NAG Library Function Document

nag_zpftri (f07wwc)

1 Purpose
nag_zpftri (f07wwc) computes the inverse of a complex Hermitian positive definite matrix using the Cholesky factorization computed by nag_zpftrf (f07wrc) stored in Rectangular Full Packed (RFP) format.

2 Specification
#include <nag.h>
#include <nagf07.h>
void nag_zpftri (Nag_OrderType order, Nag_RFP_Store transr,
                 Nag_UploType uplo, Integer n, Complex ar[], NagError *fail)

3 Description
nag_zpftri (f07wwc) is used to compute the inverse of a complex Hermitian positive definite matrix $A$, stored in RFP format. The RFP storage format is described in Section 3.3.3 in the f07 Chapter Introduction. The function must be preceded by a call to nag_zpftrf (f07wrc), which computes the Cholesky factorization of $A$.

If $\text{uplo} = \text{Nag_Upper}$, $A = U^H U$ and $A^{-1}$ is computed by first inverting $U$ and then forming $(U^{-1})U^{-H}$.

If $\text{uplo} = \text{Nag_Lower}$, $A = LL^H$ and $A^{-1}$ is computed by first inverting $L$ and then forming $L^{-H}(L^{-1})$.

4 References

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

   *Constraint:* \text{order} = \text{Nag_RowMajor} or \text{Nag_ColMajor}.

2: transr – Nag_RFP_Store
   
   **Input**
   
   *On entry:* specifies whether the normal RFP representation of $A$ or its conjugate transpose is stored.

   transr = Nag_RFP_Normal
   
   The matrix $A$ is stored in normal RFP format.

   transr = Nag_RFP_ConjTrans
   
   The conjugate transpose of the RFP representation of the matrix $A$ is stored.

   *Constraint:* transr = Nag_RFP_Normal or Nag_RFP_ConjTrans.
3: \texttt{uplo} \texttt{– Nag} \texttt{UploType} \texttt{Input}

\textit{On entry:} specifies how \(A\) has been factorized.

\texttt{uplo} = \texttt{Nag} \texttt{Upper}
\(A = U^H U\), where \(U\) is upper triangular.

\texttt{uplo} = \texttt{Nag} \texttt{Lower}
\(A = L L^H\), where \(L\) is lower triangular.

\textit{Constraint:} \texttt{uplo} = \texttt{Nag} \texttt{Upper} or \texttt{Nag} \texttt{Lower}.

4: \texttt{n} \texttt{– Integer} \texttt{Input}

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

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

5: \texttt{ar[|n \times (n + 1)/2]} \texttt{– Complex} \texttt{Input/Output}

\textit{On entry:} the Cholesky factorization of \(A\) stored in RFP format, as returned by \texttt{nag} \texttt{zpfrtf} (\texttt{f07wrc}).

\textit{On exit:} the factorization is overwritten by the \(n\) by \(n\) matrix \(A^{-1}\) stored in RFP format.

6: \texttt{fail} \texttt{– NagError *} \texttt{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

\textbf{6 Error Indicators and Warnings}

\textbf{NE_ALLOC_FAIL}
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}
On entry, argument \textit{\langle value\rangle} had an illegal value.

\textbf{NE_INT}
On entry, \texttt{n} = \textit{\langle value\rangle}.

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

\textbf{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.

\textbf{NE_MAT_NOT_POS_DEF}
The leading minor of order \textit{\langle value\rangle} is not positive definite and the factorization could not be completed. Hence \(A\) itself is not positive definite. This may indicate an error in forming the matrix \(A\). There is no function specifically designed to invert a Hermitian matrix stored in RFP format which is not positive definite; the matrix must be treated as a full Hermitian matrix, by calling \texttt{nag} \texttt{zhetri} (\texttt{f07mwc}).

\textbf{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 inverse $X$ satisfies

$$
\|X A - I\|_2 \leq c(n)\epsilon\kappa_2(A) \quad \text{and} \quad \|X A - I\|_2 \leq c(n)\epsilon\kappa_2(A),
$$

where $c(n)$ is a modest function of $n$, $\epsilon$ is the machine precision and $\kappa_2(A)$ is the condition number of $A$ defined by

$$
\kappa_2(A) = \|A\|_2\|A^{-1}\|_2.
$$

8 Parallelism and Performance

nag_zpftri (f07wwc) is not threaded by NAG in any implementation.

nag_zpftri (f07wwc) 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 $8n^3$.

