NAG Toolbox

nag_sparse_complex_gen_precon_ssor_solve (f11dr)

1 Purpose

nag_sparse_complex_gen_precon_ssor_solve (f11dr) solves a system of linear equations involving the preconditioning matrix corresponding to SSOR applied to a complex sparse non-Hermitian matrix, represented in coordinate storage format.

2 Syntax

```
[x, ifail] = nag_sparse_complex_gen_precon_ssor_solve(trans, a, irow, icol, rdiag, omega, check, y, 'n', n, 'nnz', nnz)
[x, ifail] = f11dr(trans, a, irow, icol, rdiag, omega, check, y, 'n', n, 'nnz', nnz)
```

3 Description

nag_sparse_complex_gen_precon_ssor_solve (f11dr) solves a system of linear equations

$$Mx = y, \qquad \text{or} \qquad M^{\mathrm{H}}x = y,$$

according to the value of the argument **trans**, where the matrix

$$M = \frac{1}{\omega(2-\omega)}(D+\omega L)D^{-1}(D+\omega U)$$

corresponds to symmetric successive-over-relaxation (SSOR) Young (1971) applied to a linear system $Ax = b$, where A is a complex sparse non-Hermitian matrix stored in coordinate storage (CS) format (see Section 2.1.1 in the F11 Chapter Introduction).

In the definition of M given above D is the diagonal part of A, L is the strictly lower triangular part of A, U is the strictly upper triangular part of A, and ω is a user-defined relaxation parameter.

It is envisaged that a common use of nag_sparse_complex_gen_precon_ssor_solve (f11dr) will be to carry out the preconditioning step required in the application of nag_sparse_complex_gen_basic_solver (f11bs) to sparse linear systems. For an illustration of this use of nag_sparse_complex_gen_precon_ssor_solve (f11dr) see the example program given in Section 10. nag_sparse_complex_gen_precon_ssor_solve (f11dr) is also used for this purpose by the Black Box function nag_sparse_complex_gen_solve_jacssor (f11ds).

4 References

Young D (1971) *Iterative Solution of Large Linear Systems* Academic Press, New York

5 Parameters

5.1 Compulsory Input Parameters

1: **trans** – CHARACTER(1)

Specifies whether or not the matrix M is transposed.

trans = 'N'
$Mx = y$ is solved.

trans = 'T'
$M^{\mathrm{H}}x = y$ is solved.

Constraint: **trans** = 'N' or 'T'.

2: **a**(**nnz**) – COMPLEX (KIND=nag_wp) array

The nonzero elements in the matrix A, ordered by increasing row index, and by increasing column index within each row. Multiple entries for the same row and column indices are not permitted. The function nag_sparse_complex_gen_sort (f11zn) may be used to order the elements in this way.

3: **irow**(**nnz**) – INTEGER array
4: **icol**(**nnz**) – INTEGER array

The row and column indices of the nonzero elements supplied in **a**.

Constraints:

irow and **icol** must satisfy the following constraints (which may be imposed by a call to nag_sparse_complex_gen_sort (f11zn)):

$1 \leq \mathbf{irow}(i) \leq \mathbf{n}$ and $1 \leq \mathbf{icol}(i) \leq \mathbf{n}$, for $i = 1, 2, \ldots, \mathbf{nnz}$;
either $\mathbf{irow}(i-1) < \mathbf{irow}(i)$ or both $\mathbf{irow}(i-1) = \mathbf{irow}(i)$ and $\mathbf{icol}(i-1) < \mathbf{icol}(i)$, for $i = 2, 3, \ldots, \mathbf{nnz}$.

5: **rdiag**(**n**) – COMPLEX (KIND=nag_wp) array

The elements of the diagonal matrix D^{-1}, where D is the diagonal part of A.

6: **omega** – REAL (KIND=nag_wp)

The relaxation parameter ω.

Constraint: $0.0 < \mathbf{omega} < 2.0$.

7: **check** – CHARACTER(1)

Specifies whether or not the CS representation of the matrix M should be checked.

check = 'C'
Checks are carried on the values of **n**, **nnz**, **irow**, **icol** and **omega**.

check = 'N'
None of these checks are carried out.

