NAG Toolbox for MATLAB

f11js

1 Purpose

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

2 Syntax

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

3 Description

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.

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 f11jq.

The matrix A is represented in symmetric co-ordinate 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 – string**

Specifies the iterative method to be used.

method = 'CG'

Conjugate gradient method.

method = 'SYMMLQ'

Lanczos method (SYMMLQ).

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

2: **precon – string**

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 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 f11zp may be used to order the elements in this way.

4: **irow(nnz) – int32 array**
5: **icol(nnz) – int32 array**

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

Constraints:

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

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

6: **omega – double scalar**

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

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

7: **b(n) – complex array**

The right-hand side vector b.

8: **tol – double scalar**

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

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

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

Constraint: $\mathbf{tol} < 1.0$.

9: **maxitn – int32 scalar**

The maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

10: **x(n) – complex array**

An initial approximation to the solution vector x.

5.2 Optional Input Parameters

1: **n – int32 scalar**

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

n, the order of the matrix A.

Constraint: **n** ≥ 1.

2: **nnz – int32 scalar**

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

5.3 Input Parameters Omitted from the MATLAB Interface

work, lwork, iwork

5.4 Output Parameters

1: **x(n) – complex array**

An improved approximation to the solution vector x.

2: **rnorm – double scalar**

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

3: **itn – int32 scalar**

The number of iterations carried out.

4: **rdiag(n) – double 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 – int32 scalar**

0 unless the function detects an error (see Section 6).

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** $>$ **n** $\times$ (**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:

$1 \leq$ **irow**$(i) \leq$ **n** and $1 \leq$ **icol**$(i) \leq$ **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 in the lower triangular part of A, is out of order, or has duplicate row and column indices. Call 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 (sub)program 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.

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 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

```
method = 'CG    ';
precon = 'S';
a = [complex(6, +0);
     complex(-1, +1);
     complex(6, +0);
     complex(0, +1);
     complex(5, +0);
     complex(5, +0);
     complex(2, -2);
     complex(4, +0);
     complex(1, +1);
     complex(2, +0);
     complex(6, +0);
     complex(-4, +3);
     complex(0, +1);
     complex(-1, +0);
     complex(6, +0);
     complex(-1, -1);
     complex(0, -1);
     complex(9, +0);
     complex(1, +3);
     complex(1, +2);
     complex(-1, +0);
     complex(1, +4);
     complex(9, +0)];
irow = [int32(1);
     int32(2);
     int32(2);
     int32(3);
     int32(3);
     int32(4);
     int32(5);
     int32(5);
     int32(6);
     int32(6);
     int32(6);
     int32(7);
     int32(7);
     int32(7);
     int32(7);
     int32(8);
     int32(8);
     int32(8);
     int32(9);
     int32(9);
     int32(9);
     int32(9);
     int32(9)];
icol = [int32(1);
     int32(1);
     int32(2);
     int32(2);
     int32(3);
     int32(4);
     int32(1);
     int32(5);
     int32(3);
     int32(4);
     int32(6);
     int32(2);
     int32(5);
     int32(6);
     int32(7);
     int32(4);
     int32(6);
     int32(8);
     int32(1);
     int32(5);
     int32(6);
     int32(8);
     int32(9)];
omega = 1.1;
b = [complex(8, +54);
     complex(-10, -92);
     complex(25, +27);
     complex(26, -28);
     complex(54, +12);
     complex(26, -22);
     complex(47, +65);
     complex(71, -57);
     complex(60, +70)];
tol = 1e-06;
maxitn = int32(100);
x = [complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0)];
[xOut, rnorm, itn, rdiag, ifail] = ...
    f11js(method, precon, a, irow, icol, omega, b, tol, maxitn, x)
```

```
xOut =
   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
rnorm =
   1.4773e-05
itn =
           7
rdiag =
    0.1667
    0.1667
    0.2000
    0.2000
    0.2500
    0.1667
    0.1667
    0.1111
    0.1111
ifail =
           0
```