NAG Library Function Document

nag_matop_real_gen_matrix_fun_std (f01ekc)

1 Purpose

nag_matop_real_gen_matrix_fun_std (f01ekc) computes the matrix exponential, sine, cosine, sinh or cosh, of a real \( n \) by \( n \) matrix \( A \) using the Schur–Parlett algorithm.

2 Specification

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

void nag_matop_real_gen_matrix_fun_std (Nag_OrderType order,
                                       Nag_MatFunType fun, Integer n, double a[], Integer pda, double *imnorm,
                                       NagError *fail)
```

3 Description

\( f(A) \), where \( f \) is either the exponential, sine, cosine, sinh or cosh, is computed using the Schur–Parlett algorithm described in Higham (2008) and Davies and Higham (2003).

4 References


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: 
   *fun* – Nag_MatFunType
   
   *Input*
   
   On entry: indicates which matrix function will be computed.
   
   \( f = \) Nag_Exp
   
   The matrix exponential, \( e^A \), will be computed.
   
   \( f = \) Nag_Sin
   
   The matrix sine, \( \sin(A) \), will be computed.
   
   \( f = \) Nag_Cos
   
   The matrix cosine, \( \cos(A) \), will be computed.
   
   \( f = \) Nag_Sinh
   
   The hyperbolic matrix sine, \( \sinh(A) \), will be computed.
   
   \( f = \) Nag_Cosh
   
   The hyperbolic matrix cosine, \( \cosh(A) \), will be computed.
   
   Constraint: *fun* = Nag_Exp, Nag_Sin, Nag_Cos, Nag_Sinh or Nag_Cosh.
f01ekc

3:  n – Integer
    
    *Input*
    
    On entry: \(n\), the order of the matrix \(A\).
    
    Constraint: \(n \geq 0\).

4:  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
    
    \[ a[(j-1) \times pda + i - 1] \text{ when } order = \text{Nag\_ColMajor}; \]
    \[ a[(i-1) \times pda + j - 1] \text{ when } order = \text{Nag\_RowMajor}. \]
    
    On entry: the \(n\) by \(n\) matrix \(A\).
    
    On exit: the \(n\) by \(n\) matrix, \(f(A)\).

5:  pda – Integer
    
    *Input*
    
    On entry: the stride separating row or column elements (depending on the value of \(order\)) in the array \(a\).
    
    Constraint: \(pda \geq n\).

6:  imnorm – double *
    
    *Output*
    
    On exit: if \(A\) has complex eigenvalues, \text{nag\_matop\_real\_gen\_matrix\_fun\_std\ (f01ekc)} will use complex arithmetic to compute the matrix function. The imaginary part is discarded at the end of the computation, because it will theoretically vanish. \(imnorm\) contains the 1-norm of the imaginary part, which should be used to check that the routine has given a reliable answer.
    
    If \(A\) has real eigenvalues, \text{nag\_matop\_real\_gen\_matrix\_fun\_std\ (f01ekc)} uses real arithmetic and \(imnorm = 0\).

7:  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 \(\langle\text{value}\rangle\) had an illegal value.

NE_CONVERGENCE
    
    A Taylor series failed to converge.

NE_INT
    
    On entry, \(n = \langle\text{value}\rangle\).
    
    Constraint: \(n \geq 0\).

NE_INT_2
    
    On entry, \(pda = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
    
    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 unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

An unexpected internal error occurred when evaluating the function at a point. Please contact NAG.

An unexpected internal error occurred when ordering the eigenvalues of $A$. Please contact NAG.

The function was unable to compute the Schur decomposition of $A$.

Note: this failure should not occur and suggests that the function has been called incorrectly.

There was an error whilst reordering the Schur form of $A$.

Note: this failure should not occur and suggests that the function has been called incorrectly.

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_SINGULAR

The linear equations to be solved are nearly singular and the Padé approximant used to compute the exponential may have no correct figures.

Note: this failure should not occur and suggests that the function has been called incorrectly.

7 Accuracy

For a normal matrix $A$ (for which $A^TA = AA^T$), the Schur decomposition is diagonal and the algorithm reduces to evaluating $f$ at the eigenvalues of $A$ and then constructing $f(A)$ using the Schur vectors. This should give a very accurate result. In general, however, no error bounds are available for the algorithm. For further discussion of the Schur–Parlett algorithm see Section 9.4 of Higham (2008).

8 Parallelism and Performance

nag_matop_real_gen_matrix_fun_std ($f01ekc$) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library. In these implementations, this function may make calls to the user-supplied functions from within an OpenMP parallel region. Thus OpenMP pragmas within the user functions can only be used if you are compiling the user-supplied function and linking the executable in accordance with the instructions in the Users’ Note for your implementation.

nag_matop_real_gen_matrix_fun_std ($f01ekc$) 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 $n$. If $A$ has real eigenvalues then up to $9n^2$ of double allocatable memory may be required. If $A$ has complex eigenvalues then up to $9n^2$ of Complex allocatable memory may be required.

The cost of the Schur–Parlett algorithm depends on the spectrum of $A$, but is roughly between $28n^3$ and $n^4/3$ floating-point operations; see Algorithm 9.6 of Higham (2008).
If the matrix exponential is required then it is recommended that nag_real_gen_matrix_exp (f01ecc) be used. nag_real_gen_matrix_exp (f01ecc) uses an algorithm which is, in general, more accurate than the Schur–Parlett algorithm used by nag_matop_real_gen_matrix_fun_std (f01ekc).

If estimates of the condition number of the matrix function are required then nag_matop_real_gen_matrix_cond_std (f01jac) should be used.

nag_matop_complex_gen_matrix_fun_std (f01fkc) can be used to find the matrix exponential, sin, cos, sinh or cosh of a complex matrix.

10 Example

This example finds the matrix cosine of the matrix

\[
A = \begin{pmatrix}
2 & 0 & 1 & 0 \\
0 & 2 & -2 & 1 \\
0 & 2 & 3 & 1 \\
1 & 4 & 0 & 0
\end{pmatrix}.
\]

10.1 Program Text

/* nag_matop_real_gen_matrix_fun_std (f01ekc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */
*
#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, n, pda;
    double imnorm;

    /* Arrays */
    double *a = 0;
    char nag_enum_arg[100];

    /* Nag Types */
    Nag_OrderType order;
    Nag_MatFunType fun;
    NagError fail;

    INIT_FAIL(fail);

    #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

    /* Output preamble */
    printf("nag_matop_real_gen_matrix_fun_std (f01ekc) ");
    printf("Example Program Results\n\n");
    fflush(stdout);

    /* Skip heading in data file */
```c
#ifndef _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif

