NAG Library Function Document

nag_matop_real_symm_matrix_fun (f01efc)

1 Purpose

nag_matop_real_symm_matrix_fun (f01efc) computes the matrix function, $f(A)$, of a real symmetric n by n matrix A. $f(A)$ must also be a real symmetric matrix.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_matop_real_symm_matrix_fun (Nag_OrderType order, Nag_UploType uplo,
     Integer n, double a[], Integer pda,
     void (*f)(Integer *flag, Integer n, const double x[], double fx[],
          Nag_Comm *comm),
     Nag_Comm *comm, Integer *flag, NagError *fail)
```

3 Description

$f(A)$ is computed using a spectral factorization of A

$$A = QDQ^{\mathrm{T}},$$

where D is the diagonal matrix whose diagonal elements, d_i, are the eigenvalues of A, and Q is an orthogonal matrix whose columns are the eigenvectors of A. $f(A)$ is then given by

$$f(A) = Qf(D)Q^{\mathrm{T}},$$

where $f(D)$ is the diagonal matrix whose ith diagonal element is $f(d_i)$. See for example Section 4.5 of Higham (2008). $f(d_i)$ is assumed to be real.

4 References

Higham N J (2008) *Functions of Matrices: Theory and Computation* SIAM, Philadelphia, PA, USA

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **uplo** – Nag_UploType *Input*

On entry: if **uplo** = Nag_Upper, the upper triangle of the matrix A is stored.

If **uplo** = Nag_Lower, the lower triangle of the matrix A is stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

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

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

4: **a**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **a** must be at least $\mathbf{pda} \times \mathbf{n}$.

On entry: the n by n symmetric matrix A.

If **order** = Nag_ColMajor, A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = Nag_RowMajor, A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

If **uplo** = Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = Nag_Lower, the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

On exit: if **fail.code** = NE_NOERROR, the upper or lower triangular part of the n by n matrix function, $f(A)$.

5: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix A in the array **a**.

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

6: **f** – function, supplied by the user *External Function*

The function **f** evaluates $f(z_i)$ at a number of points z_i.

The specification of **f** is:

