NAG Library Function Document

nag_superlu_refine_lu (f11mhc)

1 Purpose

nag_superlu_refine_lu (f11mhc) 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 Specification

```
#include <nag.h>
#include <nagf11.h>
void nag_superlu_refine_lu (Nag_OrderType order, Nag_TransType trans,
     Integer n, const Integer icolzp[], const Integer irowix[],
     const double a[], const Integer iprm[], const Integer il[],
     const double lval[], const Integer iu[], const double uval[],
     Integer nrhs, const double b[], Integer pdb, double x[], Integer pdx,
     double ferr[], double berr[], NagError *fail)
```

3 Description

nag_superlu_refine_lu (f11mhc) 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 nag_superlu_refine_lu (f11mhc) 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 nag_superlu_lu_factorize (f11mec) and the solution computed by nag_superlu_solve_lu (f11mfc).

4 References

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

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by

order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **trans** – Nag_TransType *Input*

On entry: specifies whether $AX = B$ or $A^{\mathrm{T}}X = B$ is solved.

trans = Nag_NoTrans
$AX = B$ is solved.

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

Constraint: **trans** = Nag_NoTrans or Nag_Trans.

3: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

4: **icolzp**[dim] – const Integer *Input*

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

On entry: **icolzp**[$i - 1$] contains the index in A of the start of a new column. See Section 2.1.3 in the f11 Chapter Introduction.

5: **irowix**[dim] – const Integer *Input*

Note: the dimension, *dim*, of the array **irowix** must be at least **icolzp**[**n**] $- 1$, the number of nonzeros of the sparse matrix A.

On entry: the row index array of sparse matrix A.

6: **a**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least **icolzp**[**n**] $- 1$, the number of nonzeros of the sparse matrix A.

On entry: the array of nonzero values in the sparse matrix A.

7: **iprm**[$7 \times \mathbf{n}$] – const Integer *Input*

On entry: the column permutation which defines P_c, the row permutation which defines P_r, plus associated data structures as computed by nag_superlu_lu_factorize (f11mec).

8: **il**[dim] – const Integer *Input*

Note: the dimension, *dim*, of the array **il** must be at least as large as the dimension of the array of the same name in nag_superlu_lu_factorize (f11mec).

On entry: records the sparsity pattern of matrix L as computed by nag_superlu_lu_factorize (f11mec).

9: **lval**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **lval** must be at least as large as the dimension of the array of the same name in nag_superlu_lu_factorize (f11mec).

On entry: records the nonzero values of matrix L and some nonzero values of matrix U as computed by nag_superlu_lu_factorize (f11mec).

10: **iu**[dim] – const Integer *Input*

Note: the dimension, *dim*, of the array **iu** must be at least as large as the dimension of the array of the same name in nag_superlu_lu_factorize (f11mec).

On entry: records the sparsity pattern of matrix U as computed by nag_superlu_lu_factorize (f11mec).

11: **uval**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **uval** must be at least as large as the dimension of the array of the same name in nag_superlu_lu_factorize (f11mec).

On entry: records some nonzero values of matrix U as computed by nag_superlu_lu_factorize (f11mec).

12: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0.

13: **b**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix B is stored in

b$[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
b$[(i-1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by $nrhs$ right-hand side matrix B.

14: **pdb** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **b**.

Constraints:

if **order** = Nag_ColMajor, **pdb** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

15: **x**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **x** must be at least

$\max(1, \mathbf{pdx} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix X is stored in

x$[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;
x$[(i-1) \times \mathbf{pdx} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by $nrhs$ solution matrix X, as returned by nag_superlu_solve_lu (f11mfc).

On exit: the n by $nrhs$ improved solution matrix X.

16: **pdx** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **x**.

Constraints:

if **order** = Nag_ColMajor, **pdx** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdx** $\geq \max(1, \mathbf{nrhs})$.

17: **ferr**[**nrhs**] – double *Output*

On exit: **ferr**$[j-1]$ contains an estimated error bound for the jth solution vector, that is, the jth column of X, for $j = 1, 2, \ldots, nrhs$.

18: **berr**[**nrhs**] – double *Output*

On exit: **berr**$[j-1]$ contains the component-wise backward error bound β for the jth solution vector, that is, the jth column of X, for $j = 1, 2, \ldots, nrhs$.

