NAG Library Function Document

nag_zgtcon (f07cuc)

1 Purpose

nag_zgtcon (f07cuc) estimates the reciprocal condition number of a complex n by n tridiagonal matrix A, using the LU factorization returned by nag_zgttrf (f07crc).

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_zgtcon (Nag_NormType norm, Integer n, const Complex dl[],
     const Complex d[], const Complex du[], const Complex du2[],
     const Integer ipiv[], double anorm, double *rcond, NagError *fail)
```

3 Description

nag_zgtcon (f07cuc) should be preceded by a call to nag_zgttrf (f07crc), which uses Gaussian elimination with partial pivoting and row interchanges to factorize the matrix A as

$$A = PLU,$$

where P is a permutation matrix, L is unit lower triangular with at most one nonzero subdiagonal element in each column, and U is an upper triangular band matrix, with two superdiagonals. nag_zgtcon (f07cuc) then utilizes the factorization to estimate either $\left\|A^{-1}\right\|_1$ or $\left\|A^{-1}\right\|_\infty$, from which the estimate of the reciprocal of the condition number of A, $1/\kappa(A)$ is computed as either

$$1/\kappa_1(A) = 1/\left(\|A\|_1\left\|A^{-1}\right\|_1\right)$$

or

$$1/\kappa_\infty(A) = 1/\left(\|A\|_\infty\left\|A^{-1}\right\|_\infty\right).$$

$1/\kappa(A)$ is returned, rather than $\kappa(A)$, since when A is singular $\kappa(A)$ is infinite.

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

4 References

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

5 Arguments

1: **norm** – Nag_NormType *Input*

On entry: specifies the norm to be used to estimate $\kappa(A)$.

norm = Nag_OneNorm
Estimate $\kappa_1(A)$.

norm = Nag_InfNorm
Estimate $\kappa_\infty(A)$.

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

2: **n** – Integer *Input*

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

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

3: **dl**[dim] – const Complex *Input*

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

On entry: must contain the $(n - 1)$ multipliers that define the matrix L of the LU factorization of A.

4: **d**[dim] – const Complex *Input*

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

On entry: must contain the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

5: **du**[dim] – const Complex *Input*

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

On entry: must contain the $(n - 1)$ elements of the first superdiagonal of U.

6: **du2**[dim] – const Complex *Input*

Note: the dimension, *dim*, of the array **du2** must be at least $\max(1, \mathbf{n} - 2)$.

On entry: must contain the $(n - 2)$ elements of the second superdiagonal of U.

7: **ipiv**[dim] – const Integer *Input*

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

On entry: must contain the n pivot indices that define the permutation matrix P. At the ith step, row i of the matrix was interchanged with row **ipiv**$[i - 1]$, and **ipiv**$[i - 1]$ must always be either i or $(i + 1)$, **ipiv**$[i - 1] = i$ indicating that a row interchange was not performed.

8: **anorm** – double *Input*

On entry: if **norm** = Nag_OneNorm, the 1-norm of the **original** matrix A.

If **norm** = Nag_InfNorm, the ∞-norm of the **original** matrix A.

anorm may be computed as demonstrated in Section 10 for the 1-norm. The ∞-norm may be similarly computed by swapping the **dl** and **du** arrays in the code for the 1-norm.

anorm must be computed either **before** calling nag_zgttrf (f07crc) or else from a **copy** of the original matrix A (see Section 10).

Constraint: **anorm** ≥ 0.0.

9: **rcond** – double * *Output*

On exit: contains an estimate of the reciprocal condition number.

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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

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.

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_REAL

On entry, **anorm** $= \langle value \rangle$.
Constraint: **anorm** ≥ 0.0.

7 Accuracy