/* Read in the problem size and the required function */
#ifndef _WIN32
    scanf_s("%"NAG_IFMT"%99s%*[\n"]", &n, nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%"NAG_IFMT"%99s%*[\n"]", &n, nag_enum_arg);
#endif

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

/* nag_enum_name_to_value (x04nac)
* Converts Nag enum member name to value
*/
fun = (Nag_MatFunType) nag_enum_name_to_value(nag_enum_arg);

/* Read in the matrix a from data file */
for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++) scanf_s("%lf", &A(i,j));
#endif

/* Find the matrix function using
* nag_matop_real_gen_matrix_fun_std (f01ekc)
* Real matrix function */
#undef _WIN32
    scanf_s("%*[\n"]);
#endif
    scanf("%*[\n"]);

nag_matop_real_gen_matrix_fun_std (order, fun, n, a, pda, &imnorm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_matop_real_gen_matrix_fun_std (f01ekc)\n\n", fail.message);
    exit_status = 1;
    goto END;
}

if (fabs(imnorm) > pow(nag_machine_precision,0.8)) {
    printf("WARNING: the error estimate returned in imnorm is larger than" "
expected;\n");
    printf(" the matrix solution printed below is suspect:\n");
    printf(" imnorm = %13.4e.\n", imnorm);
}

/* Print solution using
* nag_gen_real_mat_print (x04cac)
* Print real general matrix (easy-to-use) */
#undef _WIN32
    scanf("%*[\n"]);
#endif
    scanf("%*[\n"]);

nag_gen_real_mat_print (order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a, n, "f(A)", NULL, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac)\n\n", fail.message);
    exit_status = 2;
    goto END;
}
```

Mark 25

f01ec

f01 – Matrix Factorizations

f01ekc

Mark 25

f01ec.5
END:
   NAG_FREE(a);
   return exit_status;
}

10.2 Program Data

nag_matop_real_gen_matrix_fun_std (f01ekc) Example Program Data
4 Nag_Cos :Values of n and fun
2.0  0.0  1.0  0.0
0.0  2.0  -2.0  1.0
0.0  2.0   3.0  1.0
1.0  4.0  0.0  0.0 :End of matrix a

10.3 Program Results

nag_matop_real_gen_matrix_fun_std (f01ekc) Example Program Results

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.2998</td>
<td>1.5003</td>
<td>-0.7849</td>
<td>0.4677</td>
</tr>
<tr>
<td>2</td>
<td>-0.2385</td>
<td>-3.2657</td>
<td>0.5812</td>
<td>-1.1460</td>
</tr>
<tr>
<td>3</td>
<td>0.4677</td>
<td>0.3008</td>
<td>-4.0853</td>
<td>-0.2200</td>
</tr>
<tr>
<td>4</td>
<td>-0.2107</td>
<td>-2.8199</td>
<td>-1.2964</td>
<td>-0.8325</td>
</tr>
</tbody>
</table>