19: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **nrhs** = ⟨*value*⟩.
Constraint: **nrhs** ≥ 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

On entry, **pdx** = ⟨*value*⟩.
Constraint: **pdx** > 0.

NE_INT_2

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = ⟨*value*⟩ and **nrhs** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{nrhs})$.

On entry, **pdx** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdx** $\geq \max(1, \mathbf{n})$.

On entry, **pdx** = ⟨*value*⟩ and **nrhs** = ⟨*value*⟩.
Constraint: **pdx** $\geq \max(1, \mathbf{nrhs})$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_INVALID_PERM_COL

Incorrect column permutations in array **iprm**.

NE_INVALID_PERM_ROW

Incorrect Row Permutations in array **iprm**.

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 Parallelism and Performance

nag_superlu_refine_lu (f11mhc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_superlu_refine_lu (f11mhc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 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 $A^{\mathrm{T}}x = b$;

10 Example

This example solves the system of equations $AX = B$ using iterative refinement and to compute the forward and backward error bounds, where

$$A = \begin{pmatrix} 2.00 & 1.00 & 0 & 0 & 0 \\ 0 & 0 & 1.00 & -1.00 & 0 \\ 4.00 & 0 & 1.00 & 0 & 1.00 \\ 0 & 0 & 0 & 1.00 & 2.00 \\ 0 & -2.00 & 0 & 0 & 3.00 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1.56 & 3.12 \\ -0.25 & -0.50 \\ 3.60 & 7.20 \\ 1.33 & 2.66 \\ 0.52 & 1.04 \end{pmatrix}.$$

Here A is nonsymmetric and must first be factorized by nag_superlu_lu_factorize (f11mec).

