

NAG C Library Function Document

nag_superlu_lu_factorize (f11mec)

1 Purpose

nag_superlu_lu_factorize (f11mec) computes the LU factorization of a real sparse matrix in compressed column (Harwell–Boeing), column-permuted format.

2 Specification

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

void nag_superlu_lu_factorize (Integer n, const Integer irowix[],
    const double a[], Integer iprm[], double thresh, Integer nzlmx,
    Integer *nzlumnx, Integer nzumnx, Integer il[], double lval[], Integer iu[],
    double uval[], Integer *nnzl, Integer *nnzu, double *flop, NagError *fail)
```

3 Description

Given a real sparse matrix A , nag_superlu_lu_factorize (f11mec) computes an LU factorization of A with partial pivoting, $P_r A P_c = LU$, where P_r is a row permutation matrix (computed by nag_superlu_lu_factorize (f11mec)), P_c is a (supplied) column permutation matrix, L is unit lower triangular and U is upper triangular. The column permutation matrix, P_c , must be computed by a prior call to nag_superlu_column_permutation (f11mdc). The matrix A must be presented in the column permuted, compressed column (Harwell–Boeing) format.

The LU factorization is output in the form of four one-dimensional arrays: integer arrays **il** and **iu** and real-valued arrays **lval** and **uval**. These describe the sparsity pattern and numerical values in the L and U matrices. The minimum required dimensions of these arrays cannot be given as a simple function of the size arguments (order and number of non-zero values) of the matrix A . This is due to unpredictable fill-in created by partial pivoting. nag_superlu_lu_factorize (f11mec) will, on return, indicate which dimensions of these arrays were not adequate for the computation or (in the case of one of them) give a firm bound. You should then allocate more storage and try again.

4 References

Demmel J W, Eisenstat S C, Gilbert J R, Li X S and Li J W H (1999) A Supernodal Approach to Sparse Partial Pivoting *SIAM J. Matrix Anal. Appl.* **20** 720–755

Demmel J W, Gilbert J R and Li X S (1999) An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination *SIAM J. Matrix Anal. Appl.* **20** 915–952

5 Arguments

- 1: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.
- 2: **irowix**[*dim*] – const Integer *Input*
Note: the dimension, *dim*, of the array **irowix** must be at least *nnz*, the number of non-zeros of the sparse matrix A .
On entry: the row index array of sparse matrix A .

- 3: **a**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **a** must be at least *nnz*, the number of non-zeros of the sparse matrix *A*.
On entry: the array of non-zero values in the sparse matrix *A*.
- 4: **iprm**[7 × *n*] – Integer *Input/Output*
On entry: contains the column permutation which defines the permutation P_c and associated data structures as computed by function nag_superlu_column_permutation (f11mdc).
On exit: part of the array is modified to record the row permutation P_r determined by pivoting.
- 5: **thresh** – double *Input*
On entry: the diagonal pivoting threshold, *t*. At step *j* of the Gaussian elimination, if $|A_{jj}| \geq t \left(\max_{i \geq j} |A_{ij}| \right)$, use A_{jj} as a pivot, otherwise use $\max_{i \geq j} |A_{ij}|$. A value of $t = 1$ corresponds to partial pivoting, a value of $t = 0$ corresponds to always choosing the pivot on the diagonal (unless it is zero).
Constraint: $0 \leq \text{thresh} \leq 1$.
Suggested value: **thresh** = 1.0. Smaller values may result in a faster factorization, but the benefits are likely to be small in most cases. It might be possible to use **thresh** = 0.0 if you are confident about the stability of the factorization, for example, if *A* is diagonally dominant.
- 6: **nzlmx** – Integer *Input*
On entry: indicates the available size of array **il**. The dimension of **il** should be at least $7 \times n + \text{nzlmx} + 4$. A good range for **nzlmx** that works for many problems is *nnz* to $8 \times \text{nnz}$, where *nnz* is the number of non-zeros in the sparse matrix *A*. If, on exit, **fail** is set to 2, the given **nzlmx** was too small and you should attempt to provide more storage and call the function again.
Constraint: **nzlmx** ≥ 1.
- 7: **nzlumx** – Integer * *Input/Output*
On entry: indicates the available size of array **lval**. The dimension of **lval** should be at least **nzlumx**.
Constraint: **nzlumx** ≥ 1.
On exit: if **fail** is set to 4, the given **nzlumx** was too small and is reset to a value that will be sufficient. You should then provide the indicated storage and call the function again.
- 8: **nzumx** – Integer *Input*
On entry: indicates the available sizes of arrays **iu** and **uval**. The dimension of **iu** should be at least $2 \times n + \text{nzumx} + 1$ and the dimension of **uval** should be at least **nzumx**. A good range for **nzumx** that works for many problems is *nnz* to $8 \times \text{nnz}$, where *nnz* is the number of non-zeros in the sparse matrix *A*. If, on exit, **fail** is set to 3, the given **nzumx** was too small and you should attempt to provide more storage and call the function again.
Constraint: **nzumx** ≥ 1.
- 9: **il**[*dim*] – Integer *Output*
Note: the dimension, *dim*, of the array **il** must be at least $7 \times n + \text{nzlmx} + 4$.
On exit: encapsulates the sparsity pattern of matrix *L*.
- 10: **lval**[*dim*] – double *Output*
Note: the dimension, *dim*, of the array **lval** must be at least **nzlumx**.
On exit: records the non-zero values of matrix *L* and some of the non-zero values of matrix *U*.

- 11: **iu**[*dim*] – Integer *Output*
Note: the dimension, *dim*, of the array **iu** must be at least $2 \times \mathbf{n} + \mathbf{nzumx} + 1$.
On exit: encapsulates the sparsity pattern of matrix *U*.
- 12: **uval**[**nzumx**] – double *Output*
On exit: records some of the non-zero values of matrix *U*.
- 13: **nnzl** – Integer * *Output*
On exit: the number of non-zero values in the matrix *L*.
- 14: **nnzu** – Integer * *Output*
On exit: the number of non-zero values in the matrix *U*.
- 15: **flop** – double * *Output*
On exit: the number of floating-point operations performed.
- 16: **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, **n** = $\langle value \rangle$.

Constraint: **n** ≥ 0 .

On entry, **nzlmx** = $\langle value \rangle$.

Constraint: **nzlmx** ≥ 1 .

On entry, **nzlumx** = $\langle value \rangle$.

Constraint: **nzlumx** ≥ 1 .

On entry, **nzumx** = $\langle value \rangle$.

Constraint: **nzumx** ≥ 1 .

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_NZLMX_TOO_SMALL

Insufficient **nzlmx**.

NE_NZLUMX_TOO_SMALL

Insufficient **nzlumx**.

NE_NZUMX_TOO_SMALL

Insufficient **nzumx**.

NE_REAL

On entry, **thresh** = $\langle value \rangle$.
 Constraint: $0 \leq \mathbf{thresh} \leq 1$.

NE_SINGULAR_MATRIX

The matrix is singular – no factorization possible.

7 Accuracy

The computed factors L and U are the exact factors of a perturbed matrix $A + E$, 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 no partial pivoting is used, the factorization accuracy can be considerably worse. A call to `nag_superlu_diagnostic_lu` (f11mmc) after `nag_superlu_lu_factorize` (f11mec) can help determine the quality of the factorization.

8 Further Comments

The total number of floating-point operations depends on the sparsity pattern of the matrix A .

A call to this function may be followed by calls to the functions:

`nag_superlu_solve_lu` (f11mfc) to solve $AX = B$ or $A^T X = B$;

`nag_superlu_condition_number_lu` (f11mgc) to estimate the condition number of A ;

`nag_superlu_diagnostic_lu` (f11mmc) to estimate the reciprocal pivot growth of the LU factorization.

9 Example

To compute the LU factorization of the matrix A , where

$$A = \begin{pmatrix} 2.00 & 1.00 & 0 & 0 & 0 \\ 0 & 0 & 1.00 & -1.00 & 0 \\ 4.00 & 0 & 1.00 & 0 & 1.00 \\ 0 & 0 & 0 & 1.00 & 2.00 \\ 0 & -2.00 & 0 & 0 & 3.00 \end{pmatrix}.$$

9.1 Program Text

