NAG Library Function Document

nag_sparse_herm_sort (f11zpc)

1   Purpose

nag_sparse_herm_sort (f11zpc) sorts the nonzero elements of a sparse complex Hermitian matrix, represented in symmetric coordinate storage format.

2   Specification

```
#include <nag.h>
#include <nagf11.h>
void nag_sparse_herm_sort (Integer n, Integer *nnz, Complex a[],
     Integer irow[], Integer icol[], Nag_SparseSym_Dups dup,
     Nag_SparseSym_Zeros zero, Integer istr[], NagError *fail)
```

3   Description

nag_sparse_herm_sort (f11zpc) takes a symmetric coordinate storage (SCS) representation (see Section 2.1.2 in the f11 Chapter Introduction) of a sparse n by n complex Hermitian 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.

The function also returns a pointer array **istr** to the starting address of each row in A.

4   References

None.

5   Arguments

1:     **n** – Integer   *Input*

On entry: n, 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 lower triangular part of the matrix A.

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

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

3:     **a**[dim] – Complex   *Input/Output*

Note: the dimension, dim, of the array **a** must be at least $\max(1, \mathbf{nnz})$.

On entry: the nonzero elements of the lower triangular part of the complex 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**[*dim*] – Integer   Input/Output

   Note: the dimension, *dim*, of the array **irow** must be at least max(1, **nnz**).

   On entry: the row indices corresponding to the nonzero elements supplied in the array **a**.

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

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

5:   **icol**[*dim*] – Integer   Input/Output

   Note: the dimension, *dim*, of the array **icol** must be at least max(1, **nnz**).

   On entry: the column indices corresponding to the nonzero elements supplied in the array **a**.

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

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

   dup = Nag_SparseSym_RemoveDups
        The entries are removed.

   dup = Nag_SparseSym_SumDups
        The relevant values in **a** are summed.

   dup = Nag_SparseSym_FailDups
        The function fails with **fail.code** = NE_NON_ZERO_DUP on detecting a duplicate.

   Constraint: **dup** = Nag_SparseSym_RemoveDups, Nag_SparseSym_SumDups or Nag_SparseSym_FailDups.

7:   **zero** – Nag_SparseSym_Zeros   Input

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

   zero = Nag_SparseSym_RemoveZeros
        The entries are removed.

   zero = Nag_SparseSym_KeepZeros
        The entries are kept.

   zero = Nag_SparseSym_FailZeros
        The function fails with **fail.code** = NE_ZERO_COEFF on detecting a zero.

   Constraint: **zero** = Nag_SparseSym_RemoveZeros, Nag_SparseSym_KeepZeros or Nag_SparseSym_FailZeros.

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

   On exit: $\textbf{istr}[i-1] - 1$, for $i = 1, 2, \ldots, \textbf{n}$, is the starting address in the arrays **a**, **irow** and **icol** of row i of the matrix A. $\textbf{istr}[\textbf{n}] - 1$ is the address 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 $\langle value \rangle$ had an illegal value.

NE_INT

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_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.

NE_INVALID_SCS

On entry, $I = \langle value \rangle$, $\mathbf{icol}[I - 1] = \langle value \rangle$ and $\mathbf{irow}[I - 1] = \langle value \rangle$.
Constraint: $\mathbf{icol}[I - 1] \geq 1$ and $\mathbf{icol}[I - 1] \leq \mathbf{irow}[I - 1]$.

On entry, $i = \langle value \rangle$, $\mathbf{irow}[i - 1] = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{irow}[i - 1] \geq 1$ and $\mathbf{irow}[i - 1] \leq \mathbf{n}$.

NE_NON_ZERO_DUP

On entry, a duplicate entry has been found in row I and column J: $I = \langle value \rangle$, $J = \langle value \rangle$.

NE_ZERO_COEFF

On entry, a zero entry has been found in row I and column J: $I = \langle value \rangle$, $J = \langle value \rangle$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken for a call to nag_sparse_herm_sort (f11zpc) 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 $\mathbf{istr}[i] = \mathbf{istr}[i + 1]$.

10 Example

This example reads the SCS representation of a complex sparse Hermitian matrix A, calls nag_sparse_herm_sort (f11zpc) to reorder the nonzero elements, and outputs the original and the reordered representations.

