NAG Library Function Document

nag_sparse_sym_rcm (f11yec)

1 Purpose

nag_sparse_sym_rcm (f11yec) reduces the bandwidth of a sparse symmetric matrix stored in compressed column storage format using the Reverse Cuthill–McKee algorithm.

2 Specification

```
#include <nag.h>
#include <nagf11.h>
void nag_sparse_sym_rcm (Integer n, Integer nnz, const Integer icolzp[],
     const Integer irowix[], const Nag_Boolean lopts[], const Integer mask[],
     Integer perm[], Integer info[], NagError *fail)
```

3 Description

nag_sparse_sym_rcm (f11yec) takes the compressed column storage (CCS) representation (see Section 2.1.3 in the f11 Chapter Introduction) of an n by n symmetric matrix A and applies the Reverse Cuthill–McKee (RCM) algorithm which aims to minimize the bandwidth of the matrix A by reordering the rows and columns symmetrically. This also results in a lower profile of the matrix (see Section 9).

nag_sparse_sym_rcm (f11yec) can be useful for solving systems of equations $Ax = b$, as the permuted system $PAP^{\mathrm{T}}(Px) = Pb$ (where P is the permutation matrix described by the vector **perm** returned by nag_sparse_sym_rcm (f11yec)) may require less storage space and/or less computational steps when solving (see Wai-Hung and Sherman (1976)).

nag_sparse_sym_rcm (f11yec) may be used prior to nag_sparse_sym_chol_fac (f11jac) and nag_sparse_sym_precon_ichol_solve (f11jbc) (see Section 10 in nag_sparse_sym_precon_ichol_solve (f11jbc)).

4 References

Pissanetsky S (1984) *Sparse Matrix Technology* Academic Press

Wai-Hung L and Sherman A H (1976) Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices *SIAM J. Numer. Anal.* **13(2)** 198–213

5 Arguments

1: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: **n** ≥ 1.

2: **nnz** – Integer *Input*

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

Constraint: **nnz** ≥ 0.

3: **icolzp**[**n** + **1**] – const Integer *Input*

On entry: **icolzp** records the index into **irowix** which starts each new column.

Constraints:

$1 \leq \mathbf{icolzp}[i-1] \leq \mathbf{nnz}+1$, for $i = 2, 3, \ldots, \mathbf{n}$;
$\mathbf{icolzp}[0] = 1$;
$\mathbf{icolzp}[\mathbf{n}] = \mathbf{nnz}+1$, where $\mathbf{icolzp}[i-1]$ holds the position integer for the starts of the columns in **irowix**.

4: **irowix**[**nnz**] – const Integer *Input*

On entry: the row indices corresponding to the nonzero elements in the matrix A.

Constraint: $1 \leq \mathbf{irowix}[i-1] \leq \mathbf{n}$, for $i = 1, 2, \ldots, \mathbf{nnz}$.

5: **lopts**[**5**] – const Nag_Boolean *Input*

On entry: the options to be used by nag_sparse_sym_rcm (f11yec).

lopts[0] = Nag_TRUE
Row/column i of the matrix A will only be referenced if $\mathbf{mask}[i-1] \neq 0$, otherwise **mask** will be ignored.

lopts[1] = Nag_TRUE
The final permutation will not be reversed, that is, the Cuthill–McKee ordering will be returned. The bandwidth of the non-reversed matrix will be the same but the profile will be the same or larger (see Wai-Hung and Sherman (1976)).

lopts[2] = Nag_TRUE
The matrix A will be checked for symmetrical sparsity pattern, otherwise not.

lopts[3] = Nag_TRUE
The bandwidth and profile of the unpermuted matrix will be calculated, otherwise not.

lopts[4] = Nag_TRUE
The bandwidth and profile of the permuted matrix will be calculated, otherwise not.

6: **mask**[*dim*] – const Integer *Input*

Note: the dimension, *dim*, of the array **mask** must be at least

n when **lopts**[0] = Nag_TRUE;
otherwise **mask** may be **NULL**.

On entry: **mask** is only referenced if **lopts**[0] is Nag_TRUE otherwise **mask** may be set to **NULL**. A value of $\mathbf{mask}[i-1] = 0$ indicates that the node corresponding to row or column i is not to be referenced. A value of $\mathbf{mask}[i-1] \neq 0$ indicates that the node corresponding to row or column i is to be referenced. In particular, rows and columns not referenced will not be permuted.

