

NAG C Library Function Document

nag_ztbcon (f07vuc)

1 Purpose

nag_ztbcon (f07vuc) estimates the condition number of a complex triangular band matrix.

2 Specification

```
void nag_ztbcon (Nag_OrderType order, Nag_NormType norm, Nag_UploType uplo,
                 Nag_DiagType diag, Integer n, Integer kd, const Complex ab[], Integer pdab,
                 double *rcond, NagError *fail)
```

3 Description

nag_ztbcon (f07vuc) estimates the condition number of a complex triangular band matrix A , in either the 1-norm or the infinity-norm:

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad \text{or} \quad \kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty.$$

Note that $\kappa_\infty(A) = \kappa_1(A^T)$.

Because the condition number is infinite if A is singular, the function actually returns an estimate of the **reciprocal** of the condition number.

The function computes $\|A\|_1$ or $\|A\|_\infty$ exactly, and uses Higham's implementation of Hager's method (see Higham (1988)) to estimate $\|A^{-1}\|_1$ or $\|A^{-1}\|_\infty$.

4 References

Higham N J (1988) FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation *ACM Trans. Math. Software* **14** 381–396

5 Parameters

- 1: **order** – Nag_OrderType *Input*
On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.
Constraint: **order = Nag_RowMajor** or **Nag_ColMajor**.
- 2: **norm** – Nag_NormType *Input*
On entry: indicates whether $\kappa_1(A)$ or $\kappa_\infty(A)$ is estimated as follows:
 - if **norm = Nag_OneNorm**, $\kappa_1(A)$ is estimated;
 - if **norm = Nag_InfNorm**, $\kappa_\infty(A)$ is estimated.*Constraint:* **norm = Nag_OneNorm** or **Nag_InfNorm**.
- 3: **uplo** – Nag_UploType *Input*
On entry: indicates whether A is upper or lower triangular as follows:

if **uplo** = **Nag_Upper**, A is upper triangular;

if **uplo** = **Nag_Lower**, A is lower triangular.

Constraint: **uplo** = **Nag_Upper** or **Nag_Lower**.

4: **diag** – Nag_DiagType *Input*

On entry: indicates whether A is a non-unit or unit triangular matrix as follows:

if **diag** = **Nag_NonUnitDiag**, A is a non-unit triangular matrix;

if **diag** = **Nag_UnitDiag**, A is a unit triangular matrix; the diagonal elements are not referenced and are assumed to be 1.

Constraint: **diag** = **Nag_NonUnitDiag** or **Nag_UnitDiag**.

5: **n** – Integer *Input*

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

Constraint: $n \geq 0$.

6: **kd** – Integer *Input*

On entry: k , the number of super-diagonals of the matrix A if **uplo** = **Nag_Upper** or the number of sub-diagonals if **uplo** = **Nag_Lower**.

Constraint: $kd \geq 0$.

7: **ab**[*dim*] – const Complex *Input*

Note: the dimension, *dim*, of the array **ab** must be at least $\max(1, \mathbf{pdab} \times \mathbf{n})$.

On entry: the n by n triangular matrix A . This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements a_{ij} depends on the **order** and **uplo** parameters as follows:

if **order** = **Nag_ColMajor** and **uplo** = **Nag_Upper**,

a_{ij} is stored in **ab**[$k + i - j + (j - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = i, \dots, \min(n, i + k)$;

if **order** = **Nag_ColMajor** and **uplo** = **Nag_Lower**,

a_{ij} is stored in **ab**[$i - j + (j - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = \max(1, i - k), \dots, i$;

if **order** = **Nag_RowMajor** and **uplo** = **Nag_Upper**,

a_{ij} is stored in **ab**[$j - i + (i - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = i, \dots, \min(n, i + k)$;

if **order** = **Nag_RowMajor** and **uplo** = **Nag_Lower**,

a_{ij} is stored in **ab**[$k + j - i + (i - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = \max(1, i - k), \dots, i$.

8: **pdab** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix A in the array **ab**.

Constraint: $\mathbf{pdab} \geq \mathbf{kd} + 1$.

9: **rcond** – double * *Output*

On exit: an estimate of the reciprocal of the condition number of A . **rcond** is set to zero if exact singularity is detected or the estimate underflows. If **rcond** is less than *machine precision*, A is singular to working precision.

10: **fail** – NagError *

Output

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = $\langle value \rangle$.

Constraint: **n** ≥ 0 .

On entry, **kd** = $\langle value \rangle$.

Constraint: **kd** ≥ 0 .

On entry, **pdab** = $\langle value \rangle$.

Constraint: **pdab** > 0 .

