NAG Toolbox for MATLAB

f11mh

1 Purpose

f11mh returns error bounds for the solution of a real sparse system of linear equations with multiple right-hand sides, $AX = B$ or $A^{\mathrm{T}}X = B$. It improves the solution by iterative refinement in standard precision, in order to reduce the backward error as much as possible.

2 Syntax

```
[x, ferr, berr, ifail] = f11mh(trans, icolzp, irowix, a, iprm, il, lval,
iu, uval, b, x, 'n', n, 'nrhs_p', nrhs_p)
```

3 Description

f11mh returns the backward errors and estimated bounds on the forward errors for the solution of a real system of linear equations with multiple right-hand sides $AX = B$ or $A^{\mathrm{T}}X = B$. The function handles each right-hand side vector (stored as a column of the matrix B) independently, so we describe the function of f11mh in terms of a single right-hand side b and solution x.

Given a computed solution x, the function computes the *component-wise backward error* β. This is the size of the smallest relative perturbation in each element of A and b such that if x is the exact solution of a perturbed system:

$$(A + \delta A)x = b + \delta b$$
$$\text{then} \quad |\delta a_{ij}| \leq \beta |a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta |b_i|.$$

Then the function estimates a bound for the *component-wise forward error* in the computed solution, defined by:

$$\max_i |x_i - \hat{x}_i| / \max_i |x_i|$$

where $\hat{x}$ is the true solution.

The function uses the LU factorization $P_r A P_c = LU$ computed by f11me and the solution computed by f11mf.

4 References

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

5 Parameters

5.1 Compulsory Input Parameters

1: **trans – string**

Specifies whether $AX = B$ or $A^{\mathrm{T}}X = B$ is solved.

trans = 'N'

$AX = B$ is solved.

trans = 'T'

$A^{\mathrm{T}}X = B$ is solved.

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

2: **icolzp**$(*)$ – **int32 array**

Note: the dimension of the array **icolzp** must be at least $\mathbf{n}+1$.

icolzp(i) contains the index in A of the start of a new column. See Section 2.1.3 in the F11 Chapter Introduction.

3: **irowix**$(*)$ – **int32 array**

Note: the dimension of the array **irowix** must be at least **icolzp**$(\mathbf{n}+1)-1$, the number of nonzeros of the sparse matrix A.

The row index array of the sparse matrix A.

4: **a**$(*)$ – **double array**

Note: the dimension of the array **a** must be at least **icolzp**$(\mathbf{n}+1)-1$, the number of nonzeros of the sparse matrix A.

The array of nonzero values in the sparse matrix A.

5: **iprm**$(\mathbf{7}\times\mathbf{n})$ – **int32 array**

The column permutation which defines P_c, the row permutation which defines P_r, plus associated data structures as computed by f11me.

6: **il**$(*)$ – **int32 array**

Note: the dimension of the array **il** must be at least as large as the dimension of the array of the same name in f11me.

Records the sparsity pattern of matrix L as computed by f11me.

7: **lval**$(*)$ – **double array**

Note: the dimension of the array **lval** must be at least as large as the dimension of the array of the same name in f11me.

Records the nonzero values of matrix L and some nonzero values of matrix U as computed by f11me.

8: **iu**$(*)$ – **int32 array**

Note: the dimension of the array **iu** must be at least as large as the dimension of the array of the same name in f11me.

Records the sparsity pattern of matrix U as computed by f11me.

9: **uval**$(*)$ – **double array**

Note: the dimension of the array **uval** must be at least as large as the dimension of the array of the same name in f11me.

Records some nonzero values of matrix U as computed by f11me.

10: **b**(**ldb,**$*$) – **double array**

The first dimension of the array **b** must be at least $\max(1,\mathbf{n})$

The second dimension of the array must be at least $\max(1,\mathbf{nrhs\_p})$

The n by $nrhs$ right-hand side matrix B.

11: **x**(**ldx,**$*$) – **double array**

The first dimension of the array **x** must be at least $\max(1,\mathbf{n})$

The second dimension of the array must be at least $\max(1, \mathbf{nrhs\_p})$

The n by $nrhs$ solution matrix X, as returned by f11mf.

5.2 Optional Input Parameters

1: **n – int32 scalar**

n, the order of the matrix A.

Constraint: $\mathbf{n} \geq 0$.

2: **nrhs_p – int32 scalar**

Default: The second dimension of the array **b** The second dimension of the array **x**.

$nrhs$, the number of right-hand sides in B.

Constraint: $\mathbf{nrhs\_p} \geq 0$.

5.3 Input Parameters Omitted from the MATLAB Interface

ldb, ldx

5.4 Output Parameters

1: **x(ldx,$*$) – double array**

The first dimension of the array **x** must be at least $\max(1, \mathbf{n})$

The second dimension of the array must be at least $\max(1, \mathbf{nrhs\_p})$

The n by $nrhs$ improved solution matrix X.