10.1 Program Text

```
/* nag_superlu_refine_lu (f11mhc) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf11.h>

/* Table of constant values */

static Integer c__1 = 1;
static Integer c__80 = 80;
static Integer c__0 = 0;

int main(void)
{
```

```
  double                    flop, thresh;
  Integer                   exit_status = 0, i, j;
  Integer                   n, nnz, nnzl, nnzu, nrhs, nzlmx, nzlumx, nzumx;
  double                    *a = 0, *b = 0, *berr = 0, *ferr = 0, *lval = 0;
  double                    *uval = 0, *x = 0;
  Integer                   *icolzp = 0, *il = 0, *iprm = 0, *irowix = 0;
  Integer                   *iu = 0;
  /* Nag types */
  Nag_OrderType             order = Nag_ColMajor;
  Nag_MatrixType            matrix = Nag_GeneralMatrix;
  Nag_DiagType              diag = Nag_NonUnitDiag;
  Nag_ColumnPermutationType ispec;
  Nag_TransType             trans;
  NagError                  fail;

  INIT_FAIL(fail);

  printf("nag_superlu_refine_lu (f11mhc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Read order of matrix and number of right hand sides */
  scanf("%ld%ld%*[^\n] ", &n, &nrhs);
  /* Read the matrix A */
  if (!(icolzp = NAG_ALLOC(n+1, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (i = 1; i <= n + 1; ++i)
    scanf("%ld%*[^\n] ", &icolzp[i - 1]);
  nnz = icolzp[n] - 1;
  /* Allocate memory */
  if (!(irowix = NAG_ALLOC(nnz, Integer)) ||
      !(a = NAG_ALLOC(nnz, double)) ||
      !(il = NAG_ALLOC(7*n+8*nnz+4, Integer)) ||
      !(iu = NAG_ALLOC(2*n+8*nnz+1, Integer)) ||
      !(uval = NAG_ALLOC(8*nnz, double)) ||
      !(lval = NAG_ALLOC(8*nnz, double)) ||
      !(b = NAG_ALLOC(n * nrhs, double)) ||
      !(x = NAG_ALLOC(n * nrhs, double)) ||
      !(berr = NAG_ALLOC(nrhs, double)) ||
      !(ferr = NAG_ALLOC(nrhs, double)) ||
      !(iprm = NAG_ALLOC(7*n, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (i = 0; i < nnz; ++i)
    scanf("%lf%ld%*[^\n] ", &a[i], &irowix[i]);
  /* Read the right hand sides */
  for (j = 0; j < nrhs; ++j)
    {
      for (i = 0; i < n; ++i)
        {
          scanf("%lf", &x[j*n + i]);
          b[j*n + i] = x[j*n + i];
        }
      scanf("%*[^\n] ");
    }
  /* Calculate COLAMD permutation */
  ispec = Nag_Sparse_Colamd;
  /* nag_superlu_column_permutation (f11mdc).
   * Real sparse nonsymmetric linear systems, setup for
   * nag_superlu_lu_factorize (f11mec)
   */
  nag_superlu_column_permutation(ispec, n, icolzp, irowix, iprm, &fail);
  if (fail.code != NE_NOERROR)
    {
```

```
    printf("Error from nag_superlu_column_permutation (f11mdc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

/* Factorise */
thresh = 1.;
nzlmx = 8*nnz;
nzlumx = 8*nnz;
nzumx = 8*nnz;
/* nag_superlu_lu_factorize (f11mec).
 * LU factorization of real sparse matrix
 */
nag_superlu_lu_factorize(n, irowix, a, iprm, thresh, nzlmx, &nzlumx, nzumx,
                         il, lval, iu, uval, &nnzl, &nnzu, &flop, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_superlu_lu_factorize (f11mec).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

/* Compute solution in array X */
trans = Nag_NoTrans;
/* nag_superlu_solve_lu (f11mfc).
 * Solution of real sparse simultaneous linear equations
 * (coefficient matrix already factorized)
 */
nag_superlu_solve_lu(order, trans, n, iprm, il, lval, iu, uval, nrhs, x,
                     n, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_superlu_solve_lu (f11mfc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

/* Improve solution, and compute backward errors and estimated */
/* bounds on the forward errors */
/* nag_superlu_refine_lu (f11mhc).
 * Refined solution with error bounds of real system of
 * linear equations, multiple right-hand sides
 */
nag_superlu_refine_lu(order, trans, n, icolzp, irowix, a, iprm, il, lval,
                      iu, uval, nrhs, b, n, x, n, ferr, berr,
                      &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_superlu_refine_lu (f11mhc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

/* Print solution */
printf("\n");
/* nag_gen_real_mat_print (x04cac).
 * Print real general matrix (easy-to-use)
 */
fflush(stdout);
nag_gen_real_mat_print(order, matrix, diag, n, nrhs,
                       x, n, "Solutions", 0, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
```

```
    }

  /* nag_gen_real_mat_print_comp (x04cbc).
   * Print real general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_real_mat_print_comp(order, matrix, diag, nrhs, c__1, ferr, nrhs,
                              "%8.2g", "Estimated Forward Error", Nag_NoLabels,
                              NULL, Nag_NoLabels, NULL, c__80, c__0, 0,
                              &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_gen_real_mat_print_comp (x04cbc), see above. */
  fflush(stdout);
  nag_gen_real_mat_print_comp(order, matrix, diag, nrhs, c__1, berr, nrhs,
                              "%8.2g", "Backward Error", Nag_NoLabels, NULL,
                              Nag_NoLabels, NULL, c__80, c__0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(berr);
  NAG_FREE(ferr);
  NAG_FREE(lval);
  NAG_FREE(uval);
  NAG_FREE(x);
  NAG_FREE(icolzp);
  NAG_FREE(il);
  NAG_FREE(iprm);
  NAG_FREE(irowix);
  NAG_FREE(iu);
  return exit_status;
}
```

10.2 Program Data

```
nag_superlu_refine_lu (f11mhc) Example Program Data
  5 2  n, nrhs
 1
 3
 5
 7
 9
 12   icolzp(i) i=0..n
  2.   1
  4.   3
  1.   1
 -2.   5
  1.   2
  1.   3
 -1.   2
  1.   4
  1.   3
  2.   4
  3.   5    a(i) irowix(i) i=0..nnz-1
 1.56 -.25 3.6 1.33 .52
 3.12 -.50 7.2 2.66 1.04  matrix x
```

10.3 Program Results

```
nag_superlu_refine_lu (f11mhc) Example Program Results


 Solutions
             1          2
 1      0.7000     1.4000
 2      0.1600     0.3200
 3      0.5200     1.0400
 4      0.7700     1.5400
 5      0.2800     0.5600
 Estimated Forward Error
     5e-15
     5e-15
 Backward Error
   3.6e-17
   3.6e-17
```