10.1 Program Text

```
/* nag_sparse_herm_sort (f11zpc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{
  /* Scalars */
  Integer            exit_status = 0;
  Integer            i, n, nnz;
  /* Arrays */
  char               nag_enum_arg[40];
  Integer            *irow = 0, *icol = 0, *istr = 0;
  Complex            *a = 0;
  /* NAG types */
  NagError           fail;
  Nag_SparseSym_Dups  dup;
  Nag_SparseSym_Zeros zero;

  INIT_FAIL(fail);

  printf("nag_sparse_herm_sort (f11zpc) Example Program Results\n");

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

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

  /* Allocate memory */
  if ( !(a = NAG_ALLOC(nnz, Complex)) ||
       !(icol = NAG_ALLOC(nnz, Integer)) ||
       !(irow = NAG_ALLOC(nnz, Integer)) ||
       !(istr = NAG_ALLOC((n+1), Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read and output the original non-zero elements */
  for (i = 0; i < nnz; i++)
    scanf(" ( %lf , %lf ) %ld%ld%*[^\n]", &a[i].re, &a[i].im,
          &irow[i], &icol[i]);

  /* Reorder, sum duplicates and remove zeros */
  /* Nag_SparseSym_SumDups */
  scanf("%39s %*[^\n]", nag_enum_arg);
  dup = (Nag_SparseSym_Dups) nag_enum_name_to_value (nag_enum_arg);

  /* Nag_SparseSym_RemoveZeros */
  scanf("%39s %*[^\n]", nag_enum_arg);
  zero = (Nag_SparseSym_Zeros) nag_enum_name_to_value (nag_enum_arg);

  /* Output original */
  printf("\nOriginal elements\n");
  printf("%s%4ld\n"," n   =", n);
  printf("%s%4ld\n"," nnz =", nnz);

  printf("%9s%14s%22s%9s\n","i","a","irow","icol");
  for (i = 0; i < nnz; i++)
    printf("%9ld (%13.4e, %13.4e)%9ld%9ld\n",
```

```
                    i, a[i].re, a[i].im, irow[i], icol[i]);

  /* nag_sparse_herm_sort (f11zpc).
   * Complex sparse Hermitian matrix reorder function.
   */
  nag_sparse_herm_sort(n, &nnz, a, irow, icol, dup, zero, istr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_herm_sort (f11zpc)\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Output results */
  printf("\nReordered elements\n");
  printf("%s%4ld\n"," nnz =", nnz);

  printf("%9s%14s%22s%9s\n","i","a","irow","icol");
  for (i = 0; i < nnz; i++)
    printf("%9ld (%13.4e, %13.4e)%9ld%9ld\n",
           i, a[i].re, a[i].im, irow[i], icol[i]);

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

  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_herm_sort (f11zpc) Example Program Data
   4                         : n
   9                         : nnz
 ( 1.,  2.)        3    2
 ( 0.,  0.)        2    1
 ( 0.,  3.)        3    2
 ( 3., -5.)        4    4
 ( 4.,  2.)        1    1
 ( 0.,  3.)        2    2
 ( 2.,  4.)        3    3
 ( 1., -1.)        3    2
 ( 1.,  3.)        3    2    : (a, irow, icol)[i], i=0,...,nnz-1
   Nag_SparseSym_SumDups      : dup
   Nag_SparseSym_RemoveZeros  : zero
```

10.3 Program Results

```
nag_sparse_herm_sort (f11zpc) Example Program Results

Original elements
 n   =   4
 nnz =   9
      i             a                   irow     icol
      0 (   1.0000e+00,    2.0000e+00)        3        2
      1 (   0.0000e+00,    0.0000e+00)        2        1
      2 (   0.0000e+00,    3.0000e+00)        3        2
      3 (   3.0000e+00,   -5.0000e+00)        4        4
      4 (   4.0000e+00,    2.0000e+00)        1        1
      5 (   0.0000e+00,    3.0000e+00)        2        2
      6 (   2.0000e+00,    4.0000e+00)        3        3
      7 (   1.0000e+00,   -1.0000e+00)        3        2
      8 (   1.0000e+00,    3.0000e+00)        3        2

Reordered elements
 nnz =   5
      i             a                   irow     icol
      0 (   4.0000e+00,    2.0000e+00)        1        1
```

```
1 (    0.0000e+00,     3.0000e+00)          2           2
2 (    3.0000e+00,     7.0000e+00)          3           2
3 (    2.0000e+00,     4.0000e+00)          3           3
4 (    3.0000e+00,    -5.0000e+00)          4           4
```