

NAG 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 *nzlumx, Integer nzumx, 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, $P_r A P_c = \bar{L} U$, 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 \bar{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 nonzero 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 nonzeros 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 nonzeros of the sparse matrix *A*.
On entry: the array of nonzero 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).
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.
Constraint: $0.0 \leq \mathbf{thresh} \leq 1.0$.
- 6: **nzlmx** – Integer *Input*
On entry: indicates the available size of array **il**. The dimension of **il** should be at least $7 \times \mathbf{n} + \mathbf{nzlmx} + 4$. A good range for **nzlmx** that works for many problems is *nnz* to $8 \times \mathbf{nnz}$, where *nnz* is the number of nonzeros in the sparse matrix *A*. If, on exit, **fail.code** = NE_NZLMX_TOO_SMALL, 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.code** = NE_NZLUMX_TOO_SMALL, 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 $2 \times \mathbf{n} + \mathbf{nzumx} + 1$ and the dimension of **uval** should be at least **nzumx**. A good range for **nzumx** that works for many problems is *nnz* to $8 \times \mathbf{nnz}$, where *nnz* is the number of nonzeros in the sparse matrix *A*. If, on exit, **fail.code** = NE_NZUMX_TOO_SMALL, the given **nzumx** was too small and you should attempt to provide more storage and call the function again.
Constraint: **nzumx** ≥ 1.
- 9: **il**[7 × *n* + **nzlmx** + 4] – Integer *Output*
On exit: encapsulates the sparsity pattern of matrix *L*.
- 10: **lval**[**nzlumx**] – double *Output*
On exit: records the nonzero values of matrix *L* and some of the nonzero values of matrix *U*.

- 11: **iu**[$2 \times \mathbf{n} + \mathbf{nzumx} + 1$] – Integer *Output*
On exit: encapsulates the sparsity pattern of matrix U .
- 12: **uval**[\mathbf{nzumx}] – double *Output*
On exit: records some of the nonzero values of matrix U .
- 13: **nnzl** – Integer * *Output*
On exit: the number of nonzero values in the matrix L .
- 14: **nnzu** – Integer * *Output*
On exit: the number of nonzero 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 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

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, $\mathbf{nzlmx} = \langle value \rangle$.
Constraint: $\mathbf{nzlmx} \geq 1$.

On entry, $\mathbf{nzlumx} = \langle value \rangle$.
Constraint: $\mathbf{nzlumx} \geq 1$.

On entry, $\mathbf{nzumx} = \langle value \rangle$.
Constraint: $\mathbf{nzumx} \geq 1$.

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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.

See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

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: $0.0 \leq \mathbf{thresh} \leq 1.0$.

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 Parallelism and Performance

`nag_superlu_lu_factorize` (f11mec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_superlu_lu_factorize` (f11mec) 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 x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

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

A call to `nag_superlu_lu_factorize` (f11mec) 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.

10 Example

This example computes the LU factorization of the matrix A , where

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

10.1 Program Text

```

/* nag_superlu_lu_factorize (f11mec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

/* Table of constant values */

int main(void)
{
    double flop, thresh;
    Integer exit_status = 0, i, n, nnz, nnzl, nnzu, nzlmx, nzlumx, nzumx;
    double *a = 0, *lval = 0, *uval = 0;
    Integer *icolzp = 0, *il = 0, *iprm = 0, *irowix = 0;
    Integer *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
     */
    printf("nag_superlu_lu_factorize (f11mec) Example Program Results\n\n");
    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read order of matrix */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &n);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &n);
#endif
    /* Read the matrix A */
    /* Allocate memory */
    if (!(icolzp = NAG_ALLOC(n + 1, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}

```

```

#ifdef _WIN32
    for (i = 1; i <= n + 1; ++i)
        scanf_s("%" NAG_IFMT "%*[\n] ", &icolzp[i - 1]);
#else
    for (i = 1; i <= n + 1; ++i)
        scanf("%" NAG_IFMT "%*[\n] ", &icolzp[i - 1]);
#endif
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)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
for (i = 1; i <= nnz; ++i)
#ifdef _WIN32
    scanf_s("%lf%" NAG_IFMT "%*[\n] ", &a[i - 1], &irowix[i - 1]);
#else
    scanf("%lf%" NAG_IFMT "%*[\n] ", &a[i - 1], &irowix[i - 1]);
#endif
/* 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), 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) {
    printf("Error from nag_superlu_lu_factorize (f11mec).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Output results */
printf("\n");
printf("Number of nonzeros in factors (excluding unit diagonal)\n");
printf("%8" NAG_IFMT "\n\n", nnzl + nnzu - n);
/* nag_gen_real_mat_print_comp (x04cbc).
 * Print real general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_real_mat_print_comp(order, matrix, diag, 1, 10, lval, 1, "%6.2f",
    "Factor elements in LVAL", Nag_NoLabels, NULL,
    Nag_NoLabels, NULL, 80, 1, 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;
}

```

```

}

printf("\n");
/* nag_gen_real_mat_print_comp (x04cbc), see above. */
fflush(stdout);
nag_gen_real_mat_print_comp(order, matrix, diag, 1, 4, uval, 1, "%6.2f",
                           "Factor elements in LVAL", Nag_NoLabels, NULL,
                           Nag_NoLabels, NULL, 80, 1, 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(lval);
NAG_FREE(uval);
NAG_FREE(icolzp);
NAG_FREE(il);
NAG_FREE(iprm);
NAG_FREE(irowix);
NAG_FREE(iu);

return exit_status;
}

```

10.2 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

```

10.3 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
