LSTRS: MATLAB Software for Large-Scale Trust-Region Subproblems and Regularization

Marielba Rojas* Sandra A. Santos †
Danny C. Sorensen ‡

August 26, 2003. Revised September 27, 2006.

IMM-Technical Report-2006-26
Informatics and Mathematical Modelling,
Technical University of Denmark, Kgs. Lyngby, Denmark.

A MATLAB 6.0 implementation of the LSTRS method is presented. LSTRS was described in M. Rojas, S.A. Santos and D.C. Sorensen, A new matrix-free method for the large-scale trust-region subproblem, *SIAM J. Optim.*, 11(3):611-646, 2000. LSTRS is designed for large-scale quadratic problems with one norm constraint. The method is based on a reformulation of the trust-region subproblem as a parameterized eigenvalue problem, and consists of an iterative procedure that finds the optimal value for the parameter. The adjustment of the parameter requires the solution of a large-scale eigenvalue problem at each step. LSTRS relies on matrix-vector products only

*Informatics and Mathematical Modelling, Technical University of Denmark, Building 305, Kgs. Lyngby, Denmark (mr@imm.dtu.dk). This author was supported in part by NSF cooperative agreement CCR-9120008, the Research Council of Norway, and the Science Research Fund of Wake Forest University.

†Department of Applied Mathematics, State University of Campinas, CP 6065, 13081-970, Campinas, SP, Brazil (sandra@ime.unicamp.br). This author was supported by FAPESP (93/4907-5 and 01/04597-4), CNPq, FINEP and FAEP-UNICAMP.

‡Department of Computational and Applied Mathematics, Rice University, 6100 Main St., Houston, TX 77005-1892, USA (sorensen@caam.rice.edu). This author was supported in part by NSF Grant CCR-9988393 and in part by the Los Alamos National Laboratory Computer Science Institute (LACSI) through LANL contract number 03891-99-23, as part of the prime contract (W-7405-ENG-36) between the Department of Energy and the Regents of the University of California.

and has low and fixed storage requirements, features that make it suitable for large-scale computations. In the MATLAB implementation, the Hessian matrix of the quadratic objective function can be specified either explicitly, or in the form of a matrix-vector multiplication routine. Therefore, the implementation preserves the matrix-free nature of the method. A description of the LSTRS method and of the MATLAB software, version 1.2, is presented. Comparisons with other techniques and applications of the method are also included. A guide for using the software and examples are provided.

AMS Classification: 65K05,65F22,65-04.

Keywords: Regularization, constrained quadratic optimization, trust region, Lanczos method, MATLAB, ARPACK.

General Terms: Regularization, Optimization, Software.

1 Introduction

We describe version 1.2 of a MATLAB [22] 6.0 implementation of the LSTRS method [33] for large-scale quadratic problems with a quadratic constraint, or trust-region subproblems:

$$\min \quad \frac{1}{2}x^T H x + g^T x \quad \text{subject to (s.t.)} \quad \|x\| \leq \Delta, \tag{1}$$

where H is an $n \times n$, real, symmetric matrix, g is an n-dimensional real vector, and Δ is a positive scalar. In (1), and throughout the paper, $\|\cdot\|$ denotes the Euclidean norm. The following notation is also used throughout the paper: δ_1 denotes the algebraically smallest eigenvalue of H, $\mathcal{S}_1 \equiv \mathcal{N}(H - \delta_1 I)$ denotes the corresponding eigenspace, $\mathcal{N}(\cdot)$ denotes the nullspace of a matrix, and $\dagger$ denotes the pseudoinverse.

Problem (1) arises in connection with the trust-region globalization strategy in optimization. A special case of problem (1), namely, a least squares problem with a norm constraint, is equivalent to Tikhonov regularization [42] for discrete forms of ill-posed problems. The Lagrange multiplier associated with the constraint is the Levenberg-Marquardt parameter in optimization and the Tikhonov parameter in regularization. A constraint of the form $\|Cx\| \leq \Delta$ for a matrix $C \neq I$ is not considered in this work. The matrix C can be used, for example, as a scaling matrix in optimization or to impose a

smoothness condition on the solution in regularization. Note that when C is nonsingular, a change of variables can be used to reduce the problem to the case we are considering.

The trust-region subproblem has very interesting theoretical properties that lead to the design of efficient solution methods. In particular, if it is possible to compute the Cholesky factorization of matrices of the form $H - \lambda I$, the method of choice is probably the one proposed by Moré and Sorensen in [23]. The algorithm uses Newton's method to find a root of a scalar function that is almost linear on the interval of interest. The authors also proposed a computationally-efficient strategy for dealing with a special and usually difficult case, known since then in the optimization literature as the *hard case*. The hard case is discussed in detail in Section 2.

If the matrix H is very large or not explicitly available, factoring or even forming the matrices $H - \lambda I$ may be prohibitive and a different approach is needed to solve the problem. Possibly, the most popular method for the large-scale trust-region subproblem is the one of Steihaug [40] and Toint [43]. The method computes the solution to the problem in a Krylov space and is efficient in conjunction with optimization methods. An improvement upon the Steihaug-Toint's approach, based on the truncated Lanczos idea, was proposed by Gould et al. in [11]. Hager in [13] adopts an SQP approach to solve the trust-region subproblem in a special Krylov subspace. New properties of the trust-region subproblem that provide useful tools for the development of new classes of algorithms in the large-scale scenario are presented by Lucidi et al. [21]. Other authors that have considered large-scale problems are Golub and von Matt [10], Sorensen [39], Rendl and Wolkowicz [31] (revisited by Fortin and Wolkowicz in [7]), Rojas et al. [33] and Pham Dinh and Le Thi [30]. The theory of Gauss quadrature, matrix moments and Lanczos diagonalization is used in [10] to compute bounds for the optimal Lagrange multiplier and solution. The hard case is not analyzed in [10]. The algorithm in [30] is based on differences of convex functions, and is inexpensive due to its projective nature. However, a restarting mechanism is needed in order to guarantee convergence to a global solution. The approaches in [31], [33], and [39] recast the trust-region subproblem as a parameterized eigenvalue problem and design an iteration to find an optimal value for the parameter. A primal-dual semidefinite framework is proposed in [31], with a dual simplex-type method for the basic iteration and a primal simplex-type method for the hard-case iteration. In [33] and [39], two different rational

interpolation schemes are used for the update of the scalar parameter. In [39], a superlinearly-convergent scheme is developed for the adjustment of the parameter, as long as the hard case does not occur. In the presence of the hard case, the algorithm in [39] is linearly convergent. In [33], a unified iterative scheme is proposed which converges superlinearly in all cases.

It is possible to classify methods for the trust-region subproblem based on the properties of the computed solution. We will call an approximation to an "optimal" solution of problem (1) (see Section 2.1), a *nearly-exact solution*, and any other approximation, an *approximate solution*. Accordingly, we can make a distinction between nearly-exact methods and approximate methods. The methods in [7, 10, 23, 30, 31, 33, 39] are nearly exact, while the methods in [11, 13, 40, 43] are approximate. Approximate solutions (and methods) are of particular interest in the context of trust-region methods for optimization. In regularization, nearly exact solutions are often required.

In this paper, we describe a set of MATLAB 6.0 routines implementing the nearly-exact method LSTRS from [33]. LSTRS is suitable for large-scale computations since it relies on matrix-vector products only and has low and fixed storage requirements. As mentioned above, LSTRS is based on a reformulation of problem (1) as a parameterized eigenvalue problem. The goal of the method is to compute the optimal value for a scalar parameter, which is then used to compute a solution for problem (1). The method requires the solution of an eigenvalue problem at each step. LSTRS can handle all instances of the trust-region subproblem, including those arising in the regularization of ill-posed problems. The method has been successfully used for computing regularized solutions of large-scale inverse problems in several areas (see [6, 32, 34, 35]).

The MATLAB implementation of LSTRS described in this paper allows the user to specify the matrix H both explicitly, a feature that can be useful for small test problems, and implicitly, in the form of a matrix-vector multiplication routine, hence preserving the matrix-free nature of the original method. Several options are available for the solution of the eigenvalue problems, namely: the MATLAB routine `eig` (QR method), a slightly modified version of `eigs` (a MEX-file interface for ARPACK [20]), a combination of `eigs` with a Tchebyshev Spectral Transformation as in [34], or a user-provided routine.

The paper is organized as follows. In Section 2, we present the properties of the trust-region subproblem and its connection with regularization. In

Section 3, we describe the method LSTRS from [33]. In Section 4, we describe the main aspects of the software: data structures, interface and components, as well as the instructions for installing and running the software. We discuss the use of the software for regularization problems in Section 5. In Section 6, we present comparisons of LSTRS with other methods for the large-scale trust-region subproblem. In Section 7, we discuss the use of LSTRS on large-scale problems and present an application from image restoration. In Section 8, we illustrate the use of the software with several examples.

2 Trust Regions and Regularization

In this section, we describe the trust-region subproblem as well as its connection with the regularization of discrete forms of ill-posed problems. We present the properties of the trust-region subproblem in Section 2.1 and discuss regularization issues in Section 2.2.

2.1 The structure of the trust-region subproblem

The trust-region subproblem always has a solution, which lies either in the interior or on the boundary of the (feasible) set $\{x \in \mathbb{R}^n,\ \|x\| \leq \Delta\}$. A characterization of the solutions of problem (1), found independently by Gay [8] and Sorensen [37], is given in the following lemma where we have followed [39] in the non-standard but notationally more convenient use of a non-positive multiplier.

Lemma 2.1 ([37]) *A feasible vector $x_* \in \mathbb{R}^n$ is a solution to (1) with corresponding Lagrange multiplier λ_* if and only if x_*, λ_* satisfy $(H - \lambda_* I)x_* = -g$ with $H - \lambda_* I$ positive semidefinite, $\lambda_* \leq 0$ and $\lambda_*(\Delta - \|x_*\|) = 0$.*

Proof. For the proof see [37]. □

Lemma 2.1 implies that all solutions to the trust-region subproblem are of the form $x = -(H - \lambda I)^\dagger g + z$ for $z \in \mathcal{N}(H - \lambda I)$. If the Hessian matrix H is positive definite and $\|H^{-1}g\| < \Delta$, problem (1) has a unique interior solution given by $x = -H^{-1}g$, with Lagrange multiplier $\lambda = 0$. If the Hessian is positive semidefinite or indefinite, there exist boundary solutions satisfying $\|x\| = \Delta$ with $\lambda \leq \delta_1$.

The case $\lambda = \delta_1$ is usually called the *hard case* in the literature (cf. [23]). The hard case can only occur when $\delta_1 \leq 0$, $g \perp \mathcal{S}_1$ and $\|(H - \delta_1 I)^{\dagger} g\| \leq \Delta$. For most problems of interest, solving the trust-region problem in the hard case can be an expensive and difficult task since it requires the computation of an approximate eigenvector associated with the smallest eigenvalue of H. Moreover, in practice g will be nearly orthogonal to $\mathcal{S}_1$ and we can expect greater numerical difficulties in this case. As in [32, 34], we call this situation a *near hard case*. Note that whenever g is nearly orthogonal to $\mathcal{S}_1$ there is the possibility for the hard case or near hard case to occur, depending on the value of Δ. Therefore we call this situation a *potential hard case*.

The occurrence of the exact, near or potential hard case is structural, i.e. it depends on the relationship between the matrix H, the vector g and the scalar Δ. Although not too common in optimization, the near hard case is rather frequent in regularization. Indeed, it was shown in [32, 34] that the potential hard case is precisely the common case for discrete forms of ill-posed problems, where it occurs in a multiple instance in which the vector g is orthogonal or nearly orthogonal to several eigenspaces of H. We discuss these issues in Section 2.2.

2.2 The trust-region approach to regularization

In this section, we first describe the properties of discrete forms of ill-posed problems and show how they lead to the use of regularization. We then discuss the connection of trust regions and regularization. Finally, we describe the properties of the trust-region subproblem in the regularization context.

Discrete forms of *linear* ill-posed problems consist of linear systems or linear least squares problems in which the coefficient matrices come from the discretization of the continuous operator in an ill-posed problem and the right-hand side contains experimental data contaminated by noise. The discretization of continuous problem in inversion (eg. [1, 24, 27, 41]) usually lead to highly ill-conditioned problems, called discrete forms of ill-posed problems or discrete ill-posed problems in the literature. Reasonably accurate discretizations will produce coefficient matrices whose properties are the discrete analogs of those of the continuous operators. In particular, the matrices will be highly ill-conditioned with singular spectra that decay to zero gradually with no particular gap, and will have a large cluster of very small singular values [17]. Moreover, as observed in [17], the high-frequency com-

ponents (those with more sign changes) of the singular vectors will usually correspond to the smallest singular values.

We consider the problem of recovering x_{LS}, the minimum-norm solution to

$$\begin{aligned} \min \quad & \|Ax - b\| \\ x \in \mathbb{R}^n & \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $m \geq n$, when the *exact* data vector b is not known, and instead, only a *perturbed* data vector $\bar{b}$ is available. Specifically, we regard $\bar{b}$ as $\bar{b} = b + s$, where s is a random vector of uncorrelated noise. Considering that only $\bar{b}$ is available, we could try to approximate x_{LS} by $\bar{x}_{LS}$, the minimum-norm solution to

$$\begin{aligned} \min \quad & \|Ax - \bar{b}\|. \\ x \in \mathbb{R}^n & \end{aligned} \tag{2}$$

Unfortunately, as we now show, the two solutions might differ considerably.

