NAG Library Function Document

nag_sparse_nsym_sort (f11zac)

1 Purpose

nag_sparse_nsym_sort (f11zac) sorts the nonzero 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 the f11 Chapter Introduction) of a real n by n sparse nonsymmetric 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_nsym_sort (f11zac) also returns **istr** which contains the starting indices of each row in A. This can be used to construct a compressed column storage (CCS) representation of the matrix (see Section 9).

4 References

None.

5 Arguments

1: **n** – Integer *Input*

On entry: the order of the matrix A.

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

2: **nnz** – Integer * *Input/Output*

On entry: the number of nonzero elements in the matrix A.

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

On exit: the number of nonzero elements with unique row and column indices.

3: **a**[$\mathbf{max}(\mathbf{1}, \mathbf{nnz})$] – double *Input/Output*

On entry: the nonzero elements 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 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: $1 \leq \mathbf{irow}[i] \leq \mathbf{n}$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$.

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

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

6: **dup** – Nag_SparseNsym_Dups *Input*

On entry: indicates how any nonzero 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 function fails on detecting a duplicate.

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

7: **zero** – Nag_SparseNsym_Zeros *Input*

On entry: 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 function fails on detecting a zero.

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

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

On exit: **istr**$[i-1]-1$, for $i = 1, 2, \ldots, \mathbf{n}$, is the starting index in the arrays **a**, **irow** and **icol** of each row i of the matrix A. **istr**$[n]$ contains the number of nonzero elements 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, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 1$.

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

NE_NON_ZERO_DUP

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

NE_NONSYMM_MATRIX

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

NE_ZERO_COEFF

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 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 $\mathbf{istr}[i-1] = \mathbf{istr}[i]$.

It is also possible to use this function to convert between coordinate storage (CS) and compressed column storage (CCS) formats. To achieve this the CS storage format arrays **irow** and **icol** must be interchanged in the call to nag_sparse_nsym_sort (f11zac). On exit from nag_sparse_nsym_sort (f11zac), the CCS representation of the matrix is then defined by arrays **a**, **irow** and **istr**. This is illustrated in Section 10.

10 Example

This example program reads the CS representation of a real sparse matrix A, calls nag_sparse_nsym_sort (f11zac) to reorder the nonzero elements, and outputs the original and the reordered representations.It then calls nag_sparse_nsym_sort (f11zac) again with the alternative ordering, creating a CCS representation which is then passed to a function that computes a matrix norm for that representation.

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

10.1 Program Text

```
/* nag_sparse_nsym_sort (f11zac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 */

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

int main(void)
{
  double               *a = 0, anorm = 0.0;
  Integer              *icol = 0;
  Integer              *irow = 0, *istr = 0;
  Integer              exit_status = 0, i, n, nnz;
  Nag_SparseNsym_Zeros zero;
  Nag_SparseNsym_Dups  dup;
  Nag_NormType         norm;
  NagError             fail;

  INIT_FAIL(fail);

  printf("nag_sparse_nsym_sort (f11zac) Example Program Results\n\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 = 0; i < nnz; ++i)
#ifdef _WIN32
    scanf_s("%lf%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &a[i], &irow[i], &icol[i]);
#else
    scanf("%lf%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &a[i], &irow[i], &icol[i]);
#endif

  printf("Original elements \n");
```

```
  printf("nnz =  %4"NAG_IFMT"\n", nnz);
  printf("%8s%16s%8s%8s\n","","a","irow","icol");
  for (i = 0; i < nnz; ++i)
    printf("%8"NAG_IFMT"%16.4e%8"NAG_IFMT"%8"NAG_IFMT"\n", i, a[i],
           irow[i], icol[i]);

  /* Reorder along rows, sum duplicates and remove zeros */

  dup = Nag_SparseNsym_SumDups;
  zero = Nag_SparseNsym_RemoveZeros;

  /* nag_sparse_nsym_sort (f11zac).
   * Sparse sort (nonsymmetric)
   */
  nag_sparse_nsym_sort(n, &nnz, a, irow, icol, dup, zero, istr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_nsym_sort (f11zac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

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

  for (i = 0; i < nnz; ++i)
    printf("%8"NAG_IFMT"%16.4e%8"NAG_IFMT"%8"NAG_IFMT"\n", i, a[i],
           irow[i], icol[i]);

  /* Reorder down columns, fail on duplicates or zeros.
   *  Creates CCS storage format as side-effect
   */

  dup = Nag_SparseNsym_FailDups;
  zero = Nag_SparseNsym_FailZeros;
  INIT_FAIL(fail);

  /* nag_sparse_nsym_sort (f11zac).
   * Sparse sort (nonsymmetric)
   */
  nag_sparse_nsym_sort(n, &nnz, a, icol, irow, dup, zero, istr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_nsym_sort (f11zac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

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

  for (i = 0; i < nnz; ++i)
    printf("%8"NAG_IFMT"%16.4e%8"NAG_IFMT"%8"NAG_IFMT"\n", i, a[i],
           irow[i], icol[i]);
  printf("%8s%8s\n","","istr");
  for (i = 0; i <= n; ++i)
    printf("%8"NAG_IFMT"%8"NAG_IFMT"\n", i, istr[i]);

  /* Calculate 1-norm in Compressed Column Storage format */
  norm = Nag_RealOneNorm;
  INIT_FAIL(fail);

  /* nag_superlu_matrix_norm (f11mlc).
   * 1-norm, infinity-norm, largest absolute element, real
   * general matrix
   */
  nag_superlu_matrix_norm(norm, &anorm, n, istr, irow, a, &fail);
```

```
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_superlu_matrix_norm (f11mlc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /* Output norm */
  printf("%s %16.4e\n", "One-norm", anorm);


 END:
  NAG_FREE(istr);
  NAG_FREE(irow);
  NAG_FREE(icol);
  NAG_FREE(a);

  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_nsym_sort (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
```

10.3 Program Results

```
nag_sparse_nsym_sort (f11zac) Example Program Results

Original elements
nnz =    15
                       a    irow    icol
       0      4.0000e+00       3       1
       1     -2.0000e+00       5       2
       2      1.0000e+00       4       4
       3     -2.0000e+00       4       2
       4     -3.0000e+00       5       5
       5      1.0000e+00       1       2
       6      0.0000e+00       1       5
       7      1.0000e+00       3       5
       8     -1.0000e+00       2       4
       9      6.0000e+00       5       5
      10      2.0000e+00       1       1
      11      2.0000e+00       4       2
      12      1.0000e+00       2       3
      13      1.0000e+00       3       3
      14      2.0000e+00       4       5
Reordered elements, along rows first
nnz =   11
       0      2.0000e+00       1       1
       1      1.0000e+00       1       2
       2      1.0000e+00       2       3
       3     -1.0000e+00       2       4
       4      4.0000e+00       3       1
```

```
       5      1.0000e+00       3       3
       6      1.0000e+00       3       5
       7      1.0000e+00       4       4
       8      2.0000e+00       4       5
       9     -2.0000e+00       5       2
      10      3.0000e+00       5       5
Reordered elements, along columns first
nnz =   11
       0      2.0000e+00       1       1
       1      4.0000e+00       3       1
       2      1.0000e+00       1       2
       3     -2.0000e+00       5       2
       4      1.0000e+00       2       3
       5      1.0000e+00       3       3
       6     -1.0000e+00       2       4
       7      1.0000e+00       4       4
       8      1.0000e+00       3       5
       9      2.0000e+00       4       5
      10      3.0000e+00       5       5
            istr
       0       1
       1       3
       2       5
       3       7
       4       9
       5      12
One-norm       6.0000e+00
```