2: **ferr($*$) – double array**

Note: the dimension of the array **ferr** must be at least $\max(1, \mathbf{nrhs\_p})$.

ferr(j) contains an estimated error bound for the jth solution vector, that is, the jth column of X, for $j = 1, 2, \ldots, nrhs$.

3: **berr($*$) – double array**

Note: the dimension of the array **berr** must be at least $\max(1, \mathbf{nrhs\_p})$.

berr(j) contains the component-wise backward error bound β for the jth solution vector, that is, the jth column of X, for $j = 1, 2, \ldots, nrhs$.

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, **trans** $\neq$ 'N' or 'T',
or **n** < 0,
or **nrhs_p** < 0,
or **ldb** $< \max(1, \mathbf{n})$,
or **ldx** $< \max(1, \mathbf{n})$.

ifail $= 2$

Ill-defined row permutation in array **iprm**. Internal checks have revealed that the **iprm** array is corrupted.

ifail $= 3$

Ill-defined column permutations in array **iprm**. Internal checks have revealed that the **iprm** array is corrupted.

ifail $= 301$

Unable to allocate required internal workspace.

7 Accuracy

The bounds returned in **ferr** are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

8 Further Comments

At most five steps of iterative refinement are performed, but usually only one or two steps are required.

Estimating the forward error involves solving a number of systems of linear equations of the form $Ax = b$ or $A^{\mathrm{T}}x = b$;

9 Example

```
trans = 'N';
icolzp = [int32(1);
     int32(3);
     int32(5);
     int32(7);
     int32(9);
     int32(12)];
irowix = [int32(1);
     int32(3);
     int32(1);
     int32(5);
     int32(2);
     int32(3);
     int32(2);
     int32(4);
     int32(3);
     int32(4);
     int32(5)];
a = [2;
     4;
     1;
     -2;
     1;
     1;
     -1;
     1;
     1;
     2;
     3];
iprm = [int32(1);
     int32(0);
     int32(4);
     int32(3);
     int32(2);
     int32(4);
     int32(3);
```

```
     int32(1);
     int32(2);
     int32(0);
     int32(2);
     int32(0);
     int32(8);
     int32(6);
     int32(4);
     int32(4);
     int32(2);
     int32(11);
     int32(8);
     int32(6);
     int32(1);
     int32(2);
     int32(3);
     int32(4);
     int32(5);
     int32(2);
     int32(2);
     int32(2);
     int32(2);
     int32(1);
     int32(1);
     int32(1);
     int32(1);
     int32(2);
     int32(0)];
il = [int32(0);
     int32(1);
     int32(2);
     int32(3);
     int32(-1);
     int32(-1);
     int32(1);
     int32(2);
     int32(3);
     int32(5);
     int32(-1);
     int32(0);
     int32(1);
     int32(2);
     int32(3);
     int32(3);
     int32(3);
     int32(0);
     int32(2);
     int32(4);
     int32(6);
     int32(14);
     int32(8);
     int32(2);
     int32(4);
     int32(6);
     int32(8);
     int32(15);
     int32(0);
     int32(2);
     int32(4);
     int32(6);
     int32(8);
     int32(-1);
     int32(2);
     int32(4);
     int32(6);
     int32(8);
     int32(10);
     int32(0);
     int32(4);
     int32(1);
```

```
      int32(4);
      int32(2);
      int32(4);
      int32(3);
      int32(4);
      int32(3);
      int32(0);
      int32(3);
      int32(0);
      int32(1);
      int32(0);
      int32(0);
      int32(0);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
```

```
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1)];
lval = [-2;
      -0.5;
      4;
      0.5;
      2;
      0.5;
      -1;
      0.5;
      1;
      -1;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
```

```
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0];
iu = [int32(0);
      int32(0);
      int32(0);
      int32(2);
      int32(3);
      int32(-1);
      int32(0);
      int32(0);
      int32(2);
      int32(3);
      int32(4);
      int32(1);
      int32(0);
      int32(2);
      int32(1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
```

```
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1);
      int32(-1)];
uval = [1;
      3;
      1;
      1;
      0;
      0;
      0;
      0;
```

```
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
      0;
```

```
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0];
b = [1.56, 3.12;
     -0.25, -0.5;
     3.6, 7.2;
     1.33, 2.66;
     0.52, 1.04];
x = [0.7, 1.4;
     0.16, 0.32000000000000001;
     0.52, 1.04;
     0.77, 1.54;
     0.28, 0.56000000000000001];
[xOut, ferr, berr, ifail] = ...
    f11mh(trans, icolzp, irowix, a, iprm, il, lval, iu, uval, b, x)
```

```
xOut =
    0.7000    1.4000
    0.1600    0.3200
    0.5200    1.0400
    0.7700    1.5400
    0.2800    0.5600
ferr =
   1.0e-14 *
    0.5027
    0.4954
berr =
   1.0e-16 *
    0.4448
    0.3084
ifail =
           0
```