NAG Library Function Document

nag_matop_real_gen_matrix_actexp_rcomm (f01gbc)

1 Purpose

nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) 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. It uses reverse communication for evaluating matrix products, so that the matrix A is not accessed explicitly.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_matop_real_gen_matrix_actexp_rcomm (Integer *irevcm, Integer n,
     Integer m, double b[], Integer pdb, double t, double tr, double b2[],
     Integer pdb2, double x[], Integer pdx, double y[], Integer pdy,
     double p[], double r[], double z[], double comm[], Integer icomm[],
     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 $e^{tA}B$ without explicitly forming e^{tA}.

The algorithm does not explicity need to access the elements of A; it only requires the result of matrix multiplications of the form AX or $A^{\mathrm{T}}Y$. A reverse communication interface is used, in which control is returned to the calling program whenever a matrix product is required.

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

Note: this function uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than b2, x, y, p and r must remain unchanged**.

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

On initial entry: must be set to 0.

On intermediate exit: **irevcm** = 1, 2, 3, 4 or 5. The calling program must:

(a) if **irevcm** = 1: evaluate $B_2 = AB$, where B_2 is an n by m matrix, and store the result in **b2**;
if **irevcm** = 2: evaluate $Y = AX$, where X and Y are n by 2 matrices, and store the result in **y**;
if **irevcm** = 3: evaluate $X = A^{\mathrm{T}}Y$ and store the result in **x**;
if **irevcm** = 4: evaluate $p = Az$ and store the result in **p**;
if **irevcm** = 5: evaluate $r = A^{\mathrm{T}}z$ and store the result in **r**.

(b) call nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) again with all other parameters unchanged.

On final exit: **irevcm** $= 0$.

2: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

3: **m** – Integer *Input*

On entry: the number of columns of the matrix B.

Constraint: **m** ≥ 0.

4: **b**$[dim]$ – double *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least **pdb** $\times$ **m**.

The (i,j)th element of the matrix B is stored in **b**$[(j-1) \times$ **pdb** $+ i - 1]$.

On initial entry: the n by m matrix B.

On intermediate exit: if **irevcm** $= 1$, contains the n by m matrix B.

On intermediate re-entry: must not be changed.

On final exit: the n by m matrix $e^{tA}B$.

5: **pdb** – Integer *Input*

On entry: the stride separating matrix row elements in the array **b**.

Constraint: **pdb** $\geq$ **n**.

6: **t** – double *Input*

On entry: the scalar t.

7: **tr** – double *Input*

On entry: the trace of A. If this is not available then any number can be supplied (0 is a reasonable default); however, in the trivial case, $n = 1$, the result $e^{\mathbf{tr}t}B$ is immediately returned in the first row of B. See Section 9.

8: **b2**$[dim]$ – double *Input/Output*

Note: the dimension, *dim*, of the array **b2** must be at least **pdb2** $\times$ **m**.

The (i,j)th element of the matrix is stored in **b2**$[(j-1) \times$ **pdb2** $+ i - 1]$.

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 1$, must contain AB.

On final exit: the array is undefined.

9: **pdb2** – Integer *Input*

On entry: the stride separating matrix row elements in the array **b2**.

Constraint: **pdb2** $\geq$ **n**.

10: **x**$[dim]$ – double *Input/Output*

Note: the dimension, *dim*, of the array **x** must be at least **pdx** $\times 2$.

The (i,j)th element of the matrix X is stored in **x**$[(j-1) \times$ **pdx** $+ i - 1]$.

On initial entry: need not be set.

On intermediate exit: if **irevcm** $= 2$, contains the current n by 2 matrix X.

On intermediate re-entry: if **irevcm** $= 3$, must contain $A^{\mathrm{T}}Y$.

On final exit: the array is undefined.

11: **pdx** – Integer *Input*

On entry: the stride separating matrix row elements in the array **x**.

Constraint: **pdx** $\geq$ **n**.

12: **y**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **y** must be at least **pdy** $\times$ 2.

The (i,j)th element of the matrix Y is stored in **y**[$(j-1) \times$ **pdy** $+ i - 1$].

On initial entry: need not be set.

On intermediate exit: if **irevcm** $= 3$, contains the current n by 2 matrix Y.

On intermediate re-entry: if **irevcm** $= 2$, must contain AX.

On final exit: the array is undefined.

13: **pdy** – Integer *Input*

On entry: the stride separating matrix row elements in the array **y**.

Constraint: **pdy** $\geq$ **n**.

14: **p**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 4$, must contain Az.

On final exit: the array is undefined.

15: **r**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 5$, must contain $A^{\mathrm{T}}z$.

On final exit: the array is undefined.

16: **z**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate exit: if **irevcm** $= 4$ or 5, contains the vector z.

On intermediate re-entry: must not be changed.

On final exit: the array is undefined.