7: **perm**[**n**] – Integer *Output*

On exit: this will contain the permutation vector that describes the permutation matrix P for the reordering of the matrix A. The elements of the permutation matrix P are zero except for the unit elements in row i and column $\mathbf{perm}[i-1]$, $i = 1, 2, \ldots n$.

8: **info**[**4**] – Integer *Output*

On exit: statistics about the matrix A and the permuted matrix. The quantities below are calculated using any masking in effect otherwise the value zero is returned.

info[0]
The bandwidth of the matrix A, if **lopts**[3] = Nag_TRUE.

info[1]
The profile of the matrix A, if **lopts**[3] = Nag_TRUE.

info[2]
The bandwidth of the permuted matrix PAP^{T}, if **lopts**[4] = Nag_TRUE.

info[3]
The profile of the permuted matrix PAP^{T}, if **lopts**[4] = Nag_TRUE.

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** ≥ 1.

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, **icolzp**[0] = ⟨*value*⟩.
Constraint: **icolzp**[0] = 1.

On entry, **icolzp**[**n**] = ⟨*value*⟩ and **nnz** = ⟨*value*⟩.
Constraint: **icolzp**[**n**] = **nnz** + 1.

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_NONSYMM_MATRIX

On entry, the matrix A is not symmetric.
Element (⟨*value*⟩, ⟨*value*⟩) has no symmetric element.

NE_SPARSE_COL

On entry, **icolzp**[⟨*value*⟩] = ⟨*value*⟩ and **nnz** = ⟨*value*⟩.
Constraint: 1 ≤ **icolzp**[$i-1$] ≤ **nnz** for all i.

NE_SPARSE_ROW

On entry, **irowix**[⟨*value*⟩] = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: 1 ≤ **irowix**[$i-1$] ≤ **n** for all i.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_sparse_sym_rcm (f11yec) is not threaded in any implementation.

9 Further Comments

The bandwidth for a matrix $A = \left(a_{ij}\right)$ is defined as

$$b = \max_{ij} |i - j|, \quad i, j = 1, 2, \ldots, n \text{ s.t. } a_{ij} \neq 0.$$

The profile is defined as

$$p = \sum_{j=1}^{n} b_j, \quad \text{where } b_j = \max_{i} |i - j|, \quad i = 1, 2, \ldots n \text{ s.t. } a_{ij} \neq 0.$$

10 Example

This example reads the CCS representation of a real sparse matrix A and calls nag_sparse_sym_rcm (f11yec) to reorder the rows and columns and displays the results.