Let $A = U\Sigma V^T$ be a Singular Value Decomposition (SVD) of A, where $U \in \mathbb{R}^{m \times n}$ has orthormal columns, $V \in \mathbb{R}^{n \times n}$ is orthogonal, and Σ is a diagonal matrix with elements $\sigma_1, \sigma_2, \ldots, \sigma_n$. The σ_i's are the singular values of A in non-increasing order. The solution of problem (2) in terms of the SVD of A is given by:

$$\bar{x}_{LS} = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i + \sum_{i=1}^{n} \frac{u_i^T s}{\sigma_i} v_i. \tag{3}$$

As usual in the analysis of discrete forms of ill-posed problems, we assume that the Discrete Picard Condition (DPC) [15] holds, i.e. that the values $|u_i^T b|$ overall decay to zero faster than σ_i as the index i increases. Assuming that the DPC holds, the first term in the right-hand side of (3) is bounded. However, the second term might become very large since the expansion coefficients of the uncorrelated noise vector ($u_i^T s$) remain constant while the singular values decay to zero. Therefore, the components of $\bar{x}_{LS}$ corresponding to small singular values are magnified by the noise and $\bar{x}_{LS}$ might be dominated by the high-frequency components. Consequently, standard methods such as those in [2], [9, Ch. 5] and [19] applied to problem (2) usually produce meaningless solutions with very large norm. Note that even in the noise-free case, the ill conditioning of the matrix A will pose difficulties to most

numerical methods. Therefore, to solve these problems, special techniques known as *regularization* are needed.

In regularization, we aim to recover an approximation to the desired solution of the unknown problem with exact data from the solution of a better-conditioned problem that is related to the problem with noisy data but incorporates additional information about the desired solution. The conditioning of the new problem depends on the choice of a special parameter known as the *regularization parameter*. Excellent surveys on regularization methods can be found for example in [14], [17] and [25].

One of the most popular regularization approaches is Tikhonov regularization [42], which consists of adding a penalty term to problem (2) to obtain:

$$\min_{x \in \mathbb{R}^n} \quad \|Ax - \bar{b}\|^2 + \mu\|x\|^2, \tag{4}$$

where $\mu > 0$ is the Tikhonov regularization parameter. It is well known (cf. [5, 34]) that this approach is equivalent to a special instance of the trust-region subproblem, namely, to a least squares problem with a quadratic constraint:

$$\min \ \|Ax - \bar{b}\|^2 \ \text{ s.t. } \ \|x\| \leq \Delta, \tag{5}$$

where $H = A^T A$ and $g = -A^T\bar{b}$. Therefore, in principle, methods for the trust-region subproblem could be used to solve regularization problems of type (5), where instead of specifying a value for the Tikhonov parameter as required for (4), we need to prescribe a bound on the norm of the desired solution. However, as we shall see, the trust-region subproblem (5) has special properties in the regularization context and these properties should be taken into consideration when developing solution methods. The following analysis is based on [32] and [34].

We now show that the potential (near) hard case is the common case for ill-posed problems, where it occurs in a *multiple* instance, with g nearly orthogonal to the eigenpaces associated with *several* of the smallest eigenvalues of H. This was first shown in [32]. Assume that the singular values of A are not zero and that σ_n, the smallest singular value, has multiplicity k. Let $n - k + 1 \leq i \leq n$ and let v_i be a right-singular vector of A associated with σ_n. Then:

$$g^T v_i \ = \ -\bar{b}^T U \Sigma V^T v_i = -\sigma_n u_i^T \bar{b} = -\sigma_n (u_i^T b + u_i^T s).$$

If there is no noise in the data ($s = 0$) and if the DPC holds, the coefficients $u_i^T b$, for $n-k+1 \leq i \leq n$, are small and since σ_n is also small, it follows that g is nearly orthogonal to v_i in this case. For noisy data, $g^T v_i$ might not be small due to the possible contribution of the term $u_i^T s$. However, for severely ill-conditioned problems, the smallest singular value σ_n is so close to zero that even if $u_i^T s$ is large, g will still be nearly orthogonal to v_i. Since v_i is an eigenvector corresponding to $\delta_1 = {\sigma_n}^2$, the smallest eigenvalue of $A^T A$, we have that g will be nearly orthogonal to the eigenspace corresponding to the smallest eigenvalue and therefore, the potential (near) hard case will occur.

Observe that in ill-posed problems, the matrix A usually has a large cluster of singular values very close to zero. Therefore, following the previous argument, we see that the vector g will be orthogonal or nearly orthogonal to the eigenspaces corresponding to several of the smallest eigenvalues of the matrix $A^T A$, and the potential hard case will occur in a multiple instance. The numerical experimentation presented in [32, 34] indicates that the algoritm LSTRS can efficiently handle the multiple instances of orthogonality (or near orthogonality) based on the complete characterization of the hard case given in [32].

3 The LSTRS method

In this section, we present a description of the LSTRS method with special emphasis on the computational aspects. For more details, as well as for the theoretical foundations and the convergence properties of the method, we refer the reader to [32, 33].

LSTRS is based on a reformulation of the trust-region subproblem (1) as a parameterized eigenvalue problem. The new formulation is based on the fact that there exists a value of a scalar parameter α such that problem (1) is equivalent to:

$$\begin{array}{ll} \min & \frac{1}{2} y^T B_\alpha y \\ \text{s.t.} & y^T y \leq 1 + \Delta^2, \; e_1^T y = 1, \end{array} \tag{6}$$

where B_α is the *bordered matrix* $B_\alpha = \begin{pmatrix} \alpha & g^T \\ g & H \end{pmatrix}$, and e_1 is the first canonical vector in $\mathbb{R}^{n+1}$. The optimal value for α is given by $\alpha_* = \lambda_* - g^T x_*$, with $\lambda_*, \; x_*$ the optimal pair in Lemma 2.1. Observe that if we knew α_*,

we could compute a solution to the trust-region subproblem from the algebraically smallest eigenvalue of B_{α_*} and a corresponding eigenvector with special structure. The solution would consist of the last n components of the eigenvector and the Lagrange multiplier would be the eigenvalue. LSTRS starts with an initial guess for α and iteratively adjusts this parameter toward the optimal value. This is accomplished by solving a sequence of eigenvalue problems for B_α, for different α's, as we now show.

Let α be a scalar, let λ be the algebraically smallest eigenvalue of B_α, and assume that there exists a corresponding eigenvector that can be safely normalized to have first component equal to one. For such an eigenvector, $(1, x^T)^T$, we have:

$$\begin{pmatrix} \alpha & g^T \\ g & H \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix} = \lambda \begin{pmatrix} 1 \\ x \end{pmatrix} \quad \Leftrightarrow \quad \begin{array}{rcl} \alpha - \lambda & = & -g^T x \\ (H - \lambda I)x & = & -g \end{array} \tag{7}$$

and consequently, two of the optimality conditions in Lemma 2.1 are automatically satisfied by the pair λ, x. Namely, $(H - \lambda I)x = -g$ with $H - \lambda I$ positive semidefinite. The latter holds by Cauchy Interlace Theorem (cf. [29]), which states that the eigenvalues of H interlace the eigenvalues of B_α, for any value of α. In particular, λ, the algebraically smallest eigenvalue of B_α is a lower bound for δ_1, the algebraically smallest eigenvalue of H and therefore, $H - \lambda I$ is positive semidefinite.

The relationship $\alpha = \lambda - g^T x$ could provide a way of updating α. Indeed, LSTRS uses this relationship to adjust the parameter. Note that, from (7), $-g^T x = g^T (H - \lambda I)^\dagger g = \phi(\lambda)$, which is a rational function in λ with poles at the distinct eigenvalues of H. Therefore, the first equation in (7) can be written as $\alpha = \lambda + \phi(\lambda)$. Since ϕ is expensive to compute, instead of using this function directly to update α, LSTRS uses a rational interpolant for ϕ. The interpolation points are obtained by solving an eigenvalue problem for the algebraically smallest eigenvalue of B_α and a corresponding eigenvector, since the eigenpair provides suitable values for $\lambda, \phi(\lambda)$ and also for $\phi'(\lambda) = g^T((H - \lambda I)^\dagger)^2 g = x^T x$. The value of α is then computed as $\alpha = \widehat{\lambda} + \widehat{\phi}(\widehat{\lambda})$, where $\widehat{\phi}$ is the rational interpolant, and $\widehat{\lambda}$ satisfies $\widehat{\phi}'(\widehat{\lambda}) = \Delta^2$. One could regard the LSTRS iteration as translating the line $\alpha - \lambda$ until it intersects the graph of ϕ at the point where ϕ has slope Δ^2, as Figure 1 illustrates. Each new value of α replaces the 1,1 entry of B_α and an eigenvalue problem is solved for each new bordered matrix. A safeguarding strategy is used to ensure the convergence of α to its optimal value.

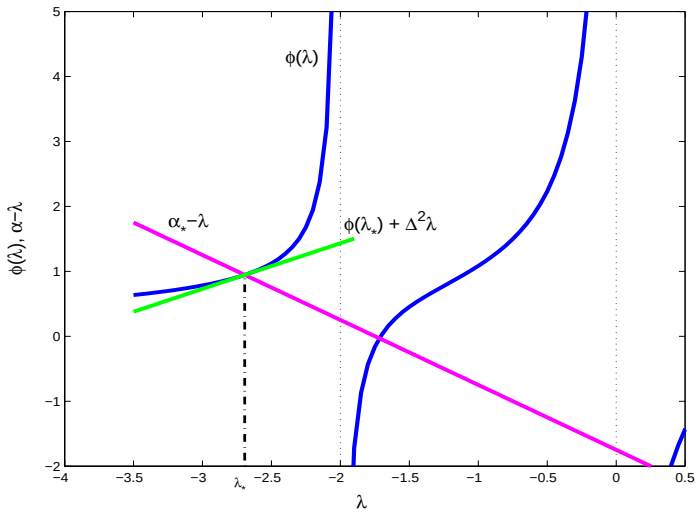


Figure 1: LSTRS method: the standard case.

The procedure we just described relies on the assumption that there exists an eigenvector corresponding to the algebraically smallest eigenvalue of the bordered matrix that can be safely normalized to have its first component equal to one. The strategy breaks down in the presence of a zero or very small first component. This situation is equivalent to one of the conditions for the hard case and is illustrated in Figure 2. The eigenvector of interest will have a first component zero or nearly zero if and only if the vector g is orthogonal or nearly orthogonal to $\mathcal{S}_1$, the eigenspace corresponding to the algebraically smallest eigenvalue of H. Therefore, a small first component indicates the potential occurrence of the hard case. In terms of the function ϕ, this means that δ_1 is not a pole or a very weak one, and ϕ will be very steep around such a pole, causing difficulties to the interpolation procedure. LSTRS handles this case by computing *two* eigenpairs of the bordered matrix at each step: one corresponding to the algebraically smallest eigenvalue of B_α, and the other, corresponding to another eigenvalue of B_α. Under certain conditions, both eigenpairs can be used to construct an approximate solution for the trust-region subproblem.

We will now describe the main components of LSTRS: the computation of initial values, the interpolation schemes, the safeguarding strategies, and the stopping criteria. We will also describe the different tolerances needed

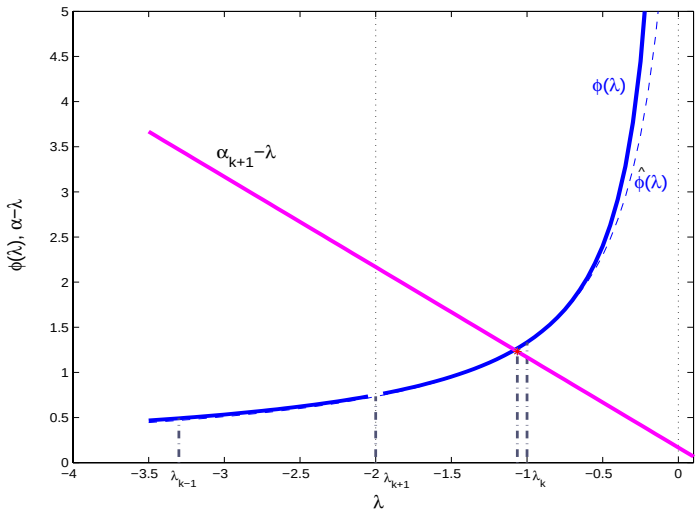


Figure 2: LSTRS method: the (near) hard case. ϕ (solid), $\widehat{\phi}$ (dashed).

by the method. We will focus on the results leading to the computational formulas and omit their derivations. We refer the reader to [32, 33] for more details.

In the remainder of this section, λ_1 refers to the algebraically smallest eigenvalue of B_α and λ_i to *any* of the remaining ones. An eigenvector of B_α is denoted by $(\nu, u^T)^T$, where ν is a scalar and u is an n-dimensional vector.

3.1 Initial values

Initial values are needed for $\delta_L, \delta_U, \alpha_L, \alpha_U$, and α. The values δ_L and δ_U are lower and upper bounds for δ_1, the algebraically smallest eigenvalue of H. The values α_L, α_U are lower and upper bounds for α_*, the optimal value for the parameter α.

