NAG Library Function Document

nag_ztbcon (f07vuc)

1 Purpose

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

2 Specification

```
#include <nag.h>
#include <nagf07.h>
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 ∞-norm:

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

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

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 $\left\|A^{-1}\right\|_1$ or $\left\|A^{-1}\right\|_\infty$.

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 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: **norm** – Nag_NormType *Input*

On entry: indicates whether $\kappa_1(A)$ or $\kappa_\infty(A)$ is estimated.

norm = Nag_OneNorm
$\kappa_1(A)$ is estimated.

norm = Nag_InfNorm
$\kappa_\infty(A)$ is estimated.

Constraint: **norm** = Nag_OneNorm or Nag_InfNorm.

3: **uplo** – Nag_UploType *Input*

On entry: specifies whether A is upper or lower triangular.

uplo = Nag_Upper
A is upper triangular.

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 nonunit or unit triangular matrix.

diag = Nag_NonUnitDiag
A is a nonunit triangular matrix.

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

6: **kd** – Integer *Input*

On entry: k_d, the number of superdiagonals of the matrix A if **uplo** = Nag_Upper, or the number of subdiagonals if **uplo** = Nag_Lower.

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

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 band matrix A.

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of A_{ij}, depends on the **order** and **uplo** arguments as follows:

if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in **ab**$[k_d + i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = \max(1, j - k_d), \ldots, j$;
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in **ab**$[i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, j + k_d)$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in **ab**$[j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_d)$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in **ab**$[k_d + j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_d), \ldots, i$.

If **diag** = 'Nag_UnitDiag', the diagonal elements of AB are assumed to be 1, and are not referenced.

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: **pdab** $\geq$ **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 * *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.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

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

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

On entry, **pdab** $=$ ⟨*value*⟩.
Constraint: **pdab** > 0.

NE_INT_2

On entry, **pdab** $=$ ⟨*value*⟩ and **kd** $=$ ⟨*value*⟩.
Constraint: **pdab** $\geq$ **kd** $+ 1$.

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.

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 Parallelism and Performance

nag_ztbcon (f07vuc) is not threaded by NAG in any implementation.

nag_ztbcon (f07vuc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

A call to nag_ztbcon (f07vuc) involves solving a number of systems of linear equations of the form $Ax = b$ or $A^{\mathrm{H}}x = b$; 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).

10 Example

This example estimates 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 two subdiagonals. The true condition number in the 1-norm is 71.51.

10.1 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;
  Nag_OrderType order;
  /* Arrays */
  char          nag_enum_arg[40];
  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);

  printf("nag_ztbcon (f07vuc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%*[^\n] ", &n, &kd);
```

```
  pdab = kd + 1;

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

  /* Read A from data file */
  scanf(" %39s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  k = kd + 1;
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= MIN(i+kd, n); ++j)
            {
              scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re,
                     &AB_UPPER(i, j).im);
            }
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = MAX(1, i-kd); j <= i; ++j)
            {
              scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re,
                     &AB_LOWER(i, j).im);
            }
        }
      scanf("%*[^\n] ");
    }
  /* Estimate condition number */
  /* nag_ztbcon (f07vuc).
   * Estimate condition number of complex band triangular
   * matrix
   */
  nag_ztbcon(order, Nag_OneNorm, uplo, Nag_NonUnitDiag, n,
             kd, ab, pdab, &rcond, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztbcon (f07vuc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* nag_machine_precision (x02ajc).
   * The machine precision
   */
  if (rcond >= nag_machine_precision)
    printf("Estimate of condition number =%11.2e\n\n", 1.0/rcond);
  else
    printf("A is singular to working precision\n");
 END:
  NAG_FREE(ab);
  return exit_status;
}
```

10.2 Program Data

```
nag_ztbcon (f07vuc) Example Program Data
  4  2                                                        :Values of n and kd
  Nag_Lower                                                   :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
```

10.3 Program Results

```
nag_ztbcon (f07vuc) Example Program Results

Estimate of condition number =   3.35e+01
```