

## NAG Library Function Document

### nag\_superlu\_column\_permutation (f11mdc)

#### 1 Purpose

nag\_superlu\_column\_permutation (f11mdc) computes a column permutation suitable for  $LU$  factorization (by nag\_superlu\_lu\_factorize (f11mec)) of a real sparse matrix in compressed column (Harwell–Boeing) format and applies it to the matrix. This function must be called prior to nag\_superlu\_lu\_factorize (f11mec).

#### 2 Specification

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

void nag_superlu_column_permutation (Nag_ColumnPermutationType spec,
    Integer n, const Integer icolzp[], const Integer irowix[],
    Integer iprm[], NagError *fail)
```

#### 3 Description

Given a sparse matrix in compressed column (Harwell–Boeing) format  $A$  and a choice of column permutation schemes, the function computes those data structures that will be needed by the  $LU$  factorization function nag\_superlu\_lu\_factorize (f11mec) and associated functions nag\_superlu\_diagnostic\_lu (f11mmc), nag\_superlu\_solve\_lu (f11mfc) and nag\_superlu\_refine\_lu (f11mhc). The column permutation choices are:

original order (that is, no permutation);

user-supplied permutation;

a permutation, computed by the function, designed to minimize fill-in during the  $LU$  factorization.

The algorithm for this computed permutation is based on the approximate minimum degree column ordering algorithm COLAMD. The computed permutation is not sensitive to the magnitude of the nonzero values of  $A$ .

#### 4 References

Amestoy P R, Davis T A and Duff I S (1996) An approximate minimum degree ordering algorithm *SIAM J. Matrix Anal. Appl.* **17** 886–905

Gilbert J R and Larimore S I (2004) A column approximate minimum degree ordering algorithm *ACM Trans. Math. Software* **30,3** 353–376

Gilbert J R, Larimore S I and Ng E G (2004) Algorithm 836: COLAMD, an approximate minimum degree ordering algorithm *ACM Trans. Math. Software* **30, 3** 377–380

#### 5 Arguments

1: **spec** – Nag\_ColumnPermutationType *Input*

*On entry:* indicates the permutation to be applied.

**spec** = Nag\_Sparse\_Identity

The identity permutation is used (i.e., the columns are not permuted).

**spec** = Nag\_Sparse\_User

The permutation in the **iprm** array is used, as supplied by you.

**spec** = Nag\_Sparse\_Colamd

The permutation computed by the COLAMD algorithm is used

*Constraint:* **spec** = Nag\_Sparse\_Identity, Nag\_Sparse\_User or Nag\_Sparse\_Colamd.

- 2: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $A$ .  
*Constraint:*  $n \geq 0$ .
- 3: **icolzp**[ $dim$ ] – const Integer *Input*  
**Note:** the dimension,  $dim$ , of the array **icolzp** must be at least  $n + 1$ .  
*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.
- 4: **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:* **irowix**[ $i - 1$ ] contains the row index in  $A$  for element  $A(i)$ . See Section 2.1.3 in the f11 Chapter Introduction.
- 5: **iprm**[ $7 \times n$ ] – Integer *Input/Output*  
*On entry:* the first  $n$  entries contain the column permutation supplied by you. This will be used if **spec** = Nag\_Sparse\_User, and ignored otherwise. If used, it must consist of a permutation of all the integers in the range  $[0, (n - 1)]$ , the leftmost column of the matrix  $A$  denoted by 0 and the rightmost by  $n - 1$ . Labelling columns in this way, **iprm**[ $i$ ] =  $j$  means that column  $i - 1$  of  $A$  is in position  $j$  in  $AP_c$ , where  $P_rAP_c = LU$  expresses the factorization to be performed.  
*On exit:* a new permutation is returned in the first  $n$  entries. The rest of the array contains data structures that will be used by other functions. The function computes the column elimination tree for  $A$  and a post-order permutation on the tree. It then compounds the **iprm** permutation given or computed by the COLAMD algorithm with the post-order permutation. This array is needed by the  $LU$  factorization function nag\_superlu\_lu\_factorize (f11mec) and associated functions nag\_superlu\_solve\_lu (f11mfc), nag\_superlu\_refine\_lu (f11mhc) and nag\_superlu\_diagnostic\_lu (f11mmc) and should be passed to them unchanged.