NE_INT_2

On entry, **pdab** = $\langle value \rangle$, **kd** = $\langle value \rangle$.

Constraint: **pdab** $\geq \mathbf{kd} + 1$.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ had an illegal value.

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 consult NAG for assistance.

7 Accuracy

The computed estimate **rcond** is never less than the true value ρ , and in practice is nearly always less than 10ρ , although examples can be constructed where **rcond** is much larger.

8 Further Comments

A call to nag_ztbcon (f07vuc) involves solving a number of systems of linear equations of the form $Ax = b$ or $A^H x = b$; the number is usually 5 and never more than 11. Each solution involves approximately $8nk$ real floating-point operations (assuming $n \gg k$) but takes considerably longer than a call to nag_ztbtrs (f07vsc) with one right-hand side, because extra care is taken to avoid overflow when A is approximately singular.

The real analogue of this function is nag_dtbcon (f07vgc).

9 Example

To estimate the condition number in the 1-norm of the matrix A , where

$$A = \begin{pmatrix} -1.94 + 4.43i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\ -3.39 + 3.44i & 4.12 - 4.27i & 0.00 + 0.00i & 0.00 + 0.00i \\ 1.62 + 3.68i & -1.84 + 5.53i & 0.43 - 2.66i & 0.00 + 0.00i \\ 0.00 + 0.00i & -2.77 - 1.93i & 1.74 - 0.04i & 0.44 + 0.10i \end{pmatrix}.$$

Here A is treated as a lower triangular band matrix with 2 sub-diagonals. The true condition number in the 1-norm is 71.51.

9.1 Program Text

```

/* nag_ztbcon (f07vuc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx02.h>

int main(void)
{
    /* Scalars */
    Integer i, j, k, kd, n, pdab;
    Integer exit_status=0;
    double rcond;
    NagError fail;
    Nag_UploType uplo_enum;
    Nag_OrderType order;

    /* Arrays */
    char uplo[2];
    Complex *ab=0;

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I,J) ab[(J-1)*pdab + k + I - J - 1]
#define AB_LOWER(I,J) ab[(J-1)*pdab + I - J]
    order = Nag_ColMajor;
#else
#define AB_UPPER(I,J) ab[(I-1)*pdab + J - I]
#define AB_LOWER(I,J) ab[(I-1)*pdab + k + J - I - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);
    Vprintf("f07vuc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[\n] ");
    Vscanf("%ld%ld%*[\n] ", &n, &kd);
    pdab = kd + 1;

    /* Allocate memory */
    if ( !(ab = NAG_ALLOC((kd+1) * n, Complex)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read A from data file */
    Vscanf(" ' %1s '%*[\n] ", uplo);
    if (*(unsigned char *)uplo == 'L')
        uplo_enum = Nag_Lower;
    else if (*(unsigned char *)uplo == 'U')
        uplo_enum = Nag_Upper;
    else
    {
        Vprintf("Unrecognised character for Nag_UploType type\n");
        exit_status = -1;
        goto END;
    }
    k = kd + 1;
    if (uplo_enum == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)

```

```

        {
            for (j = i; j <= MIN(i+kd,n); ++j)
            {
                Vscanf(" ( %lf , %lf )", &AB_UPPER(i,j).re,
                        &AB_UPPER(i,j).im);
            }
        }
        Vscanf("%*[^\\n] ");
    }
else
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = MAX(1,i-kd); j <= i; ++j)
            {
                Vscanf(" ( %lf , %lf )", &AB_LOWER(i,j).re,
                        &AB_LOWER(i,j).im);
            }
        }
        Vscanf("%*[^\\n] ");
    }
/* Estimate condition number */
f07vuc(order, Nag_OneNorm, uplo_enum, Nag_NonUnitDiag, n,
       kd, ab, pdab, &rcond, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07vuc.\\n%s\\n", fail.message);
    exit_status = 1;
    goto END;
}
if (rcond >= X02AJC)
    Vprintf("Estimate of condition number =%10.2e\\n\\n", 1.0/rcond);
else
    Vprintf("A is singular to working precision\\n");
END:
if (ab) NAG_FREE(ab);
return exit_status;
}

```

9.2 Program Data

f07vuc Example Program Data

```

4 2                                     :Values of N and KD
'L'                                   :Value of UPLO
(-1.94, 4.43)
(-3.39, 3.44) ( 4.12,-4.27)
( 1.62, 3.68) (-1.84, 5.53) ( 0.43,-2.66)
                (-2.77,-1.93) ( 1.74,-0.04) ( 0.44, 0.10) :End of matrix A

```

9.3 Program Results

f07vuc Example Program Results

Estimate of condition number = 3.35e+01
