# NAG Library Function Document

# nag_matop_real_gen_matrix_cond_usd (f01jcc)

## 1 Purpose

nag_matop_real_gen_matrix_cond_usd (f01jcc) computes an estimate of the absolute condition number of a matrix function $f$ at a real $n$ by $n$ matrix $A$ in the 1-norm, using analytical derivatives of $f$ you have supplied.

## 2 Specification

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

```
void nag_matop_real_gen_matrix_cond_usd (Integer n, double a[], Integer pda,
     void (*f)(Integer m, Integer *iflag, Integer nz, const Complex z[],
          Complex fz[], Nag_Comm *comm),
     Nag_Comm *comm, Integer *iflag, double *conda, double *norma,
     double *normfa, NagError *fail)
```

## 3 Description

The absolute condition number of $f$ at $A$, $\text{cond}_{\text{abs}}(f, A)$ is given by the norm of the Fréchet derivative of $f$, $L(A)$, which is defined by

$$\|L(X)\| := \max{}_{E \neq 0} \frac{\|L(X, E)\|}{\|E\|},$$

where $L(X, E)$ is the Fréchet derivative in the direction $E$. $L(X, E)$ is linear in $E$ and can therefore be written as

$$\text{vec}(L(X, E)) = K(X)\text{vec}(E),$$

where the vec operator stacks the columns of a matrix into one vector, so that $K(X)$ is $n^2 \times n^2$. nag_matop_real_gen_matrix_cond_usd (f01jcc) computes an estimate $\gamma$ such that $\gamma \leq \|K(X)\|_1$, where $\|K(X)\|_1 \in \left[n^{-1}\|L(X)\|_1, n\|L(X)\|_1\right]$. The relative condition number can then be computed via

$$\text{cond}_{\text{rel}}(f, A) = \frac{\text{cond}_{\text{abs}}(f, A)\|A\|_1}{\|f(A)\|_1}.$$

The algorithm used to find $\gamma$ is detailed in Section 3.4 of Higham (2008).

The function $f$, and the derivatives of $f$, are returned by function **f** which, given an integer $m$, evaluates $f^{(m)}(z_i)$ at a number of (generally complex) points $z_i$, for $i = 1, 2, \ldots, n_z$. For any $z$ on the real line, $f(z)$ must also be real. nag_matop_real_gen_matrix_cond_usd (f01jcc) is therefore appropriate for functions that can be evaluated on the complex plane and whose derivatives, of arbitrary order, can also be evaluated on the complex plane.

## 4 References

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

2: **a**[$dim$] – double *Input/Output*

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

The $(i, j)$th element of the matrix $A$ is stored in $\mathbf{a}[(j - 1) \times \mathbf{pda} + i - 1]$.

*On entry*: the $n$ by $n$ matrix $A$.

*On exit*: the $n$ by $n$ matrix, $f(A)$.

3: **pda** – Integer *Input*

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

*Constraint*: $\mathbf{pda} \geq \mathbf{n}$.

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

Given an integer $m$, the function **f** evaluates $f^{(m)}(z_i)$ at a number of points $z_i$.

The specification of **f** is:

```
void f (Integer m, Integer *iflag, Integer nz, const Complex z[],
     Complex fz[], Nag_Comm *comm)
```

1: **m** – Integer *Input*

*On entry*: the order, $m$, of the derivative required.

If $\mathbf{m} = 0$, $f(z_i)$ should be returned. For $\mathbf{m} > 0$, $f^{(m)}(z_i)$ should be returned.

2: **iflag** – Integer * *Input/Output*

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

*On exit*: **iflag** 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 $f(z)$; for instance $f(z)$ may not be defined. If **iflag** is returned as nonzero then nag_matop_real_gen_matrix_cond_usd (f01jcc) will terminate the computation, with **fail.code** = NE_USER_STOP.

3: **nz** – Integer *Input*

*On entry*: $n_z$, the number of function or derivative values required.

4: **z**[**nz**] – const Complex *Input*

*On entry*: the $n_z$ points $z_1, z_2, \dots, z_{n_z}$ at which the function $f$ is to be evaluated.

5: **fz**[**nz**] – Complex *Output*

*On exit*: the $n_z$ function or derivative values. $\mathbf{fz}[i - 1]$ should return the value $f^{(m)}(z_i)$, for $i = 1, 2, \dots, n_z$. If $z_i$ lies on the real line, then so must $f^{(m)}(z_i)$.

6: **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_gen_matrix_cond_usd (f01jcc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_matop_real_gen_matrix_cond_usd (f01jcc) (see Section 3.2.1.1 in the Essential Introduction).

5: **comm** – Nag_Comm *

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

6: **iflag** – Integer * *Output*

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

7: **conda** – double * *Output*

*On exit*: an estimate of the absolute condition number of $f$ at $A$.

8: **norma** – double * *Output*

*On exit*: the 1-norm of $A$.

9: **normfa** – double * *Output*

*On exit*: the 1-norm of $f(A)$.

10: **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 $\langle value \rangle$ had an illegal value.

**NE_INT**

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

**NE_INT_2**

On entry, **pda** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
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 internal error occurred when estimating the norm of the Fréchet derivative of $f$ at $A$. Please contact NAG.

An internal error occurred when evaluating the matrix function $f(A)$. You can investigate further by calling nag_matop_real_gen_matrix_fun_usd (f01emc) with the matrix $A$ and the function $f$.

