NAG C Library Function Document

nag_superlu_solve_lu (f11mfc)

1 Purpose

nag_superlu_solve_lu (f11mfc) solves a real sparse system of linear equations with multiple right-hand sides given an LU factorization of the sparse matrix computed by nag_superlu_lu_factorize (f11mec).

2 Specification

```
#include <nag.h>
#include <nagf11.h>
```

```
void nag_superlu_solve_lu (Nag_OrderType order, Nag_TransType trans, Integer n,
     const Integer iprm[], const Integer il[], const double lval[],
     const Integer iu[], const double uval[], Integer nrhs, double b[],
     Integer pdb, NagError *fail)
```

3 Description

nag_superlu_solve_lu (f11mfc) solves a real system of linear equations with multiple right-hand sides $AX = B$ or $A^{\mathrm{T}}X = B$, according to the value of the argument **trans**, where the matrix factorization $P_rAP_c = LU$ corresponds to an LU decomposition of a sparse matrix stored in compressed column (Harwell–Boeing) format, as computed by nag_superlu_lu_factorize (f11mec).

In the above decomposition L is a lower triangular sparse matrix with unit diagonal elements and U is an upper triangular sparse matrix; P_r and P_c are permutation matrices.

4 References

None.

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 2.2.1.4 of 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: **iprm**$[\mathbf{7} \times \mathbf{n}]$ – 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).

5: **il**$[dim]$ – const Integer *Input*

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

6: **lval**$[dim]$ – const double *Input*

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

7: **iu**$[dim]$ – const Integer *Input*

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

8: **uval**$[dim]$ – const double *Input*

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

9: **nrhs** – Integer *Input*

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

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

10: **b**$[dim]$ – double *Input/Output*

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

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

If **order** = **Nag_ColMajor**, the (i,j)th element of the matrix B is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = **Nag_RowMajor**, the (i,j)th element of the matrix B is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.

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

On exit: the **n** by **nrhs** solution matrix X.

11: **pdb** – Integer *Input*

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

Constraints:

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

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

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{nrhs} = \langle value \rangle$.
Constraint: $\mathbf{nrhs} \geq 0$.

On entry, $\mathbf{pdb} = \langle value \rangle$.
Constraint: $\mathbf{pdb} > 0$.

NE_INT_2

On entry, $\mathbf{pdb} = \langle value \rangle$, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

On entry, $\mathbf{pdb} = \langle value \rangle$, $\mathbf{nrhs} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.

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 consult 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

For each right-hand side vector b, the computed solution x is the exact solution of a perturbed system of equations $(A + E)x = b$, where

$$|E| \leq c(n)\epsilon|L||U|,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***, when partial pivoting is used.

If $\hat{x}$ is the true solution, then the computed solution x satisfies a forward error bound of the form

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq c(n)\,\mathrm{cond}(A, x)\epsilon$$

where $\mathrm{cond}(A, x) = \left\| |A^{-1}||A||x| \right\|_\infty / \|x\|_\infty \leq \mathrm{cond}(A) = \left\| |A^{-1}||A| \right\|_\infty \leq \kappa_\infty(A)$. Note that $\mathrm{cond}(A, x)$ can be much smaller than $\mathrm{cond}(A)$, and $\mathrm{cond}\left(A^{\mathrm{T}}\right)$ can be much larger (or smaller) than $\mathrm{cond}(A)$.

Forward and backward error bounds can be computed by calling nag_superlu_refine_lu (f11mhc), and an estimate for $\kappa_\infty(A)$ can be obtained by calling nag_superlu_condition_number_lu (f11mgc).

8 Further Comments

This function may be followed by a call to nag_superlu_refine_lu (f11mhc) to refine the solution and return an error estimate.

9 Example

To solve the system of equations $AX = B$, 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 non-symmetric and must first be factorized by nag_superlu_lu_factorize (f11mec).

9.1 Program Text

```
/* nag_superlu_solve_lu (f11mfc) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

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


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



  INIT_FAIL(fail);
  Vprintf( "nag_superlu_solve_lu (f11mfc) Example Program Results\n\n");
  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  /* Read order of matrix and number of right hand sides */
  Vscanf("%ld%ld%*[^\n] ",   &n,  &nrhs);
  /* Read the matrix A */
  if ( !(icolzp = NAG_ALLOC(n+1, Integer)))
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (i = 1; i <= n + 1; ++i)
    {
      Vscanf("%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)) ||
     !(iprm = NAG_ALLOC(7*n, Integer)) ||
     !(x = NAG_ALLOC(n*nrhs, double))  )
  {
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
for (i = 1; i <= nnz; ++i)
  Vscanf("%lf%ld%*[^\n] ",   &a[i - 1],  &irowix[i - 1]);
/* Read the right hand sides */
for (j = 1; j <= nrhs; ++j)
  {
    for (i = 1; i <= n; ++i)
      Vscanf("%lf", &x[j*n + i - n - 1]);
    Vscanf("%*[^\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)
  {
    Vprintf("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)
  {
    Vprintf("Error from nag_superlu_lu_factorize (f11mec).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }
/* Solve */
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)
  {
    Vprintf("Error from nag_superlu_solve_lu (f11mfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
/* Output results */
Vprintf("\n");
```

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

 END:
  if (a) NAG_FREE(a);
  if (lval) NAG_FREE(lval);
  if (uval) NAG_FREE(uval);
  if (x) NAG_FREE(x);
  if (icolzp) NAG_FREE(icolzp);
  if (il) NAG_FREE(il);
  if (iprm) NAG_FREE(iprm);
  if (irowix) NAG_FREE(irowix);
  if (iu) NAG_FREE(iu);

  return exit_status;
}
```

9.2 Program Data

```
nag_superlu_solve_lu (f11mfc) 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
```

9.3 Program Results

```
nag_superlu_solve_lu (f11mfc) 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
```