NAG Toolbox

nag_sparse_complex_gen_solve_jacssor (f11ds)

1 Purpose

nag_sparse_complex_gen_solve_jacssor (f11ds) solves a complex sparse non-Hermitian system of linear equations, represented in coordinate storage format, using a restarted generalized minimal residual (RGMRES), conjugate gradient squared (CGS), stabilized bi-conjugate gradient (Bi-CGSTAB), or transpose-free quasi-minimal residual (TFQMR) method, without preconditioning, with Jacobi, or with SSOR preconditioning.

2 Syntax

```
[x, rnorm, itn, ifail] = nag_sparse_complex_gen_solve_jacssor(method, precon, a, irow, icol, omega, b, m, tol, maxitn, x, 'n', n, 'nnz', nnz)
[x, rnorm, itn, ifail] = f11ds(method, precon, a, irow, icol, omega, b, m, tol, maxitn, x, 'n', n, 'nnz', nnz)
```

3 Description

nag_sparse_complex_gen_solve_jacssor (f11ds) solves a complex sparse non-Hermitian system of linear equations:

$$Ax = b,$$

using an RGMRES (see Saad and Schultz (1986)), CGS (see Sonneveld (1989)), Bi-CGSTAB(ℓ) (see Van der Vorst (1989) and Sleijpen and Fokkema (1993)), or TFQMR (see Freund and Nachtigal (1991) and Freund (1993)) method.

nag_sparse_complex_gen_solve_jacssor (f11ds) 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 LU (ILU) preconditioning see nag_sparse_complex_gen_solve_ilu (f11dq).

The matrix A is represented in coordinate storage (CS) format (see Section 2.1.1 in the F11 Chapter Introduction) in the arrays **a**, **irow** and **icol**. The array **a** holds the nonzero entries in the matrix, while **irow** and **icol** hold the corresponding row and column indices.

nag_sparse_complex_gen_solve_jacssor (f11ds) is a Black Box function which calls nag_sparse_complex_gen_basic_setup (f11br), nag_sparse_complex_gen_basic_solver (f11bs) and nag_sparse_complex_gen_basic_diag (f11bt). If you wish to use an alternative storage scheme, preconditioner, or termination criterion, or require additional diagnostic information, you should call these underlying functions directly.

4 References

Freund R W (1993) A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems *SIAM J. Sci. Comput.* **14** 470–482

Freund R W and Nachtigal N (1991) QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems *Numer. Math.* **60** 315–339

Saad Y and Schultz M (1986) GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **7** 856–869

Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(ℓ) for linear equations involving matrices with complex spectrum *ETNA* **1** 11–32

Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **10** 36–52

Van der Vorst H (1989) Bi-CGSTAB, a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **13** 631–644

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 = 'RGMRES'
Restarted generalized minimum residual method.

method = 'CGS'
Conjugate gradient squared method.

method = 'BICGSTAB'
Bi-conjugate gradient stabilized (ℓ) method.

method = 'TFQMR'
Transpose-free quasi-minimal residual method.

Constraint: **method** = 'RGMRES', 'CGS', 'BICGSTAB' or 'TFQMR'.

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 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.

4: **irow**(**nnz**) – INTEGER array

5: **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 $i = 2, 3, \ldots, \mathbf{nnz}$.

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 and is not referenced.

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

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

The right-hand side vector b.

8: **m** – INTEGER

If **method** = 'RGMRES', **m** is the dimension of the restart subspace.

If **method** = 'BICGSTAB', **m** is the order ℓ of the polynomial Bi-CGSTAB method.

Otherwise, **m** is not referenced.

Constraints:

if **method** = 'RGMRES', $0 < \mathbf{m} \leq \min(\mathbf{n}, 50)$;
if **method** = 'BICGSTAB', $0 < \mathbf{m} \leq \min(\mathbf{n}, 10)$.

9: **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 $\mathbf{tol} \leq 0.0$, $\tau = \max\left(\sqrt{\epsilon}, 10\epsilon, \sqrt{n}\epsilon\right)$ is used, where ϵ is the ***machine precision***. Otherwise $\tau = \max(\mathbf{tol}, 10\epsilon, \sqrt{n}\epsilon)$ is used.

Constraint: **tol** < 1.0.

10: **maxitn** – INTEGER

The maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

11: **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 matrix A.

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

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 $\|r_k\|_\infty$, where k is the output value of **itn**.

3: **itn** – INTEGER

The number of iterations carried out.

4: **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** $\neq$ 'RGMRES', 'CGS', 'BICGSTAB' or 'TFQMR',
or **precon** $\neq$ 'N', 'J' or 'S',
or **n** < 1,
or **nnz** < 1,
or **nnz** $>$ **n**^{2},
or **precon** $=$ 'S' and **omega** lies outside the interval $(0.0, 2.0)$,
or **m** < 1,
or **m** $> \min($**n**$, 50)$, when **method** $=$ 'RGMRES',
or **m** $> \min($**n**$, 10)$, when **method** $=$ 'BICGSTAB',
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:

$1 \leq$ **irow**$(i) \leq$ **n** and $1 \leq$ **icol**$(i) \leq$ **n**, 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 nag_sparse_complex_gen_sort (f11zn) to reorder and sum or remove duplicates.

ifail $= 3$

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

ifail $= 4$ (*warning*)

The required accuracy could not be obtained. However, a reasonable accuracy may have been obtained, and further iterations could not improve the result. You should check the output value of **rnorm** for acceptability. This error code usually implies that your problem has been fully and satisfactorily solved to within or close to the accuracy available on your system. Further iterations are unlikely to improve on this situation.

ifail $= 5$

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

ifail $= 6$

Algorithmic breakdown. A solution is returned, although it is possible that it is completely inaccurate.

ifail $= 7$ (nag_sparse_complex_gen_basic_setup (f11br), nag_sparse_complex_gen_basic_solver (f11bs) or nag_sparse_complex_gen_basic_diag (f11bt))

A serious error has occurred in an internal call to one of the specified functions. Check all function calls and array sizes. Seek expert help.

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_gen_solve_jacssor (f11ds) for each iteration is roughly proportional to **nnz**.

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 coefficient matrix $\bar{A} = M^{-1}A$, for some preconditioning matrix M.

9 Example

This example solves a complex sparse non-Hermitian system of equations using the CGS method, with no preconditioning.

9.1 Program Text

```
function f11ds_example

fprintf('f11ds example results\n\n');

% Solve sparse system Ax = b using preconditioned CGS

% Sparse A and b
n    = nag_int(5);
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 = 'N';
omega  = 1.05;
m      = nag_int(1);
tol    = 1e-10;
maxitn = nag_int(1000);
x      = complex(zeros(n, 1));

[x, rnorm, itn, ifail] = ...
f11ds( ...
       method, precon, a, irow, icol, omega, b, m, tol, maxitn, x);

fprintf('Converged in %d iterations\n', itn);
fprintf('Final residual norm = %16.4e\n\n', rnorm);
disp('Solution');
disp(x);
```

9.2 Program Results

```
     f11ds example results

Converged in 5 iterations
Final residual norm =       1.0520e-10

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