NAG Toolbox for MATLAB

f11dq

1 Purpose

f11dq solves a complex sparse non-Hermitian system of linear equations, represented in co-ordinate 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, with incomplete LU preconditioning.

2 Syntax

```
[x, rnorm, itn, ifail] = f11dq(method, nnz, a, irow, icol, ipivp, ipivq,
istr, idiag, b, m, tol, maxitn, x, 'n', n, 'la', la)
```

3 Description

f11dq solves a complex sparse non-Hermitian linear system of equations

$$Ax = b,$$

using a preconditioned 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.

f11dq uses the incomplete LU factorization determined by f11dn as the preconditioning matrix. A call to f11dq must always be preceded by a call to f11dn. Alternative preconditioners for the same storage scheme are available by calling f11ds.

The matrix A, and the preconditioning matrix M, are represented in co-ordinate storage (CS) format (see Section 2.1.1 in the F11 Chapter Introduction) in the arrays **a**, **irow** and **icol**, as returned from f11dn. The array **a** holds the nonzero entries in these matrices, while **irow** and **icol** hold the corresponding row and column indices.

f11dq is a Black Box function which calls f11br, f11bs and 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

5 Parameters

5.1 Compulsory Input Parameters

1: **method – string**

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: **nnz – int32 scalar**

The number of nonzero elements in the matrix A. This **must** be the same value as was supplied in the preceding call to f11dn.

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

3: **a(la) – complex array**

The values returned in the array **a** by a previous call to f11dn.

4: **irow(la) – int32 array**
5: **icol(la) – int32 array**
6: **ipivp(n) – int32 array**
7: **ipivq(n) – int32 array**
8: **istr(n + 1) – int32 array**
9: **idiag(n) – int32 array**

The values returned in arrays **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** by a previous call to f11dn.

ipivp and **ipivq** are restored on exit.

10: **b(n) – complex array**

The right-hand side vector b.

11: **m – int32 scalar**

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', $0 < \mathbf{m} \leq \min(\mathbf{n}, 10)$.

12: **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

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

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

Constraint: **tol** < 1.0.

13: **maxitn – int32 scalar**

The maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

14: **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 **ipivp**, **ipivq**, **idiag**, **b**, **x**. (An error is raised if these dimensions are not equal.)

n, the order of the matrix A. This **must** be the same value as was supplied in the preceding call to f11dn.

Constraint: **n** ≥ 1.

2: **la – int32 scalar**

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

This **must** be the same value as was supplied in the preceding call to f11dn.

Constraint: **la** $\geq 2 \times$ **nnz**.

5.3 Input Parameters Omitted from the MATLAB Interface

work, lwork

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: **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** $\neq$ 'RGMRES','CGS','BICGSTAB', or 'TFQMR',
or **n** < 1,
or **nnz** < 1,
or **nnz** $> \mathbf{n}^2$,
or **la** $< 2 \times$ **nnz**,
or **m** < 1 and **method** $=$ 'RGMRES' or **method** $=$ 'BICGSTAB',
or **m** $> \min(\mathbf{n}, 50)$, with **method** $=$ 'RGMRES',
or **m** $> \min(\mathbf{n}, 10)$, with **method** $=$ 'BICGSTAB',
or **tol** ≥ 1.0,
or **maxitn** < 1,
or **lwork** too small.

ifail $= 2$

On entry, the CS representation of A is invalid. Further details are given in the error message. Check that the call to f11dq has been preceded by a valid call to f11dn, and that the arrays **a**, **irow**, and **icol** have not been corrupted between the two calls.

ifail $= 3$

On entry, the CS representation of the preconditioning matrix M is invalid. Further details are given in the error message. Check that the call to f11dq has been preceded by a valid call to f11dn, and that the arrays **a**, **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** have not been corrupted between the two calls.

ifail $= 4$

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 $= 7$ (f11br, f11bs or f11bt)

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

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 f11dq for each iteration is roughly proportional to the value of **nnzc** returned from the preceding call to f11dn.