Initial values are computed as in [33]: δ_U is chosen as either the Rayleigh quotient $\frac{u^T H u}{u^T u}$, for a random vector u, or as the minimum diagonal element of H; α_U is set to $\delta_U + \|g\|\Delta$. An initial value for α can be chosen as either $\alpha^{(0)} = \min\{0, \alpha_U\}$ or $\alpha^{(0)} = \delta_U$. The value $\alpha^{(0)}$ is used to construct a first bordered matrix $B_{\alpha^{(0)}}$ for which two eigenpairs, corresponding to λ_1 and to λ_i, are computed. As discussed before, the algebraically smallest eigenvalue is a lower bound for δ_1, and consequently we set $\delta_L = \lambda_1$. A lower bound for

α is given by $\alpha_{_L} = \delta_{_L} - \frac{\|g\|}{\Delta}$. It was shown in [33] that the interval $[\alpha_{_L}, \alpha_{_U}]$ contains α_* and therefore, it is used as the initial safeguarding interval for the parameter α. We remark that an adjusting procedure is applied to $\alpha^{(0)}$ in order to ensure that one of the two eigenvectors of $B_{\alpha^{(0)}}$ can be safely normalized to have first component one. The existence of an eigenvector with this special structure is guaranteed by the theory (cf. [33]). This eigenvector, $(\nu, u^{^T})^{^T}$ and the corresponding eigenvalue λ provide an initial iterate $\{\lambda^{(0)}, x^{(0)}\}$, with $\lambda^{(0)} = \lambda$ and $x^{(0)} = \frac{u}{\nu}$. This iterate will be used in the computation of $\alpha^{(1)}$ by the 1-point rational interpolation scheme [33], used to interpolate the pair $(\lambda^{(0)}, \phi(\lambda^{(0)}))$. The scheme yields:

$$\alpha^{(1)} = \widehat{\lambda} + \widehat{\phi}(\widehat{\lambda}) = \alpha^{(0)} + \frac{\alpha^{(0)} - \lambda^{(0)}}{\|x^{(0)}\|} \left(\frac{\Delta - \|x^{(0)}\|}{\Delta} \right) \left(\Delta + \frac{1}{\|x^{(0)}\|} \right), \qquad (8)$$

where $\widehat{\lambda} = \dfrac{(x^{(0)})^{^T} H x^{(0)}}{(x^{(0)})^{^T} x^{(0)}} + \dfrac{g^{^T} x^{(0)}}{\|x^{(0)}\| \Delta}$.

The value $\alpha^{(1)}$ is used to construct a second bordered matrix $B_{\alpha^{(1)}}$ for which two eigenpairs are computed. As before, an adjusting procedure is applied to $\alpha^{(1)}$ to ensure the availability of an eigenvector with the required structure. This eigenvector, $(\nu, u^{^T})^{^T}$ and the corresponding eigenvalue λ provide the new iterate $\{\lambda^{(1)}, x^{(1)}\}$, with $\lambda^{(1)} = \lambda$ and $x^{(1)} = \frac{u}{\nu}$. Observe that from the k-th LSTRS iterate we have $\lambda = \lambda^{(k)}$, $\phi(\lambda) = -g^{^T} x^{(k)}$, and $\phi'(\lambda) = (x^{(k)})^{^T} x^{(k)}$. Therefore, the first two iterates, $\{\lambda^{(0)}, x^{(0)}\}$ and $\{\lambda^{(1)}, x^{(1)}\}$, provide the first six values required in the 2-point rational interpolation scheme used to construct an interpolant for ϕ, which in turn is used to update the parameter α in the main iteration of LSTRS.

3.2 Update of α

The 2-point interpolation scheme (cf. [33]) used to compute $\alpha^{(k+1)}$, $k \geq 1$, yields:

$$\begin{aligned} \alpha^{(k+1)} &= \omega \alpha^{(k-1)} + (1-\omega)\alpha^{(k)} \\ &+ \frac{\|x^{(k-1)}\| \|x^{(k)}\| (\|x^{(k)}\| - \|x^{(k-1)}\|)}{\omega \|x^{(k)}\| + (1-\omega)\|x^{(k-1)}\|} \frac{(\lambda^{(k-1)} - \widehat{\lambda})(\lambda^{(k)} - \widehat{\lambda})}{(\lambda^{(k)} - \lambda^{(k-1)})}, \end{aligned} \qquad (9)$$

where $\omega = \dfrac{\lambda^{(k)} - \widehat{\lambda}}{\lambda^{(k)} - \lambda^{(k-1)}}$, $\alpha^{(k-1)} = \lambda^{(k-1)} + \phi(\lambda^{(k-1)})$ and $\alpha^{(k)} = \lambda^{(k)} + \phi(\lambda^{(k)})$,

and where

$$\widehat{\lambda} = \frac{\lambda^{(k-1)}\|x^{(k-1)}\|(\|x^{(k)}\| - \Delta) + \lambda^{(k)}\|x^{(k)}\|(\Delta - \|x^{(k-1)}\|)}{\Delta(\|x^{(k)}\| - \|x^{(k-1)}\|)} \ .$$

3.3 Adjustment of α

Each computed value of $\alpha^{(k)}$, $k \geq 0$, is adjusted to ensure that one of the two eigenpairs of $B_{\alpha^{(k)}}$ has an eigenvector that can be safely normalized to have first component equal to one. As previously mentioned, the existence of such an eigenvector is guaranteed by the theory (see Theorem 3.1 in [33]). This eigenvector is needed to construct the rational interpolants used to derive the updates (8) and (9), and continue the iterations of LSTRS. Figure 3 presents the adjusting procedure.

Adjust α.

Input: $\varepsilon_\nu, \varepsilon_\alpha \in (0,1)$, α_L, α_U, α with $\alpha \in [\alpha_L, \alpha_U]$,
eigenpairs $\{\lambda_1, (\nu_1, u_1^T)^T\}$ and $\{\lambda_i, (\nu_i, u_i^T)^T\}$ of B_α
Output: $\alpha, \{\lambda_1, (\nu_1, u_1^T)^T\}$ and $\{\lambda_i, (\nu_i, u_i^T)^T\}$.

while
$\|g\||\nu_1| \leq \varepsilon_\nu\sqrt{1-\nu_1{}^2}$ **and** $\|g\||\nu_i| \leq \varepsilon_\nu\sqrt{1-\nu_i{}^2}$
and $|\alpha_U - \alpha_L| > \varepsilon_\alpha * \max\{|\alpha_L|, |\alpha_U|\}$ **do**
$\alpha_U = \alpha$
$\alpha = (\alpha_L + \alpha_U)/2$
Compute eigenpairs $\{\lambda_1, (\nu_1, u_1^T)^T\}$ and $\{\lambda_i, (\nu_i, u_i^T)^T\}$ of B_α
end while

Figure 3: Adjustment of α.

3.4 Safeguarding of α

The use of an interpolant for the update of α might yield values that are far from the desired optimal value α_*. Therefore, a safeguarding interval $[\alpha_L, \alpha_U]$, containing α_*, is maintained and updated throughout the iterations, and each

value of α is safeguarded so it belongs to this interval. The safeguarding strategy is presented in Figure 4.

Safeguard α.

Input: α, $\delta_U \geq \delta_1$, α_L, α_U,
$\phi_i = -g^T x^{(i)}$ and $\phi'_i = \|x^{(i)}\|^2$, for $i = k-1, k$.
Output: $\alpha \in [\alpha_L, \alpha_U]$.

if $\alpha \notin [\alpha_L, \alpha_U]$
  if $k = 0$ **then** $\alpha = \delta_U + \phi_k + \phi'_k(\delta_U - \lambda^{(k)})$
  else if $\|x^{(k)}\| < \|x^{(k-1)}\|$ **then** $\alpha = \delta_U + \phi_k + \phi'_k(\delta_U - \lambda^{(k)})$
  else $\alpha = \delta_U + \phi_{k-1} + \phi'_{k-1}(\delta_U - \lambda^{(k-1)})$
  end if
  if $\alpha \notin [\alpha_L, \alpha_U]$ **then** set $\alpha = (\alpha_L + \alpha_U)/2$ **end if**
end if

Figure 4: Safeguarding of α.

3.5 Stopping criteria

3.5.1 Boundary solution.

We detect a boundary solution, according to Lemma 2.1, whenever the following two optimality conditions are satisfied:

$$\left(\left| \left\| \frac{u_1}{\nu_1} \right\| - \Delta \right| \leq \varepsilon_\Delta * \Delta \right) \quad \text{and} \quad (\lambda_1 \leq 0)$$

for a given $\varepsilon_\Delta \in (0, 1)$. It suffices to check these two conditions since, as shown in the analysis of (7), the other two optimality conditions are satisfied by the eigenpair corresponding to the algebraically smallest eigenvalue of each of the bordered matrices generated in LSTRS. The solution to (1) in this case is $\lambda_* = \lambda_1$ and $x_* = \dfrac{u_1}{\nu_1}$.

3.5.2 Interior solution.

We detect an interior solution when

$$(\|u_1\| \ < \ \Delta|\nu_1|) \quad \text{and} \quad (\lambda_1 > -\varepsilon_{Int}),$$

for a given $\varepsilon_{Int} \in [0, 1)$. In this case, the solution is $\lambda_* = 0$ and x_* such that $Hx_* = -g$, with H positive definite. An unpreconditioned version of the Conjugate Gradient Method is used to solve the system in this case.

3.5.3 Quasi-optimal solution.

Let $\psi(x) = \frac{1}{2}x^T Hx + g^T x$ be the quadratic objective function of problem (1). Then, we say that a vector $\widetilde{x}$ is a *quasi-optimal* or *nearly-optimal* solution for problem (1), if $\|\widetilde{x}\| = \Delta$ and if $\psi(\widetilde{x})$ is sufficiently close to $\psi(x_*)$, the value of the objective function at the true solution of (1), i.e. if for a given tolerance $\eta \in (0, 1)$,

$$|\psi(\widetilde{x}) - \psi(x_*)| \leq \eta|\psi(x_*)|. \tag{10}$$

A quasi-optimal solution can only occur in the hard case or near hard case. A sufficient condition for (10) to hold is the basis for the stopping criterion in the hard case. The condition has the same flavor as Lemmas 3.4 and 3.13 in [23], and was established in Theorem 3.2 and Lemmas 3.5 and 3.6 of [33]. Theorem 3.2 establishes that, under certain conditions, the last n components of a special linear combination of eigenvectors of B_α form a nearly-optimal solution for problem (1). Lemma 3.5 establishes the conditions under which the special linear combination can be computed, and Lemma 3.6 shows how to compute it. The three results combined yield the stopping criterion presented in Figure 5.

3.5.4 The safeguarding interval is too small.

If the safeguarding interval for α, namely, $[\alpha_L, \alpha_U]$ satisfies $|\alpha_U - \alpha_L| \leq \varepsilon_\alpha \max\{|\alpha_L|, |\alpha_U|\}$ for a given tolerance $\varepsilon_\alpha \in (0, 1)$, then the interval cannot be further decreased and we stop the iteration. If $(\nu, u^T)^T$ is one of the two available eigenvectors of the bordered matrix such that ν is not small, then $x = \dfrac{u}{\nu}$ and $\lambda = \lambda_1$ are, in general, good approximations to x_*, λ_*, and we

Conditions for a Quasi-Optimal Solution

Input: $\lambda_1,\ (\nu_1, u_1^T)^T,\ \lambda_i,\ (\nu_i, u_i^T)^T,\ \varepsilon_{HC} \in (0,1)$
Output: True or False: quasi-optimal condition found or not.
In case a solution has been found, $\widetilde{\lambda},\ \widetilde{x}$ are also returned.

found = **false**, $\eta = \dfrac{\varepsilon_{HC}}{1-\varepsilon_{HC}}$

if $(1+\Delta^2)({\nu_1}^2 + {\nu_i}^2) > 1$ **then**

$$\tau_1 = \frac{\nu_1 - \nu_i\sqrt{(1+\Delta^2)({\nu_1}^2+{\nu_i}^2)-1}}{({\nu_1}^2+{\nu_i}^2)\sqrt{(1+\Delta^2)}},\quad \tau_2 = \frac{\nu_i + \nu_1\sqrt{(1+\Delta^2)({\nu_1}^2+{\nu_i}^2)-1}}{({\nu_1}^2+{\nu_i}^2)\sqrt{1+\Delta^2}}$$

else

$$\tau_1 = \frac{\nu_1}{\sqrt{{\nu_1}^2+{\nu_i}^2}},\quad \tau_2 = \frac{\nu_i}{\sqrt{{\nu_1}^2+{\nu_i}^2}}$$

end if

$$\widetilde{x} = \frac{\tau_1 u_1 + \tau_2 u_i}{\tau_1\nu_1 + \tau_2\nu_i},\quad \widetilde{\lambda} = {\tau_1}^2\lambda_1 + {\tau_2}^2\lambda_i,\quad \psi(\widetilde{x}) = \tfrac{1}{2}\widetilde{x}^T H\widetilde{x} + g^T\widetilde{x}$$

if $(\lambda_i - \lambda_1){\tau_2}^2(1+\Delta^2) \le -2\eta\psi(\widetilde{x})$ **then**

$\widetilde{\lambda},\ \widetilde{x}$ is a nearly-optimal pair, found = **true**

else

if $(1+\Delta^2)({\nu_1}^2 + {\nu_i}^2) > 1$ **then**

$$\tau_1 = \frac{\nu_1 + \nu_i\sqrt{(1+\Delta^2)({\nu_1}^2+{\nu_i}^2)-1}}{({\nu_1}^2+{\nu_i}^2)\sqrt{(1+\Delta^2)}},\quad \tau_2 = \frac{\nu_i - \nu_1\sqrt{(1+\Delta^2)({\nu_1}^2+{\nu_i}^2)-1}}{({\nu_1}^2+{\nu_i}^2)\sqrt{1+\Delta^2}}$$

$$\widetilde{x} = \frac{\tau_1 u_1 + \tau_2 u_i}{\tau_1\nu_1 + \tau_2\nu_i},\quad \widetilde{\lambda} = {\tau_1}^2\lambda_1 + {\tau_2}^2\lambda_i,\quad \psi(\widetilde{x}) = \tfrac{1}{2}\widetilde{x}^T H\widetilde{x} + g^T\widetilde{x}$$

if $(\lambda_i - \lambda_1){\tau_2}^2(1+\Delta^2) \le -2\eta\psi(\widetilde{x})$ **then**

$\widetilde{\lambda},\ \widetilde{x}$ is a nearly-optimal pair, found = **true**

end if

end if

end if

Figure 5: Conditions for a Quasi-Optimal Solution

return them as the approximate solution pair. If, in addition, $\|x\| < \Delta$ (hard case) and if an approximate eigenvector corresponding to the algebraically smallest eigenvalue of H is available, we add to x a component in the direction of this eigenvector to obtain x_* such that $\|x_*\| = \Delta$. This strategy was thoroughly described in [23] and [39, Section 5], and was also adopted in [32, 33]. Note that the necessary eigenvector will be usually available from the LSTRS iteration. The *updated* x_* is returned along with λ_1 as a solution pair.