The real analogue of this function is nag_dpfrtri (f07wjc).

10 Example

This example computes the inverse of the matrix $A$, where

$$
A = \begin{pmatrix}
3.23 + 0.00i & 1.51 - 1.92i & 1.90 + 0.84i & 0.42 + 2.50i \\
1.51 + 1.92i & 3.58 + 0.00i & -0.23 + 1.11i & -1.18 + 1.37i \\
1.90 - 0.84i & -0.23 - 1.11i & 4.09 + 0.00i & 2.33 - 0.14i \\
0.42 - 2.50i & -1.18 - 1.37i & 2.33 + 0.14i & 4.29 + 0.00i 
\end{pmatrix}.
$$

Here $A$ is Hermitian positive definite, stored in RFP format, and must first be factorized by nag_zpftrf (f07wrc).

10.1 Program Text

/* nag_zpftri (f07wwc) Example Program. */
* * Copyright 2014 Numerical Algorithms Group. *
* * Mark 25, 2014. *
* /

#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    Integer i, j, k, lar1, lar2, lenar, n, pdar, pda, q;
    /* Arrays */
    Complex *ar = 0, *a = 0;
    char nag_enum_arg[40];
    /* NAG types */
Nag_RFP_Store  transr;
Nag_UploType  uplo;
Nag_OrderType  order;
Nag_MatrixType  matrix;
Nag_DiagType  diag = Nag_NonUnitDiag;
NagError  fail;

#ifdef NAG_COLUMN_MAJOR
   order = Nag_ColMajor;
#endif
#define AR(I,J) ar[J*pdar + I]
#endif
#define AR(I,J) ar[I*pdar + J]
#endif
INIT_FAIL(fail);
printf("nag_zpftri (f07wwc) Example Program Results\n\n");
/* Skip heading in data file*/
#else
    scanf("%*\[^
");
#endif

#ifdef NAG_COLUMN_MAJOR
#elifdef _WIN32
    scanf("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s", nag_enum_arg);
#endif
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
#else
    scanf("%39s%*\[^
");
#endif
transr = (Nag_RFP_Store) nag_enum_name_to_value(nag_enum_arg);
lenar = (n * (n + 1))/2;
if (!(ar = NAG_ALLOC(lenar, Complex)) ||
 !(a = NAG_ALLOC(n*n, Complex)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
pda = n;
/* Setup dimensions for RFP array ar. */
k = n/2;
q = n - k;
if (transr==Nag_RFP_Normal) {
   lar1 = 2*k+1;
   lar2 = q;
} else {
   lar1 = q;
   lar2 = 2*k+1;
}
if (order==Nag_RowMajor) {
   pdar = lar2;
} else {
   pdar = lar1;
}
/* Read matrix into RFP array ar. */
for (i = 0; i < lar1; i++) {
    for (j = 0; j < lar2; j++) {
#ifdef _WIN32
        scanf_s(" ( %lf , %lf ) ", &AR(i,j).re, &AR(i,j).im);
#else
        scanf(" ( %lf , %lf ) ", &AR(i,j).re, &AR(i,j).im);
#endif
    } /* j */
} /* i */

f07wwc.4  Mark 25
f07 – Linear Equations (LAPACK)

10.2 Program Data

nag_zpftri (f07wwc) Example Program Data

4 Nag_Lower Nag_RFP_Normal : n, uplo, transr
( 4.09, 0.00) ( 2.33,-0.14)
( 3.23, 0.00) ( 4.29, 0.00)
( 1.51, 1.92) ( 3.58, 0.00)
( 1.90,-0.84) (-0.23,-1.11)
( 0.42,-2.50) (-1.18,-1.37) : ar[]
### 10.3 Program Results

*nag_zpftri (f07wwc)* Example Program Results

<table>
<thead>
<tr>
<th>Inverse</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(5.4691, 0.0000)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(-1.2624, -1.5491)</td>
<td>(1.1024, 0.0000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(-2.9746, -0.9616)</td>
<td>(0.8989, -0.5672)</td>
<td>(2.1589, -0.0000)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(1.1962, 2.9772)</td>
<td>(-0.9826, -0.2566)</td>
<td>(-1.3756, -1.4550)</td>
<td>(2.2934, 0.0000)</td>
</tr>
</tbody>
</table>