

NAG C Library Function Document

nag_ztrtri (f07twc)

1 Purpose

nag_ztrtri (f07twc) computes the inverse of a complex triangular matrix.

2 Specification

```
void nag_ztrtri (Nag_OrderType order, Nag_UploType uplo, Nag_DiagType diag,
                Integer n, Complex a[], Integer pda, NagError *fail)
```

3 Description

nag_ztrtri (f07twc) forms the inverse of a complex triangular matrix A . Note that the inverse of an upper (lower) triangular matrix is also upper (lower) triangular.

4 References

Du Croz J J and Higham N J (1992) Stability of methods for matrix inversion *IMA J. Numer. Anal.* **12** 1–19

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: **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**.
- 3: **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**.
- 4: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.

5: **a**[*dim*] – Complex *Input/Output*

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

If **order** = **Nag_ColMajor**, the (*i*, *j*)th element of the matrix *A* is stored in **a**[(*j* – 1) × **pda** + *i* – 1] and if **order** = **Nag_RowMajor**, the (*i*, *j*)th element of the matrix *A* is stored in **a**[(*i* – 1) × **pda** + *j* – 1].

On entry: the *n* by *n* triangular matrix *A*. If **uplo** = **Nag_Upper**, *A* is upper triangular and the elements of the array below the diagonal are not referenced; if **uplo** = **Nag_Lower**, *A* is lower triangular and the elements of the array above the diagonal are not referenced. If **diag** = **Nag_UnitDiag**, the diagonal elements of *A* are not referenced, but are assumed to be 1.

On exit: *A* is overwritten by A^{-1} , using the same storage format as described above.

6: **pda** – Integer *Input*

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

Constraint: **pda** ≥ max(1, **n**).

7: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = *<value>*.

Constraint: **n** ≥ 0.

On entry, **pda** = *<value>*.

Constraint: **pda** > 0.

NE_INT_2

On entry, **pda** = *<value>*, **n** = *<value>*.

Constraint: **pda** ≥ max(1, **n**).

NE_SINGULAR

a(*<value>*, *<value>*) is zero, and the matrix *A* is singular.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter *<value>* 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 inverse *X* satisfies

$$|XA - I| \leq c(n)\epsilon |X| |A|,$$

where *c*(*n*) is a modest linear function of *n*, and ϵ is the *machine precision*.

Note that a similar bound for $|AX - I|$ cannot be guaranteed, although it is almost always satisfied.

The computed inverse satisfies the forward error bound

$$|X - A^{-1}| \leq c(n)\epsilon|A^{-1}||A||X|.$$

See Du Croz and Higham (1992).

8 Further Comments

The total number of real floating-point operations is approximately $\frac{4}{3}n^3$.

The real analogue of this function is nag_dtrtri (f07tjc).

9 Example

To compute the inverse of the matrix A , where

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

9.1 Program Text

```
/* nag_ztrtri (f07twc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, j, n, pda;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    Nag_MatrixType matrix;

    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char uplo[2];
    Complex *a=0;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
    order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);
    Vprintf("f07twc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%s[^\n] ");
    Vscanf("%ld%s[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
    pda = n;
#else
    pda = n;
#endif
}
```

```

#endif

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

/* Read A from data file */
Vscanf(" ' %ls '%*[^\\n] ", uplo);
if (*(unsigned char *)uplo == 'L')
{
    uplo_enum = Nag_Lower;
    matrix = Nag_LowerMatrix;
}
else if (*(unsigned char *)uplo == 'U')
{
    uplo_enum = Nag_Upper;
    matrix = Nag_UpperMatrix;
}
else
{
    Vprintf("Unrecognised character for Nag_UploType type\n");
    exit_status = -1;
    goto END;
}
if (uplo_enum == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
        Vscanf("%*[^\\n] ");
    }
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
        Vscanf("%*[^\\n] ");
    }
}

/* Compute inverse of A */
f07twc(order, uplo_enum, Nag_NonUnitDiag, n, a, pda, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07twc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print inverse */
x04dbc(order, matrix, Nag_NonUnitDiag, n, n, a, pda,
        Nag_BracketForm, "%7.4f", "Inverse", Nag_IntegerLabels,
        0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04dbc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
if (a) NAG_FREE(a);

return exit_status;
}

```

9.2 Program Data

f07twc Example Program Data

```

4                                     :Value of N
'L'                                 :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

```

9.3 Program Results

f07twc Example Program Results

```

Inverse
          1          2          3          4
1 ( 0.1095,-0.1045)
2 ( 0.0582,-0.0411) (-0.2227,-0.0677)
3 ( 0.0032, 0.1905) ( 0.1538,-0.2192) ( 0.2323,-0.0448)
4 ( 0.7602, 0.2814) ( 1.6184,-1.4346) ( 0.1289,-0.2250) ( 1.8697, 1.4731)

```
