NAG Library Function Document

nag_zsytrf (f07nrc)

1 Purpose
nag_zsytrf (f07nrc) computes the Bunch–Kaufman factorization of a complex symmetric matrix.

2 Specification
#include <nag.h>
#include <nagf07.h>
void nag_zsytrf (Nag_OrderType order, Nag_UploType uplo, Integer n,
                Complex a[], Integer pda, Integer ipiv[], NagError *fail)

3 Description
nag_zsytrf (f07nrc) factorizes a complex symmetric matrix $A$, using the Bunch–Kaufman diagonal pivoting method. $A$ is factorized as either $A = PUDU^T P^T$ if $\text{uplo} = \text{Nag}_\text{Upper}$ or $A = PLDL^T P^T$ if $\text{uplo} = \text{Nag}_\text{Lower}$, where $P$ is a permutation matrix, $U$ (or $L$) is a unit upper (or lower) triangular matrix and $D$ is a symmetric block diagonal matrix with 1 by 1 and 2 by 2 diagonal blocks; $U$ (or $L$) has 2 by 2 unit diagonal blocks corresponding to the 2 by 2 blocks of $D$. Row and column interchanges are performed to ensure numerical stability while preserving symmetry.

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 order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.
   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: uplo – Nag_UploType  
   Input
   On entry: specifies whether the upper or lower triangular part of $A$ is stored and how $A$ is to be factorized.
   uplo = Nag_Upper
   The upper triangular part of $A$ is stored and $A$ is factorized as $PUDU^T P^T$, where $U$ is upper triangular.
   uplo = Nag_Lower
   The lower triangular part of $A$ is stored and $A$ is factorized as $PLDL^T P^T$, where $L$ is lower triangular.
   Constraint: uplo = Nag_Upper or Nag_Lower.
3: \( \mathbf{n} \) – Integer

\textit{Input}

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

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

4: \( \mathbf{a}[\text{dim}] \) – Complex

\textit{Input/Output}

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

On entry: the \( n \) by \( n \) symmetric indefinite matrix \( \mathbf{A} \).

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

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

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

On exit: the upper or lower triangle of \( \mathbf{A} \) is overwritten by details of the block diagonal matrix \( \mathbf{D} \) and the multipliers used to obtain the factor \( \mathbf{U} \) or \( \mathbf{L} \) as specified by \( \text{uplo} \).

5: \( \text{pda} \) – Integer

\textit{Input}

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

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

6: \( \text{ipiv}[\text{dim}] \) – Integer

\textit{Output}

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

On exit: details of the interchanges and the block structure of \( \mathbf{D} \). More precisely,

- if \( \text{ipiv}[i - 1] = k > 0 \), \( d_{ii} \) is a 1 by 1 pivot block and the \( i \)th row and column of \( \mathbf{A} \) were interchanged with the \( k \)th row and column;

- if \( \text{uplo} = \text{Nag\_Upper} \) and \( \text{ipiv}[i - 2] = \text{ipiv}[i - 1] = -l < 0 \), \( \begin{pmatrix} d_{i-1,i-1} & d_{i,i-1} \\ d_{i,i-1} & d_{ii} \end{pmatrix} \) is a 2 by 2 pivot block and the \( (i - 1) \)th row and column of \( \mathbf{A} \) were interchanged with the \( l \)th row and column;

- if \( \text{uplo} = \text{Nag\_Lower} \) and \( \text{ipiv}[i - 1] = \text{ipiv}[i] = -m < 0 \), \( \begin{pmatrix} d_{ii} & d_{i+1,i} \\ d_{i+1,i} & d_{i+1,i+1} \end{pmatrix} \) is a 2 by 2 pivot block and the \( (i + 1) \)th row and column of \( \mathbf{A} \) were interchanged with the \( m \)th row and column.

7: \( \text{fail} \) – NagError *

\textit{Input/Output}

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

6 \hspace{1em} \textbf{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 \( \langle \text{value} \rangle \) had an illegal value.
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

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

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

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.

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.

Element \( \langle \text{value} \rangle \) of the diagonal is exactly zero. The factorization has been completed, but the block diagonal matrix \( D \) is exactly singular, and division by zero will occur if it is used to solve a system of equations.

If \( \text{uplo} = \text{Nag}_\text{Upper} \), the computed factors \( U \) and \( D \) are the exact factors of a perturbed matrix \( A + E \), where

\[
|E| \leq c(n)\epsilon P||U||D||U^T||P^T,\]

\( c(n) \) is a modest linear function of \( n \), and \( \epsilon \) is the machine precision.

If \( \text{uplo} = \text{Nag}_\text{Lower} \), a similar statement holds for the computed factors \( L \) and \( D \).

The elements of \( D \) overwrite the corresponding elements of \( A \); if \( D \) has 2 by 2 blocks, only the upper or lower triangle is stored, as specified by \text{uplo}.