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

9 Example

```
method = 'CGS     ';
nnz = int32(24);
a = [complex(2, +1);
     complex(-1, +1);
     complex(1, -3);
     complex(4, +7);
     complex(-3, +0);
     complex(2, +4);
     complex(-7, -5);
     complex(2, +1);
     complex(3, +2);
     complex(-4, +2);
     complex(0, +1);
     complex(5, -3);
     complex(-1, +2);
     complex(8, +6);
     complex(-3, -4);
     complex(-6, -2);
     complex(5, -2);
     complex(2, +0);
     complex(0, -5);
     complex(-1, +5);
     complex(6, +2);
     complex(-1, +4);
     complex(2, +0);
     complex(3, +3);
     complex(-0.0945945945945946, +0.06756756756756757);
     complex(-0.2567567567567568, +0.04054054054054054);
     complex(-0.3378378378378378, +0.5270270270270271);
     complex(-0.15, +0.05);
     complex(-0.4445945945945946, +0.2675675675675676);
     complex(-0.35, -0.04999999999999999);
     complex(0.1, +0.3);
     complex(-0.4, -0.2);
     complex(-0.95, -0.85);
     complex(0.1, -0.2);
     complex(-0.3, +0.6000000000000001);
     complex(0.3378378378378378, +0.472972972972973);
     complex(0.9, +0.7000000000000001);
     complex(0.15, -0.05);
     complex(0.2432432432432433, -0.4594594594594595);
     complex(-0.55, -0.15);
     complex(0.6, -0.7);
     complex(0, -1);
     complex(2, -1);
     complex(-0.65, -0.45);
     complex(0.1923076923076923, +0.03846153846153846);
     complex(-0.6, +1.2);
     complex(-0.3461538461538461, +0.7307692307692306);
     complex(0.5, -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);
     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);
     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);
     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);
     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);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0);
     complex(0, +0)];
irow = [int32(1);
     int32(1);
     int32(1);
     int32(2);
     int32(2);
     int32(2);
     int32(3);
     int32(3);
     int32(4);
     int32(4);
     int32(4);
     int32(4);
     int32(5);
     int32(5);
     int32(5);
     int32(6);
     int32(6);
     int32(6);
     int32(7);
     int32(7);
     int32(7);
     int32(8);
     int32(8);
```

```
    int32(8);
    int32(1);
    int32(1);
    int32(2);
    int32(2);
    int32(2);
    int32(3);
    int32(3);
    int32(3);
    int32(4);
    int32(4);
    int32(4);
    int32(5);
    int32(5);
    int32(5);
    int32(6);
    int32(6);
    int32(6);
    int32(6);
    int32(7);
    int32(7);
    int32(7);
    int32(8);
    int32(8);
    int32(8);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
    int32(0);
```

```
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0)];
icol = [int32(1);
     int32(4);
     int32(8);
     int32(1);
     int32(2);
     int32(5);
     int32(3);
     int32(6);
     int32(1);
     int32(3);
     int32(4);
     int32(7);
     int32(2);
     int32(5);
     int32(7);
     int32(1);
     int32(3);
     int32(6);
     int32(3);
     int32(5);
     int32(7);
     int32(2);
     int32(6);
     int32(8);
     int32(1);
     int32(8);
     int32(1);
     int32(2);
     int32(8);
     int32(2);
     int32(3);
     int32(6);
     int32(2);
     int32(4);
     int32(7);
     int32(1);
     int32(4);
     int32(5);
     int32(1);
     int32(2);
     int32(5);
     int32(6);
     int32(4);
     int32(5);
     int32(7);
     int32(3);
     int32(7);
     int32(8);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
```

```
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0);
     int32(0)];
ipivp = [int32(3);
     int32(6);
     int32(1);
     int32(2);
     int32(7);
     int32(4);
     int32(5);
     int32(8)];
ipivq = [int32(3);
     int32(1);
     int32(8);
     int32(5);
     int32(7);
     int32(4);
     int32(2);
     int32(6)];
istr = [int32(25);
     int32(27);
     int32(30);
     int32(33);
     int32(36);
     int32(39);
     int32(43);
     int32(46);
     int32(49)];
idiag = [int32(25);
     int32(28);
     int32(31);
     int32(34);
     int32(38);
     int32(42);
     int32(45);
     int32(48)];
```

```
b = [complex(7, +11);
     complex(1, +24);
     complex(-13, -18);
     complex(-10, +3);
     complex(23, +14);
     complex(17, -7);
     complex(15, -3);
     complex(-3, +20)];
m = int32(4);
tol = 1e-10;
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)];
[xOut, rnorm, itn, ifail] = ...
      f11dq(method, nnz, a, irow, icol, ipivp, ipivq, istr, idiag, b, m,
tol, maxitn, x)
```

```
xOut =
   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
rnorm =
   8.6757e-12
itn =
           4
ifail =
           0
```