10.1 Program Text

```
/* nag_sparse_sym_rcm (f11yec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf11.h>

void plot(const Integer n, const Integer nnz, Integer *perm,
          Integer *icolzp, Integer *irowix);
void uncompress(Integer n, Integer *icolzp, Integer *icol);

int main(void)
{
  /* Scalars */
  Integer n, nnz, exit_status = 0, doplot = 0, i;
  /* Arrays */
  Integer *icolzp = 0, *irowix = 0, *mask = 0, *perm = 0;
  Integer info[4];
  Nag_Boolean lopts[5];
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);
  printf("nag_sparse_sym_rcm (f11yec) Example Program Results\n");
  /* Skip heading in data file and
   * read Size of the matrix and Number of nonzero elements
   */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
  scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
  scanf_s("%" NAG_IFMT "%*[^\n] ", &nnz);
#else
  scanf("%*[^\n] ");
```

```
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
  scanf("%" NAG_IFMT "%*[^\n] ", &nnz);
#endif
  if (!(icolzp = NAG_ALLOC((n + 1), Integer)) ||
      !(irowix = NAG_ALLOC((nnz), Integer)) ||
      !(mask = NAG_ALLOC((n), Integer)) || !(perm = NAG_ALLOC((n), Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  /* Read in data */
#ifdef _WIN32
  for (i = 0; i < nnz; i += 1)
    scanf_s("%" NAG_IFMT "", &irowix[i]);
  scanf_s("%*[^\n] ");
  for (i = 0; i < (n + 1); i += 1)
    scanf_s("%" NAG_IFMT "", &icolzp[i]);
  scanf_s("%*[^\n] ");
#else
  for (i = 0; i < nnz; i += 1)
    scanf("%" NAG_IFMT "", &irowix[i]);
  scanf("%*[^\n] ");
  for (i = 0; i < (n + 1); i += 1)
    scanf("%" NAG_IFMT "", &icolzp[i]);
  scanf("%*[^\n] ");
#endif
  /* Set options */
  lopts[0 /* Use Mask                */ ] = Nag_FALSE;
  lopts[1 /* Don't reverse           */ ] = Nag_FALSE;
  lopts[2 /* Check symmetry          */ ] = Nag_TRUE;
  lopts[3 /* Compute bandwidth before */ ] = Nag_TRUE;
  lopts[4 /* Compute bandwidth after  */ ] = Nag_TRUE;
  /* nag_sparse_sym_rcm (f11yec).
   * Reverse Cuthill-McKee reordering of a real sparse symmetric
   * matrix in CCS format
   */
  nag_sparse_sym_rcm(n, nnz, icolzp, irowix, lopts, mask, perm, info, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_sym_rcm (f11yec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Print results */
  printf("Permutation (perm):\n");
  for (i = 0; i < n; i += 1) {
    printf("    %3" NAG_IFMT "", perm[i]);
    if (i % 6 == 5)
      printf("\n");
  }
  printf("\n\nStatistics:\n");
  printf(" %s%6" NAG_IFMT "\n", " Before:  Bandwidth = ", info[0]);
  printf(" %s%6" NAG_IFMT "\n", " Before:  Profile   = ", info[1]);
  printf(" %s%6" NAG_IFMT "\n", " After :  Bandwidth = ", info[2]);
  printf(" %s%6" NAG_IFMT "\n", " After :  Profile   = ", info[3]);
  /* Print matrix entries and permuted entries in form suitable
   * for plotting
   */
  if (doplot)
    plot(n, nnz, perm, icolzp, irowix);
END:
  NAG_FREE(icolzp);
  NAG_FREE(irowix);
  NAG_FREE(mask);
  NAG_FREE(perm);

  return exit_status;
}

void uncompress(Integer n, Integer *icolzp, Integer *icol)
{
```

```
  Integer i, j, col_beg, col_end;
  for (i = 0; i < n; i++) {
    col_end = icolzp[i + 1] - 1;
    col_beg = icolzp[i];
    for (j = col_beg; j <= col_end; j++)
      icol[j - 1] = i + 1;
  }

}

void plot(const Integer n, const Integer nnz, Integer *perm,
          Integer *icolzp, Integer *irowix)
{
  /* Put data, suitable for plotting matrix structure, in data file */

  /* Scalars */
  Integer i, nnz2;
  /* Arrays */
  double *a = 0;
  Integer *icolix = 0, *ipcolix = 0, *iperm = 0, *iprowix = 0, *istr = 0;
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);
  if (!(icolix = NAG_ALLOC(nnz, Integer)) ||
      !(ipcolix = NAG_ALLOC(nnz, Integer)) ||
      !(iprowix = NAG_ALLOC(nnz, Integer)) ||
      !(iperm = NAG_ALLOC(n, Integer)) ||
      !(a = NAG_ALLOC(nnz, double)) || !(istr = NAG_ALLOC(n + 1, Integer)))
  {
    printf("Allocation failure\n");
    return;
  }
  /* Decompress icolzp to full set of column indices (icolix)
   * and compute inverse permutation
   */
  uncompress(n, icolzp, icolix);
  for (i = 0; i < n; i++) {
    iperm[perm[i] - 1] = i + 1;
  }
  /* Original matrix structure */
  for (i = 0; i < nnz; i++) {
    a[i] = icolix[i] * .01 + 1.0 * irowix[i];
    printf("%8" NAG_IFMT "    ", irowix[i]);
    printf("%8" NAG_IFMT "    ", icolix[i]);
    printf("%8.2f\n", a[i]);
  }
  printf("\n");
  /* Apply Inverse Permutation */
  for (i = 0; i < nnz; i++) {
    ipcolix[i] = iperm[icolix[i] - 1];
    iprowix[i] = iperm[irowix[i] - 1];
  }
  /* Reorder (in exit: istr contains new CCS icolzp) */
  nnz2 = nnz;
  nag_sparse_nsym_sort(n, &nnz2, a, ipcolix, iprowix, Nag_SparseNsym_FailDups,
                       Nag_SparseNsym_KeepZeros, istr, &fail);
  /* Permuted matrix structure */
  for (i = 0; i < nnz2; i++) {
    printf("%8" NAG_IFMT "    ", iprowix[i]);
    printf("%8" NAG_IFMT "    ", ipcolix[i]);
    printf("%8.2f\n", a[i]);
  }
  NAG_FREE(icolix);
  NAG_FREE(ipcolix);
  NAG_FREE(iprowix);
  NAG_FREE(iperm);
  NAG_FREE(a);
  NAG_FREE(istr);
}
```

