```
function [ x, istop, itn, r1norm, r2norm, Anorm, Acond, Arnorm, xnorm, var ]...
   = lsqrSOL( m, n, A, b, damp, atol, btol, conlim, itnlim, show )

%        [ x, istop, itn, r1norm, r2norm, Anorm, Acond, Arnorm, xnorm, var ]...
%  = lsqrSOL( m, n, A, b, damp, atol, btol, conlim, itnlim, show );
%
% LSQR solves Ax = b or min ||b - Ax||_2 if damp = 0,
% or   min ||(b) - (  A   )x||   otherwise.
%          ||(0)    (damp*I) ||_2
% A is an m by n matrix (ideally sparse),
% or a function handle such that
%    y = A(x,1) returns y = A*x   (where x will be an n-vector);
%    y = A(x,2) returns y = A'*x  (where x will be an m-vector).

%----------------------------------------------------------------------
% LSQR uses an iterative (conjugate-gradient-like) method.
% For further information, see
% 1. C. C. Paige and M. A. Saunders (1982a).
%    LSQR: An algorithm for sparse linear equations and sparse least squares,
%    ACM TOMS 8(1), 43-71.
% 2. C. C. Paige and M. A. Saunders (1982b).
%    Algorithm 583.  LSQR: Sparse linear equations and least squares problems,
%    ACM TOMS 8(2), 195-209.
% 3. M. A. Saunders (1995).  Solution of sparse rectangular systems using
%    LSQR and CRAIG, BIT 35, 588-604.
%
% Input parameters:
% m   , n     are the dimensions of A.
% atol, btol  are stopping tolerances.  If both are 1.0e-9 (say),
%             the final residual norm should be accurate to about 9 digits.
%             (The final x will usually have fewer correct digits,
%             depending on cond(A) and the size of damp.)
% conlim      is also a stopping tolerance.  lsqr terminates if an estimate
%             of cond(A) exceeds conlim.  For compatible systems Ax = b,
%             conlim could be as large as 1.0e+12 (say).  For least-squares
%             problems, conlim should be less than 1.0e+8.
%             Maximum precision can be obtained by setting
%             atol = btol = conlim = zero, but the number of iterations
%             may then be excessive.
% itnlim      is an explicit limit on iterations (for safety).
% show = 1    gives an iteration log,
% show = 0    suppresses output.
%
% Output parameters:
% x           is the final solution.
% istop       gives the reason for termination.
% istop       = 1 means x is an approximate solution to Ax = b.
%             = 2 means x approximately solves the least-squares problem.
% r1norm      = norm(r), where r = b - Ax.
% r2norm      = sqrt( norm(r)^2  +  damp^2 * norm(x)^2 )
%             = r1norm if damp = 0.
% Anorm       = estimate of Frobenius norm of Abar = [  A   ].
%                                                    [damp*I]
% Acond       = estimate of cond(Abar).
% Arnorm      = estimate of norm(A'*r - damp^2*x).
% xnorm       = norm(x).
```

```
% var         (if present) estimates all diagonals of (A'A)^{-1} (if damp=0)
%             or (A'A + damp^2*I)^{-1} if damp > 0.
%             This is well defined if A has full column rank or damp > 0.
%             More precisely, var = diag(Dk*Dk'), where Dk is the n*k
%             matrix of search directions after k iterations.  Theoretically
%             Dk satisfies Dk'(A'A + damp^2*I)Dk = I for any A or damp.
%
%
%        1990: Derived from Fortran 77 version of LSQR.
% 22 May 1992: bbnorm was used incorrectly.  Replaced by Anorm.
% 26 Oct 1992: More input and output parameters added.
% 01 Sep 1994: Print log reformatted.
% 14 Jun 1997: show  added to allow printing or not.
% 30 Jun 1997: var   added as an optional output parameter.
% 07 Aug 2002: Output parameter rnorm replaced by r1norm and r2norm.
% 03 May 2007: Allow A to be a matrix or a function handle.
% 04 Sep 2011: Description of y = A(x,1) and y = A(x,2) corrected.
% 04 Sep 2011: I would like to allow an input x0.
%              If damp = 0 and x0 is nonzero, we could compute
%              r0 = b - A*x0, solve min ||r0 - A*dx||, and return
%              x = x0 + dx.  The current updating of "xnorm" would
%              give norm(dx), which we don't really need.  Instead
%              we would compute xnorm = norm(x0+dx) directly.
%
%              If damp is nonzero,  we would have to solve the bigger system
%                 min ||(    r0    ) - (   A    )dx||
%                     ||(-damp*x0)    (damp*I)  ||_2
%              with no benefit from the special structure.
%              Forget x0 for now and leave it to the user.
