NAG Library Function Document

nag_sparse_sym_matvec (f11xec)

1 Purpose

nag_sparse_sym_matvec (f11xec) computes a matrix-vector product involving a real sparse symmetric matrix stored in symmetric coordinate storage format.

2 Specification

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

void nag_sparse_sym_matvec (Integer n, Integer nnz, const double a[],
     const Integer irow[], const Integer icol[],
     Nag_SparseSym_CheckData check, const double x[], double y[],
     NagError *fail)
```

3 Description

nag_sparse_sym_matvec (f11xec) computes the matrix-vector product

$$y = Ax$$

where A is an n by n symmetric sparse matrix, of arbitrary sparsity pattern, stored in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the f11 Chapter Introduction). The array **a** stores all nonzero elements in the lower triangular part of A, while arrays **irow** and **icol** store the corresponding row and column indices respectively.

It is envisaged that a common use of nag_sparse_sym_matvec (f11xec) will be to compute the matrix-vector product required in the application of nag_sparse_sym_basic_solver (f11gec) to sparse symmetric linear systems. An illustration of this usage appears in nag_sparse_sym_precon_ssor_solve (f11jdc).

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*

On entry: the number of nonzero elements in the lower triangular part of A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n} + 1)/2$.

3: **a**[**nnz**] – const double *Input*

On entry: the nonzero elements in the lower triangular part of the matrix A, ordered by increasing row index, and by increasing column index within each row. Multiple entries for the same row and column indices are not permitted. The function nag_sparse_sym_sort (f11zbc) may be used to order the elements in this way.

4: **irow**[**nnz**] – const Integer *Input*
5: **icol**[**nnz**] – const Integer *Input*

On entry: the row and column indices of the nonzero elements supplied in array **a**.

Constraints:

irow and **icol** must satisfy these constraints (which may be imposed by a call to nag_sparse_sym_sort (f11zbc)):

$1 \le \textbf{irow}[i] \le \textbf{n}$ and $1 \le \textbf{icol}[i] \le \textbf{irow}[i]$, for $i = 0, 1, \ldots, \textbf{nnz} - 1$;
$\textbf{irow}[i-1] < \textbf{irow}[i]$ or $\textbf{irow}[i-1] = \textbf{irow}[i]$ and $\textbf{icol}[i-1] < \textbf{icol}[i]$, for $i = 1, 2, \ldots, \textbf{nnz} - 1$.

6: **check** – Nag_SparseSym_CheckData *Input*

On entry: specifies whether or not the SCS representation of the matrix A, values of **n**, **nnz**, **irow** and **icol** should be checked.

check = Nag_SparseSym_Check
Checks are carried out on the values of **n**, **nnz**, **irow** and **icol**.

check = Nag_SparseSym_NoCheck
None of these checks are carried out.

See also Section 9.2.

Constraint: **check** = Nag_SparseSym_Check or Nag_SparseSym_NoCheck.

7: **x**[**n**] – const double *Input*

On entry: the vector x.

8: **y**[**n**] – double *Output*

On exit: the vector y.

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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\textbf{n} \ge 1$.

On entry, **nnz** = ⟨*value*⟩.
Constraint: $\textbf{nnz} \ge 1$.

NE_INT_2

On entry, **nnz** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: $\textbf{nnz} \le \textbf{n} \times (\textbf{n} + 1)/2$.

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 3.6.6 in the Essential Introduction for further information.

NE_INVALID_SCS

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

On entry, $i = \langle value \rangle$, **irow**$[i-1] = \langle value \rangle$ and **n** $= \langle value \rangle$.
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 3.6.5 in the Essential Introduction for further information.

NE_NOT_STRICTLY_INCREASING

On entry, **a**$[i-1]$ is out of order: $i = \langle value \rangle$.

On entry, the location (**irow**$[I-1]$, **icol**$[I-1]$) is a duplicate: $I = \langle value \rangle$. Consider calling nag_sparse_sym_sort (f11zbc) to reorder and sum or remove duplicates.

7 Accuracy

The computed vector y satisfies the error bound

$$\|y - Ax\|_\infty \leq c(n)\epsilon\|A\|_\infty\|x\|_\infty,$$

where $c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_sparse_sym_matvec (f11xec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sparse_sym_matvec (f11xec) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

9.1 Timing

The time taken for a call to nag_sparse_sym_matvec (f11xec) is proportional to **nnz**.

9.2 Use of check

It is expected that a common use of nag_sparse_sym_matvec (f11xec) will be to compute the matrix-vector product required in the application of nag_sparse_sym_basic_solver (f11gec) to sparse symmetric linear systems. In this situation nag_sparse_sym_matvec (f11xec) is likely to be called many times with the same matrix A. In the interests of both reliability and efficiency you are recommended to set **check** = Nag_SparseSym_Check for the first of such calls, and to set **check** = Nag_SparseSym_NoCheck for all subsequent calls.

10 Example

This example reads in a symmetric positive definite sparse matrix A and a vector x. It then calls nag_sparse_sym_matvec (f11xec) to compute the matrix-vector product $y = Ax$.

10.1 Program Text

