NAG Toolbox

nag_sparse_complex_herm_solve_jacssor (f11js)

1 Purpose

nag_sparse_complex_herm_solve_jacssor (f11js) solves a complex sparse Hermitian system of linear equations, represented in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, without preconditioning, with Jacobi or with SSOR preconditioning.

2 Syntax

```
[x, rnorm, itn, rdiag, ifail] = nag_sparse_complex_herm_solve_jacssor(method, precon, a, irow, icol, omega, b, tol, maxitn, x, 'n', n, 'nnz', nnz)
[x, rnorm, itn, rdiag, ifail] = f11js(method, precon, a, irow, icol, omega, b, tol, maxitn, x, 'n', n, 'nnz', nnz)
```

3 Description

nag_sparse_complex_herm_solve_jacssor (f11js) solves a complex sparse Hermitian linear system of equations

$$Ax = b,$$

using a preconditioned conjugate gradient method (see Barrett *et al.* (1994)), or a preconditioned Lanczos method based on the algorithm SYMMLQ (see Paige and Saunders (1975)). The conjugate gradient method is more efficient if A is positive definite, but may fail to converge for indefinite matrices. In this case the Lanczos method should be used instead. For further details see Barrett *et al.* (1994).

nag_sparse_complex_herm_solve_jacssor (f11js) allows the following choices for the preconditioner:

- no preconditioning;
- Jacobi preconditioning (see Young (1971));
- symmetric successive-over-relaxation (SSOR) preconditioning (see Young (1971)).

For incomplete Cholesky (IC) preconditioning see nag_sparse_complex_herm_solve_ilu (f11jq).

The matrix A is represented in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the F11 Chapter Introduction) in the arrays **a**, **irow** and **icol**. The array **a** holds the nonzero entries in the lower triangular part of the matrix, while **irow** and **icol** hold the corresponding row and column indices.

4 References

Barrett R, Berry M, Chan T F, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C and Van der Vorst H (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM, Philadelphia

Paige C C and Saunders M A (1975) Solution of sparse indefinite systems of linear equations *SIAM J. Numer. Anal.* **12** 617–629

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

5 Parameters

5.1 Compulsory Input Parameters

1: **method** – CHARACTER(*)

Specifies the iterative method to be used.

method = 'CG'
Conjugate gradient method.

method = 'SYMMLQ'
Lanczos method (SYMMLQ).

Constraint: **method** = 'CG' or 'SYMMLQ'.

2: **precon** – CHARACTER(1)

Specifies the type of preconditioning to be used.

precon = 'N'
No preconditioning.

precon = 'J'
Jacobi.

precon = 'S'
Symmetric successive-over-relaxation (SSOR).

Constraint: **precon** = 'N', 'J' or 'S'.

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

The nonzero elements of the lower triangular part of 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_herm_sort (f11zp) may be used to order the elements in this way.

4: **irow**(**nnz**) – INTEGER array

5: **icol**(**nnz**) – INTEGER array

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

Constraints:

irow and **icol** must satisfy these constraints (which may be imposed by a call to nag_sparse_complex_herm_sort (f11zp)):

$1 \leq \mathbf{irow}(i) \leq \mathbf{n}$ and $1 \leq \mathbf{icol}(i) \leq \mathbf{irow}(i)$, 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 $i = 2, 3, \ldots, \mathbf{nnz}$.

6: **omega** – REAL (KIND=nag_wp)

If **precon** = 'S', **omega** is the relaxation parameter ω to be used in the SSOR method. Otherwise **omega** need not be initialized.

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

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

The right-hand side vector b.

8: **tol** – REAL (KIND=nag_wp)

The required tolerance. Let x_k denote the approximate solution at iteration k, and r_k the corresponding residual. The algorithm is considered to have converged at iteration k if

$$\|r_k\|_\infty \leq \tau \times \left(\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty\right).$$

If **tol** ≤ 0.0, $\tau = \max\sqrt{\epsilon}, 10\epsilon, \sqrt{n}\epsilon$ is used, where ϵ is the ***machine precision***. Otherwise $\tau = \max(\textbf{tol}, 10\epsilon, \sqrt{n}\epsilon)$ is used.

Constraint: **tol** < 1.0.

9: **maxitn** – INTEGER

The maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

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

An initial approximation to the solution vector x.

5.2 Optional Input Parameters

1: **n** – INTEGER

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

n, the order of the matrix A.

Constraint: **n** ≥ 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 lower triangular part of the matrix A.

Constraint: $1 \leq \textbf{nnz} \leq \textbf{n} \times (\textbf{n} + 1)/2$.

5.3 Output Parameters

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

An improved approximation to the solution vector x.

2: **rnorm** – REAL (KIND=nag_wp)

The final value of the residual norm $\|r_k\|$, where k is the output value of **itn**.

3: **itn** – INTEGER

The number of iterations carried out.