%
%              Michael Saunders, Systems Optimization Laboratory,
%              Dept of MS&E, Stanford University.
%-----------------------------------------------------------------------

% Initialize.

if isa(A,'numeric')
  explicitA = true;
elseif isa(A,'function_handle')
  explicitA = false;
else
  error('SOL:lsqrSOL:Atype','%s','A must be numeric or a function handle');
end

wantvar   = nargout >= 10;
if wantvar, var = zeros(n,1); end

msg=['The exact solution is  x = 0                              '
     'Ax - b is small enough, given atol, btol                  '
     'The least-squares solution is good enough, given atol     '
     'The estimate of cond(Abar) has exceeded conlim            '
     'Ax - b is small enough for this machine                   '
     'The least-squares solution is good enough for this machine'
     'Cond(Abar) seems to be too large for this machine         '
     'The iteration limit has been reached                      '];

```

```
if show
   disp(' ')
   disp('LSQR            Least-squares solution of  Ax = b')
   str1 = sprintf('The matrix A has %8g rows  and %8g cols', m,n);
   str2 = sprintf('damp = %20.14e    wantvar = %8g', damp,wantvar);
   str3 = sprintf('atol = %8.2e                 conlim = %8.2e', atol,conlim);
   str4 = sprintf('btol = %8.2e                 itnlim = %8g'  , btol,itnlim);
   disp(str1);   disp(str2);   disp(str3);   disp(str4);
end

itn    = 0;             istop  = 0;
ctol   = 0;             if conlim > 0, ctol = 1/conlim; end;
Anorm  = 0;             Acond  = 0;
dampsq = damp^2;        ddnorm = 0;             res2   = 0;
xnorm  = 0;             xxnorm = 0;             z      = 0;
cs2    = -1;            sn2    = 0;

% Set up the first vectors u and v for the bidiagonalization.
% These satisfy  beta*u = b,  alfa*v = A'u.

u      = b(1:m);        x    = zeros(n,1);
alfa   = 0;             beta = norm(u);
if beta > 0
   u = (1/beta)*u;
   if explicitA
     v = A'*u;
   else
     v = A(u,2);
   end
   alfa = norm(v);
end
if alfa > 0
   v = (1/alfa)*v;      w = v;
end

Arnorm = alfa*beta;     if Arnorm == 0, disp(msg(1,:)); return, end

rhobar = alfa;          phibar = beta;          bnorm  = beta;
rnorm  = beta;
r1norm = rnorm;
r2norm = rnorm;
head1  = '   Itn      x(1)       r1norm     r2norm ';
head2  = ' Compatible   LS      Norm A   Cond A';

if show
   disp(' ')
   disp([head1 head2])
   test1  = 1;          test2  = alfa / beta;
   str1   = sprintf( '%6g %12.5e',        itn,   x(1) );
   str2   = sprintf( ' %10.3e %10.3e', r1norm, r2norm );
   str3   = sprintf( '  %8.1e %8.1e',   test1,  test2 );
   disp([str1 str2 str3])
end

%-----------------------------------------------------------------
%     Main iteration loop.
```

```
%------------------------------------------------------------------
while itn < itnlim
  itn = itn + 1;

% Perform the next step of the bidiagonalization to obtain the
% next beta, u, alfa, v.  These satisfy the relations
%      beta*u  =  A*v  - alfa*u,
%      alfa*v  =  A'*u - beta*v.

  if explicitA
    u = A*v    - alfa*u;
  else
    u = A(v,1) - alfa*u;
  end
  beta = norm(u);
  if beta > 0
    u     = (1/beta)*u;
    Anorm = norm([Anorm alfa beta damp]);
    if explicitA
      v = A'*u   - beta*v;
    else
      v = A(u,2) - beta*v;
    end
    alfa  = norm(v);
    if alfa > 0,  v = (1/alfa)*v; end
  end

% Use a plane rotation to eliminate the damping parameter.
% This alters the diagonal (rhobar) of the lower-bidiagonal matrix.

  rhobar1 = norm([rhobar damp]);
  cs1     = rhobar/rhobar1;
  sn1     = damp  /rhobar1;
  psi     = sn1*phibar;
  phibar  = cs1*phibar;

% Use a plane rotation to eliminate the subdiagonal element (beta)
% of the lower-bidiagonal matrix, giving an upper-bidiagonal matrix.

  rho     =   norm([rhobar1 beta]);
  cs      =   rhobar1/rho;
  sn      =   beta   /rho;
  theta   =   sn*alfa;
  rhobar  = - cs*alfa;
  phi     =   cs*phibar;
  phibar  =   sn*phibar;
  tau     =   sn*phi;

% Update x and w.