3.5.5 Maximum number of iterations reached.

The user may specify the maximum number of LSTRS iterations allowed and the method will stop when this number is reached.

3.6 Tolerances

LSTRS requires a few tolerances for the stopping criteria and also for some computations. The different tolerances and their meanings are summarized in Table 1. The MATLAB implementation of LSTRS provides a set of default values for the tolerances. The values will be presented in Section 4.

3.7 Algorithm

The strategies and procedures described in Sections 3.1 through 3.5 are the building blocks for the LSTRS method, shown in Figure 6.

4 The MATLAB software

In this section, we describe our MATLAB 6.0 implementation of the LSTRS method presented in [33] and summarized in the previous section. In the following, the `teletype` font is used for MATLAB codes, built-in types and routines; **boldface** is used for file names, parameters, variables, including structure fields, and also to highlight parts of MATLAB codes.

LSTRS

Input: $H \in \mathbb{R}^{n\times n}$, $g \in \mathbb{R}^n$, $\Delta > 0$,
Tolerances: $\varepsilon_\Delta, \varepsilon_\nu, \varepsilon_{HC}, \varepsilon_\alpha \in (0,1)$, $\varepsilon_{Int} \in [0,1)$.
Output: λ_*, x_* satisfying conditions of Lemma 2.1,
or $\widetilde{\lambda}$, $\widetilde{x}$, a quasi-optimal pair as in Figure 5.

1. Initialization
1.1 Compute: $\delta_U \geq \delta_1$, $\alpha_U \geq \alpha_*$, $\alpha^{(0)}$ as in Section 3.1
1.2 Compute eigenpairs $\{\lambda_1, (\nu_1, u_1^T)^T\}$ and $\{\lambda_i, (\nu_i, u_i^T)^T\}$ of $B_{\alpha^{(0)}}$
1.3 Compute $\delta_L \leq \delta_1$, $\alpha_L \leq \alpha_*$ as in Section 3.1
1.4 Set $k = 0$
2. repeat
2.1 Update $\delta_U = \min\left\{\delta_U, \frac{u_1^T H u_1}{u_1^T u_1}\right\}$
2.2 Adjust $\alpha^{(k)}$ using procedure in Figure 3
2.3 **if** $\|g\||\nu_1| > \varepsilon_\nu\sqrt{1-\nu_1{}^2}$ **then**
set $\lambda^{(k)} = \lambda_1$ and $x^{(k)} = \dfrac{u_1}{\nu_1}$
if $\|x^{(k)}\| < \Delta$ **then** $\alpha_L = \alpha^{(k)}$ **end if**
if $\|x^{(k)}\| > \Delta$ **then** $\alpha_U = \alpha^{(k)}$
else set $\lambda^{(k)} = \lambda_i$, $x^{(k)} = \dfrac{u_i}{\nu_i}$ and $\alpha_U = \alpha^{(k)}$ **end if**
end if
2.4 Compute $\alpha^{(k+1)}$ by rational interpolation schemes using (8) from Section 3.1 if $k = 0$, and (9) from Section 3.2, otherwise
2.5 Safeguard $\alpha^{(k+1)}$ using procedure in Figure 4
2.6 Compute eigenpairs $\{\lambda_1, (\nu_1, u_1^T)^T\}$ and $\{\lambda_i, (\nu_i, u_i^T)^T\}$ of $B_{\alpha^{(k+1)}}$
2.7 Set $k = k + 1$
until convergence

Figure 6: LSTRS: an algorithm for Large-Scale Trust-Region Subproblems.

ε_Δ	The desired relative accuracy in the norm of the trust-region solution. A boundary solution x satisfies $\frac{\lvert\lVert x\rVert-\Delta\rvert}{\Delta} \leq \varepsilon_\Delta$.
ε_{HC}	The desired accuracy of a quasi-optimal solution. If x_* is the true solution and $\widetilde{x}$ is the quasi-optimal solution, then $\psi(x_*) \leq \psi(\widetilde{x}) \leq \varepsilon_{HC}\psi(x_*)$, where $\psi(x) = \frac{1}{2}x^T Hx + g^T x$.
ε_{Int}	Used to declare the algebraically smallest eigenvalue of B_α positive in the test for an interior solution: λ_1 is considered positive if $\lambda_1 > -\varepsilon_{Int}$.
ε_α	The minimum relative length of the safeguarding interval for α. The interval is too small when $\lvert\alpha_U - \alpha_L\rvert \leq \varepsilon_\alpha * \max\{\lvert\alpha_L\rvert, \lvert\alpha_U\rvert\}$.
ε_ν	The minimum relative size of an eigenvector component. The component ν is small when $\lvert\nu\rvert \leq \varepsilon_\nu \frac{\lVert u\rVert}{\lVert g\rVert}$.
maxiter	The maximum number of iterations allowed.

Table 1: Tolerances for LSTRS

4.1 Data structures

The main data structures, implemented with the MATLAB type `struct`, are the following:

- A structure for the bordered matrix B_α, with fields: **H** (the Hessian matrix), **g** (the gradient vector), **alpha** (the scalar parameter α), **dim** (one plus the dimension of the trust-region subproblem), **bord** (scalar indicating if the structure represents a bordered matrix (1), or if only the Hessian is to be used (0)), and **Hpar** (parameters for **H**, whenever **H** is a matrix-vector multiplication routine, cf. Section 4.2.1).

- A structure for the LSTRS iterate chosen from two eigenpairs of B_α. The fields of the structure are: **lambda** (the eigenvalue), **nu** (the first component of the eigenvector), **anu** (the absolute value of **nu**), **u** (an n-dimensional vector consisting of the last n components of the eigenvector), and **noru** (the norm of the vector **u**).

- A structure for the interpolation points, with fields: **lambda** (λ), **fi** ($\phi(\lambda)$), and **norx** ($\sqrt{\phi'(\lambda)}$).

4.2 Interface

The front-end routine is called **lstrs**. The most general call to this routine is of the form:

```
[x,lambda,info,moreinfo] = ...
lstrs(H,g,delta,epsilon,eigensolver,lopts,Hpar,eigensolverpar);
```

The parameter **H** specifies either the Hessian matrix or a matrix-vector multiplication routine; **eigensolver** specifies the eigensolver routine. The *required* input parameters are: **H, g, delta**. The remaining parameters are *optional* with default values provided where appropriate. A detailed specification of the parameters follows. The type and default values for the optional parameters are given in curly brackets.

4.2.1 Input parameters

Required (3):

1. **H** {`string`, `function_handle`, or `double`}: matrix-vector multiplication routine, or an $n \times n$ array containing a symmetric matrix.
2. **g** {`double`}: $n \times 1$ array.
3. **delta** {`double`}: positive scalar (trust-region radius).

Optional (5):

1. **epsilon** {`struct`}: contains the tolerances described in Table 1. The fields are:
   - **Delta** {`double`, 10^{-4}}: boundary solutions.
   - **HC** {`double`, 10^{-4}}: quasi-optimal solutions.
   - **Int** {`double`, 10^{-10}}: interior solutions.
   - **alpha** {`double`, 10^{-8}}: size of the safeguarding interval for α.
   - **nu** {`double`, 10^{-2}}: small components.

2. **eigensolver** {`string` or `function_handle`, `'eigs_lstrs_gateway'`}: specifies which eigensolver to use. Current choices for the eigensolver are:

   - User-provided. See Section 4.2.3 for the calling sequence.
   - **eig_gateway**: gateway to MATLAB routine `eig` (QR method).
   - **eigs_lstrs_gateway**: gateway to `eigs_lstrs`, a modified version of MATLAB's `eigs` (ARPACK [20] implementation of the Implicitly Restarted Arnoldi Method [38]). The modified routine returns more information, including the number of converged eigenvalues and the smallest Ritz value.
   - **tcheigs_lstrs_gateway**: gateway to a routine that computes the eigenpairs of a given matrix from the eigenpairs of a Tchebyshev matrix polynomial of degree 10. It is a combination of **eigs_lstrs** and a Tchebyshev Spectral Transformation [34]. The transformation will be described in detail in Section 5.

3. **lopts** {`struct`}: options for **lstrs** with fields:

   - **maxiter** {`double`, 50}: scalar indicating the maximum number of LSTRS iterations allowed.
   - **message_level** {`double`, 1}: scalar indicating the level of messages desired. The options are no messages (0), a message per iteration plus a summary at the end (1), and more detailed messages (2).
   - **name** {`string`}: the problem name.
   - **plot** {`string`, `'no'`}: indicates if a plot of the solution is desired. The possible values are: a string beginning with **`'y'`** or **`'Y'`** (plot), or any other string (no plot).
   - **correction** {`string`, `'yes'`}: indicates if, in the hard case, a correction term in the direction of an eigenvector corresponding to the smallest eigenvalue of the Hessian matrix H, should be added. The possible values are: a string beginning with **`'y'`** or **`'Y'`** (add), or any other string (do not add).

- **interior** {`string`, `'yes'`}: indicates if, when the existence of an interior solution is detected, such solution should be computed. The possible values are: a string beginning with **`'y'`** or **`'Y'`** (compute), other string (do not compute).
- **intsoltol** {`double`, `epsilon.Delta`}: a scalar indicating the accuracy with which an interior solution should be computed.
- **deltaU** {`string` or `double`, `'rayleigh'`}: a string indicating how to initialize δ_U (an upper bound for δ_1), or a scalar with the initial value. Possible values: **`'rayleigh'`**, a Rayleigh quotient with a random vector; **`'mindiag'`**, the minimum of the diagonal of H; or a scalar. Note that the **`'mindiag'`** option is only available for problems where the Hessian is given as an array. For problems where the Hessian is available implicitly as a matrix-vector multiplication routine, the minimum of the diagonal is still a good choice to initialize δ_U. However, in this case, the user must provide this value.
- **alpha** {`string` or `double`, `'min'`}: a string indicating how to initialize the parameter α, or a scalar with the initial value. Possible values: **`'min'`**, $\alpha^{(0)} = \min\{0, \alpha_U\}$; **`'deltaU'`**, $\alpha^{(0)} = \delta_U$; or a scalar.
- **maxeigentol** {`double`, `[ ]`}: the desired maximum relative accuracy in the eigenpairs, in case the user wants to adjust this accuracy at each iteration. Possible values are `[ ]` for no adjustment, a scalar (maximum relative accuracy), or a structure containing the maximum relative accuracy of the eigenpairs (**maxeigentol**) and the accuracy of the norm of the current iterate (**itermaxacc**), i.e. $\frac{|\Delta - ||x_k|||}{\Delta}$. Two different adjustment strategies are implemented in the routine **adjust_eigentol**.
- **heuristics** {`double`, `0`}: a scalar indicating if eigenvalues equal to zero and Lanczos vectors (not converged eigenvectors) should be used to construct a LSTRS iterate. When set to 0, the heuristics is not used. The strategy is only available in combination with the eigensolver **'eigs_lstrs'**. Possible values: any scalar.

4. **Hpar** {`struct`}: parameters for **H**, whenever **H** is a matrix-vector multiplication routine. See Section 4.2.2 for more details.

5. **eigensolverpar** {`struct`}: parameters for the eigensolver routine.

   If the eigensolver is **eigs_lstrs_gateway** or **tcheigs_lstrs_gateway**, the parameter **eigensolverpar** should be used as the parameter **OPTS** in MATLAB's `eigs`, which specifies the options for ARPACK. LSTRS uses the following default values for `eigs`' options: **eigensolverpar.tol** $= 10^{-2}$, **eigensolverpar.maxit** $= 13$, **eigensolverpar.p** $= 7$, and **eigensolverpar.issym** $= 1$.

   The variable **eigensolverpar.v0** allows the user to specify an initial vector for the Arnoldi/Lanczos process. For LSTRS, **eigensolverpar.v0** must be an $(n + 1) \times 1$ array of type `double`. In the software, the first column of the Lanczos-basis matrix for the bordered matrix in a given iteration is used as the initial vector for the Lanczos process on the bordered matrix in the next iteration. Finally, LSTRS allows a new field **k** to be added to **eigensolverpar**. This field is used to specify the number of wanted (small) eigenvalues. The default value for **eigensolverpar.k** is 2. If a number less than 2 is specified, the parameter is set to 2. A value greater than 2 is allowed.