- 6: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALG\_FAIL

COLAMD algorithm failed.

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

### NE\_BAD\_PARAM

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

**NE\_INT**

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

**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 3.6.6 in the Essential Introduction for further information.

**NE\_INVALID\_PERM\_COL**

Incorrect column permutations in array **iprm**.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
 See Section 3.6.5 in the Essential Introduction for further information.

**NE\_SPARSE\_COL**

Incorrect specification of argument **icolzp**.

**NE\_SPARSE\_ROW**

Incorrect specification of argument **irrowix**.

**7 Accuracy**

Not applicable. This computation does not use floating-point numbers.

**8 Parallelism and Performance**

`nag_superlu_column_permutation` (f11mdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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**

We recommend calling this function with **spec** = Nag\_Sparse\_Colamd before calling `nag_superlu_lu_factorize` (f11mec). The COLAMD algorithm computes a sparsity-preserving permutation  $P_c$  solely from the pattern of  $A$  such that the  $LU$  factorization  $P_r A P_c = LU$  remains as sparse as possible, regardless of the subsequent choice of  $P_r$ . The algorithm takes advantage of the existence of super-columns (columns with the same sparsity pattern) to reduce running time.

**10 Example**

This example computes a sparsity preserving column permutation for 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_column_permutation (f11mdc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

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

int main(void)
{
    Integer                exit_status = 0, i, n, nnz;
    Integer                *icolzp = 0, *iprm = 0, *irowix = 0;
    /* Nag types */
    Nag_ColumnPermutationType ispec;
    NagError               fail;

    INIT_FAIL(fail);

    printf("nag_superlu_column_permutation (f11mdc) 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
    if (!(icolzp = NAG_ALLOC(n+1, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read the matrix A */
#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;
    if (!(irowix = NAG_ALLOC(nnz, Integer)) ||
        !(iprm = NAG_ALLOC(7*n, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#ifdef _WIN32
    for (i = 1; i <= nnz; ++i) scanf_s("%"NAG_IFMT"%*[\n] ", &irowix[i - 1]);
#else
    for (i = 1; i <= nnz; ++i) scanf("%"NAG_IFMT"%*[\n] ", &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;
    }

/* Output results */
printf("\n");
printf("%s\n", "COLAMD Permutation");
for (i = 1; i <= n; ++i)
    printf("%6"NAG_IFMT"%s", iprm[i - 1], i%10 == 0 || i == n?"\n":" ");

/* Calculate user permutation */
ispec = Nag_Sparse_User;
iprm[0] = 4;
iprm[1] = 3;
iprm[2] = 2;
iprm[3] = 1;
iprm[4] = 0;
/* nag_superlu_column_permutation (f11mdc), see above. */
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;
    }
/* Output results */
printf("\n");
printf("%s", "User Permutation");
printf("\n");
for (i = 1; i <= n; ++i)
    printf("%6"NAG_IFMT"%s", iprm[i - 1], i%10 == 0 || i == n?"\n":" ");

/* Calculate natural permutation */
ispec = Nag_Sparse_Identity;
/* nag_superlu_column_permutation (f11mdc), see above. */
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;
    }
/* Output results */
printf("\n");
printf("%s\n", "Natural Permutation");
for (i = 1; i <= n; ++i)
    printf("%6"NAG_IFMT"%s", iprm[i - 1], i%10 == 0 || i == n?"\n":" ");
printf("\n");

END:
NAG_FREE(icolzp);
NAG_FREE(iprm);
NAG_FREE(irowix);

return exit_status;
}

```

## 10.2 Program Data

```
nag_superlu_column_permutation (f11mdc) Example Program Data
  5      n
  1
  3
  5
  7
  9
 12      icolzp(i) i=0..n
  1
  3
  1
  5
  2
  3
  2
  4
  3
  4
  5      irowix(i) i=0..nnz-1
```

## 10.3 Program Results

```
nag_superlu_column_permutation (f11mdc) Example Program Results
```

```
COLAMD Permutation
  1      0      4      3      2

User Permutation
  4      3      2      1      0

Natural Permutation
  0      1      2      3      4
```

---