1 Purpose
nag_ztrcon (f07tuc) estimates the condition number of a complex triangular matrix.

2 Specification
#include <nag.h>
#include <nagf07.h>
void nag_ztrcon (Nag_OrderType order, Nag_NormType norm, Nag_UploType uplo,
    Nag_DiagType diag, Integer n, const Complex a[], Integer pda,
    double *rcond, NagError *fail)

3 Description
nag_ztrcon (f07tuc) estimates the condition number of a complex triangular matrix \( A \), in either the 1-norm or the \( \infty \)-norm:

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

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 Arguments
1: \textbf{order} – Nag_OrderType  
\textit{Input}

\textit{On entry}: the \textit{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 \textit{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint}: \textit{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{norm} – Nag_NormType  
\textit{Input}

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

\textit{norm} = Nag_OneNorm  
\( \kappa_1(A) \) is estimated.

\textit{norm} = Nag_InfNorm  
\( \kappa_\infty(A) \) is estimated.

\textit{Constraint}: \textit{norm} = Nag_OneNorm or Nag_InfNorm.
3: \texttt{uplo} – Nag_UploType \textit{Input}

\textit{On entry:} specifies whether \(A\) is upper or lower triangular.

\texttt{uplo} = Nag_Upper
\hspace{1em}A is upper triangular.

\texttt{uplo} = Nag_Lower
\hspace{1em}A is lower triangular.

\textit{Constraint:} \texttt{uplo} = Nag_Upper or Nag_Lower.

4: \texttt{diag} – Nag_DiagType \textit{Input}

\textit{On entry:} indicates whether \(A\) is a nonunit or unit triangular matrix.

\texttt{diag} = Nag_NonUnitDiag
\hspace{1em}A is a nonunit triangular matrix.

\texttt{diag} = Nag_UnitDiag
\hspace{1em}A is a unit triangular matrix; the diagonal elements are not referenced and are assumed to be 1.

\textit{Constraint:} \texttt{diag} = Nag_NonUnitDiag or Nag_UnitDiag.

5: \texttt{n} – Integer \textit{Input}

\textit{On entry:} \(n\), the order of the matrix \(A\).

\textit{Constraint:} \(n \geq 0\).

6: \texttt{a[dim]} – const Complex \textit{Input}

\textit{Note:} the dimension, \(dim\), of the array \texttt{a} must be at least \(\max(1, \texttt{pda} \times n)\).

\textit{On entry:} the \(n\) by \(n\) triangular matrix \(A\).

If \texttt{order} = Nag_ColMajor, \(A_{ij}\) is stored in \(\texttt{a}[(j-1) \times \texttt{pda} + i - 1]\).

If \texttt{order} = Nag_RowMajor, \(A_{ij}\) is stored in \(\texttt{a}[(i-1) \times \texttt{pda} + j - 1]\).

If \texttt{uplo} = Nag_Upper, the upper triangular part of \(A\) must be stored and the elements of the array below the diagonal are not referenced.

If \texttt{uplo} = Nag_Lower, the lower triangular part of \(A\) must be stored and the elements of the array above the diagonal are not referenced.

If \texttt{diag} = Nag_UnitDiag, the diagonal elements of \(A\) are assumed to be 1, and are not referenced.

7: \texttt{pda} – Integer \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) of the matrix \(A\) in the array \texttt{a}.

\textit{Constraint:} \texttt{pda} \(\geq \max(1, n)\).

8: \texttt{rcond} – double * \textit{Output}

\textit{On exit:} an estimate of the reciprocal of the condition number of \(A\). \texttt{rcond} is set to zero if exact singularity is detected or the estimate underflows. If \texttt{rcond} is less than \textit{machine precision}, \(A\) is singular to working precision.

9: \texttt{fail} – NagError * \textit{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: \(n \geq 0\).

On entry, \(pda = \langle value\rangle\).
Constraint: \(pda > 0\).

**NE_INT_2**

On entry, \(pda = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: \(pda \geq \max(1, n)\).

**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.

**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.

7 Accuracy

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

8 Parallelism and Performance

_nag_ztrcon_ (f07tuc) is not threaded by NAG in any implementation.

_nag_ztrcon_ (f07tuc) 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

A call to _nag_ztrcon_ (f07tuc) involves solving a number of systems of linear equations of the form \(Ax = b\) or \(A^T x = b\); the number is usually 5 and never more than 11. Each solution involves approximately \(4n^2\) real floating-point operations but takes considerably longer than a call to _nag_ztrtrs_ (f07tsc) 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_dtrcon_ (f07tgc).
10 Example

This example estimates the condition number in the 1-norm of the matrix $A$, where

$$A = \begin{pmatrix}
4.78 + 4.56i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\
2.00 - 0.30i & -4.11 + 1.25i & 0.00 + 0.00i & 0.00 + 0.00i \\
2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.80i & 0.00 + 0.00i \\
-1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 0.26i
\end{pmatrix}. $$

The true condition number in the 1-norm is 70.27.

10.1 Program Text

```c
/* nag_ztrcon (f07tuc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* /

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

int main(void)
{
    /* Scalars */
    double rcond;
    Integer i, j, n, pda;
    Integer exit_status = 0;
    Nag_UploType uplo;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char nag_enum_arg[40];
    Complex *a = 0;

    INIT_FAIL(fail);
    printf("nag_ztrcon (f07tuc) Example Program Results\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
    #ifdef NAG_COLUMN_MAJOR
    pda = n;
    #else
    pda = n;
    #endif

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

/* Read A from data file */
#ifdef _WIN32
    scanf_s(" %39s%*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n] ", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}

/* Estimate condition number */
/* nag_ztrcon (f07tuc).
* Estimate condition number of complex triangular matrix */
nag_ztrcon(order, Nag_OneNorm, uplo, Nag_NonUnitDiag, n, a, pda, &rcond, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_ztrcon (f07tuc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
printf("\n");
/* 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(a);
    return exit_status;
}

\section*{10.2 Program Data}

\begin{verbatim}
nag_ztrcon (f07tuc) Example Program Data
\end{verbatim}
\begin{verbatim}
4 :Value of n
Nag_Lower :Value of uplo
( 4.78, 4.56)
( 2.00,-0.30) (-4.11, 1.25)
( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.80)
(-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-0.26) :End of matrix A
\end{verbatim}

\section*{10.3 Program Results}

\begin{verbatim}
nag_ztrcon (f07tuc) Example Program Results
\end{verbatim}

Estimate of condition number = 3.74e+01