NAG Library Function Document

nag_ztrtri (f07twc)

1 Purpose

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

2 Specification

```c
#include <nag.h>
#include <nagf07.h>
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


5 Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry}: the \texttt{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 \texttt{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}: \texttt{order} = Nag_RowMajor or Nag_COLMajor.

2: \textbf{uplo} – Nag_UploType \hspace{1cm} \textit{Input}

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

\texttt{uplo} = Nag_Upper
\hspace{1cm} \( A \) is upper triangular.

\texttt{uplo} = Nag_Lower
\hspace{1cm} \( A \) is lower triangular.

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

3: \textbf{diag} – Nag_DiagType \hspace{1cm} \textit{Input}

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

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

\texttt{diag} = Nag_UnitDiag
\hspace{1cm} \( A \) 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.
4: \( n \) – Integer

\textit{Input}

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

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

5: \( a[\text{dim}] \) – Complex

\textit{Input/Output}

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

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

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

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

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

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

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

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

6: \( pda \) – Integer

\textit{Input}

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

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

7: \( \text{fail} \) – 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 \( \langle \text{value} \rangle \) had an illegal value.

**NE_INT**

On entry, \( n = \langle \text{value} \rangle \).

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

On entry, \( pda = \langle \text{value} \rangle \).

\textit{Constraint}: \( pda > 0 \).

**NE_INT_2**

On entry, \( pda = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

\textit{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.

NE_SINGULAR
Element (value) of the diagonal is exactly zero. A is singular its inverse cannot be computed.

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 \( \epsilon \) 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

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


8 Parallelism and Performance
nag_ztrtri (f07twc) is not threaded by NAG in any implementation.

nag_ztrtri (f07twc) 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 total number of real floating-point operations is approximately \( \frac{4}{3}n^3 \).

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

10 Example
This example computes the inverse of the matrix \( A \), where

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

10.1 Program Text
/* nag_ztrtri (f07twc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* * Mark 7, 2001. */
```c
#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;
    Nag_MatrixType matrix;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char nag_enum_arg[40];
    Complex *a = 0;
    
    #ifdef NAG_LOAD_FP
    /* The following line is needed to force the Microsoft linker
       to load floating point support */
    float force_loading_of_ms_float_support = 0;
    #endif /* NAG_LOAD_FP */
    
    #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);
    printf("nag_ztrtri (f07twc) 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
    #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_UplType) nag_enum_name_to_value(nag_enum_arg);

if (uplo == Nag_Upper)
{  
  matrix = Nag_UpperMatrix;
  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
{
  matrix = Nag_LowerMatrix;
  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
}

/* Compute inverse of A */
/* nag_ztrtri (f07twc).  
  * Inverse of complex triangular matrix  
*/
if (fail.code != NE_NOERROR)
{  
  printf("Error from nag_ztrtri (f07twc)\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print inverse */
/* nag_gen_complx_mat_print_comp (x04dbc).  
  * Print complex general matrix (comprehensive)  
*/
fflush(stdout);
nag_gen_complx_mat_print_comp(order, matrix, Nag_NonUnitDiag, n, n, a, pda, &fail);
if (fail.code != NE_NOERROR)
{  
  printf("Error from nag_gen_complx_mat_print_comp (x04dbc)\n", fail.message);
  exit_status = 1;
  goto END;
}
END:
    NAG_FREE(a);

    return exit_status;
}

10.2 Program Data

nag_ztrtri (f07twc) Example Program Data
  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

10.3 Program Results

nag_ztrtri (f07twc) Example Program Results

Inverse

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0.1095,-0.1045)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(0.0582,-0.0411)</td>
<td>(-0.2227,-0.0677)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(0.0032, 0.1905)</td>
<td>(0.1538,-0.2192)</td>
<td>(0.2323,-0.0448)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(0.7602, 0.2814)</td>
<td>(1.6184,-1.4346)</td>
<td>(0.1289,-0.2250)</td>
<td>(1.8697, 1.4731)</td>
</tr>
</tbody>
</table>