadi. 1987. Some NP-complete problems in quadratic and nonlinear programming. *Math. Program.* 39, 2 (1987), 117–129.

J. Nocedal. 1980. Updating quasi-newton matrices with limited storage. *Math. Comput.* 35 (1980), 773–782.

J. Nocedal and S. J. Wright. 2006. *Numerical Optimization* (2nd ed.). Springer, New York.

M. Rojas, S. A. Santos, and D. C. Sorensen. 2008. Algorithm 873: LSTRS: MATLAB software for large-scale trust-region subproblems and regularization. *ACM Trans. Math. Softw.* 34, 2 (2008), 1–28.

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

T. Steihaug. 1983. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.* 20 (1983), 626–637.

W. Sun and Y.-X. Yuan. 2006. *Optimization Theory and Methods*. Springer Optimization and Its Applications, Vol. 1. Springer, New York.

B. Vavasis. 1992. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, Oxford, UK.

H. Wolkowicz. 1994. Measures for symmetric rank-one updates. *Math. Operat. Res.* 19, 4 (1994), 815–830.

Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.* 23, 4 (December 1997), 550–560. https://doi.org/10.1145/279232.279236

Received 18 May 2021; revised 5 February 2022; accepted 9 June 2022