10.2 Program Data

```
nag_sparse_sym_rcm (f11yec) Example Program Data
60       n
180      nnz

  2   5   6   1   3  11   2   4  16
  3   5  21   1   4  26   1   7  10
  6   8  30   7   9  42   8  10  38
  6   9  12   2  12  15  10  11  13
 12  14  37  13  15  33  11  14  17
  3  17  20  15  16  18  17  19  32
 18  20  53  16  19  22   4  22  25
 20  21  23  22  24  52  23  25  48
 21  24  27   5  27  30  25  26  28
 27  29  47  28  30  43   7  26  29
 32  35  54  18  31  33  14  32  34
 33  35  36  31  34  56  34  37  40
 13  36  38   9  37  39  38  40  41
 36  39  57  39  42  45   8  41  43
 29  42  44  43  45  46  41  44  58
 44  47  50  28  46  48  24  47  49
 48  50  51  46  49  59  49  52  55
 23  51  53  19  52  54  31  53  55
 51  54  60  35  57  60  40  56  58
 45  57  59  50  58  60  55  56  59   irowix

  1   4   7  10  13  16
 19  22  25  28  31  34
 37  40  43  46  49  52
 55  58  61  64  67  70
 73  76  79  82  85  88
 91  94  97 100 103 106
109 112 115 118 121 124
127 130 133 136 139 142
145 148 151 154 157 160
163 166 169 172 175 178
181                       icolzp
```

10.3 Program Results

```
nag_sparse_sym_rcm (f11yec) Example Program Results
Permutation (perm):
     1     5     2     6     4     3
    26     7    30    11    12    10
    21    16    27    25     8    29
    17    15    13     9    22    20
    28    24    42    43    18    14
    37    38    23    19    47    48
    41    44    32    33    36    39
    52    53    46    49    45    31
    34    40    51    54    50    58
    35    57    55    59    56    60


Statistics:
  Before:  Bandwidth =    34
  Before:  Profile   =   490
  After :  Bandwidth =    10
  After :  Profile   =   458
```

Example Program

Figure 1 : Original Matrix Ordering

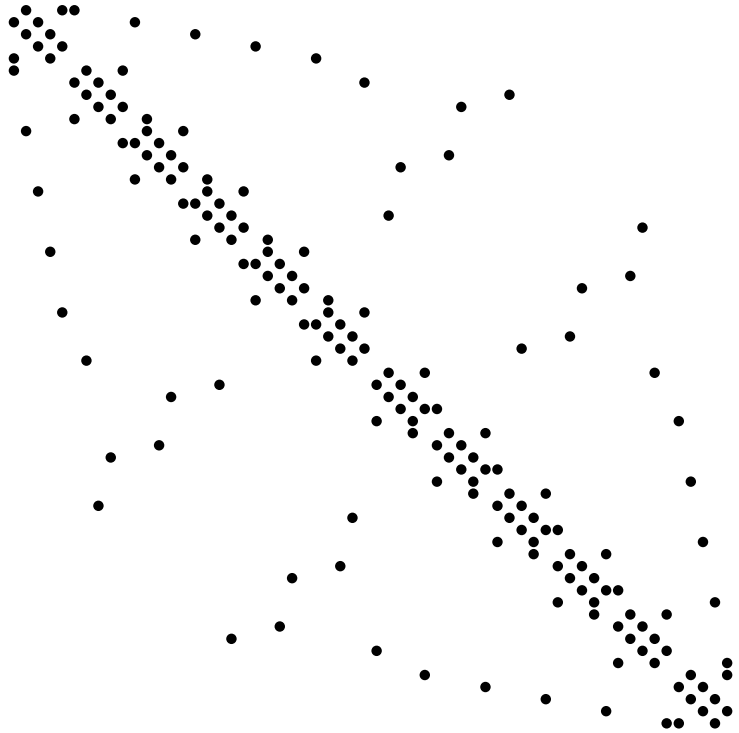

Figure 2 : Reverse Cuthill-McKee Reordering