

## NAG Library Function Document

### nag\_sparse\_sym\_sort (f11zbc)

#### 1 Purpose

nag\_sparse\_sym\_sort (f11zbc) sorts the nonzero elements of a real sparse symmetric matrix, represented in symmetric coordinate storage format.

#### 2 Specification

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

void nag_sparse_sym_sort (Integer n, Integer *nnz, double a[],
    Integer irow[], Integer icol[], Nag_SparseSym_Dups dup,
    Nag_SparseSym_Zeros zero, Integer istr[], NagError *fail)
```

#### 3 Description

nag\_sparse\_sym\_sort (f11zbc) takes a symmetric coordinate storage (SCS) representation (see the f11 Chapter Introduction) of a real  $n$  by  $n$  sparse symmetric matrix  $A$ , and reorders the nonzero elements by increasing row index and increasing column index within each row. Entries with duplicate row and column indices may be removed, or the values may be summed. Any entries with zero values may optionally be removed.

nag\_sparse\_sym\_sort (f11zbc) also returns **istr** which contains the starting indices of each row in  $A$ .

#### 4 References

None.

#### 5 Arguments

- 1: **n** – Integer *Input*  
*On entry:* the order of the matrix  $A$ .  
*Constraint:*  $n \geq 1$ .
- 2: **nnz** – Integer \* *Input/Output*  
*On entry:* the number of nonzero elements in the lower triangular part of the matrix  $A$ .  
*Constraint:*  $nnz \geq 0$ .  
*On exit:* the number of lower triangular nonzero elements with unique row and column indices.
- 3: **a**[**max(1, nnz)**] – double *Input/Output*  
*On entry:* the nonzero elements of the lower triangular part of the matrix  $A$ . These may be in any order and there may be multiple nonzero elements with the same row and column indices.  
*On exit:* the lower triangular nonzero elements ordered by increasing row index, and by increasing column index within each row. Each nonzero element has a unique row and column index.
- 4: **irow**[**max(1, nnz)**] – Integer *Input/Output*  
*On entry:* the row indices of the elements supplied in array **a**.  
*Constraint:*  $1 \leq \mathbf{irow}[i] \leq n$ , for  $i = 0, 1, \dots, nnz - 1$ .

*On exit:* the first **nnz** elements contain the row indices corresponding to the elements returned in array **a**.

5: **icol**[**max**(1, **nnz**)] – Integer *Input/Output*

*On entry:* the column indices of the elements supplied in array **a**

*Constraint:*  $1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i]$ , for  $i = 0, 1, \dots, \mathbf{nnz} - 1$ .

*On exit:* the first **nnz** elements contain the column indices corresponding to the elements returned in array **a**.

6: **dup** – Nag\_SparseSym\_Dups *Input*

*On entry:* indicates how any nonzero elements with duplicate row and column indices are to be treated:

if **dup** = Nag\_SparseSym\_RemoveDups then duplicate elements are removed;

if **dup** = Nag\_SparseSym\_SumDups then duplicate elements are summed.

*Constraint:* **dup** = Nag\_SparseSym\_RemoveDups or Nag\_SparseSym\_SumDups.

7: **zero** – Nag\_SparseSym\_Zeros *Input*

*On entry:* indicates how any elements with zero values in **a** are to be treated:

if **zero** = Nag\_SparseSym\_RemoveZeros then elements with zero value are removed;

if **zero** = Nag\_SparseSym\_KeepZeros then elements with zero value are kept.

*Constraint:* **zero** = Nag\_SparseSym\_RemoveZeros or Nag\_SparseSym\_KeepZeros.

8: **istr**[**n** + 1] – Integer *Output*

*On exit:* **istr**[ $i - 1$ ] – 1, for  $i = 1, 2, \dots, \mathbf{n}$ , is the starting index in the arrays **a**, **irow** and **icol** of each row  $i$  of the matrix  $A$ . **istr**[ $\mathbf{n}$ ] – 1 is the index of the last nonzero element in  $A$  plus one.

9: **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 **dup** had an illegal value.

On entry, argument **zero** had an illegal value.

### NE\_INT\_ARG\_LT

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

Constraint:  $\mathbf{n} \geq 1$ .

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

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

### NE\_SYMM\_MATRIX

A nonzero element has been supplied which does not lie in the lower triangular part of the matrix  $A$ , i.e., one or more of the following constraints has been violated:  $1 \leq \mathbf{irow}[i] \leq \mathbf{n}$ ,  $1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i]$ , for  $i = 0, 1, \dots, \mathbf{nnz} - 1$ .

## 7 Accuracy

Not applicable.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

The time taken for a call to `nag_sparse_sym_sort` (f11zbc) is proportional to **nnz**. Note that the resulting matrix may have either rows or columns with no entries. If row  $i$  has no entries then  $\mathbf{istr}[i - 1] = \mathbf{istr}[i]$ .

## 10 Example

This example program reads the SCS representation of a real sparse symmetric matrix  $A$ , reorders the nonzero elements, and outputs the original and the reordered representations.

### 10.1 Program Text

