NAG Library Function Document

nag_sparse_nherm_sort (f11znc)

1 Purpose

nag_sparse_nherm_sort (f11znc) sorts the nonzero elements of a complex sparse non-Hermitian matrix, represented in coordinate storage format.

2 Specification

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

3 Description

nag_sparse_nherm_sort (f11znc) takes a coordinate storage (CS) representation (see Section 2.1.1 in the f11 Chapter Introduction) of a sparse n by n complex non-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 matrix A.

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

On exit: the number of 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 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**$[dim]$ – Integer *Input/Output*

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

On entry: the row indices corresponding to the nonzero elements supplied in the 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 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 $\max(1, \mathbf{nnz})$.

On entry: the column indices corresponding to the nonzero elements supplied in the 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 row indices corresponding to the nonzero elements returned in the 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.

dup = Nag_SparseNsym_RemoveDups
The entries are removed.

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

dup = Nag_SparseNsym_FailDups
The function fails with **fail**.**code** = NE_NON_ZERO_DUP 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 array **a** are to be treated.

zero = Nag_SparseNsym_RemoveZeros
The entries are removed.

zero = Nag_SparseNsym_KeepZeros
The entries are kept.

zero = Nag_SparseNsym_FailZeros
The function fails with **fail**.**code** = NE_ZERO_COEFF 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 address in the arrays **a**, **irow** and **icol** of row i of the matrix A. **istr**[**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 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 1.

On entry, **nnz** = ⟨*value*⟩.
Constraint: **nnz** ≥ 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_INVALID_CS

On entry, $i = \langle value \rangle$, **icol**$[i-1] = \langle value \rangle$ and **n** = ⟨*value*⟩.
Constraint: **icol**$[i-1] \geq 1$ and **icol**$[i-1] \leq$ **n**.

On entry, $i = \langle value \rangle$, **irow**$[i-1] = \langle value \rangle$ and **n** = ⟨*value*⟩.
Constraint: **irow**$[i-1] \geq 1$ and **irow**$[i-1] \leq$ **n**.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

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

nag_sparse_nherm_sort (f11znc) is not threaded in any implementation.

9 Further Comments

The time taken for a call to nag_sparse_nherm_sort (f11znc) 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**$[i]$.

10 Example

This example reads the CS representation of a complex sparse matrix A, calls nag_sparse_nherm_sort (f11znc) to reorder the nonzero elements, and outputs the original and the reordered representations.

10.1 Program Text

```
/* nag_sparse_nherm_sort (f11znc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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_SparseNsym_Dups dup;
  Nag_SparseNsym_Zeros zero;

  INIT_FAIL(fail);

  printf("nag_sparse_nherm_sort (f11znc) 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 nonzero 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 */
  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 nonzero elements */
```

```
  for (i = 0; i < nnz; i++)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf ) %" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &a[i].re,
            &a[i].im, &irow[i], &icol[i]);
#else
    scanf(" ( %lf , %lf ) %" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &a[i].re,
          &a[i].im, &irow[i], &icol[i]);
#endif

  /* Reorder, sum duplicates and remove zeros */
  /* Nag_SparseNsym_SumDups */
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  dup = (Nag_SparseNsym_Dups) nag_enum_name_to_value(nag_enum_arg);

  /* Nag_SparseNsym_RemoveZeros */
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  zero = (Nag_SparseNsym_Zeros) nag_enum_name_to_value(nag_enum_arg);

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

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

  /* nag_sparse_nherm_sort (f11znc).
   * Complex sparse non-Hermitian matrix reorder function.
   */
  nag_sparse_nherm_sort(n, &nnz, a, irow, icol, dup, zero, istr, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nherm_sort (f11znc)\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

  printf("%9s%14s%22s%9s\n", "i", "a", "irow", "icol");
  for (i = 0; i < nnz; i++)
    printf("%9" NAG_IFMT " (%13.4e, %13.4e)%9" NAG_IFMT "%9" NAG_IFMT "\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_nherm_sort (f11znc) Example Program Data
  5                          : n
  15                         : nnz
 ( 4.,  1.)   3    1
 (-2.,  6.)   5    2
 ( 1., -3.)   4    4
 (-2., -1.)   4    2
 (-3.,  0.)   5    5
 ( 1.,  2.)   1    2
 ( 0.,  0.)   1    5
 ( 1.,  3.)   3    5
 (-1., -1.)   2    4
 ( 6., -3.)   5    5
 ( 2.,  6.)   1    1
 ( 2.,  1.)   4    2
 ( 1.,  0.)   2    3
 ( 0., -3.)   3    3
 ( 2.,  2.)   4    5       : (a, irow, icol)[i], i=0,...,nnz-1
Nag_SparseNsym_SumDups     : dup
Nag_SparseNsym_RemoveZeros : zero
```

10.3 Program Results

```
nag_sparse_nherm_sort (f11znc) Example Program Results

Original elements
 n   =    5
 nnz =   15
        i             a                   irow     icol
        0 (   4.0000e+00,    1.0000e+00)        3        1
        1 (  -2.0000e+00,    6.0000e+00)        5        2
        2 (   1.0000e+00,   -3.0000e+00)        4        4
        3 (  -2.0000e+00,   -1.0000e+00)        4        2
        4 (  -3.0000e+00,    0.0000e+00)        5        5
        5 (   1.0000e+00,    2.0000e+00)        1        2
        6 (   0.0000e+00,    0.0000e+00)        1        5
        7 (   1.0000e+00,    3.0000e+00)        3        5
        8 (  -1.0000e+00,   -1.0000e+00)        2        4
        9 (   6.0000e+00,   -3.0000e+00)        5        5
       10 (   2.0000e+00,    6.0000e+00)        1        1
       11 (   2.0000e+00,    1.0000e+00)        4        2
       12 (   1.0000e+00,    0.0000e+00)        2        3
       13 (   0.0000e+00,   -3.0000e+00)        3        3
       14 (   2.0000e+00,    2.0000e+00)        4        5

Reordered elements
 nnz =   11
        i             a                   irow     icol
        0 (   2.0000e+00,    6.0000e+00)        1        1
        1 (   1.0000e+00,    2.0000e+00)        1        2
        2 (   1.0000e+00,    0.0000e+00)        2        3
        3 (  -1.0000e+00,   -1.0000e+00)        2        4
        4 (   4.0000e+00,    1.0000e+00)        3        1
        5 (   0.0000e+00,   -3.0000e+00)        3        3
        6 (   1.0000e+00,    3.0000e+00)        3        5
        7 (   1.0000e+00,   -3.0000e+00)        4        4
        8 (   2.0000e+00,    2.0000e+00)        4        5
        9 (  -2.0000e+00,    6.0000e+00)        5        2
       10 (   3.0000e+00,   -3.0000e+00)        5        5
```