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

8 Parallelism and Performance

nag_zgtcon (f07cuc) is not threaded by NAG in any implementation.

nag_zgtcon (f07cuc) 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 condition number estimation typically requires between four and five solves and never more than eleven solves, following the factorization. The total number of floating-point operations required to perform a solve is proportional to n.

The real analogue of this function is nag_dgtcon (f07cgc).

10 Example

This example estimates the condition number in the 1-norm of the tridiagonal matrix A given by

$$A = \begin{pmatrix} -1.3+1.3i & 2.0-1.0i & 0 & 0 & 0 \\ 1.0-2.0i & -1.3+1.3i & 2.0+1.0i & 0 & 0 \\ 0 & 1.0+1.0i & -1.3+3.3i & -1.0+1.0i & 0 \\ 0 & 0 & 2.0-3.0i & -0.3+4.3i & 1.0-1.0i \\ 0 & 0 & 0 & 1.0+1.0i & -3.3+1.3i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zgtcon (f07cuc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 *
 */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx02.h>

int main(void)
{
  /* Scalars */
  double   anorm, rcond;
  Integer  exit_status = 0, i, n;

  /* Arrays */
  Complex  *d = 0, *dl = 0, *du = 0, *du2 = 0;
  Integer  *ipiv = 0;

  /* Nag Types */
  NagError fail;

#define CABS(dl) sqrt(dl.re * dl.re + dl.im * dl.im)

  INIT_FAIL(fail);

  printf("nag_zgtcon (f07cuc) Example Program Results\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif
  if (n < 0)
    {
      printf("Invalid n\n");
      exit_status = -2;
      goto END;
    }
  /* Allocate memory */
  if (!(d = NAG_ALLOC(n, Complex)) ||
      !(dl = NAG_ALLOC(n-1, Complex)) ||
      !(du = NAG_ALLOC(n-1, Complex)) ||
      !(du2 = NAG_ALLOC(n-2, Complex)) ||
      !(ipiv = NAG_ALLOC(n, Integer)))
    {
```

```
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the tridiagonal matrix A from data file */
#ifdef _WIN32
  for (i = 0; i < n - 1; ++i) scanf_s(" ( %lf , %lf )", &du[i].re, &du[i].im);
#else
  for (i = 0; i < n - 1; ++i) scanf(" ( %lf , %lf )", &du[i].re, &du[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n; ++i) scanf_s(" ( %lf , %lf )", &d[i].re, &d[i].im);
#else
  for (i = 0; i < n; ++i) scanf(" ( %lf , %lf )", &d[i].re, &d[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n - 1; ++i) scanf_s(" ( %lf , %lf )", &dl[i].re, &dl[i].im);
#else
  for (i = 0; i < n - 1; ++i) scanf(" ( %lf , %lf )", &dl[i].re, &dl[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Compute the 1-norm of A */

  if (n == 1)
    {
      anorm = CABS(d[0]);
    }
  else
    {
      anorm = MAX(CABS(d[0])+CABS(dl[0]), CABS(d[n-1])+CABS(du[n-2]));
      for (i = 1; i < n-1; ++i)
        anorm = MAX(anorm, CABS(d[i])+CABS(dl[i])+CABS(du[i-1]));
    }
  /* nag_zgttrf (f07crc)
   * LU factorization of complex tridiagonal matrix A
   */
  nag_zgttrf(n, dl, d, du, du2, ipiv, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgttrf (f07crc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Estimate the condition number of A using
   * nag_zgtcon (f07cuc).
   * Estimates the reciprocal of the condition number of a LU factorized
   * complex tridiagonal matrix.
   */
  nag_zgtcon(Nag_OneNorm, n, dl, d, du, du2, ipiv, anorm, &rcond, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgtcon (f07cuc).\n%s\n", fail.message);
      exit_status = 2;
      goto END;
```

```
    }

  /* Print the estimated condition number */
  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. RCOND = %11.2e\n\n", rcond);
 END:
  NAG_FREE(d);
  NAG_FREE(dl);
  NAG_FREE(du);
  NAG_FREE(du2);
  NAG_FREE(ipiv);

  return exit_status;
}
```

10.2 Program Data

```
nag_zgtcon (f07cuc) Example Program Data
   5                                                          : n
            ( 2.0,-1.0) ( 2.0, 1.0) (-1.0, 1.0) ( 1.0,-1.0) : du
 (-1.3, 1.3) (-1.3, 1.3) (-1.3, 3.3) (-0.3, 4.3) (-3.3, 1.3) : d
 ( 1.0,-2.0) ( 1.0, 1.0) ( 2.0,-3.0) ( 1.0, 1.0)            : dl
```

10.3 Program Results

```
nag_zgtcon (f07cuc) Example Program Results

Estimate of condition number =    1.84e+02
```