4: **rdiag**(**n**) – REAL (KIND=nag_wp) array

The elements of the diagonal matrix D^{-1}, where D is the diagonal part of A. Note that since A is Hermitian the elements of D^{-1} are necessarily real.

5: **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, **method** $\neq$ 'CG' or 'SYMMLQ',
or **precon** $\neq$ 'N', 'J' or 'S',
or **n** < 1,
or **nnz** < 1,
or **nnz** $> \mathbf{n} \times (\mathbf{n}+1)/2$,
or **omega** lies outside the interval $(0.0, 2.0)$,
or **tol** ≥ 1.0,
or **maxitn** < 1,
or *lwork* is too small.

ifail $= 2$

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

$1 \leq \mathbf{irow}(i) \leq \mathbf{n}$ and $1 \leq \mathbf{icol}(i) \leq \mathbf{irow}(i)$, 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 lower triangular part of A, is out of order, or has duplicate row and column indices. Call nag_sparse_complex_herm_sort (f11zp) to reorder and sum or remove duplicates.

ifail $= 3$

On entry, the matrix A has a zero diagonal element. Jacobi and SSOR preconditioners are not appropriate for this problem.

ifail $= 4$

The required accuracy could not be obtained. However, a reasonable accuracy has been obtained and further iterations could not improve the result.

ifail $= 5$

Required accuracy not obtained in **maxitn** iterations.

ifail $= 6$

The preconditioner appears not to be positive definite.

ifail $= 7$

The matrix of the coefficients appears not to be positive definite (conjugate gradient method only).

ifail $= 8$

A serious error has occurred in an internal call to an auxiliary function. Check all function calls and array sizes. Seek expert help.

ifail $= 9$

The matrix of the coefficients has a non-real diagonal entry, and is therefore not Hermitian.

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

On successful termination, the final residual $r_k = b - Ax_k$, where $k =$ **itn**, satisfies the termination criterion

$$\|r_k\|_\infty \leq \tau \times \left(\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty\right).$$

The value of the final residual norm is returned in **rnorm**.

8 Further Comments

The time taken by nag_sparse_complex_herm_solve_jacssor (f11js) for each iteration is roughly proportional to **nnz**. One iteration with the Lanczos method (SYMMLQ) requires a slightly larger number of operations than one iteration with the conjugate gradient method.

The number of iterations required to achieve a prescribed accuracy cannot easily be determined *a priori*, as it can depend dramatically on the conditioning and spectrum of the preconditioned matrix of the coefficients $\bar{A} = M^{-1}A$.

9 Example

This example solves a complex sparse Hermitian positive definite system of equations using the conjugate gradient method, with SSOR preconditioning.

9.1 Program Text

```
     function f11js_example

fprintf('f11js example results\n\n');

% Solve sparse Hermitian system Ax = b using CG method with
% SSOR preconditioning.

% Define A and b
n  = nag_int(9);
nz = nag_int(23);
a  = [ 6 + 0.i; -1 + 1.i;  6 + 0.i;  0 + 1.i;
       5 + 0.i;  5 + 0.i;  2 - 2.i;  4 + 0.i;
       1 + 1.i;  2 + 0.i;  6 + 0.i; -4 + 3.i;
       0 + 1.i; -1 + 0.i;  6 + 0.i; -1 - 1.i;
       0 - 1.i;  9 + 0.i;  1 + 3.i;  1 + 2.i;
      -1 + 0.i;  1 + 4.i;  9 + 0.i];
b  = [ 8 + 54i;-10 - 92i; 25 + 27i; 26 - 28i;
      54 + 12i; 26 - 22i; 47 + 65i; 71 - 57i;
      60 + 70i];
irow = nag_int([1;2;2;3;3;4;5;5;6;6;6;7;7;7;7;8;8;8;9;9;9;9;9]);
icol = nag_int([1;1;2;2;3;4;1;5;3;4;6;2;5;6;7;4;6;8;1;5;6;8;9]);

% Solve
method = 'CG';
precon = 'S';
omega  = 1.1;
tol    = 1e-06;
maxitn = nag_int(100);
x      = complex(zeros(n,1));

[x, rnorm, itn, rdiag, ifail] = ...
f11js( ...
```

```
       method, precon, a, irow, icol, omega, b, tol, maxitn, x);

fprintf('Converged in %d iterations\n', itn);
fprintf('Final redidual norm = %16.3d\n\n', rnorm);
disp('Solution');
disp(x);
```

9.2 Program Results

```
     f11js example results

Converged in 7 iterations
Final redidual norm =        1.477e-05

Solution
   1.0000 + 9.0000i
   2.0000 - 8.0000i
   3.0000 + 7.0000i
   4.0000 - 6.0000i
   5.0000 + 5.0000i
   6.0000 - 4.0000i
   7.0000 + 3.0000i
   8.0000 - 2.0000i
   9.0000 + 1.0000i
```