NAG Toolbox

nag_matop_real_gen_matrix_actexp_rcomm (f01gb)

1 Purpose

nag_matop_real_gen_matrix_actexp_rcomm (f01gb) 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 Syntax

```
[irevcm, b, b2, x, y, p, r, z, comm, icomm, ifail] =
nag_matop_real_gen_matrix_actexp_rcomm(irevcm, m, b, t, b2, x, y, p, r, z, comm,
icomm, 'n', n, 'tr', tr)
[irevcm, b, b2, x, y, p, r, z, comm, icomm, ifail] = f01gb(irevcm, m, b, t, b2,
x, y, p, r, z, comm, icomm, 'n', n, 'tr', tr)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 25: **m** was made optional.

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 Parameters

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**.

5.1 Compulsory Input Parameters

1: **irevcm** – INTEGER

On initial entry: must be set to 0.

2: **m** – INTEGER

The number of columns of the matrix B.

Constraint: **m** ≥ 0.

3: **b**$(ldb,:)$ – REAL (KIND=nag_wp) array
The first dimension of the array **b** must be at least **n**.
The second dimension of the array **b** must be at least **m**.
On initial entry: the n by m matrix B.
On intermediate re-entry: must not be changed.

4: **t** – REAL (KIND=nag_wp)
The scalar t.

5: **b2**$(ldb2,:)$ – REAL (KIND=nag_wp) array
The first dimension of the array **b2** must be at least **n**.
The second dimension of the array **b2** must be at least **m**.
On initial entry: need not be set.
On intermediate re-entry: if **irevcm** $= 1$, must contain AB.

6: **x**$(ldx,:)$ – REAL (KIND=nag_wp) array
The first dimension of the array **x** must be at least **n**.
The second dimension of the array **x** must be at least 2.
On initial entry: need not be set.
On intermediate re-entry: if **irevcm** $= 3$, must contain $A^{\mathrm{T}}Y$.

7: **y**$(ldy,:)$ – REAL (KIND=nag_wp) array
The first dimension of the array **y** must be at least **n**.
The second dimension of the array **y** must be at least 2.
On initial entry: need not be set.
On intermediate re-entry: if **irevcm** $= 2$, must contain AX.

8: **p**(**n**) – REAL (KIND=nag_wp) array
On initial entry: need not be set.
On intermediate re-entry: if **irevcm** $= 4$, must contain Az.

9: **r**(**n**) – REAL (KIND=nag_wp) array
On initial entry: need not be set.
On intermediate re-entry: if **irevcm** $= 5$, must contain $A^{\mathrm{T}}z$.

10: **z**(**n**) – REAL (KIND=nag_wp) array
On initial entry: need not be set.
On intermediate re-entry: must not be changed.

11: **comm**($\mathbf{n} \times \mathbf{m} + \mathbf{3} \times \mathbf{n} + \mathbf{12}$) – REAL (KIND=nag_wp) array

12: **icomm**($\mathbf{2} \times \mathbf{n} + \mathbf{40}$) – INTEGER array

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the first dimension of the arrays **b**, **b2**, **x**, **y**, **comm** and the dimension of the arrays **icomm**, **p**, **r**, **z**. (An error is raised if these dimensions are not equal.)

n, the order of the matrix A.

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

2: **tr** – REAL (KIND=nag_wp)

Default: 0

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.

5.3 Output Parameters

1: **irevcm** – INTEGER

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 (f01gb) again with all other parameters unchanged.

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

2: **b**($ldb, :$) – REAL (KIND=nag_wp) array

The first dimension of the array **b** will be **n**.

The second dimension of the array **b** will be **m**.

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

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

3: **b2**($ldb2, :$) – REAL (KIND=nag_wp) array

The first dimension of the array **b2** will be **n**.

The second dimension of the array **b2** will be **m**.

On final exit: the array is undefined.

4: **x**($ldx, :$) – REAL (KIND=nag_wp) array

The first dimension of the array **x** will be **n**.

The second dimension of the array **x** will be 2.

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

On final exit: the array is undefined.

5: **y**$(ldy, :)$ – REAL (KIND=nag_wp) array

The first dimension of the array **y** will be **n**.

The second dimension of the array **y** will be 2.

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

On final exit: the array is undefined.

6: **p**$(\mathbf{n})$ – REAL (KIND=nag_wp) array

On final exit: the array is undefined.

7: **r**$(\mathbf{n})$ – REAL (KIND=nag_wp) array

On final exit: the array is undefined.

8: **z**$(\mathbf{n})$ – REAL (KIND=nag_wp) array

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

On final exit: the array is undefined.

9: **comm**$(\mathbf{n} \times \mathbf{m} + \mathbf{3} \times \mathbf{n} + \mathbf{12})$ – REAL (KIND=nag_wp) array

10: **icomm**$(\mathbf{2} \times \mathbf{n} + \mathbf{40})$ – INTEGER array

11: **ifail** – INTEGER

On final exit: **ifail** $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 2$ (*warning*)

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

ifail $= -1$

On initial entry, **irevcm** $= \langle value \rangle$.
Constraint: **irevcm** $= 0$.

On intermediate re-entry, **irevcm** $= \langle value \rangle$.
Constraint: **irevcm** $= 1$, 2, 3, 4 or 5.

ifail $= -2$

On initial entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

ifail $= -3$

On initial entry, **m** $= \langle value \rangle$.
Constraint: **m** ≥ 0.

ifail $= -5$

On initial entry, $ldb = \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: $ldb \geq$ **n**.

ifail $= -9$

On initial entry, $ldb2 = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $ldb2 \geq \mathbf{n}$.

ifail $= -11$

On initial entry, $ldx = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $ldx \geq \mathbf{n}$.

ifail $= -13$

On initial entry, $ldy = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $ldy \geq \mathbf{n}$.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

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 Further Comments

8.1 Use of $Tr(A)$

The elements of A are not explicitly required by nag_matop_real_gen_matrix_actexp_rcomm (f01gb). 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.

8.2 When to use nag_matop_real_gen_matrix_actexp_rcomm (f01gb)

nag_matop_real_gen_matrix_actexp_rcomm (f01gb) 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 (f01ga) 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 (f01gb) is nag_matop_complex_gen_matrix_actexp_rcomm (f01hb).

8.3 Use in Conjunction with NAG Toolbox Functions

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