17: **comm**[**n** $\times$ **m** $+$ **3** $\times$ **n** $+$ **12**] – double *Communication Array*

18: **icomm**[**2** $\times$ **n** $+$ **40**] – Integer *Communication Array*

19: **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 3.2.1.2 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, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

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

On initial entry, **irevcm** = ⟨*value*⟩.
Constraint: **irevcm** = 0.

On intermediate re-entry, **irevcm** = ⟨*value*⟩.
Constraint: **irevcm** = 1, 2, 3, 4 or 5.

NE_INT_2

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

On entry, **pdb2** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb2** $\geq$ **n**.

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

On entry, **pdy** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdy** $\geq$ **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 unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation 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 How to Use the NAG Library and its Documentation for further information.

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 $A^{\mathrm{T}} = A$) 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_rcomm (f01gbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

9.1 Use of $Tr(A)$

The elements of A are not explicitly required by nag_matop_real_gen_matrix_actexp_rcomm (f01gbc). However, the trace of A is used in the preprocessing phase of the algorithm. If $Tr(A)$ is not available to the calling function then any number can be supplied (0 is recommended). This will not affect the stability of the algorithm, but it may reduce its efficiency.

9.2 When to use nag_matop_real_gen_matrix_actexp_rcomm (f01gbc)

nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) is designed to be used when A is large and sparse. Whenever a matrix multiplication is required, the function will return control to the calling program so that the multiplication can be done in the most efficient way possible. Note that $e^{tA}B$ will not, in general, be sparse even if A is sparse.

If A is small and dense then nag_matop_real_gen_matrix_actexp (f01gac) can be used to compute $e^{tA}B$ without the use of a reverse communication interface.

The complex analog of nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) is nag_matop_complex_gen_matrix_actexp_rcomm (f01hbc).

9.3 Use in Conjunction with NAG C Library Functions

To compute $e^{tA}B$, the following skeleton code can normally be used:

```
do {
 f01gbc(&irevcm,n,m,b,tdb,t,tr,b2,tdb2,x,tdx,y,tdy,p,r,z,comm,icomm,&fail);
   if (irevcm == 1) {
     .. Code to compute B2=AB ..
   }
   else if (irevcm == 2){
     .. Code to compute Y=AX ..
   }
   else if (irevcm == 3){
     .. Code to compute X=A^T Y ..
   }
   else if (irevcm == 4){
     .. Code to compute P=AZ ..
   }
   else if (irevcm == 5){
     .. Code to compute R=A^T Z ..
   }
 } (while irevcm !=0)
```

The code used to compute the matrix products will vary depending on the way A is stored. If all the elements of A are stored explicitly, then nag_dgemm (f16yac)) can be used. If A is triangular then nag_dtrmm (f16yfc) should be used. If A is symmetric, then nag_dsymm (f16ycc) should be used. For sparse A stored in coordinate storage format nag_sparse_nsym_matvec (f11xac) and nag_sparse_sym_matvec (f11xec) can be used. Alternatively if A is stored in compressed column format nag_superlu_matrix_product (f11mkc) can be used.

10 Example

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

$$A = \begin{pmatrix} 0.4 & -0.2 & 1.3 & 0.6 \\ 0.3 & 0.8 & 1.0 & 1.0 \\ 3.0 & 4.8 & 0.2 & 0.7 \\ 0.5 & 0.0 & -5.0 & 0.7 \end{pmatrix},$$

$$B = \begin{pmatrix} 0.1 & 1.1 \\ 1.7 & -0.2 \\ 0.5 & 1.0 \\ 0.4 & -0.2 \end{pmatrix},$$