See also Section 9.2.

Constraint: **check** = 'C' or 'N'.

8: **y**(**n**) – COMPLEX (KIND=nag_wp) array

The right-hand side vector y.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **rdiag**, **y**. (An error is raised if these dimensions are not equal.)

n, the order of the matrix A.

Constraint: $\mathbf{n} \geq 1$.

2: **nz** – INTEGER

Default: the dimension of the arrays **a**, **irow**, **icol**. (An error is raised if these dimensions are not equal.)

The number of nonzero elements in the matrix A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

5.3 Output Parameters

1: **x**(**n**) – COMPLEX (KIND=nag_wp) array

The solution vector x.

2: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, **trans** $\neq$ 'N' or 'T',
or **check** $\neq$ 'C' or 'N'.

ifail $= 2$

On entry, **n** < 1,
or **nnz** < 1,
or **nnz** $> \mathbf{n}^2$,
or **omega** lies outside the interval $(0.0, 2.0)$,

ifail $= 3$

On entry, the arrays **irow** and **icol** fail to satisfy the following constraints:

$1 \leq \mathbf{irow}(i) \leq \mathbf{n}$ and $1 \leq \mathbf{icol}(i) \leq \mathbf{n}$, for $i = 1, 2, \ldots, \mathbf{nnz}$;

$\mathbf{irow}(i-1) < \mathbf{irow}(i)$ or $\mathbf{irow}(i-1) = \mathbf{irow}(i)$ and $\mathbf{icol}(i-1) < \mathbf{icol}(i)$, for $i = 2, 3, \ldots, \mathbf{nnz}$.

Therefore a nonzero element has been supplied which does not lie in the matrix A, is out of order, or has duplicate row and column indices. Call nag_sparse_complex_gen_sort (f11zn) to reorder and sum or remove duplicates.

ifail $= 4$

On entry, the matrix A has a zero diagonal element. The SSOR preconditioner is not appropriate for this problem.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

If **trans** = 'N' the computed solution x is the exact solution of a perturbed system of equations $(M + \delta M)x = y$, where

$$|\delta M| \leq c(n)\epsilon|D + \omega L|\left|D^{-1}\right||D + \omega U|,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***. An equivalent result holds when **trans** = 'T'.

8 Further Comments

8.1 Timing

The time taken for a call to nag_sparse_complex_gen_precon_ssor_solve (f11dr) is proportional to **nnz**.

8.2 Use of check

It is expected that a common use of nag_sparse_complex_gen_precon_ssor_solve (f11dr) will be to carry out the preconditioning step required in the application of nag_sparse_complex_gen_basic_solver (f11bs) to sparse linear systems. In this situation nag_sparse_complex_gen_precon_ssor_solve (f11dr) is likely to be called many times with the same matrix M. In the interests of both reliability and efficiency, you are recommended to set **check** = 'C' for the first of such calls, and **check** = 'N' for all subsequent calls.

9 Example

This example solves a complex sparse linear system of equations

$$Ax = b,$$

using RGMRES with SSOR preconditioning.

The RGMRES algorithm itself is implemented by the reverse communication function nag_sparse_complex_gen_basic_solver (f11bs), which returns repeatedly to the calling program with various values of the argument **irevcm**. This argument indicates the action to be taken by the calling program.

If **irevcm** = 1, a matrix-vector product $v = Au$ is required. This is implemented by a call to nag_sparse_complex_gen_matvec (f11xn).

If **irevcm** = −1, a conjugate transposed matrix-vector product $v = A^{\mathrm{H}}u$ is required in the estimation of the norm of A. This is implemented by a call to nag_sparse_complex_gen_matvec (f11xn).

If **irevcm** = 2, a solution of the preconditioning equation $Mv = u$ is required. This is achieved by a call to nag_sparse_complex_gen_precon_ssor_solve (f11dr).

If **irevcm** = 4, nag_sparse_complex_gen_basic_solver (f11bs) has completed its tasks. Either the iteration has terminated, or an error condition has arisen.

For further details see the function document for nag_sparse_complex_gen_basic_solver (f11bs).