```
while irevcm ~= 0  switch irevcm    case {1}      % Compute ab and store in b2
case {2}      % Compute ax and store in y    case {3}      % Compute a'y and
store in x    case {4}      % compute az and store in p    otherwise      %
compute a'z and store in r  end
```

9 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

$$t = -0.2.$$

9.1 Program Text

```
function f01gb_example

fprintf('f01gb example results\n\n');

% A and B
a = [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];
b = [0.1,  1.1;
     1.7, -0.2;
     0.5,  1.0;
     0.4, -0.2];
n = 4;
m = 2;

% Initial input parameters
t = -0.2;
b2 = b;  x = b;   y = b;
p = zeros(n, 1);  r = p;  z = p;
comm =  zeros(n*m+3*n+12, 1);
icomm = zeros(2*n+40, 1, nag_int_name);
tr = trace(a);

% Compute exp(ta)b by reverse communication
irevcm = nag_int(0);
done = false;

while ~done
  [irevcm, b, b2, x, y, p, r, z, comm, icomm, ifail] = ...
  f01gb( ...
         irevcm, b, t, b2, x, y, p, r, z, comm, icomm, 'tr', tr);

  switch irevcm
    case {0}
      done = true;
    case {1}
      % Compute ab and store in b2
      b2 = a*b;
    case {2}
      % Compute ax and store in y
      y = a*x;
    case {3}
      % Compute a'y and store in x
      x = a'*y;
    case {4}
      % compute az and store in p
      p = a*z;
    otherwise
```

```
      % compute a'z and store in r
      r = a'*z;
  end
end

fprintf('exp(tA)B\n');
disp(b);
```

9.2 Program Results

```
     f01gb example results

exp(tA)B
    0.1933    0.7812
    1.4423   -0.4055
   -1.0756    0.6686
    0.0276    0.4900
```