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**

**iflag** has been set nonzero by the user-supplied function.

## 7 Accuracy

nag_matop_real_gen_matrix_cond_usd (f01jcc) uses the norm estimation routine nag_linsys_real_gen_norm_rcomm (f04ydc) to estimate a quantity $\gamma$, where $\gamma \leq \|K(X)\|_1$ and $\|K(X)\|_1 \in \left[n^{-1}\|L(X)\|_1, n\|L(X)\|_1\right]$. For further details on the accuracy of norm estimation, see the documentation for nag_linsys_real_gen_norm_rcomm (f04ydc).

## 8 Parallelism and Performance

nag_matop_real_gen_matrix_cond_usd (f01jcc) is threaded by NAG for parallel execution in multithreaded 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. You must also ensure that you use the NAG communication argument **comm** in a thread safe manner, which is best achieved by only using it to supply read-only data to the user functions.

nag_matop_real_gen_matrix_cond_usd (f01jcc) 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 matrix function is computed using the underlying matrix function routine nag_matop_real_gen_matrix_fun_usd (f01emc). Approximately $6n^2$ of real allocatable memory is required by the routine, in addition to the memory used by the underlying matrix function routine.

If only $f(A)$ is required, without an estimate of the condition number, then it is far more efficient to use the underlying matrix function routine directly.

The complex analogue of this function is nag_matop_complex_gen_matrix_cond_usd (f01kcc).

## 10 Example

This example estimates the absolute and relative condition numbers of the matrix function $e^{2A}$ where

$$A = \begin{pmatrix} 0 & -1 & -1 & 1 \\ -2 & 0 & 1 & -1 \\ 2 & -1 & 2 & -2 \\ -1 & -2 & 0 & -1 \end{pmatrix}.$$

### 10.1 Program Text

```
/* nag_matop_real_gen_matrix_cond_usd (f01jcc) Example Program.
 *
 * Copyright 2014 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>

#ifdef __cplusplus
extern "C" {
#endif
   static void NAG_CALL f(Integer m, Integer *iflag, Integer nz,
                          const Complex z[], Complex fz[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

#define A(I,J) a[J*pda + I]

int main(void)
{

  /* Scalars */
  Integer        exit_status = 0;
  Integer        i, iflag, j, n, pda;
  double         conda, cond_rel, eps, norma, normfa;
  /* Arrays */
  static double ruser[1] = {-1.0};
  double         *a = 0;
  /* Nag Types */
  Nag_OrderType  order = Nag_ColMajor;
  Nag_Comm       comm;
  NagError       fail;

  INIT_FAIL(fail);

  /* Output preamble */
  printf("nag_matop_real_gen_matrix_cond_usd (f01jcc) ");
  printf("Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  fflush(stdout);

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

  /* Read in the problem size */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif

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

```
  }

  /* Read in the matrix A from data file */
  for (i = 0; i < n; i++)
#ifdef _WIN32
    for (j = 0; j < n; j++) scanf_s("%lf", &A(i, j));
#else
    for (j = 0; j < n; j++) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Print real general matrix A using the easy-to-use function
   * nag_gen_real_mat_print (x04cac).
   */
  nag_gen_real_mat_print (order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                          n, n, a, pda, "A", 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;
  }
  /* Find absolute condition number estimate of f(A) for real matrix A using
   * nag_matop_real_gen_matrix_cond_usd (f01jcc),
   * which requires user-supplied derivatives.
   */
  nag_matop_real_gen_matrix_cond_usd (n, a, pda, f, &comm, &iflag,
                                      &conda, &norma, &normfa, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_real_gen_matrix_cond_usd (f01jcc)\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print absolute condition number estimate */
  printf("\nf(A) = exp(2A)\n");
  printf("Estimated absolute condition number is: %7.2f\n",conda);

  /* nag_machine_precision (x02ajc)  The machine precision */
  eps = nag_machine_precision;

  /* Find relative condition number estimate */
  if ( normfa>eps) {
    cond_rel = conda * norma/normfa;
    printf("Estimated relative condition number is: %7.2f\n",cond_rel);
  }
  else {
    printf("The estimated norm of f(A) is effectively zero");
    printf("and so the relative condition number is undefined.\n");
  }

 END:
  NAG_FREE(a);
  return exit_status;
}

static void NAG_CALL f(Integer m, Integer *iflag, Integer nz,
                       const Complex z[], Complex fz[], Nag_Comm *comm)
{
  /* Scalars */
  Integer j;
#pragma omp master
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback f, first invocation.)\n");
      comm->user[0] = 0.0;
    }
```

```
  for (j = 0; j < nz; j++) {
    /* The m^th derivative of exp 2z for complex z*/
    fz[j].re = pow(2.0,m)*exp(2.0*z[j].re)*cos(2.0*z[j].im);
    fz[j].im = pow(2.0,m)*exp(2.0*z[j].re)*sin(2.0*z[j].im);
  }
  /* Set iflag nonzero to terminate execution for any reason. */
  *iflag = 0;
}
```

## 10.2 Program Data

```
nag_matop_real_gen_matrix_cond_usd (f01jcc) Example Program Data

 4                        :Value of n

 0.0   -1.0   -1.0    1.0
-2.0    0.0    1.0   -1.0
 2.0   -1.0    2.0   -2.0
-1.0   -2.0    0.0   -1.0  :End of matrix a
```

## 10.3 Program Results

```
nag_matop_real_gen_matrix_cond_usd (f01jcc) Example Program Results

 A
            1          2          3          4
 1      0.0000    -1.0000    -1.0000     1.0000
 2     -2.0000     0.0000     1.0000    -1.0000
 3      2.0000    -1.0000     2.0000    -2.0000
 4     -1.0000    -2.0000     0.0000    -1.0000
(User-supplied callback f, first invocation.)

f(A) = exp(2A)
Estimated absolute condition number is:  183.90
Estimated relative condition number is:   13.90
```