All the optional parameters can be set to the empty array `[ ]`. This is useful when we want to use the default value for one parameter but choose the value of the next. In this way, the value `[ ]` is used to *skip* a parameter. The order in which the parameters appear in the header of the function determines which parameter is skipped. For example, the first `[ ]` to appear in a calling sequence corresponds to **epsilon**.

4.2.2 Calling specifications for the matrix-vector multiplication routine

If **H** is a matrix-vector multiplication routine, it is called as **H(v,Hpar)**, where **v** is an $n \times 1$ array of type `double`, and **Hpar** is a structure containing parameters for **H**. If H is the Hessian matrix, the routine **H** should compute:

$$\mathbf{w} = H\ \mathbf{v}.$$

If **H** does not require any parameters besides **v**, MATLAB's **varargin** mechanism can be used in the specification of the function, as in the function **mv** in Figure 10.

4.2.3 Calling specifications for the eigensolver routine

As explained in Section 4.2.1, the user may provide the eigensolver routine, which will be called as:

```
[nconv,lambda1,y1,lambda2,y2] = ...
eigensolver(Balpha,eigensolverpar);
```

As before, if only `Balpha` is needed as parameter, MATLAB's **varargin** can be used to define the routine, as in:

```
function [nconv,lambda1,y1,lambda2,y2,it,mvp] = ...
user_eigensolver(Balpha,varargin);
```

The eigensolver routine should return:

- **nconv**: number of converged eigenvalues.
- **lambda1, y1**: the smallest eigenvalue of B_α, and a corresponding eigenvector.
- **lambda2, y2**: any of the remaining eigenvalues of B_α, and a corresponding eigenvector. In practice, faster convergence can be expected if this eigenvalue is either the second or a value close to the second smallest eigenvalue.

The eigensolver routine should receive the following input parameters:

- **Balpha**: a bordered matrix data structure as described in Section 4.1.
- **eigensolverpar**: parameters (usually of type `struct`) for the eigensolver routine.

4.2.4 Output parameters

The routine **lstrs** returns four parameters:

- **x**: the solution to the trust-region subproblem.
- **lambda**: the corresponding Lagrange multiplier.
- **info**: an integer representing the result of the computation, with the following possible values:

  0: **x** is a boundary solution.

1: **x** is an interior solution.

2: **x** is a quasi-optimal solution.

-1: an interior solution was detected and, as instructed by the user, the linear system was not solved, **x** is the current iterate.

-2: **x** is an approximation to the solution corresponding to the last value of α available when the safeguarding interval could not be further decreased. Note that **x** might contain a correction term in the direction of an eigenvector corresponding to the smallest eigenvalue of the Hessian matrix, as described in Section 3.5.4. Note also that **x** can take the value empty (`[ ]`) if there is no iterate available.

-3: the maximum number of iterations was reached, **x** is the current iterate, or empty if there is no iterate available.

-4: it was not possible to compute an iterate. This can happen when the eigensolver cannot compute the necessary eigenvectors, **x** is empty.

- **moreinfo**: a structure with fields **exitcond**, **mvp**, **iter**, **solves**, **kkt** and **alpha**, which contain, respectively, strings indicating all the stopping criteria that were satisfied, the number of matrix-vector products, the number of LSTRS iterations, the number of calls to the eigensolver, the value of $\frac{\|(H-\lambda I)x+g\|}{\|g\|}$, and the final value of the parameter α.

4.3 Global variables

The global variable **mvp_lstrs** is used to count the number of matrix-vector products performed. The variable is used only in three routines: `lstrs_method` (initialization), `matvec` (update), and `output`.

4.4 Output

In addition to the output parameters previously described, when the message level is chosen as 1 or 2, the following information is displayed: information on each iteration, and at the end, a summary of cost indicators (iterations, matrix-vector products). The value $\frac{\|(H-\lambda I)x+g\|}{\|g\|}$ is provided as an indication

of how well the solution pair satisfies this optimality condition. The Lagrange multiplier λ is also displayed.

The program then displays the first stopping criterion that x satisfies. In case more than one stopping criteria are satisfied, these are displayed separately.

When **lopts.message_level** is 1 or 2, the name (if provided) of the problem, its dimension, and the value of Δ are displayed at the beginning of the execution, followed by the name of the eigensolver routine used. Additionally, a plot of the LSTRS solution (blue on the screen, dashed in Figure 17) can be provided, depending on the value of the input parameter **lopts.plot**. This information can be particularly useful when only *one* trust-region problem needs to be solved, as in regularization.

4.5 Files

The LSTRS software follows a structured, top-down design. The MATLAB M-files containing the components of the software are the following:

lstrs.m:	the front-end routine.
lstrs_method.m:	the main LSTRS iteration.
init_up_bounds.m:	initializes the upper bounds for α, δ_1.
b_epairs.m:	front-end routine for eigensolver.
adjust_eigentol.m:	adjusts the desired relative eigenpair accuracy.
init_lo_bounds.m:	initializes the lower bound for α.
upd_deltaU.m:	updates the upper bound for δ_1.
adjust_alpha.m:	adjusts α, might need to compute eigenpairs.
convergence.m:	checks the stopping criteria.
boundary_sol.m:	the boundary-solution stopping criterion.
interior_sol.m:	the interior-solution stopping criterion.
quasioptimal_sol.m:	the quasi-optimal-solution stopping criterion.
upd_alpha_safe.m:	updates the safeguarding interval for α.
upd_param0.m:	updates $\alpha^{(0)}$ by one-point interpolation scheme.
interpol1.m:	one-point rational interpolation scheme.
inter_point.m:	chooses the interpolation point from two eigenpairs of the bordered matrix.
safe_alpha1.m:	safeguards $\alpha^{(1)}$.

upd_paramk.m:	updates $\alpha^{(k)}$ by two-point interpolation scheme.
interpol2.m:	two-point rational interpolation scheme.
safe_alphak.m:	safeguards $\alpha^{(k)}$.
output.m:	sets output parameter and output messages.
cg.m:	the conjugate gradient method for computing interior solutions.
correct.m:	adds a suitable correction term to the current iterate in the hard case.
eigs_lstrs.m:	a modified version of MATLAB's `eigs`.
eig_gateway.m:	gateway routine for MATLAB's `eig`.
eigs_lstrs_gateway.m:	gateway routine for `eigs_lstrs`.
tcheigs_lstrs_gateway.m:	gateway routine for `eigs_lstrs` combined with a Tchebyshev spectral transformation.
matvec.m:	front-end routine for matrix-vector multiplication.
tchmatvec.m:	front-end routine for multiplication with a Tchebyshev matrix polynomial.
quadratic.m:	evaluates the quadratic objective function in problem (1).
smallnu.m:	determines if a scalar is small.

The files **mv.m, uutmatvec.m, simple.m, vcalls1.m, icalls.m, vcalls2.m** and **regularization.m**, containing the examples in Section 8, are also distributed with the software. The file **altmatvec.m** contains an alternative matrix-vector multiplication routine that does not use **varargin**. The file **atamv.m** contains a matrix-vector multiplication routine for the quadratically constrained least squares case, i.e. when the Hessian is the matrix $A^T A$.

4.6 Installing and Running the Software

The LSTRS MATLAB software is distributed as an archive in either tar or zip format in the files **lstrs.tar** and **lstrs.zip**, respectively. The Unix/Linux command `tar xvf lstrs.tar` will create a directory `LSTRS` in the current directory where all the M-files listed above will be stored. For the zip format we recommend that the user creates a directory `lstrs-directory` and store the LSTRS files in that directory.

In either case, the the LSTRS directory should be included in MATLAB's

search path. This can be accomplished with one of the following commands: `path(path,'lstrs-directory')` or `addpath 'lstrs-directory'`.

5 LSTRS for regularization

A few considerations are in order when using the LSTRS MATLAB software for regularization problems, for which $H = A^T A$ and $g = -A^T \bar{b}$.

In the first place, since the (potential) near hard case is the common case for these problems (cf. Section 2.2), the solution will usually be quasi-optimal. However, it might happen that the *exact* hard case is detected, which will cause a correction term to be added to the iterate so it can have the desired norm (see Section 3.5.4). The correction term is in the direction of an eigenvector corresponding to the smallest eigenvalue of H. In general, such correction term is not desirable in regularization, since it might bring high-frequency components into the solution (cf. Section 2.2). Therefore, we recommend not to add the correction term for regularization problems. To indicate this choice, the user should set the input parameter **lopts.correction** to a string that does not begin with **'y'** or **'Y'**. Note that, if the correction term is not added, the solution provided by LSTRS is not a solution to the original trust-region subproblem, but a *regularized* solution corresponding to a smaller value of Δ.

The second consideration concerns interior solutions. For problem (2), an interior solution corresponds to the unconstrained least squares solution. Such solution is unique if the Hessian matrix is positive definite. If the Hessian is positive semidefinite and singular, it might be that both interior and boundary solutions exist, as illustrated in Figure 7, where we can see that there are infinite solutions along the dashed line in the "valley". The least squares solution is of no interest in regularization since it is contaminated by the noise in the data. In general, however, we have no means of distinguishing between the positive definite and the positive semidefinite cases. Therefore, we recommend not to compute the least squares (interior) solution, when such a solution is detected for a regularization problem. To indicate this choice, the user should set the input parameter **lopts.interior** to a string that does not begin with **'y'** or **'Y'**. In this case, a message is displayed advising the user to decrease Δ. The current iterate is returned since it might be useful in some situations, as we now discuss. Note that, if λ is small and an interior

solution is detected, then the current iterate $x^{(k)}$ is an approximation to $x = -H^{-1}g = (A^TA)^{-1}(A^T\bar{b})$, which is the interior solution. If, in addition, the noise level in $\bar{b}$ is low, $x^{(k)}$ will be a reasonable approximation to the desired solution.

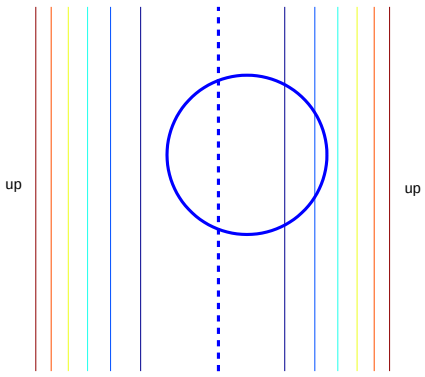


Figure 7: Interior and boundary solutions for a positive semidefinite Hessian.

A final note on regularization problems concerns the eigensolver. As we saw in Section 2.2, for these problems, the smallest eigenvalues of the Hessian matrix are usually clustered and close to zero, and because of the interlacing property the smallest eigenvalues of B_α will also be clustered and small for certain values of Δ. Computing a clustered set of small eigenvalues with a method that relies only on matrix-vector products with the original matrix is likely to fail since the multiplication will annihilate components precisely in the direction of the eigenvectors of interest. This difficulty may be overcome through the use of a spectral transformation. Instead of trying to find the smallest eigenvalue of B_α directly, we can work with a matrix function $T(B_\alpha)$ and use the fact that $B_\alpha q = q\lambda \iff T(B_\alpha)q = qT(\lambda)$. If we are able to construct T so that $|T(\lambda_1)| \gg |T(\lambda_j)|,\ j > 1$, then a Lanczos type method such as the Implicitly Restarted Lanczos Method (IRLM) [38] will converge much faster toward the eigenvector q_1 corresponding to λ_1. As in [34], we use a Tchebyshev polynomial T_ℓ of degree ℓ constructed to be as large as possible on λ_1 and as small as possible on an interval containing the remaining eigenvalues of B_α. The convergence of the IRLM is often greatly enhanced through this spectral transformation strategy. After convergence, the eigenvalues of B_α are recovered via the Rayleigh quotients with the converged

eigenvectors. We provide the routine **tcheigs_lstrs_gateway** implementing a Tchebyshev Spectral Transformation with a polynomial of degree $\ell = 10$. We recommend the use of this routine for most regularization problems.

A less expensive alternative to the Tchebyshev Spectral Transformation consists of using zero as eigenvalue and the available Lanczos vectors (*not* converged eigenvectors) returned by the eigensolver (e.g. `eigs_lstrs`) to construct an LSTRS iterate. The rationale behind this heuristics is that zero is a lower bound for the smallest eigenvalue of $H = A^T A$, and that the Lanczos vectors are rich in the direction of the eigenvectors of interest. Note that the safeguarding mechanisms of LSTRS guarantee convergence also in this case. This option is available through the input parameter **lopts.heuristics** and can only be used in combination with the eigensolver `eigs_lstrs`. If the heuristics does not yield satisfactory results, then **lopts.heuristics** must be set to zero and the Tchebyshev Spectral Transformation must be used.

6 Comparisons

In this section, we compare LSTRS with other methods for the large-scale trust-region subproblem. The methods used for comparisons were the Sequential Subspace Method (SSM) of Hager [13], the Semidefinite Programming approach (SDP) of Fortin and Wolkowicz [7], and the Generalized Lanczos Trust Region method (GLTR) of Gould, Lucidi, Roma and Toint [11]. Note that only LSTRS, SSM and SDP are limited-memory methods. For SSM, results are reported only for the two matrix-free variants SSM and SSM_d. The methods are described in Section 6.1.