9.1 Program Text

```
function f11dr_example

fprintf('f11dr example results\n\n');

% Solve sparse system Ax = b using preconditioned CGS

% Sparse A and b
n    = nag_int(5);
m    = nag_int(2);
nz   = nag_int(16);
sparseA = [ 2  3   1    1;
```

```
            1 -1   1    2;
           -1  0   1    4;
            0  2   2    2;
           -2  1   2    3;
            1  0   2    5;
            0 -1   3    1;
            5  4   3    3;
            3 -1   3    4;
            1  0   3    5;
           -2  2   4    1;
           -3  1   4    4;
            0  3   4    5;
            4 -2   5    2;
           -2  0   5    3;
           -6  1   5    5];
a(1:nz)    = sparseA(:,1) + i*sparseA(:,2);
irow(1:nz) = nag_int(sparseA(:,3));
icol(1:nz) = nag_int(sparseA(:,4));
b = [ -3 +  3i;
     -11 +  5i;
      23 + 48i;
     -41 +  2i;
     -28 - 31i];

% Solver setup input argument initialization
method = 'CGS';
precon = 'P';
norm_p = 'I';
tol    = 1e-10;
maxitn = nag_int(1000);
anorm  = 0;
sigmax = 0;
monit  = nag_int(0);
lwork  = max([121+n*(3+m)+m*(m+5),120+7*n,120+(2*n+m)*(m+2)+2*n,120+10*n]);

% Initialize solver
[lwreq, work, ifail] = f11br( ...
                             method, precon, n, m, tol, maxitn, anorm, ...
                             sigmax, monit, lwork, 'norm_p', norm_p);

rdiag = zeros(n, 1);
% Calculate reciprocal diagonal matrix elements for preconditioner
if precon == 'P' || precon == 'p'
   dcount = zeros(n, 1, nag_int_name);

   for j = 1:nz
     if irow(j) == icol(j)
       dcount(irow(j)) = dcount(irow(j)) + 1;
       if a(j) ~= 0
         rdiag(irow(j)) = 1/a(j);
       else
         error('Matrix has a zero diagonal element');
       end
     end
   end

   for j = 1:n
     if dcount(j)==0
       error('Matrix has a missing diagonal element');
     elseif dcount(j) > 1
       error('Matrix has a multiple diagonal element');
     end
   end
end
% Other preconditioner inputs
omega = 1.4;
ckdrf = 'C';

% Solver inputs
irevcm = nag_int(0);
wgt    = zeros(n, 1);
```

```
x      = complex(zeros(n, 1));

% matrix-vector input
ckxnf = 'C';

% solve the linear system by revere communication
while (irevcm ~= 4)
  [irevcm, x, b, work, ifail] = f11bs( ...
                                       irevcm, x, b, wgt, work);

  if (irevcm == -1)
    % b = A^T x
    [b, ifail] = f11xn( ...
                        'T', a, irow, icol, ckxnf, x);
    ckxnf = 'N';
  elseif (irevcm == 1)
    % b = Ax
    [b, ifail] = f11xn( ...
                        'N', a, irow, icol, ckxnf, x);
    ckxnf = 'N';
  elseif (irevcm == 2)
    % SSOR preconditioning
    [b, ifail] = f11dr( ...
                        'N', a, irow, icol, rdiag, omega, ckdrf, x);
    ckdrf = 'N';
  end
end

if ifail == 0
   % Successful termination, get solution details from communication array
  [itn, stplhs, stprhs, anorm, sigmax, work, ifail] = ...
    f11bt(work);

  fprintf('Converged in %d iterations\n', itn);
  fprintf('Matrix norm         = %16.3e\n', anorm);
  fprintf('Final residual norm = %16.4e\n\n', stplhs);
  disp('Solution');
  disp(x);

end
```

9.2 Program Results

```
     f11dr example results

Converged in 5 iterations
Matrix norm         =        1.500e+01
Final residual norm =       4.6185e-14

Solution
   1.0000 + 2.0000i
   2.0000 + 3.0000i
   3.0000 + 4.0000i
   4.0000 + 5.0000i
   5.0000 + 6.0000i
```