NAG Toolbox

nag_sparse_complex_gen_precon_jacobi (f11dx)

1 Purpose

nag_sparse_complex_gen_precon_jacobi (f11dx) computes the **approximate** solution of a complex, Hermitian or non-Hermitian, sparse system of linear equations applying a number of Jacobi iterations. It is expected that nag_sparse_complex_gen_precon_jacobi (f11dx) will be used as a preconditioner for the iterative solution of complex sparse systems of equations.

2 Syntax

```
[x, diag, ifail] = nag_sparse_complex_gen_precon_jacobi(store, trans, init, niter, a, irow, icol, check, b, diag, 'n', n, 'nnz', nnz)
[x, diag, ifail] = f11dx(store, trans, init, niter, a, irow, icol, check, b, diag, 'n', n, 'nnz', nnz)
```

3 Description

nag_sparse_complex_gen_precon_jacobi (f11dx) computes the **approximate** solution of the complex sparse system of linear equations $Ax = b$ using **niter** iterations of the Jacobi algorithm (see also Golub and Van Loan (1996) and Young (1971)):

$$x_{k+1} = x_k + D^{-1}(b - Ax_k) \tag{1}$$

where $k = 1, 2, \ldots,$ **niter** and $x_0 = 0$.

nag_sparse_complex_gen_precon_jacobi (f11dx) can be used both for non-Hermitian and Hermitian systems of equations. For Hermitian matrices, either all nonzero elements of the matrix A can be supplied using coordinate storage (CS), or only the nonzero elements of the lower triangle of A, using symmetric coordinate storage (SCS) (see the F11 Chapter Introduction).

It is expected that nag_sparse_complex_gen_precon_jacobi (f11dx) will be used as a preconditioner for the iterative solution of complex sparse systems of equations, using either the suite comprising the functions nag_sparse_complex_herm_basic_setup (f11gr), nag_sparse_complex_herm_basic_solver (f11gs) and nag_sparse_complex_herm_basic_diag (f11gt), for Hermitian systems, or the suite comprising the functions nag_sparse_complex_gen_basic_setup (f11br), nag_sparse_complex_gen_basic_solver (f11bs) and nag_sparse_complex_gen_basic_diag (f11bt), for non-Hermitian systems of equations.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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

5 Parameters

5.1 Compulsory Input Parameters

1: **store** – CHARACTER(1)

Specifies whether the matrix A is stored using symmetric coordinate storage (SCS) (applicable only to a Hermitian matrix A) or coordinate storage (CS) (applicable to both Hermitian and non-Hermitian matrices).

store = 'N'
The complete matrix A is stored in CS format.

store = 'S'
The lower triangle of the Hermitian matrix A is stored in SCS format.

Constraint: **store** = 'N' or 'S'.

2: **trans** – CHARACTER(1)

Suggested value: if the matrix A is Hermitian and stored in CS format, it is recommended that **trans** = 'N' for reasons of efficiency.

If **store** = 'N', specifies whether the approximate solution of $Ax = b$ or of $A^{\mathrm{H}}x = b$ is required.

trans = 'N'
The approximate solution of $Ax = b$ is calculated.

trans = 'T'
The approximate solution of $A^{\mathrm{H}}x = b$ is calculated.

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

3: **init** – CHARACTER(1)

Suggested value: **init** = 'I' on first entry; **init** = 'N', subsequently, unless **diag** has been overwritten.

On first entry, **init** should be set to `I', unless the diagonal elements of $A$ are already stored in the array **diag**. If **diag** already contains the diagonal of $A$, it must be set to `N'.

init = 'N'
diag must contain the diagonal of A.

init = 'I'
diag will store the diagonal of A on exit.

Constraint: **init** = 'N' or 'I'.

4: **niter** – INTEGER

The number of Jacobi iterations requested.

Constraint: **niter** ≥ 1.

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

If **store** = 'N', the nonzero elements in the matrix A (CS format).

If **store** = 'S', the nonzero elements in the lower triangle of the matrix A (SCS format).

In both cases, the elements of either A or of its lower triangle must be ordered by increasing row index and by increasing column index within each row. Multiple entries for the same row and columns indices are not permitted. The function nag_sparse_complex_gen_sort (f11zn) or nag_sparse_complex_herm_sort (f11zp) may be used to reorder the elements in this way for CS and SCS storage, respectively.

6: **irow**(**nnz**) – INTEGER array
7: **icol**(**nnz**) – INTEGER array

If **store** = 'N', the row and column indices of the nonzero elements supplied in **a**.

If **store** = 'S', the row and column indices of the nonzero elements of the lower triangle of the matrix A supplied in **a**.