We used MATLAB implementations of LSTRS, SSM and SDP, and a Fortran 90 implementation of GLTR. It is important to note that the four codes are at a different stage of maturity. The GLTR code is the routine HSL_VF05 of the HSL library [18]. The SSM and SDP codes are initial implementations not yet released publicly and that were kindly provided by their authors for the purpose of these comparisons. In particular, the SDP code is still under development. The double precision version of HSL_VF05 was used whereas the MATLAB codes were run under MATLAB 6.0. The experiments were carried out on a SUN Ultra-250 with a 400 MHZ processor and 2048 Megabytes of RAM, running Solaris 5.8. The floating point arithmetic was IEEE standard double precision with machine precision $2^{-52} \approx 2.2204 \cdot 10^{-16}$.

We ran the codes on three different families of problems whose Hessian matrices were as follows: the 2-D Discrete Laplacian, UDU^T with U orthogonal and D diagonal, and a discretized operator from the inverse heat equation. The Laplacian is a frequently used model problem in CFD applications, the UDU^T matrix allows for the exploration of ill conditioning and the effect on the hard case, while the inverse heat equation is a well-known ill-posed problem.

For the first two families of problems, we report the average number of matrix-vector products (MVP), of the number of vectors (STORAGE), and of the value of the optimality measure $\frac{\|(H - \lambda I)x+g\|}{\|g\|}$, from a sample of ten related problems in each family. Time is not reported due to the different nature of the implementations, namely, three MATLAB interpreted codes and one Fortran 90 stand-alone code. Since the methods use different stopping criteria, the tolerances were adjusted so that the methods computed a solution with $\frac{\|(H - \lambda I)x+g\|}{\|g\|}$ of the order of 10^{-6}. Finally, the results used to compute the averages correspond to the choice of options for which each method required the lowest number of matrix-vector products. For the third family of problems, we report the number of matrix-vector products, the storage, the optimality measure and the relative error of the solution with respect to the true solution to the inverse problem. We report the best results in terms of this relative error out of several trials with each method.

This section is organized as follows. A brief description of the methods used for comparisons is presented in Section 6.1. A detailed description of the three families of problems, the settings for each of the codes, and the results are presented in Sections 6.2, 6.3 and 6.4, respectively. The discussion of the results is presented in Section 6.5.

6.1 Methods for large-scale trust-region subproblems

SSM

SSM considers subproblems restricted to a Krylov subspace and by imposing that this subspace contains the iterate generated by one step of the sequential quadratic programming (SQP) algorithm applied to the trust region subproblem, a locally quadratic convergent scheme is obtained. The SQP method is

equivalent to Newton's method applied to the nonlinear system

$$\begin{aligned}(H \ - \ \lambda \ I)x + g &= 0 \\ \tfrac{1}{2}x^T x - \tfrac{\Delta^2}{2} &= 0.\end{aligned}$$

The use of the minimum residual solution ensures locally quadratic convergence even for degenerate problems with multiple solutions and a singular Jacobian for the first order optimality conditions. Hager observed in his experiments that appropriate small-dimensional subspaces could be generated by combining preconditioned Krylov spaces with minimum residual techniques (cf. [13]). Two preconditioned schemes corresponding to a diagonal preconditioner (SSM_d) and an SSOR-type of approach (SSM_{ssor}) were suggested in [13]. Note that the SSM_{ssor} variant is not matrix-free.

SDP

The SDP method is an extension of the semi-definite programming approach of Rendl and Wolkowicz [31]. The current algorithm maintains the primal-dual philosophy of the previous and introduces a novel strategy for the hard case, which combines shifting of the eigenvalues and deflation. The equality-constrained trust-region subproblem is, due to duality, equivalent to an unconstrained concave maximization problem in one variable. The evaluation of the objective function of this problem depends on the determination of the algebraically smallest eigenvalue of a parameterized eigenvalue problem, similar to the idea employed by LSTRS. In the MATLAB working code provided by the authors, the eigenvalue is computed with MATLAB's subroutine **eigs**.

GLTR

The GLTR approach is based on the Lanczos tridiagonalization of the matrix H and on the solution of a sequence of problems restricted to Krylov spaces, inspired by the Steihaug-Toint algorithm [40, 43]. GLTR uses a weighted ℓ_2 norm that defines the trust region and plays the role of preconditioning. The method is an alternative to the Steihaug-Toint algorithm that investigates further the trust-region boundary whenever it is reached, and keeps the efficiency of the (preconditioned) conjugate gradient method inside the trust region.

Although GLTR does not require any factorization of H, the Lanczos vectors are needed to recover the minimizer of the original problem. Therefore, the Lanczos vectors should be either store or re-generated (see [11, p.509]) and limited-storage requirements may be lost.

The software package is available as the Fortran 90 module HSL_VF05 in the Harwell Subroutine Library [18], and it is also part of the GALAHAD Optimization Library, version 1.0 [12].

6.2 The 2-D Discrete Laplacian family

In this family of problems, the Hessian matrix was $H = L - 5I$, where L is the standard 2-D discrete Laplacian on the unit square based upon a 5-point stencil with equally-spaced mesh points. The diagonal shift of -5 was introduced to make H indefinite. The order of H was $n = 1024$. We used the four trust-region solvers described above to solve a sequence of ten related problems, differing only by the vector g, randomly generated with entries uniformly distributed on $(0, 1)$. The trust-region radius was fixed at $\Delta = 100$.

We studied problems with and without hard case. To generate the hard case, we orthogonalized the random vectors g against the eigenvector q corresponding to the algebraically smallest eigenvalue of H. We accomplished this by setting $g := g - q(q^T g)$. For the easy and hard cases we added a noise vector to g, of norm 10^{-8}.

For the limited-memory methods (LSTRS, SSM, SSM$_d$ and SDP) the number of vectors was fixed at 12, and 10 shifts were applied in each implicit restart in ARPACK. Other parameters were as follows. LSTRS: for the easy case, $\varepsilon_\Delta = 10^{-5}$, $\varepsilon_{HC} = 10^{-11}$; for the hard case, $\varepsilon_{HC} = \varepsilon_\Delta = 10^{-11}$. In both cases, $\delta_U =$ 'mindiag' and $\alpha^{(0)} = \delta_U$ as in [33]. The initial vector for ARPACK was $e/\sqrt{n+1}$, where e is the vector of all ones. Default values were used for the remainder of the parameters. SSM, SSM$_d$: $\|(H - \lambda I)x + g\|$ was required to be less than or equal to $tol = 10^{-5}$, one of the initial vectors in a relevant Krylov subspace was chosen as the vectors of all ones. SDP: the tolerance for the duality gap was set to 10^{-10} in the easy case and 10^{-9} in the hard case. GLTR: the tolerance for the optimality measure was set to 10^{-5} and the required fraction of the optimal value of the objective function was set to 1. The desired optimality level could not be achieved for lower fractions of the optimal objective value. This was also the case in the other experiments.

The results are presented in Table 2. For the easy case, SSM required about 50% fewer matrix-vector products than LSTRS, and GLTR required a relatively low number of matrix-vector products but used more than three times the storage of the limited-memory methods. For the hard case, LSTRS required about 30% fewer matrix-vector products than SSM. GLTR required the lowest number of matrix-vector products but more than six times the storage of the limited-memory methods.

6.3 The UDU^T family

In these problems, the matrix H was of the form $H = UDU^T$, with D a diagonal matrix with elements $d_1, \ldots, d_n$ and $U = I - 2uu^T$, with $u^T u = 1$. The elements of D were randomly generated with a uniform distribution on $(-5, 5)$, then sorted in nondecreasing order and d_1 set to -5. Both vectors u and g were randomly generated with entries selected from a uniform distribution on $(-0.5, 0.5)$ and then u was normalized to have unit length. The order of H was $n = 1000$. There was a total of ten problems.

In this case, the eigenvectors of the matrix H are of the form $q_i = e_i - 2uu_i$, $i = 1, \ldots, n$ with e_i the i–th canonical vector in $\mathbb{R}^n$ and u_i the i–th component of the vector u. The vector g was orthogonalized against $q_1 = e_1 - 2uu_1$, and a noise vector was added to g. Finally, g was normalized to have unit norm. The noise vectors had norms 10^{-2} and 10^{-8}, for the easy and hard cases, respectively. To ensure that the hard case really occurred, we computed $\Delta_{min} = \|(H - d_1 I)^\dagger g\|$, and set $\Delta = 0.1\Delta_{min}$ for the easy case and $\Delta = 5\Delta_{min}$ for the hard case.

For the limited-memory methods (LSTRS, SSM, SSM$_d$ and SDP) the number of vectors was fixed at 12 in the easy case and 36 in the hard case. Other parameters were as follows. LSTRS: for the easy case, δ_U was set to the minimum of the diagonal of UDU^T, $\alpha^{(0)} = \delta_U$, the adaptive tolerance for the eigenpairs was 0.2, and 10 shifts were applied in each implicit restart; for the hard case, $\delta_U = -4.5$, $\alpha^{(0)}$ = 'min', the adaptive tolerance for the eigenpairs was 0.03, and 24 shifts were applied in each implicit restart; $\varepsilon_\Delta = 10^{-4}$, $\varepsilon_{HC} = 10^{-10}$. More basis vectors were needed in the hard case since the eigenvalues were computed to a higher accuracy. The initial vector for ARPACK was $e/\sqrt{n+1}$, where e is the vector of all ones. Default values were used for the remainder of the parameters. SSM, SSM$_d$: $tol = 10^{-6}$ and one of the initial vectors in a relevant Krylov subspace was chosen as the

METHOD	MVP	STORAGE	$\frac{\|(H - \lambda I)x+g\|}{\|g\|}$
LSTRS	127.1	12	2.32×10^{-6}
SSM	66.4	12	9.65×10^{-7}
SSM_d	66.4	12	9.65×10^{-7}
SDP	595	12	3.17×10^{-5}
GLTR	81.6	41.3	8.56×10^{-6}

(a) Easy Case

METHOD	MVP	STORAGE	$\frac{\|(H - \lambda I)x+g\|}{\|g\|}$
LSTRS	252.6	12	6.91×10^{-6}
SSM	361.4	12	1.41×10^{-6}
SSM_d	361.4	12	1.41×10^{-6}
SDP	2023.8	12	5.76×10^{-2}
GLTR	151.8	76.4	8.37×10^{-6}

(b) Hard Case

Table 2: Average results for the 2-D Laplacian, $n = 1024$.

vectors of all ones. SDP: the tolerance for the duality gap was set to 10^{-11} in the easy case and 10^{-12} in the hard case. GLTR: the tolerance for the KKT (Karush-Kuhn-Tucker) condition was set to 10^{-5} for the easy case and 10^{-7} for the hard case. The fraction of the optimal value of the objective function was set to 1.

The results are reported in Table 3. The SSM methods outperformed all methods for the hard case. GLTR was comparable in both computations and storage for the easy case, but more expensive than the SSM methods for the hard case. These results are consistent with those reported in [13] and show that SSM performs extremely well on this class of problems. LSTRS was the second best of the limited-memory methods. For these problems, we found that LSTRS was sensitive to the choice of some initial parameters such as δ_U and $\alpha^{(0)}$. This behavior, as well as the features of this particular class of problems, should be further investigated.

6.4 Regularization problems

The third family of problems comes from the Regularization Tools package by Hansen [16]. We chose problem **heat** of dimension $n = 1000$. This problem is a discretized version of the Inverse Heat Equation, which arises, for example, in the inverse heat conduction problem of determining the temperature on the surface of a body from transient measurements of the temperature at a fixed location in the interior [3]. The equation is a Volterra integral equation

$$\gamma(y) = \int_0^1 \mathcal{K}(y,t)\phi(t)dt, \quad 0 \leq y \leq 1, \tag{11}$$

where $\mathcal{K}(y,t) = k(y-t)$, with $k(t) = \dfrac{t^{-3/2}}{2\kappa\sqrt{\pi}} \exp\left(-\dfrac{1}{4\kappa^2 t^2}\right)$. The parameter κ controls the degree of ill posedness. We performed experiments with a mildly ill-posed problem ($\kappa = 5$) and a severely ill-posed one ($\kappa = 1$).

To compute regularized solutions for problem (11), we solve the following quadratically constrained least squares problem

$$\min \ \ \frac{1}{2}\|Ax - b\|^2 \ \text{ s.t. } \|x\| \ \leq \ \Delta.$$

The MATLAB routine **heat** provided the matrix A, the vector b, as well as X_{IP}, a discretized version of the analytical solution of the continuous

METHOD	MVP	STORAGE	$\frac{\|(H - \lambda I)x+g\|}{\|g\|}$
LSTRS	90.2	12	2.95×10^{-6}
SSM	33.1	12	1.29×10^{-6}
SSM_d	21.9	12	8.90×10^{-7}
SDP	950.4	12	9.65×10^{-7}
GLTR	36.8	18.9	7.37×10^{-6}

(a) Easy Case

METHOD	MVP	STORAGE	$\frac{\|(H - \lambda I)x+g\|}{\|g\|}$
LSTRS	954.1	36	9.65×10^{-6}
SSM	420.1	36	1.80×10^{-6}
SSM_d	155.7	36	1.05×10^{-6}
SDP	1720.8	36	7.86×10^{-6}
GLTR	634.6	317.8	7.64×10^{-6}

(b) Hard Case

Table 3: Average results for UDU^T, $n = 1000$.

problem. For the trust-region problem, $H = A^T A$, $g = -A^T b$, and $\Delta = \|X_{IP}\|$. Twenty percent of the singular values of the matrix A were zero to working precision. No noise was added to the vector b since, as discussed in Section 2.2, the absence of noise yields a more difficult trust-region problem.