```
/* nag_superlu_lu_factorise (f11mec) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

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

/* Table of constant values */

static Integer c__1 = 1;
static Integer c__10 = 10;
static Integer c__80 = 80;
static Integer c__4 = 4;

int main(void)
{
    double flop, thresh;
    Integer exit_status=0, i, n, nnz, nnzl, nnzu, nzlmax, nzlsum, nzumx;
```

```

double *a=0, *lval=0, *uval=0;
Integer *icolzp=0, *il=0, *iprm=0, *irowix=0, *iu=0;
/* Nag types */
NagError fail;
Nag_ColumnPermutationType ispec;
Nag_OrderType order = Nag_ColMajor;
Nag_MatrixType matrix = Nag_GeneralMatrix;
Nag_DiagType diag = Nag_NonUnitDiag;

INIT_FAIL(fail);
/* nag_superlu_lu_factorize (f11mec).
 * LU factorization of real sparse matrix
 */
Vprintf("nag_superlu_lu_factorize (f11mec) Example Program Results\n\n");
/* Skip heading in data file */
Vscanf("%*[^\\n] ");
/* Read order of matrix */
Vscanf("%ld%*[^\\n] ", &n, &n);
/* Read the matrix A */
/* Allocate memory */
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)) )
{
    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]);
/* 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), see above. */
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;
    }

    /* Output results */
    Vprintf("\n");
    Vprintf("%s\n", "Number of nonzeros in factors (excluding unit diagonal)");
    Vprintf("%8ld\n", nnz1 + nnzu - n);
    Vprintf("\n");
    /* nag_gen_real_mat_print_comp (x04cbc).
     * Print real general matrix (comprehensive)
     */
    nag_gen_real_mat_print_comp(order, matrix, diag, c__1, c__10,
                                lval, c__1, "%6.2f", "Factor elements in LVAL",
                                Nag_NoLabels, NULL, Nag_NoLabels, NULL, c__80,
                                c__1, NULL, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }

    Vprintf("\n");
    /* nag_gen_real_mat_print_comp (x04cbc), see above. */
    nag_gen_real_mat_print_comp(order, matrix, diag, c__1, c__4,
                                uval, c__1, "%6.2f", "Factor elements in LVAL",
                                Nag_NoLabels, NULL, Nag_NoLabels, NULL, c__80,
                                c__1, NULL, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from nag_gen_real_mat_print_comp (x04cbc).\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 (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_lu_factorize (f11mec) Example Program Data

```

5          n
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

```

9.3 Program Results

nag_superlu_lu_factorize (f11mec) Example Program Results

Number of nonzeros in factors (excluding unit diagonal)
14

Factor elements in LVAL

```

-2.00 -0.50  4.00  0.50  2.00  0.50 -1.00  0.50  1.00 -1.00

```

Factor elements in LVAL

```

1.00  3.00  1.00  1.00

```
