Algorithm xxx: LSTRS Software Manual

MARIELBA ROJAS
Technical University of Denmark
SANDRA A. SANTOS
State University of Campinas
and
DANNY C. SORENSEN
Rice University

The software manual of 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 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 MATLAB software, version 1.2, is presented. A guide for using the software and examples are provided.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Documentation

General Terms: Algorithms, Design

Additional Key Words and Phrases: ARPACK, constrained quadratic optimization, Lanczos method, regularization, MATLAB, trust-region

Authors's Addresses:
M. Rojas, Informatics and Mathematical Modelling, Technical University of Denmark, 2800 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.
S.A. Santos, 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 (06/53768-0), CNPq (302412/2004-2), FINEP and FAEP-UNICAMP.
D.C. Sorensen, 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 Grants CCR-0306503, ACI-0325081 and CCF-0634902.

1. INTRODUCTION

This document contains the software manual for version 1.2 of a MATLAB [The MathWorks, Inc. 2000] 6.0 implementation of the LSTRS method [Rojas et al. 2000] for large-scale quadratic problems with a quadratic constraint, or trust-region subproblems:

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

where H is an $n \times n$, real, symmetric matrix, g is an n-dimensional real vector, Δ is a positive scalar, and $\|\cdot\|$ denotes the Euclidean norm.

The MATLAB implementation of LSTRS described in this manual 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. LSTRS is an iterative method that requires, at each step, the solution of a parameterized eigenvalue problem for the *bordered* matrix

$$B_\alpha = \begin{pmatrix} \alpha & g^T \\ g & H \end{pmatrix}$$

with α a scalar parameter. An eigenpair $\{\lambda, (1, x^T)^T\}$ of B_α provides points $\lambda, \phi(\lambda) = -g^T x$, and $\phi'(\lambda) = x^T x$, used in rational interpolation schemes to update the parameter α.

In the software, 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 [Lehoucq et al. 1998]), a combination of `eigs` with a Tchebyshev Spectral Transformation, or a user-provided routine.

The remainder of this document is organized as follows. In Section 2, we describe the main features of the software: data structures, interface, and components. In Section 3, we provide instructions for installing and running the software. In Section 4, we illustrate the use of the software with several examples.

2. THE MATLAB SOFTWARE

In this section, we describe our MATLAB 6.0 implementation of the LSTRS method from [Rojas et al. 2000]. 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.

2.1 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 2.2.1).

—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** ($\sqrt{\phi'(\lambda)}$).

2.2 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 between curly brackets.

2.2.1 *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 for the stopping criteria. 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 the eigensolver routine. Current choices for the eigensolver are:
—User-provided. See Section 2.2.3 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 [Lehoucq et al. 1998] implementation of the Implicitly Restarted Arnoldi Method [Sorensen 1992]). 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 as described in [Rojas and Sorensen 2002].

(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, the smallest eigenvalue of H), 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 an 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 2.2.2 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.issym** $=$ **1**, and **eigensolverpar.p** $=$ **7** (or $n+1$ if $n < 7$). Note that **eigensolverpar.p** is the number of vectors used by ARPACK, and hence by LSTRS.
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. Note that **eigensolverpar.k** $<$ **eigensolverpar.p** $\leq n$ must hold.

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**.

2.2.2 *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 2.

2.2.3 *Calling specifications for the eigensolver routine.* As explained in Section 2.2.1, 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 2.1.
—**eigensolverpar**: parameters (usually of type `struct`) for the eigensolver routine.

2.2.4 *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. 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 $\frac{\|(H-\lambda I)x+g\|}{\|g\|}$, and the final value of the parameter α.

2.3 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`.

2.4 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 9) 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.

2.5 Files

The LSTRS software follows a structured, top-down design. The MATLAB M-files containing the components of the software are presented in Figure 1.

The files **mv.m, uutmatvec.m, simple.m, vcalls1.m, icalls.m, vcalls2.m** and **regularization.m**, containing the examples in Section 4, 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 $A^T A$, with A an $m \times n$ matrix and $m \geq n$.

3. 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'`.

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.

Fig. 1. LSTRS M-files.

4. 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;
```

Fig. 2. 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;
```

Fig. 3. A matrix-vector multiplication routine *with* additional parameters.

REFERENCES

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

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

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

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

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

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

Received ?; revised ?; accepted ?

```
%
% 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 2.
%
[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
```

Fig. 4. Simple calls to **lstrs**.

```
%
% File:  vcalls1.m
% Uses the same data as in Figure 4
%
% 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 4
%
% 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.
```

Fig. 5. Valid and invalid calls to **lstrs**.

```
%
% File:  vcalls2.m
% Redefining struct parameters.
%
% uutmatvec is the matrix-vector multiplication routine in Figure 3
% 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);
```

Fig. 6. Redefining **struct** parameters for **lstrs**.

```
%
% File:  regularization.m
% Computes a regularized solution for problem phillips from
% Regularization Tools [Hansen 1994], 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); % Initial vector for ARPACK

% atamv is the routine in Figure 8

[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
```

Fig. 7. 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)';
```

Fig. 8. A matrix-vector multiplication routine that computes $w = A^T Av$.

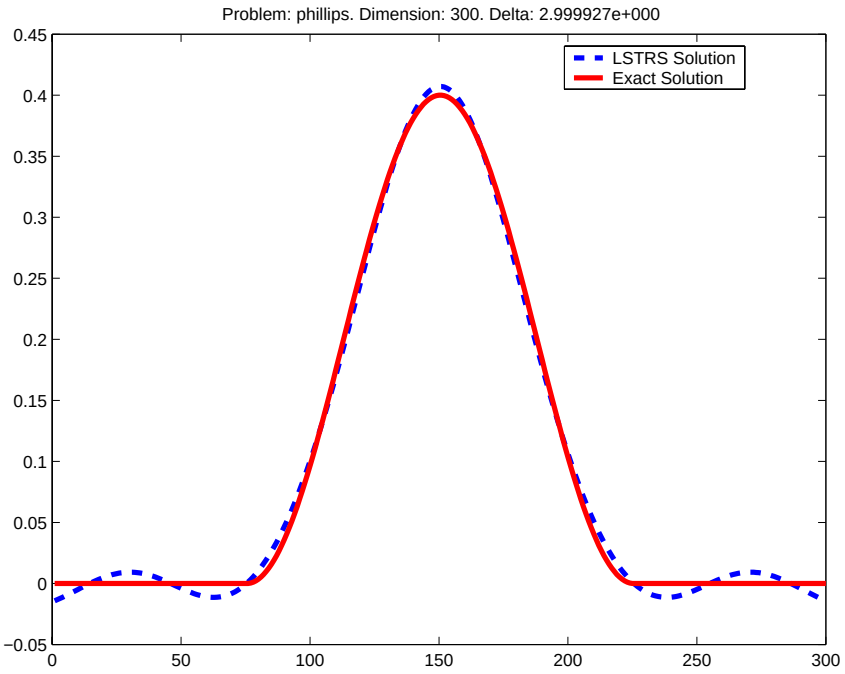

Fig. 9. LSTRS solution plot for regularization problem **phillips**. The **dashed** curve (LSTRS solution) will appear as **solid blue** on the screen when LSTRS is executed under MATLAB. The solid curve is the exact solution which has been added to the LSTRS plot for comparison (the solid curve is *not* generated by LSTRS).