

NAG Library Function Document

nag_matop_real_gen_matrix_actexp (f01gac)

1 Purpose

nag_matop_real_gen_matrix_actexp (f01gac) computes the action of the matrix exponential e^{tA} , on the matrix B , where A is a real n by n matrix, B is a real n by m matrix and t is a real scalar.

2 Specification

```
#include <nag.h>
#include <nagf01.h>

void nag_matop_real_gen_matrix_actexp (Integer n, Integer m, double a[],
    Integer pda, double b[], Integer pdb, double t, NagError *fail)
```

3 Description

$e^{tA}B$ is computed using the algorithm described in Al-Mohy and Higham (2011) which uses a truncated Taylor series to compute the product $e^{tA}B$ without explicitly forming e^{tA} .

4 References

Al-Mohy A H and Higham N J (2011) Computing the action of the matrix exponential, with an application to exponential integrators *SIAM J. Sci. Statist. Comput.* **33(2)** 488-511

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

5 Arguments

- 1: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.
- 2: **m** – Integer *Input*
On entry: m , the number of columns of the matrix B .
Constraint: $m \geq 0$.
- 3: **a**[*dim*] – double *Input/Output*
Note: the dimension, *dim*, of the array **a** must be at least $pda \times n$.
The (i, j) th element of the matrix A is stored in **a**[$(j - 1) \times pda + i - 1$].
On entry: the n by n matrix A .
On exit: A is overwritten during the computation.
- 4: **pda** – Integer *Input*
On entry: the stride separating matrix row elements in the array **a**.
Constraint: $pda \geq n$.

- 5: **b**[*dim*] – double *Input/Output*
Note: the dimension, *dim*, of the array **b** must be at least **pdb** × **m**.
The (*i*, *j*)th element of the matrix *B* is stored in **b**[(*j* – 1) × **pdb** + *i* – 1].
On entry: the *n* by *m* matrix *B*.
On exit: the *n* by *m* matrix $e^{tA}B$.
- 6: **pdb** – Integer *Input*
On entry: the stride separating matrix row elements in the array **b**.
Constraint: **pdb** ≥ **n**.
- 7: **t** – double *Input*
On entry: the scalar *t*.
- 8: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Allocation of memory failed.

NE_BAD_PARAM

On entry, argument *<value>* had an illegal value.

NE_INT

On entry, **m** = *<value>*.

Constraint: **m** ≥ 0.

On entry, **n** = *<value>*.

Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pda** = *<value>* and **n** = *<value>*.

Constraint: **pda** ≥ **n**.

On entry, **pdb** = *<value>* and **n** = *<value>*.

Constraint: **pdb** ≥ **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.

NW_SOME_PRECISION_LOSS

$e^{tA}B$ has been computed using an IEEE double precision Taylor series, although the arithmetic precision is higher than IEEE double precision.

7 Accuracy

For a symmetric matrix *A* (for which $A^T = A$) the computed matrix $e^{tA}B$ is guaranteed to be close to the exact matrix, that is, the method is forward stable. No such guarantee can be given for non-symmetric matrices. See Section 4 of Al-Mohy and Higham (2011) for details and further discussion.

8 Parallelism and Performance

`nag_matop_real_gen_matrix_actexp` (f01gac) is not threaded by NAG in any implementation.

`nag_matop_real_gen_matrix_actexp` (f01gac) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The matrix $e^{tA}B$ could be computed by explicitly forming e^{tA} using `nag_real_gen_matrix_exp` (f01ecc) and multiplying B by the result. However, experiments show that it is usually both more accurate and quicker to use `nag_matop_real_gen_matrix_actexp` (f01gac).

The cost of the algorithm is $O(n^2m)$. The precise cost depends on A since a combination of balancing, shifting and scaling is used prior to the Taylor series evaluation.

Approximately $n^2 + (2m + 8)n$ of real allocatable memory is required by `nag_matop_real_gen_matrix_actexp` (f01gac).

`nag_matop_complex_gen_matrix_actexp` (f01hac) can be used to compute $e^{tA}B$ for complex A , B , and t . `nag_matop_real_gen_matrix_actexp_rcomm` (f01gbc) provides an implementation of the algorithm with a reverse communication interface, which returns control to the user when matrix multiplications are required. This should be used if A is large and sparse.

10 Example

This example computes $e^{tA}B$, where

$$A = \begin{pmatrix} 0.7 & -0.2 & 1.0 & 0.3 \\ 0.3 & 0.7 & 1.2 & 1.0 \\ 0.9 & 0.0 & 0.2 & 0.7 \\ 2.4 & 0.1 & 0.0 & 0.2 \end{pmatrix},$$

$$B = \begin{pmatrix} 0.1 & 1.2 \\ 1.3 & 0.2 \\ 0.0 & 1.0 \\ 0.4 & -0.9 \end{pmatrix}$$

and

$$t = 1.2.$$

10.1 Program Text

```
/* nag_matop_real_gen_matrix_actexp (f01gac) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
```

```

/* Scalars */
Integer      exit_status = 0;
Integer      i, j, m, n, lda, ldb;
double       t;

/* Arrays */
double       *a = 0;
double       *b = 0;

/* Nag Types */
Nag_OrderType order;
Nag_Error    fail;

INIT_FAIL(fail);

#define A(I, J) a[(J-1)*lda + I-1]
#define B(I, J) b[(J-1)*ldb + I-1]

order = Nag_ColMajor;

/* Output preamble */
printf("nag_matop_real_gen_matrix_actexp (f01gac) ");
printf("Example Program Results\n\n");
fflush(stdout);

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

/* Read in the problem size and the value of the parameter t */
scanf("%ld %ld %lf%*[\n]", &n, &m, &t);

lda = n;
ldb = n;

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

/* Read in the matrix a from data file */
for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++) scanf("%lf", &A(i, j));
scanf("%*[\n]");

/* Read in the matrix b from data file */
for (i = 1; i <= n; i++)
    for (j = 1; j <= m; j++) scanf("%lf", &B(i, j));
scanf("%*[\n]");

/* Find exp(tA) B using
 * nag_matop_real_gen_matrix_actexp (f01gac)
 * Action of the the exponential of a real matrix on a real matrix
 */
nag_matop_real_gen_matrix_actexp(n, m, a, lda, b, ldb, t, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_matop_real_gen_matrix_actexp (f01gac)\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution using
 * nag_gen_real_mat_print (x04cac)
 * Print real general matrix (easy-to-use)
 */
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
                      b, ldb, "exp(tA) B", NULL, &fail);
if (fail.code != NE_NOERROR)

```

```

    {
      printf("Error from nag_gen_real_mat_print (x04cac)\n%s\n", fail.message);
      exit_status = 2;
      goto END;
    }

END:
  NAG_FREE(a);
  NAG_FREE(b);
  return exit_status;
}

```

10.2 Program Data

nag_matop_real_gen_matrix_actexp (f01gac) Example Program Data

```

4      2      1.2           :Values of n, m and t

0.7   -0.2   1.0   0.3
0.3   0.7   1.2   1.0
0.9   0.0   0.2   0.7
2.4   0.1   0.0   0.2 :End of matrix a

0.1   1.2
1.3   0.2
0.0   1.0
0.4   -0.9           :End of matrix b

```

10.3 Program Results

nag_matop_real_gen_matrix_actexp (f01gac) Example Program Results

```

exp(tA) B
          1          2
1      0.2138      7.6756
2      4.9980     11.6051
3      0.8307      7.5468
4      1.2406      9.7261

```