  t1      =   phi  /rho;
  t2      = - theta/rho;
  dk      =   (1/rho)*w;

  x       = x      + t1*w;
  w       = v      + t2*w;
```

```
  ddnorm  = ddnorm + norm(dk)^2;
  if wantvar, var = var + dk.*dk; end

% Use a plane rotation on the right to eliminate the
% super-diagonal element (theta) of the upper-bidiagonal matrix.
% Then use the result to estimate  norm(x).

  delta   =   sn2*rho;
  gambar  = - cs2*rho;
  rhs     =   phi - delta*z;
  zbar    =   rhs/gambar;
  xnorm   =   sqrt(xxnorm + zbar^2);
  gamma   =   norm([gambar theta]);
  cs2     =   gambar/gamma;
  sn2     =   theta /gamma;
  z       =   rhs   /gamma;
  xxnorm  =   xxnorm + z^2;

% Test for convergence.
% First, estimate the condition of the matrix  Abar,
% and the norms of  rbar  and  Abar'rbar.

  Acond   =   Anorm*sqrt(ddnorm);
  res1    =   phibar^2;
  res2    =   res2 + psi^2;
  rnorm   =   sqrt(res1 + res2);
  Arnorm  =   alfa*abs(tau);

% 07 Aug 2002:
% Distinguish between
%    r1norm = ||b - Ax|| and
%    r2norm = rnorm in current code
%           = sqrt(r1norm^2 + damp^2*||x||^2).
%    Estimate r1norm from
%    r1norm = sqrt(r2norm^2 - damp^2*||x||^2).
% Although there is cancellation, it might be accurate enough.

  r1sq    =   rnorm^2 - dampsq*xxnorm;
  r1norm  =   sqrt(abs(r1sq));   if r1sq < 0, r1norm = - r1norm; end
  r2norm  =   rnorm;

% Now use these norms to estimate certain other quantities,
% some of which will be small near a solution.

  test1   =   rnorm /bnorm;
  test2   =   Arnorm/(Anorm*rnorm);
  test3   =        1/Acond;
  t1      =   test1/(1 + Anorm*xnorm/bnorm);
  rtol    =   btol + atol*Anorm*xnorm/bnorm;

% The following tests guard against extremely small values of
% atol, btol  or  ctol.  (The user may have set any or all of
% the parameters  atol, btol, conlim  to 0.)
% The effect is equivalent to the normal tests using
% atol = eps,  btol = eps,  conlim = 1/eps.

```

```
  if itn >= itnlim,   istop = 7; end
  if 1 + test3  <= 1, istop = 6; end
  if 1 + test2  <= 1, istop = 5; end
  if 1 + t1     <= 1, istop = 4; end

% Allow for tolerances set by the user.

  if  test3 <= ctol,  istop = 3; end
  if  test2 <= atol,  istop = 2; end
  if  test1 <= rtol,  istop = 1; end

% See if it is time to print something.

  prnt = 0;
  if n     <= 40       , prnt = 1; end
  if itn   <= 10       , prnt = 1; end
  if itn   >= itnlim-10, prnt = 1; end
  if rem(itn,10) == 0  , prnt = 1; end
  if test3 <=  2*ctol  , prnt = 1; end
  if test2 <= 10*atol  , prnt = 1; end
  if test1 <= 10*rtol  , prnt = 1; end
  if istop ~=  0       , prnt = 1; end

  if prnt
    if show
      str1 = sprintf( '%6g %12.5e',        itn,   x(1) );
      str2 = sprintf( ' %10.3e %10.3e', r1norm, r2norm );
      str3 = sprintf( '  %8.1e %8.1e',   test1,  test2 );
      str4 = sprintf( ' %8.1e %8.1e',    Anorm,  Acond );
      disp([str1 str2 str3 str4])
    end
  end
  if istop > 0, break, end
end

% End of iteration loop.
% Print the stopping condition.

if show
   fprintf('\nlsqrSOL finished\n')
   disp(msg(istop+1,:))
   disp(' ')
   str1 = sprintf( 'istop =%8g   r1norm =%8.1e',   istop, r1norm );
   str2 = sprintf( 'Anorm =%8.1e   Arnorm =%8.1e', Anorm, Arnorm );
   str3 = sprintf( 'itn   =%8g   r2norm =%8.1e',     itn, r2norm );
   str4 = sprintf( 'Acond =%8.1e   xnorm  =%8.1e', Acond, xnorm  );
   disp([str1 '   ' str2])
   disp([str3 '   ' str4])
   disp(' ')
end

%-----------------------------------------------------------------------
% end function lsqrSOL
%-----------------------------------------------------------------------
```