```
/* nag_sparse_sym_matvec (f11xec) Example Program.
 *
 * Copyright 2014 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, j, n, nnz;
  /* Arrays */
  char                   nag_enum_arg[40];
  Integer                *irow = 0, *icol = 0;
  double                 *a = 0, *x = 0, *y = 0;
  /* NAG types */
  NagError               fail;
  Nag_SparseSym_CheckData check;

  INIT_FAIL(fail);

  printf("nag_sparse_sym_matvec (f11xec) Example Program Results\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 */
  if ( !(a = NAG_ALLOC(nnz, double)) ||
       !(x = NAG_ALLOC(n, double)) ||
       !(y = NAG_ALLOC(n, double)) ||
       !(icol = NAG_ALLOC(nnz, Integer)) ||
       !(irow = NAG_ALLOC(nnz, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the matrix A */
  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

  /* Read the vector x */
  for (j = 0; j < n; j++)
#ifdef _WIN32
    scanf_s("%lf", &x[j]);
#else
    scanf("%lf", &x[j]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Set matrix to be checked */
  /* Nag_SparseSym_Check */
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  check = (Nag_SparseSym_CheckData) nag_enum_name_to_value (nag_enum_arg);

  /* nag_sparse_sym_matvec (f11xec)
   * Real sparse symmetric matrix vector multiply.
   */
  nag_sparse_sym_matvec(n, nnz, a, irow, icol, check, x, y, &fail);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_sym_matvec (f11xec)\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Output results */
  printf(" Matrix-vector product\n");
  for (j = 0; j < n; j++)
    printf("%16.4e\n", y[j]);

 END:
  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(icol);
  NAG_FREE(irow);

  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_sym_matvec (f11xec) Example Program Data
  9                    : n
 23                    : nnz
  4.    1     1
 -1.    2     1
  6.    2     2
  1.    3     2
  2.    3     3
  3.    4     4
  2.    5     1
  4.    5     5
  1.    6     3
  2.    6     4
  6.    6     6
 -4.    7     2
```

```
  1.    7     5
 -1.    7     6
  6.    7     7
 -1.    8     4
 -1.    8     6
  3.    8     8
  1.    9     1
  1.    9     5
 -1.    9     6
  1.    9     8
  4.    9     9       : (a, irow, icol)[i], i=0,...,nnz-1
  0.70  0.16  0.52
  0.77  0.28  0.21
  0.93  0.20  0.90    : x[i], i=1,...,n-1
 Nag_SparseSym_Check  : check
```

10.3 Program Results

```
nag_sparse_sym_matvec (f11xec) Example Program Results
 Matrix-vector product
      4.1000e+00
     -2.9400e+00
      1.4100e+00
      2.5300e+00
      4.3500e+00
      1.2900e+00
      5.0100e+00
      5.2000e-01
      4.5700e+00
```