Constraints:

$1 \leq \mathbf{irow}(i) \leq \mathbf{n}$, for $i = 1, 2, \ldots, \mathbf{nnz}$;
if **store** = 'N', $1 \leq \mathbf{icol}(i) \leq \mathbf{n}$, for $i = 1, 2, \ldots, \mathbf{nnz}$;
if **store** = 'S', $1 \leq \mathbf{icol}(i) \leq \mathbf{irow}(i)$, 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 $i = 2, 3, \ldots, \mathbf{nnz}$.

8: **check** – CHARACTER(1)

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

check = 'C'
Checks are carried out on the values of **n**, **nnz**, **irow**, **icol**; if **init** = 'N', **diag** is also checked.

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

See also Section 9.2.

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

9: **b**(**n**) – COMPLEX (KIND=nag_wp) array

The right-hand side vector b.

10: **diag**(**n**) – COMPLEX (KIND=nag_wp) array

If **init** = 'N', the diagonal elements of A.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **b**, **diag**. (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.)

If **store** = 'N', the number of nonzero elements in the matrix A.

If **store** = 'S', the number of nonzero elements in the lower triangle of the matrix A.

Constraints:

if **store** = 'N', $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$;
if **store** = 'S', $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n}+1)/2$.

5.3 Output Parameters

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

The approximate solution vector $x_{\mathbf{niter}}$.

2: **diag**(**n**) – COMPLEX (KIND=nag_wp) array

If **init** = 'N', unchanged on exit.

If **init** = 'I', the diagonal elements of A.

3: **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, **store** ≠ 'N' or 'S',
or **trans** ≠ 'N' or 'T',
or **init** ≠ 'N' or 'I',
or **check** ≠ 'C' or 'N',
or **niter** ≤ 0.

ifail = 2

On entry, **n** < 1,
or **nnz** < 1,
or **nnz** > $\mathbf{n}^2$, if **store** = 'N',
or 1 ≤ **nnz** ≤ [**n**(**n** + 1)]/2, if **store** = 'S'.

ifail = 3

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

1 ≤ **irow**(i) ≤ **n** and

if **store** = 'N' then 1 ≤ **icol**(i) ≤ **n**, or

if **store** = 'S' then 1 ≤ **icol**(i) ≤ **irow**(i), for $i = 1, 2, \ldots,$ **nnz**.

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

Therefore a nonzero element has been supplied which does not lie within the matrix A, is out of order, or has duplicate row and column indices. Call either nag_sparse_real_gen_sort (f11za) or nag_sparse_real_symm_sort (f11zb) to reorder and sum or remove duplicates when **store** = 'N' or **store** = 'S', respectively.

ifail = 4

On entry, **init** = 'N' and some diagonal elements of A stored in **diag** are zero.

ifail = 5

On entry, **init** = 'I' and some diagonal elements of A are zero.

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

In general, the Jacobi method cannot be used on its own to solve systems of linear equations. The rate of convergence is bound by its spectral properties (see, for example, Golub and Van Loan (1996)) and as a solver, the Jacobi method can only be applied to a limited set of matrices. One condition that guarantees convergence is strict diagonal dominance.

However, the Jacobi method can be used successfully as a preconditioner to a wider class of systems of equations. The Jacobi method has good vector/parallel properties, hence it can be applied very efficiently. Unfortunately, it is not possible to provide criteria which define the applicability of the Jacobi method as a preconditioner, and its usefulness must be judged for each case.

8 Further Comments

8.1 Timing

The time taken for a call to nag_sparse_complex_gen_precon_jacobi (f11dx) is proportional to **niter** $\times$ **nnz**.

8.2 Use of check

It is expected that a common use of nag_sparse_complex_gen_precon_jacobi (f11dx) will be as preconditioner for the iterative solution of complex, Hermitian or non-Hermitian, linear systems. In this situation, nag_sparse_complex_gen_precon_jacobi (f11dx) is likely to be called many times. In the interests of both reliability and efficiency, you are recommended to set **check** $=$ 'C' for the first of such calls, and to set **check** $=$ 'N' for all subsequent calls.

9 Example

This example solves the complex sparse non-Hermitian system of equations $Ax = b$ iteratively using nag_sparse_complex_gen_precon_jacobi (f11dx) as a preconditioner.