```

/* nag_sparse_sym_sort (f11zbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 */

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

int main(void)
{
    double          *a = 0;
    Integer         *icol = 0;
    Integer         *irow = 0, *istr = 0;
    Integer         exit_status = 0, i, n, nnz;
    Nag_SparseSym_Zeros zero;
    Nag_SparseSym_Dups dup;
    NagError        fail;

    INIT_FAIL(fail);

    printf("nag_sparse_sym_sort (f11zbc) Example Program Results\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s(" %*[\n]");
#else
    scanf(" %*[\n]");
#endif

    /* Read order of matrix and number of non-zero entries */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n]", &n);
#else
    scanf("%"NAG_IFMT"%*[\n]", &n);
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n]", &nnz);
#else
    scanf("%"NAG_IFMT"%*[\n]", &nnz);
#endif
}

```

```

/* Allocate memory */
istr = NAG_ALLOC(n+1, Integer);
a = NAG_ALLOC(nnz, double);
irow = NAG_ALLOC(nnz, Integer);
icol = NAG_ALLOC(nnz, Integer);

if (!istr || !irow || !icol || !a)
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read and output the original non-zero elements */
for (i = 1; i <= nnz; ++i)
#ifdef _WIN32
    scanf_s("%lf%"NAG_IFMT%"NAG_IFMT"%*[\n]", &a[i-1], &irow[i-1], &icol[i-1]);
#else
    scanf("%lf%"NAG_IFMT%"NAG_IFMT"%*[\n]", &a[i-1], &irow[i-1], &icol[i-1]);
#endif
printf(" Original elements \n");
printf(" nnz = %6"NAG_IFMT"\n", nnz);
for (i = 1; i <= nnz; ++i)
    printf(" %8"NAG_IFMT"%16.4e%8"NAG_IFMT"%8"NAG_IFMT"\n", i, a[i-1],
        irow[i-1], icol[i-1]);

/* Reorder, sum duplicates and remove zeros */
dup = Nag_SparseSym_SumDups;
zero = Nag_SparseSym_RemoveZeros;

/* nag_sparse_sym_sort (f11zbc).
 * Sparse sort (symmetric)
 */
nag_sparse_sym_sort(n, &nnz, a, irow, icol, dup, zero, istr, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sparse_sym_sort (f11zbc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Output results */
printf(" Reordered elements \n");
printf(" nnz = %4"NAG_IFMT"\n", nnz);

for (i = 1; i <= nnz; ++i)
    printf(" %8"NAG_IFMT"%16.4e%8"NAG_IFMT"%8"NAG_IFMT"\n", i, a[i-1],
        irow[i-1], icol[i-1]);

END:
NAG_FREE(istr);
NAG_FREE(irow);
NAG_FREE(icol);
NAG_FREE(a);
return exit_status;
}

```

## 10.2 Program Data

```

nag_sparse_sym_sort (f11zbc) Example Program Data
4          n
9          nnz
1.0    3    2
0.0    2    1
1.0    3    2
3.0    4    4

```

```

4.0  1  1
6.0  2  2
2.0  3  3
1.0  3  2
1.0  3  2          a[i-1], irow[i-1], icol[i-1], i=1,...,nnz

```

### 10.3 Program Results

nag\_sparse\_sym\_sort (f11zbc) Example Program Results

Original elements

nnz = 9

1	1.0000e+00	3	2
2	0.0000e+00	2	1
3	1.0000e+00	3	2
4	3.0000e+00	4	4
5	4.0000e+00	1	1
6	6.0000e+00	2	2
7	2.0000e+00	3	3
8	1.0000e+00	3	2
9	1.0000e+00	3	2

Reordered elements

nnz = 5

1	4.0000e+00	1	1
2	6.0000e+00	2	2
3	4.0000e+00	3	2
4	2.0000e+00	3	3
5	3.0000e+00	4	4

---