Since this problem is implemented as a MATLAB routine, we only tested the methods for which a MATLAB implementation was available, i.e. LSTRS, SSM, SSM$_d$, and SDP. Several options were tried for all methods. We report the best results in terms of the relative error in the solution to the trust-region problem with respect to X_{IP}, the exact solution to the inverse problem. Note that a bound on the optimality measure was not prescribed for these problems.

The number of vectors was fixed at 10 since this choice produced the best results for all methods. Other settings were as follows. For LSTRS: for the mildly ill-posed problem, $\varepsilon_\Delta = 10^{-3}$ and `lopts.maxeigentol = 0.7`; for the severely ill-posed problem, $\varepsilon_\Delta = 10^{-2}$ and `lopts.maxeigentol = 0.4`. The initial vector for ARPACK was $e/\sqrt{n+1}$, where e is the vector of all ones. The parameter `lopts.heuristics` was set to 1. Default values were used for the remainder of the parameters. SSM, SSM$_d$: $tol = 10^{-8}$ for the mildly ill-posed problem and 10^{-7} for the severely ill-posed problem, one of the initial vectors in a relevant Krylov subspace was chosen as the vector of all ones. SDP: the tolerance for the duality gap was set to 10^{-7} for the mildly ill-posed problem and 10^{-10} for the severely ill-posed problem.

The results are reported in Table 4. In the mildly ill-posed case, all the computed solutions are indistinguishable from the exact solution to the inverse problem. Plots of the solutions in the severely ill-posed case are shown in Figure 8. The results in Table 4 show that LSTRS required the lowest number of matrix-vector products of all unpreconditioned methods, while the diagonally-preconditioned version of SSM has the best performance of all methods for the severely ill-posed problem. In Figure 8, we can observe the oscillatory pattern in the SDP computed approximation for the severely ill-posed problem. The oscillations are probably due to high-frequency components and indicate that the desired regularizing effect could not be achieved.

6.5 Discussion

We have compared four methods for the large-scale trust-region subproblem on a set of different problems. Although more experiments should be per-

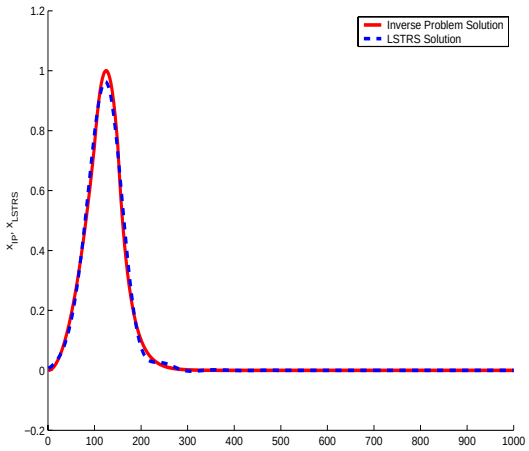
METHOD	MVP	STORAGE	$\frac{\|(H - \lambda I)x+g\|}{\|g\|}$	$\frac{\|x-X_{IP}\|}{\|X_{IP}\|}$
LSTRS	265	10	9.12×10^{-7}	6.13×10^{-4}
SSM	665	10	2.97×10^{-9}	2.69×10^{-4}
SSM_d	469	10	3.01×10^{-9}	6.07×10^{-4}
SDP	5700	10	2.73×10^{-7}	3.63×10^{-4}

(a) Mildly Ill-Posed Case

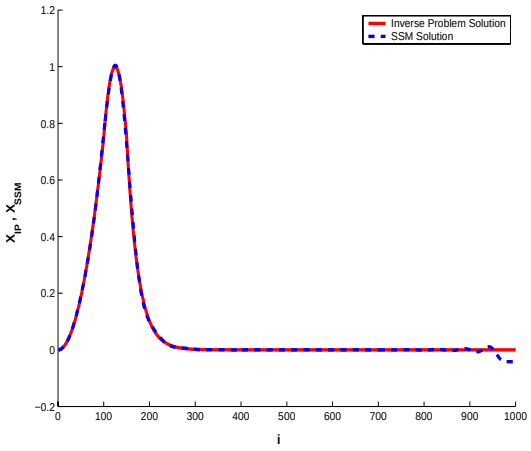
METHOD	MVP	STORAGE	$\frac{\|(H - \lambda I)x+g\|}{\|g\|}$	$\frac{\|x-X_{IP}\|}{\|X_{IP}\|}$
LSTRS	552	10	7.05×10^{-6}	5.49×10^{-2}
SSM	620	10	2.01×10^{-7}	3.63×10^{-2}
SSM_d	311	10	2.15×10^{-7}	1.89×10^{-2}
SDP	4600	10	2.27×10^{-4}	2.08×10^{-1}

(b) Severely Ill-Posed Case

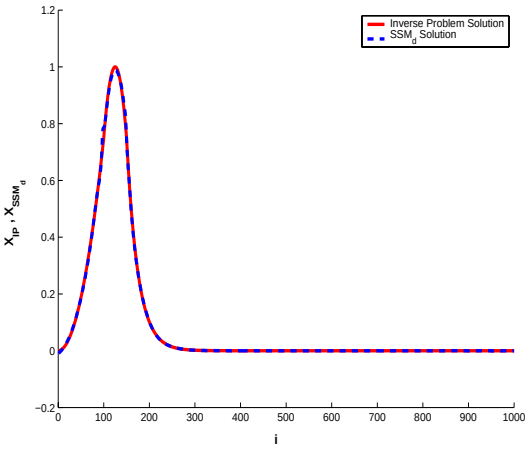
Table 4: Results for the Inverse Heat Equation, $n = 1000$.



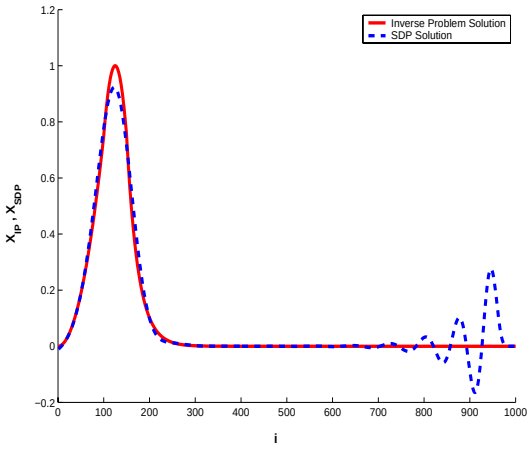
LSTRS Solution



SSM solution



SSM_d Solution



SDP Solution

Figure 8: Regularized solutions (dashed) and the exact solution (solid) to the inverse problem for a severely ill-posed Inverse Heat Equation, $n = 1000$.

formed, our results seem to indicate that LSTRS is competitive with state-of-the-art techniques for some classes of trust-region problems, including regularization problems. The latter was expected since part of the motivation for the development of the method came from the regularization of large-scale discrete forms of ill-posed problems.

The Sequential Subspace Methods performed extremely well on most problems and we believe that a publicly available, matrix-free and portable (MEX-file-free) implementation will be of interest. As we pointed out before, the Semidefinite Programming approach is still under development and we expect that some refinement of the codes and of the interesting deflation strategy will greatly improve the performance of the method. Moreover, the use of a Tchebyshev Spectral Transformation might help compute small eigenvalues to higher accuracy. Our tests with SDP were run in the summer of 2004. The authors have since reported improvements.

Except for regularization problems, the GLTR approach had the best performance in terms of the number of matrix-vector products. However, the memory requirements were larger than for the other methods. The method is a good choice when storage is not an issue and stand-alone software is desired. To the best of our knowledge, the method is yet to be tested on regularization problems.

Finally, we remark that both SDP and LSTRS required the solution of a low number of eigenproblems in all tests. As pointed out in [13], the parameterized eigenvalue approach would probably benefit from improvements in the eigenvalue computation such as the introduction of some sort of preconditioning together with ARPACK, or the use of other techniques such as the Jacobi-Davidson method [36]. For LSTRS, these possibilities are yet to be investigated and can be easily incorporated into the software.

7 Applications

A MATLAB 5.3 implementation of LSTRS has been successfully used in several large-scale applications. In [32, 34], the code was used in the study of the bathymetry of the Sea of Galilee. This study required the regularization of a linear inverse interpolation problem of dimension **40401**. The same version of the code was used to compute regularized solutions for a problem arising in the solution of the viscoacoustic wave equation in marine oil ex-

ploration with field data. The problem was of dimension **121121**. The code was also used as an inner solver in the TRUST$_\mu$ method for non-negative image restoration [35], where the typical images are digital arrays of 256×256 pixels which give rise to trust-region problems of dimension **65536**.

The MATLAB 5.3 and the current versions used the same computational routines. The versions differ only on the interface.

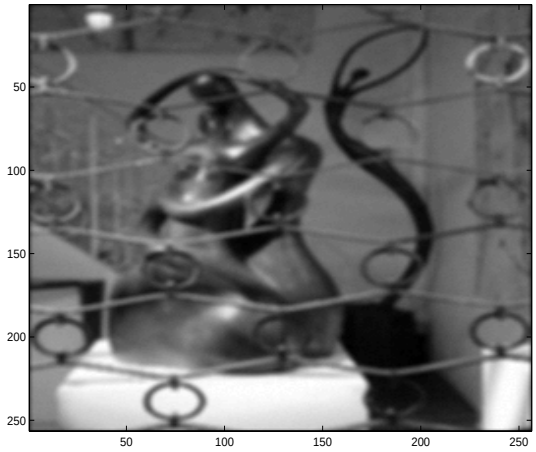
The current code was used as the inner solver in the iterative method MLFIP [6], which has been applied to the computation of confidence intervals for regularized solutions of the severely ill-posed sideways inverse heat equation. These experiments were carried out in MATLAB 6.5.

The performance of LSTRS in the context of trust-region methods for nonlinear programming is yet to be investigated. An important aspect to take into account is that nearly-exact solutions are acceptable in this context, as long as they provide a fraction of the Cauchy-point reduction (c.f. [4, 26]). We expect that LSTRS can compute such solutions with lower computational effort than in regularization, where good approximations to the exact solutions are usually required.

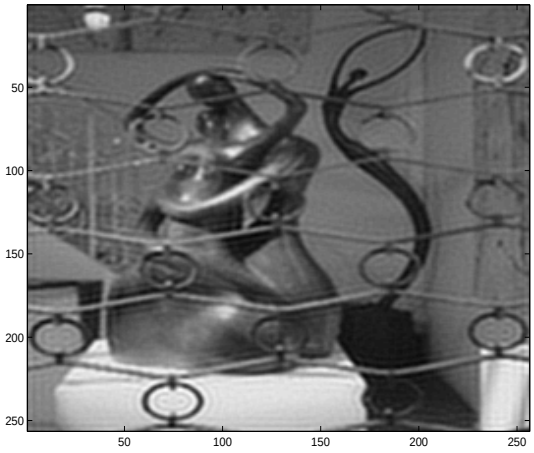
Next, we present a large-scale example in image restoration. The problem is that of recovering an image from blurred and noisy data. The problem was constructed in the following way. A digital photograph of an art gallery in Paris was blurred with the routine **blur** from the Regularization Tools package [16]. Then, a random Gaussian noise vector was added to the blurred image. The regularization problem was a constrained least squares problem of type (5), where A was the blurring operator returned by the routine **blur**, and $\bar{b}$ was the blurred and noisy image generated as above and stored columnwise as a one-dimensional array. The noise level in $\bar{b}$ was 10^{-2}. The dimension of the problem is **65536**.

The following options were used in LSTRS: **epsilon.Delta** $= 10^{-2}$ and **epsilon.HC** $= 10^{-4}$. The eigensolver was **tcheigs_lstrs_gateway** with initial vector equal to the vectors of all ones. The results are shown in Figure 9. Default values were used for the remainder of the parameters. LSTRS required 201 matrix-vector products and 9 vectors of storage to compute a quasi-optimal solution with a relative error of 1.06×10^{-1} with respect to the true solution. The optimality measure was 1.01×10^{-3}. The *waves* or *ripples* observed in the LSTRS restoration are due to the famous Gibbs phenomenon (cf. [28]) and are characteristic of least squares restorations.

True image



Blurred and noisy image



LSTRS restoration

9	Vectors
3	LSTRS iterations
201	Matrix-vector products

Cost

Figure 9: Restoration of the photograph of a Paris art gallery. Dimension: **65536**.