and

$$t = -0.2.$$

10.1 Program Text

```
/* nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagf11.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{

  /* Scalars */
  Integer exit_status = 0, irevcm = 0;
  Integer i, j, m, n, nnz, ldb, ldb2, ldx, ldy;
  double one = 1.0, zero = 0.0;
  double t, tr;

  /* Arrays */
  double *a = 0, *b = 0, *b2 = 0, *comm = 0;
  double *p = 0, *r = 0, *x = 0, *y = 0, *z = 0;
  Integer *icolzp = 0, *irowix = 0, *icomm = 0;

  /* Nag Types */
  NagError fail, fail2;
  Nag_OrderType order = Nag_ColMajor;
  Nag_TransType tran = Nag_Trans, notran = Nag_NoTrans;

  INIT_FAIL(fail);
  INIT_FAIL(fail2);

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

  /* Output preamble */
  printf("nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
#ifdef _WIN32
```

```
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read in the problem size and the value of the parameter t */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT " %" NAG_IFMT " %lf %*[^\n] ", &n, &m, &t);
#else
  scanf("%" NAG_IFMT " %" NAG_IFMT " %lf %*[^\n] ", &n, &m, &t);
#endif

  ldb = ldb2 = ldx = ldy = n;

  if (!(b = NAG_ALLOC(n * m, double)) ||
      !(b2 = NAG_ALLOC(n * m, double)) ||
      !(comm = NAG_ALLOC(n * m + 3 * n + 12, double)) ||
      !(p = NAG_ALLOC(n, double)) ||
      !(r = NAG_ALLOC(n, double)) ||
      !(x = NAG_ALLOC(n * 2, double)) ||
      !(y = NAG_ALLOC(n * 2, double)) ||
      !(z = NAG_ALLOC(n, double)) ||
      !(icolzp = NAG_ALLOC(n + 1, Integer)) ||
      !(icomm = NAG_ALLOC(2 * n + 40, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read in the sparse matrix a in compressed column format */
  for (i = 0; i < n + 1; ++i)
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &icolzp[i]);
#else
    scanf("%" NAG_IFMT "", &icolzp[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  nnz = icolzp[n] - 1;

  if (!(a = NAG_ALLOC(nnz, double)) || !(irowix = NAG_ALLOC(nnz, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -2;
    goto END;
  }

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

  /* Read in the matrix b from data file */
  for (i = 1; i <= n; i++)
#ifdef _WIN32
    for (j = 1; j <= m; j++)
      scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= m; j++)
      scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
```

```
  scanf("%*[^\n]");
#endif

  /* Compute the trace of A */
  tr = 0.0;
  j = 1;
  for (i = 0; i < nnz; ++i) {
    /* new column? */
    if (icolzp[j] <= i + 1)
      j++;
    /* diagonal element? */
    if (irowix[i] == j)
      tr += a[i];
  }

  /* Find exp(tA) B using
   * nag_matop_real_gen_matrix_actexp_rcomm (f01gbc):
   *     Action of the exponential of a complex matrix on a complex matrix.
   *
   * Sparse matrix multiplies are performed using
   * nag_superlu_matrix_product (f11mkc):
   *     Real sparse nonsymmetric matrix matrix multiply, compressed column
   *     storage.
   */
  while (1) {
    nag_matop_real_gen_matrix_actexp_rcomm(&irevcm, n, m, b, ldb, t, tr, b2,
                                           ldb2, x, ldx, y, ldy, p, r, z,
                                           comm, icomm, &fail);
    switch (irevcm) {
    case 0:
      /* Final exit. */
      goto END_REVCM;
      break;
    case 1:
      /* Compute AB and store in B2 */
      nag_superlu_matrix_product(order, notran, n, m, one, icolzp, irowix, a,
                                 b, ldb, zero, b2, ldb2, &fail2);
      break;
    case 2:
      /* Compute AX and store in Y */
      nag_superlu_matrix_product(order, notran, n, 2, one, icolzp, irowix, a,
                                 x, ldx, zero, y, ldy, &fail2);
    case 3:
      /* Compute A^T Y and store in X */
      nag_superlu_matrix_product(order, tran, n, 2, one, icolzp, irowix, a, y,
                                 ldy, zero, x, ldx, &fail2);
    case 4:
      /* Compute AZ and store in P */
      nag_superlu_matrix_product(order, notran, n, 1, one, icolzp, irowix, a,
                                 z, n, zero, p, n, &fail2);
    case 5:
      /* Compute A^T Z and store in R */
      nag_superlu_matrix_product(order, tran, n, 1, one, icolzp, irowix, a, z,
                                 n, zero, r, n, &fail2);
    }
    if (fail2.code != NE_NOERROR) {
      printf("Error from nag_superlu_matrix_product (f11mkc).\n%s\n",
             fail2.message);
      exit_status = 1;
      goto END;
    }
  }
END_REVCM:

  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_real_gen_matrix_actexp_rcomm (f01gbc)\n%s\n",
           fail.message);
    exit_status = 2;
    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 = 3;
    goto END;
  }

END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(b2);
  NAG_FREE(p);
  NAG_FREE(r);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(z);
  NAG_FREE(comm);
  NAG_FREE(icolzp);
  NAG_FREE(irowix);
  NAG_FREE(icomm);
  return exit_status;
}
```

10.2 Program Data

```
nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) Example Program Data

 4      2     -0.2                : n, m, and t

 1      5      8      12      16  : icolzp

 0.4    1
 0.3    2
 3.0    3
 0.5    4
-0.2    1
 0.8    2
 4.8    3
 1.3    1
 1.0    2
 0.2    3
-5.0    4
 0.6    1
 1.0    2
 0.7    3
 0.7    4                        : (a[i], irowix[i]) i = 0, nnz-1

0.1    1.1
1.7   -0.2
0.5    1.0
0.4   -0.2                      : matrix b
```

10.3 Program Results

```
nag_matop_real_gen_matrix_actexp_rcomm (f01gbc) Example Program Results

 exp(tA) B
              1          2
 1      0.1933     0.7812
 2      1.4423    -0.4055
 3     -1.0756     0.6686
 4      0.0276     0.4900
```