```
void f (Integer *flag, Integer n, const double x[], double fx[],
     Nag_Comm *comm)
```

1: **flag** – Integer * *Input/Output*

On entry: **flag** will be zero.

On exit: **flag** should either be unchanged from its entry value of zero, or may be set nonzero to indicate that there is a problem in evaluating the function $f(x)$; for instance $f(x)$ may not be defined, or may be complex. If **flag** is returned as nonzero then nag_matop_real_symm_matrix_fun (f01efc) will terminate the computation, with **fail.code** = NE_USER_STOP.

2: **n** – Integer *Input*

On entry: n, the number of function values required.

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

On entry: the n points $x_1, x_2, \ldots, x_n$ at which the function f is to be evaluated.

4: **fx**[**n**] – double *Output*

On exit: the n function values. **fx**[$i-1$] should return the value $f(x_i)$, for $i = 1, 2, \ldots, n$.

5: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **f**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_matop_real_symm_matrix_fun (f01efc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_matop_real_symm_matrix_fun (f01efc) (see Section 3.2.1.1 in the Essential Introduction).

7: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

8: **flag** – Integer * *Output*

On exit: **flag** $= 0$, unless you have set **flag** nonzero inside **f**, in which case **flag** will be the value you set and **fail** will be set to **fail.code** = NE_USER_STOP.

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_CONVERGENCE

The computation of the spectral factorization failed to converge.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pda** $\geq$ **n**.

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 internal error occurred when computing the spectral factorization. Please contact NAG.

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_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_USER_STOP

flag was set to a nonzero value in **f**.

7 Accuracy

Provided that $f(D)$ can be computed accurately then the computed matrix function will be close to the exact matrix function. See Section 10.2 of Higham (2008) for details and further discussion.

8 Parallelism and Performance

nag_matop_real_symm_matrix_fun (f01efc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_real_symm_matrix_fun (f01efc) 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

The Integer allocatable memory required is **n**, and the double allocatable memory required is approximately $(\mathbf{n} + nb + 4) \times \mathbf{n}$, where nb is the block size required by nag_dsyev (f08fac).

The cost of the algorithm is $O(n^3)$ plus the cost of evaluating $f(D)$. If $\hat{\lambda}_i$ is the ith computed eigenvalue of A, then the user-supplied function **f** will be asked to evaluate the function f at $f\left(\hat{\lambda}_i\right)$, $i = 1, 2, \ldots, n$.

For further information on matrix functions, see Higham (2008).

nag_matop_complex_herm_matrix_fun (f01ffc) can be used to find the matrix function $f(A)$ for a complex Hermitian matrix A.

10 Example

This example finds the matrix cosine, $\cos(A)$, of the symmetric matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_real_symm_matrix_fun (f01efc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL f(Integer *flag, Integer n, const double x[],
                       double fx[], Nag_Comm *comm);
```

```
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer        exit_status = 0;
  double         k = 1.0;
  Integer        i, flag, j, n, pda;

  /* Arrays */
  char           uplo_c[40];
  Integer        iuser[1];
  double         user[1];
  double         *a = 0;
  char           *outfile = 0;

  /* NAG types */
  Nag_UploType   uplo;
  Nag_MatrixType matrix;
  Nag_Comm       comm;
  Nag_OrderType  order;
  NagError       fail;

  INIT_FAIL(fail);

  /* Communicate constant k and initialize function counter through comm */
  comm.user = user;
  comm.iuser = iuser;
  user[0] = k;
  iuser[0] = 0;

  printf("nag_matop_real_symm_matrix_fun (f01efc) Example Program Results");
  printf("\n\n");
  fflush(stdout);

  /* Read problem parameters from data file*/
#ifdef _WIN32
   scanf_s("%*[^\n]%"NAG_IFMT"%*[^\n]  %39s%*[^\n]  ", &n, uplo_c, _countof(u-
plo_c));
#else
  scanf("%*[^\n]%"NAG_IFMT"%*[^\n] %39s%*[^\n] ", &n, uplo_c);
#endif

  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(uplo_c);

  pda = n;
  if (!(a = NAG_ALLOC((pda)*(n), double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I-1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J-1]
  order = Nag_RowMajor;
#endif

  /* Read A from data file */
  if (uplo == Nag_Upper)
    {
      matrix = Nag_UpperMatrix;
      for (i = 1; i <= n; i++)
```

```
#ifdef _WIN32
        for (j = i; j <= n; j++) scanf_s("%lf", &A(i, j));
#else
        for (j = i; j <= n; j++) scanf("%lf", &A(i, j));
#endif
    }
  else
    {
      matrix = Nag_LowerMatrix;
      for (i = 1; i <= n; i++)
#ifdef _WIN32
        for (j = 1; j <= i; j++) scanf_s("%lf", &A(i, j));
#else
        for (j = 1; j <= i; j++) scanf("%lf", &A(i, j));
#endif
    }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* nag_matop_real_symm_matrix_fun (f01efc).
   * Function of a real symmetric matrix
   */
  nag_matop_real_symm_matrix_fun(order, uplo, n, a, pda, f, &comm, &flag,
                                 &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_matop_real_symm_matrix_fun (f01efc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
  /* Expected number of function evaluations is n. */
  if (iuser[0] != n)
    printf("\nNumber of function evaluations = %"NAG_IFMT"\n\n", iuser[0]);

  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  nag_gen_real_mat_print(order, matrix, Nag_NonUnitDiag, n, n, a, pda,
                         "Symmetric f(A)=cos(kA)", outfile, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("%s\n", fail.message);
      exit_status = 2;
      goto END;
    }

 END:
  NAG_FREE(a);

  return exit_status;
}

static void NAG_CALL f(Integer *flag, Integer n, const double x[], double fx[],
                       Nag_Comm *comm)
{
  Integer j;
  double  k;
  if (!comm->user[0])
    {
      *flag = -1;
    }
  else
    {
      k = comm->user[0];

      for (j = 0; j < n; j++)
        {
```

```
            comm->iuser[0]++;
            fx[j] = cos(k*x[j]);
        }
    }
}
```

10.2 Program Data

```
nag_matop_real_symm_matrix_fun (f01efc) Example Program Data

  4                       : n
  Nag_Upper               : uplo

  1.0   2.0   3.0   4.0
        1.0   2.0   3.0
              1.0   2.0
                    1.0   : A
```

10.3 Program Results

```
nag_matop_real_symm_matrix_fun (f01efc) Example Program Results

 Symmetric f(A)=cos(kA)
          1       2       3       4
 1  -0.5420 -0.6612 -0.0261  0.1580
 2           0.2306 -0.3396 -0.0261
 3                   0.2306 -0.6612
 4                          -0.5420
```