The unit diagonal elements of \( U \) or \( L \) and the 2 by 2 unit diagonal blocks are not stored. The remaining elements of \( U \) or \( L \) are stored in the corresponding columns of the array \( a \), but additional row interchanges must be applied to recover \( U \) or \( L \) explicitly (this is seldom necessary). If \( \text{ipiv}[i-1] = i \),...
for $i = 1, 2, \ldots, n$, then $U$ or $L$ is stored explicitly (except for its unit diagonal elements which are equal to 1).

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

A call to nag_zsytrf (f07nrc) may be followed by calls to the functions:

- nag_zsytrs (f07nsc) to solve $AX = B$;
- nag_zsycon (f07nuc) to estimate the condition number of $A$;
- nag_zsytri (f07nwc) to compute the inverse of $A$.

The real analogue of this function is nag_dsytrf (f07mdc).

## 10 Example

This example computes the Bunch–Kaufman factorization of the matrix $A$, where

\[
A = \begin{pmatrix}
-0.39 - 0.71i & 5.14 - 0.64i & -7.86 - 2.96i & 3.80 + 0.92i \\
5.14 - 0.64i & 8.86 + 1.81i & -3.52 + 0.58i & 5.32 - 1.59i \\
-7.86 - 2.96i & -3.52 + 0.58i & -2.83 - 0.03i & -1.54 - 2.86i \\
3.80 + 0.92i & 5.32 - 1.59i & -1.54 - 2.86i & -0.56 + 0.12i
\end{pmatrix}
\]

### 10.1 Program Text

/* nag_zsytrf (f07nrc) Example Program. */
* * Copyright 2014 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, nrhs, pda, pdb;
    Integer exit_status = 0;
    Nag_UploType uplo;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Integer *ipiv = 0;
    char nag_enum_arg[40];
    Complex *a = 0, *b = 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]
    #define B(I, J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda + J - 1]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);

    ...
printf("nag_zsytrf (f07nrc) Example Program Results\n\n");

/* Skip heading in data file */
#endif 
scanf_s("%*[\n] ");
#endif 
scanf("%*[\n] ");
#endif 
scanf_s("%NAG_IPMT"%NAG_IPMT"%*[\n] ", &n, &nrhs);
#endif 
scanf("%NAG_IPMT"%NAG_IPMT"%*[\n] ", &n, &nrhs);
#endif 
#endif 
pda = n;
pdb = n;
#endif 
pda = n;
pdb = nrhs;
#endif 

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

/* Read A and B from data file */
#endif 
scanf_s(" %39s%*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#endif 
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)
        {
            #ifndef _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
        }
    }
#else
    scanf("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
            #ifndef _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
        }
    }
#else
    scanf("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
```c
#ifdef _WIN32
    scanf_s("%*[\n ]");
#else
    scanf("%*[\n ]");
#endif

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

/* Factorize A */
/* nag_zsytrf (f07nrc).  
   * Bunch-Kaufman factorization of complex symmetric matrix 
   */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zsytrf (f07nrc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute solution */
/* nag_zsytrs (f07nsc).  
   * Solution of complex symmetric system of linear equations, 
   * multiple right-hand sides, matrix already factorized by 
   * nag_zsytrf (f07nrc) 
   */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zsytrs (f07nsc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
/* nag_gen_complx_mat_print_comp (x04dbc).  
   * Print complex general matrix (comprehensive) 
   */
fflush(stdout);
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(ipiv);
NAG_FREE(a);
NAG_FREE(b);
return exit_status;
```
10.2 Program Data

nag_zsytrf (f07nrc) Example Program Data

Nag_Lower :Values of n and nrhs
(-0.39,-0.71)
( 5.14,-0.64) ( 8.86, 1.81)
(-7.86,-2.96) (-3.52, 0.58) (-2.83,-0.03)
( 3.80, 0.92) ( 5.32,-1.59) (-1.54,-2.86) (-0.56, 0.12) :End of matrix A
(-55.64, 41.22) (-19.09,-35.97)
(-48.18, 66.00) (-12.08,-27.02)
(-0.49, -1.47) ( 6.95, 20.49)
(-6.43, 19.24) ( -4.59,-35.53) :End of matrix B

10.3 Program Results

nag_zsytrf (f07nrc) Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1.0000,-1.0000)</td>
<td>(-2.0000,-1.0000)</td>
</tr>
<tr>
<td>2</td>
<td>(-2.0000, 5.0000)</td>
<td>( 1.0000,-3.0000)</td>
</tr>
<tr>
<td>3</td>
<td>( 3.0000,-2.0000)</td>
<td>( 3.0000, 2.0000)</td>
</tr>
<tr>
<td>4</td>
<td>(-4.0000, 3.0000)</td>
<td>(-1.0000, 1.0000)</td>
</tr>
</tbody>
</table>