8 Examples

```
%
% File: mv.m
% A simple matrix-vector multiplication routine
% that computes the Identity matrix times a vector v
%
function [w] = mv(v,varargin)
w = v;
```

Figure 10: A matrix-vector multiplication routine *without* additional parameters.

```
%
% File: uutmatvec.m
% A matrix-vector multiplication routine that
% multiplies the matrix: (I-2uu') D (I-2uu') times a vector v
% D is a diagonal matrix, u is a unit vector
% uutmatvecpar is a structure with two fields d and u
% containing the vectors that define the matrix
%
function [w] = uutmatvec(v,uutmatvecpar)

d = uutmatvecpar.d;
u = uutmatvecpar.u;

w = v - 2 * (u'*v) * u;
w = d .* w;
w = w - 2 * (u'*w) * u;
```

Figure 11: A matrix-vector multiplication routine *with* additional parameters.

```
%
% File: simple.m
% A simple problem where the Hessian is the Identity matrix.
%
name = 'Identity';
H = eye(50);
g = ones(50,1);
mu = -3;            % chosen arbitrarily
xexact = -ones(50,1)/(1-mu);
Delta = norm(xexact);

%
% The simplest possible calls to lstrs. Default values are used.
%
% The initial vector for ARPACK is random.
% mv is the matrix-vector multiplication routine in Figure 10.
%
[x,lambda,info,moreinfo] = lstrs(H,g,Delta);
[x,lambda,info,moreinfo] = lstrs(@mv,g,Delta);

                                < M A T L A B >
>> simple

Problem: no name available. Dimension: 50. Delta: 1.767767e+00

Eigensolver: eigs_lstrs_gateway

LSTRS iteration: 0
||x||: 9.317862e-01, lambda: -6.588723e+00
|||x||-Delta|/Delta: 4.729021e-01

LSTRS iteration: 1
||x||: 1.767767e+00, lambda: -3.000000e+00
|||x||-Delta|/Delta: 2.512148e-16

Number of LSTRS Iterations: 2
Number of calls to eigensolver: 2
Number of MV products: 19

(||x||-Delta)/Delta: 2.512148e-16

lambda: -3.000000e+00

||g + (H-lambda* I)x||/||g|| = 1.159851e-15

The vector x is a Boundary Solution

Other Stopping Criteria Satisfied:
    Quasi-optimal Solution
```

Figure 12: Simple calls to **lstrs**.

```
%
% File: vcalls1.m
% Uses the same data as in Figure 12
%
% Eigensolver is tcheigs_lstrs_gateway, initial vector for ARPACK is random
%
[x,lambda,info,moreinfo] = lstrs(@mv,g,Delta,[ ],@tcheigs_lstrs_gateway);
[x,lambda,info,moreinfo] = lstrs(@mv,g,Delta,[ ],'tcheigs_lstrs_gateway');

%
% Eigensolver is eig_gateway
%
[x,lambda,info,moreinfo] = lstrs(H,g,Delta,[ ],@eig_gateway);

%
% Defining maxiter, message_level, name
% Default values are used for the remaining parameters
%
lopts.maxiter = 3;
lopts.message_level = 0;
lopts.name = name;

[x,lambda,info,moreinfo] = lstrs(@mv,g,Delta,[ ],[ ],lopts);

                                        ***
%
% File: icalls.m
% Uses the same data as in Figure 12
%
% This call produces an error: H must be a matrix, not a routine

                                < M A T L A B >

>> [x,lambda,info,moreinfo] = lstrs(@mv,g,Delta,@eig_gateway);

??? Error using ==> lstrs
To use the eigensolver 'eig_gateway', 'H' must be a matrix !
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
%
% This call produces an error. The string name is
% interpreted as the name of an eigensolver routine
%
                                < M A T L A B >

>> [x,lambda,info,moreinfo] = lstrs(@mv,g,Delta,[ ],name);

??? Error using ==> lstrs Undefined eigensolver: 'Identity'. Not in search path.
```

Figure 13: Valid and invalid calls to **lstrs**.

```
%
% File: vcalls2.m
% Redefining struct parameters.
%
% uutmatvec is the matrix-vector multiplication routine in Figure 11
% uutmatvecpar contains the parameters
%
name = '(I-2uu") D (I-2uu")';
uutmatvecpar.d = rand(50,1);
uutmatvecpar.u = rand(50,1);
uutmatvecpar.u = uutmatvecpar.u/norm(uutmatvecpar.u);
g = rand(50,1);
Delta = 1;

%
% These statements redefine the values of epsilon.Delta, epsilon.HC
% lstrs will use the new values
%
epsilon.Delta = 1e-3;
epsilon.HC = 1e-8;

[x,lambda,info,moreinfo] = lstrs(@uutmatvec,g,Delta,epsilon,uutmatvecpar);

%
% These statements redefine the values of opts.tol, opts.p, opts.v0 for eigs_lstrs
% and lopts.message_level, lopts.name
%
epar.tol = 1e-3;
epar.p = 15;
epar.v0 = ones(51,1)/sqrt(51);        % Initial vector for ARPACK

lopts.message_level = 2;
lopts.name = name;

[x,lambda,info,moreinfo] = ...
lstrs(@uutmatvec,g,Delta,epsilon,[ ],lopts,uutmatvecpar,epar);

%
% The longest possible call to lstrs
%
[x,lambda,info,moreinfo] = ...
lstrs(@uutmatvec,g,Delta,epsilon,@tcheigs_lstrs_gateway,lopts,uutmatvecpar,epar);
```

Figure 14: Redefining **struct** parameters for **lstrs**.

```
%
% File: regularization.m
% Computes a regularized solution for problem phillips from
% Regularization Tools [16], available from http://www.imm.dtu.dk/~pch
%
[A,b,xexact] = phillips(300);

atamvpar = A;
g = - (b'*A)';
Delta = norm(xexact);

lopts.name = 'phillips';
lopts.plot = 'y';
lopts.correction = 'n';  lopts.interior = 'n';


epsilon.Delta = 1e-2;
epar.v0 = ones(301,1)/sqrt(301);
% atamv is the routine in Figure 16

[x,lambda,info,moreinfo] = ...
lstrs(@atamv,g,Delta,epsilon,@tcheigs_lstrs_gateway,lopts,atamvpar,epar);

                              < M A T L A B >
>> regularization

Problem: phillips. Dimension: 300. Delta: 2.999927e+00

Eigensolver: tcheigs_lstrs_gateway

LSTRS iteration: 0
||x||: 8.327280e-01, lambda: -6.913002e+01
|||x||-Delta|/Delta: 7.224172e-01

LSTRS iteration: 1
||x||: 1.746167e+00, lambda: -1.768532e+01
|||x||-Delta|/Delta: 4.179302e-01

LSTRS iteration: 2
||x||: 2.935925e+00, lambda: -3.680399e-01
|||x||-Delta|/Delta: 2.133441e-02

LSTRS iteration: 3
||x||: 3.000546e+00, lambda: 1.883676e-03
|||x||-Delta|/Delta: 2.064169e-04

Number of LSTRS Iterations:   4
Number of calls to eigensolver:  5
Number of MV products:        342

(||x||-Delta)/Delta: 4.441000e-16

lambda: 1.904289e-03

||g + (H-lambda* I)x||/||g|| = 2.501468e-05

The vector x is a Quasi-optimal Solution
```

Figure 15: Solving a regularization problem with **lstrs**.

```
%
% File: atamv.m
% A matrix-vector multiplication routine
% that computes A'*A*v
%
function [w] = atamv(v,A)
w = A*v;
w = (w'*A)';
```

Figure 16: A matrix-vector multiplication routine that computes $w = A^T Av$.

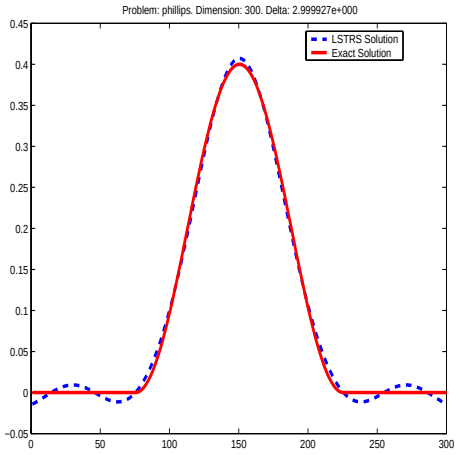


Figure 17: LSTRS solution plot for regularization problem **phillips**. The **dashed** curve (LSTRS solution) appears as **solid blue** on the screen.
The solid curve is the exact solution which has been added to the LSTRS plot for comparison.

Acknowledgements: The authors would like to thank Chao Yang for providing the code for the Tchebyshev matrix-vector multiplication. We also thank Bill Hager and Henry Wolkowicz for providing MATLAB implementations of their methods so that comparisons with LSTRS could be performed. M. Rojas would like to thank Iain Duff and the rest of the Algo Team at CERFACS for welcoming her for several extended visits, during which part of this work was done. The final revision of this work was done while M. Rojas was visiting Professor Cees Roos and the Optimization Group of the Software Technology Department at the Delft University of Technology. She would like to thank Professor Roos and his group for their hospitality. We are indebted to Martin van Gijzen for helpful discussions and suggestions. Finally, we thank the three anonymous referees and the editor for numerous helpful suggestions that greatly improved an earlier version of this manuscript.

References

[1] M. Bertero and P. Bocacci. *Introduction to Inverse Problems in Imaging*. Institute of Physics, Bristol, 1998.

[2] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[3] A. Carasso. Determining surface temperatures from interior observations. *SIAM J. Appl. Math.*, 42:558–574, 1982.

[4] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. *Trust Region Methods*. SIAM, Philadelphia, 2000.

[5] L. Eldén. Algorithms for the regularization of ill-conditioned least squares problems. *BIT*, 17:134–145, 1977.

[6] L. Eldén, P.C. Hansen, and M. Rojas. Minimization of linear functionals defined on solutions of large-scale discrete ill-posed problems. Technical Report 2003-5, Department of Mathematics, Wake Forest University, October 2003.

[7] C. Fortin and H. Wolkowicz. The trust-region subproblem and semidefinite programming. *Optim. Methods Softw.*, 19(1):41–67, 2004.

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

[9] G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, third edition, 1996.

[10] G.H. Golub and U. von Matt. Quadratically constrained least squares and quadratic problems. *Numer. Math.*, 59:561–580, 1991.

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

[12] N.I.M. Gould, D. Orban, and Ph.L. Toint. GALAHAD–a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. Technical Report RAL-TR-2002-014, Rutherford Appleton Laboratory, Chilton, England, 2002. Available from `http://galahad.rl.ac.uk/galahad-www`.

[13] W. W. Hager. Minimizing a quadratic over a sphere. *SIAM J. Optim.*, 12:188–208, 2001.

[14] M. Hanke and P.C. Hansen. Regularization methods for large–scale problems. *Surv. Math. Ind.*, 3:253–315, 1993.

[15] P.C. Hansen. The Discrete Picard Condition for discrete ill-posed problems. *BIT*, 30:658–672, 1990.

[16] P.C. Hansen. Regularization Tools: a MATLAB package for analysis and solution of discrete ill-posed problems. *Numer. Algo.*, 6:1–35, 1994.

[17] P.C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion.* SIAM, Philadelphia, 1998.

[18] HSL. HSL 2004: A collection of Fortran codes for large scale scientific computation, 2004. Available from `www.cse.clrc.ac.uk/nag/hsl`.

[19] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems.* SIAM, Philadelphia, 1995.

[20] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK User's Guide: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods.* SIAM, Philadelphia, 1998.

[21] S. Lucidi, L. Palagi, and M. Roma. On some properties of quadratic programs with a convex quadratic constraint. *SIAM J. Optim.*, 8:105–122, 1998.

[22] MATLAB:. *The Language of Technical Computing. Using MATLAB Version 6.* The MathWorks, Inc., Natick, Massachussetts, 2000.

[23] J.J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comput.*, 4(3):553–572, 1983.

[24] F. Natterer. *The Mathematics of Computerized Tomography*. John Wiley & Sons, New York, 1986.

[25] A. Neumaier. Solving ill-conditioned and singular linear systems: a tutorial on regularization. *SIAM Review*, 40(3):636–666, 1998.

[26] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[27] G. Nolet. *Seismic Tomography, with applications in global seismology and exploration geophysics*. D. Reidel Publishing Company, Dordrecht, 1987.

[28] T.W. Parks and C.S. Burrus. *Digital Filter Design*. John Wiley & Sons, New York, 1987.

[29] B.N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs NY, 1980.

[30] T. Pham-Dinh and H.A. Le-Thi. A D.C. optimization algorithm for solving the trust-region subproblem. *SIAM J. Optim.*, 8(2):476–505, 1998.

[31] F. Rendl and H. Wolkowicz. A semidefinite framework for trust region subproblems with applications to large scale minimization. *Math. Prog.*, 77(2):273–299, 1997.

[32] M. Rojas. A large-scale trust-region approach to the regularization of discrete ill-posed problems. Ph.D. thesis, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, Technical Report TR98-19, May 1998.

[33] M. Rojas, S.A. Santos, and D.C. Sorensen. A new matrix-free algorithm for the large-scale trust-region subproblem. *SIAM J. Optim.*, 11(3):611–646, 2000.

[34] M. Rojas and D.C. Sorensen. A trust-region approach to the regularization of large-scale discrete forms of ill-posed problems. *SIAM J. Sci. Comput.*, 23(3):1843–1861, 2002.

[35] M. Rojas and T. Steihaug. An interior-point trust-region-based method for large-scale non-negative regularization. *Inverse Problems*, 18(5):1291–1307, 2002.

[36] G.L.G. Sleijpen and H.A. Van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17:401–425, 1996.

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

[38] D.C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13(1):357–385, 1992.

[39] D.C. Sorensen. Minimization of a large-scale quadratic function subject to a spherical constraint. *SIAM J. Optim.*, 7(1):141–161, 1997.

[40] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, 20(3):626–637, 1983.

[41] W.W. Symes. A differential semblance criterion for inversion of multi-offset seismic reflection data. *Journal of Geophysical Research*, 98:2061–2073, 1993.

[42] A.N. Tikhonov. Regularization of incorrectly posed problems. *Soviet Math.*, 4:1624–1627, 1963.

[43] Ph.L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I.S. Duff, editor, *Sparse matrices and their uses*, pages 57–88. Academic Press, London, 1981.