9.1 Program Text

```
function f11dx_example

fprintf('f11dx example results\n\n');

% Solve sparse system Ax = b iteratively using Jacobi preconditioner

% Define sparse matrix A and RHS B
nz   = nag_int(24);
n    = nag_int(8);
a = [ 2 + 1i, -1 + 1i,  1 - 3i,  4 + 7i, -3 + 0i,  2 + 4i, ...
     -7 - 5i,  2 + 1i,  3 + 2i, -4 + 2i,  0 + 1i,  5 - 3i, ...
     -1 + 2i,  8 + 6i, -3 - 4i, -6 - 2i,  5 - 2i,  2 + 0i, ...
      0 - 5i, -1 + 5i,  6 + 2i, -1 + 4i,  2 + 0i,  3 + 3i];
b = [ 7 + 11i;
      1 + 24i;
    -13 - 18i;
    -10 +  3i;
     23 + 14i;
     17 -  7i;
     15 -  3i;
     -3 + 20i];
irow = nag_int(...
```

```
               [1; 1; 1; 2; 2; 2; 3; 3; 4; 4; 4; 4;
                5; 5; 5; 6; 6; 6; 7; 7; 7; 8; 8; 8]);
icol = nag_int(...
               [1; 4; 8; 1; 2; 5; 3; 6; 1; 3; 4; 7;
                2; 5; 7; 1; 3; 6; 3; 5; 7; 2; 6; 8]);

b = [  7 + 11i;
       1 + 24i;
     -13 - 18i;
     -10 +  3i;
      23 + 14i;
      17 -  7i;
      15 -  3i;
      -3 + 20i];

% Iterative Solution Setup
%      Input parameters
method = 'TFQMR';
precon = 'p';
m      = nag_int(2);
tol    = 1e-6;
maxitn = nag_int(20);
anorm  = 0;
sigmax = 0;
monit  = nag_int(1);
lwork = nag_int(300);

% Initialise Solver
[lwreq, work, ifail] = f11br( ...
                             method, precon, n, m, tol, maxitn, anorm, ...
                             sigmax, monit, lwork, 'norm_p', '1');

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

% Preconditioner inputs
store = 'Non Hermitian';
trans = 'N';
init  = 'I';
niter = nag_int(2);
check = 'Check';
diag  = complex(zeros(n, 1));

% Solve by reverse communication
while (irevcm ~= 4)
  [irevcm, x, b, work, ifail] = f11bs( ...
                                      irevcm, x, b, wgt, work);

  if (irevcm == -1)
    % b = A^Tx
    [b, ifail] = f11xn('T', a, irow, icol, 'N', x);
  elseif (irevcm == 1)
    % b = Ax
    [b, ifail] = f11xn('N', a, irow, icol, 'N', x);
  elseif (irevcm == 2)
    % Jacobi preconditioner
    [b, diag, ifail] = f11dx( ...
                             store, trans, init, niter, ...
                             a, irow, icol, check, x, diag);
  elseif (irevcm == 3)
    [itn, stplhs, stprhs, anorm, sigmax, work, ifail] = ...
    f11bt(work);
    fprintf('\nMonitoring at iteration number %d\n', itn);
    fprintf('residual norm: %14.4e\n', stplhs);
  end
end

% Obtain information about the computation
[itn, stplhs, stprhs, anorm, sigmax, work, ifail] = ...
```

```
f11bt(work);
fprintf('\nNumber of iterations for convergence: %d\n', itn);
fprintf('Residual norm:                          %14.4e\n', stplhs);
fprintf('Right-hand side of termination criteria: %14.4e\n', stprhs);
fprintf('i-norm of matrix a:                     %14.4e\n', anorm);
fprintf('\n   Solution Vector\n');
disp(x);
fprintf('\n   Residual Vector\n');
disp(b);
```

9.2 Program Results

```
     f11dx example results

Monitoring at iteration number 1
residual norm:     1.5062e+02

Monitoring at iteration number 2
residual norm:     1.5704e+02

Monitoring at iteration number 3
residual norm:     1.4803e+02

Monitoring at iteration number 4
residual norm:     8.5215e+01

Monitoring at iteration number 5
residual norm:     4.2951e+01

Monitoring at iteration number 6
residual norm:     2.5055e+01

Monitoring at iteration number 7
residual norm:     1.9090e-01

Number of iterations for convergence: 8
Residual norm:                                 9.5485e-08
Right-hand side of termination criteria:       8.9100e-04
i-norm of matrix a:                            2.7000e+01

   Solution Vector
   1.0000 + 1.0000i
   2.0000 - 1.0000i
   3.0000 + 1.0000i
   4.0000 - 1.0000i
   3.0000 - 1.0000i
   2.0000 + 1.0000i
   1.0000 - 1.0000i
   0.0000 + 3.0000i

   Residual Vector
   1.0e-07 *

   0.0471 - 0.0394i
   0.0877 - 0.0981i
  -0.0287 + 0.0416i
   0.0358 - 0.1112i
  -0.0692 - 0.0869i
  -0.0225 + 0.0849i
   0.0052 - 0.0334i
  -0.0558 - 0.1073i
```