nag_sparse_nsym_sort (f11zac)

1. Purpose

nag_sparse_nsym_sort (f11zac) sorts the non-zero elements of a real sparse nonsymmetric matrix, represented in coordinate storage format.

2. Specification

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

void nag_sparse_nsym_sort(Integer n, Integer *nnz, double a[], Integer irow[],
            Integer icol[], Nag_SparseNsym_Dups dup,
            Nag_SparseNsym_Zeros zero, Integer istr[], NagError *fail)
```

3. Description

nag_sparse_nsym_sort (f11zac) takes a coordinate storage (CS) representation (see Section 2.1.1 of the Chapter Introduction) of a real n by n sparse nonsymmetric matrix A, and reorders the non-zero 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_nsym_sort (f11zac) also returns **istr** which contains the starting indices of each row in A.

4. Parameters

n

Input: the order of the matrix A.

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

nnz

Input: the number of non-zero elements in the matrix A.

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

Output: the number of non-zero elements with unique row and column indices.

a[max(**1**,**nnz**)]

Input: the non-zero elements of the matrix A. These may be in any order and there may be multiple non-zero elements with the same row and column indices.

Output: the non-zero elements ordered by increasing row index, and by increasing column index within each row. Each non-zero element has a unique row and column index.

irow[max(**1**,**nnz**)]

Input: the row indices of the elements supplied in array **a**.

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

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

icol[max(**1**,**nnz**)]

Input: the column indices of the elements supplied in array **a**.

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

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

dup

Input: indicates how any non-zero elements with duplicate row and column indices are to be treated:

if **dup** = **Nag_SparseNsym_RemoveDups** then duplicate elements are removed;

if **dup** = **Nag_SparseNsym_SumDups** then the relevant values in array **a** are summed;

if **dup** = **Nag_SparseNsym_FailDups** then the routine fails on detecting a duplicate.

Constraint: **dup** = **Nag_SparseNsym_RemoveDups**, **Nag_SparseNsym_SumDups**, or **Nag_SparseNsym_FailDups**.

zero

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

if **zero** = **Nag_SparseNsym_RemoveZeros** then the entries are removed;

if **zero** = **Nag_SparseNsym_KeepZeros** then the entries are kept;

if **zero** = **Nag_SparseNsym_FailZeros** then the routine fails on detecting a zero.

Constraint: **zero** = **Nag_SparseNsym_RemoveZeros**, **Nag_SparseNsym_KeepZeros**, or **Nag_SparseNsym_FailZeros**.

istr[n+1]

Output: **istr**$[i-1]$, for $i = 1, 2, \ldots,$**n**, contains the starting index in the arrays **a**, **irow** and **icol** of each row i of the matrix A. **istr**$[n]$ contains the index of the last non-zero element in A plus one.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_BAD_PARAM

On entry, parameter **dup** had an illegal value.
On entry, parameter **zero** had an illegal value.

NE_INT_ARG_LT

On entry, **n** must not be less than 1: **n** = ⟨*value*⟩.
On entry, **nnz** must not be less than 0: **nnz** = ⟨*value*⟩.

NE_NONSYMM_MATRIX

A non-zero element has been supplied which does not lie within the matrix A,
i.e., one or more of the following constraints has been violated:
$1 \leq$ **irow**$[i] \leq$ **n**, $1 \leq$ **icol**$[i] \leq$ **n**, for $i = 0, 1. \ldots,$**nnz**-1.

NE_NON_ZERO_DUP

Non-zero elements have been supplied which have duplicate row and column indices, when **dup** = **Nag_SparseNsym_FailDups**.

NE_ZERO_COEFF

At least one matrix element has been supplied with a zero coefficient value, when **zero** = **Nag_SparseNsym_FailZeros**.

NE_ALLOC_FAIL

Memory allocation failed.

6. Further Comments

The time taken for a call to nag_sparse_nsym_sort (f11zac) 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 **istr**$[i-1]$ = **istr**$[i]$.

7. See Also

None.

8. Example

This example program reads the CS representation of a real sparse matrix A, reorders the non-zero elements, and outputs the original and the reordered representations.

8.1. Program Text

```
/* nag_sparse_nsym_sort (f11zac) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 */

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

main()
{
  double *a=0;

  Integer *icol=0;
  Integer *irow=0, *istr=0;
  Integer i, n, nnz;

  Nag_SparseNsym_Zeros zero;
  Nag_SparseNsym_Dups dup;

  Vprintf("f11zac Example Program Results\n");

  /*  Skip heading in data file */
  Vscanf(" %*[^\n]");

  /*  Read order of matrix and number of non-zero entries */
  Vscanf("%ld%*[^\n]",&n);
  Vscanf("%ld%*[^\n]",&nnz);

  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)
    {
      Vprintf("Allocation failure\n");
      exit (EXIT_FAILURE);
    }

  /*  Read and output the original non-zero elements */
  for (i = 1; i <= nnz; ++i)
    Vscanf("%lf%ld%ld%*[^\n]",&a[i-1], &irow[i-1], &icol[i-1]);

  Vprintf(" Original elements \n");
  Vprintf(" nnz =  %6ld\n",nnz);
  for (i = 1; i <= nnz; ++i)
    Vprintf(" %8ld%16.4e%8ld%8ld\n",i,a[i-1],irow[i-1],icol[i-1]);

  /*  Reorder, sum duplicates and remove zeros */

  dup = Nag_SparseNsym_SumDups;
  zero = Nag_SparseNsym_RemoveZeros;

  f11zac(n, &nnz, a, irow, icol, dup, zero, istr, NAGERR_DEFAULT);

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

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

  NAG_FREE(istr);
```

```
  NAG_FREE(irow);
  NAG_FREE(icol);
  NAG_FREE(a);
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

```
f11zac Example Program Data
  5                   n
  15                  nnz
  4.   3    1
 -2.   5    2
  1.   4    4
 -2    4    2
 -3    5    5
  1.   1    2
  0.   1    5
  1.   3    5
 -1.   2    4
  6.   5    5
  2.   1    1
  2.   4    2
  1.   2    3
  1.   3    3
  2.   4    5          a[i-1], irow[i-1], icol[i-1], i=1,...,nnz
```

8.3. Program Results

```
f11zac Example Program Results
 Original elements
 nnz =      15
        1      4.0000e+00       3       1
        2     -2.0000e+00       5       2
        3      1.0000e+00       4       4
        4     -2.0000e+00       4       2
        5     -3.0000e+00       5       5
        6      1.0000e+00       1       2
        7      0.0000e+00       1       5
        8      1.0000e+00       3       5
        9     -1.0000e+00       2       4
       10      6.0000e+00       5       5
       11      2.0000e+00       1       1
       12      2.0000e+00       4       2
       13      1.0000e+00       2       3
       14      1.0000e+00       3       3
       15      2.0000e+00       4       5
 Reordered elements
 nnz =   11
        1      2.0000e+00       1       1
        2      1.0000e+00       1       2
        3      1.0000e+00       2       3
        4     -1.0000e+00       2       4
        5      4.0000e+00       3       1
        6      1.0000e+00       3       3
        7      1.0000e+00       3       5
        8      1.0000e+00       4       4
        9      2.0000e+00       4       5
       10     -2.0000e+00       5       2
       11      3.0000e+00       5       5
```