NAG Library Function Document

nag_zhptrf (f07prc)

1 Purpose

nag_zhptrf (f07prc) computes the Bunch–Kaufman factorization of a complex Hermitian indefinite matrix, using packed storage.

2 Specification

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

```
void nag_zhptrf (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Complex ap[], Integer ipiv[], NagError *fail)
```

3 Description

nag_zhptrf (f07prc) factorizes a complex Hermitian matrix A, using the Bunch–Kaufman diagonal pivoting method and packed storage. A is factorized as either $A = PUDU^{\mathrm{H}}P^{\mathrm{T}}$ if **uplo** $=$ Nag_Upper or $A = PLDL^{\mathrm{H}}P^{\mathrm{T}}$ if **uplo** $=$ Nag_Lower, where P is a permutation matrix, U (or L) is a unit upper (or lower) triangular matrix and D is an Hermitian 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 keeping the matrix Hermitian.

This method is suitable for Hermitian matrices which are not known to be positive definite. If A is in fact positive definite, no interchanges are performed and no 2 by 2 blocks occur in D.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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 $PUDU^{\mathrm{H}}P^{\mathrm{T}}$, where U is upper triangular.

uplo $=$ Nag_Lower
: The lower triangular part of A is stored and A is factorized as $PLDL^{\mathrm{H}}P^{\mathrm{T}}$, where L is lower triangular.

Constraint: **uplo** $=$ Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: $\mathbf{n} \geq 0$.

4: **ap**[dim] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$.

On entry: the n by n Hermitian matrix A, packed by rows or columns.

The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:

if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in $\mathbf{ap}[(j-1) \times j/2 + i - 1]$, for $i \leq j$;
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in $\mathbf{ap}[(2n-j) \times (j-1)/2 + i - 1]$, for $i \geq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in $\mathbf{ap}[(2n-i) \times (i-1)/2 + j - 1]$, for $i \leq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in $\mathbf{ap}[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

On exit: A is overwritten by details of the block diagonal matrix D and the multipliers used to obtain the factor U or L as specified by **uplo**.

5: **ipiv**[**n**] – Integer *Output*

On exit: details of the interchanges and the block structure of D. More precisely,

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

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

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

6: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

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.

NE_SINGULAR

$D(\langle value \rangle, \langle value \rangle)$ 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.

7 Accuracy

If **uplo** $=$ Nag_Upper, the computed factors U and D are the exact factors of a perturbed matrix $A+E$, where

$$|E| \leq c(n)\epsilon P|U||D|\left|U^{\mathrm{H}}\right|P^{\mathrm{T}},$$

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

If **uplo** $=$ Nag_Lower, a similar statement holds for the computed factors L and D.

8 Parallelism and Performance

nag_zhptrf (f07prc) is not threaded by NAG in any implementation.

nag_zhptrf (f07prc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

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 **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 and L are stored in the corresponding columns of the array **ap**, but additional row interchanges must be applied to recover U or L explicitly (this is seldom necessary). If **ipiv**$[i-1]=i$, for $i=1,2,\ldots,n$ (as is the case when A is positive definite), then U or L are stored explicitly in packed form (except for their 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_zhptrf (f07prc) may be followed by calls to the functions:

nag_zhptrs (f07psc) to solve $AX=B$;

nag_zhpcon (f07puc) to estimate the condition number of A;

nag_zhptri (f07pwc) to compute the inverse of A.

The real analogue of this function is nag_dsptrf (f07pdc).

10 Example

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

$$A=\begin{pmatrix} -1.36+0.00i & 1.58+0.90i & 2.21-0.21i & 3.91+1.50i \\ 1.58-0.90i & -8.87+0.00i & -1.84-0.03i & -1.78+1.18i \\ 2.21+0.21i & -1.84+0.03i & -4.63+0.00i & 0.11+0.11i \\ 3.91-1.50i & -1.78-1.18i & 0.11-0.11i & -1.84+0.00i \end{pmatrix},$$

using packed storage.

10.1 Program Text

```
/* nag_zhptrf (f07prc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

#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, pdb;
  Integer       exit_status = 0;
  NagError      fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
  Integer       *ipiv = 0;
  char          nag_enum_arg[40];
  Complex       *ap = 0, *b = 0;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
#define B(I, J)       b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
#define B(I, J)       b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zhptrf (f07prc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%*[^\n] ", &n, &nrhs);
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif

  /* Allocate memory */
  if (!(ipiv = NAG_ALLOC(n, Integer)) ||
      !(ap = NAG_ALLOC(n * (n + 1)/2, Complex)) ||
      !(b = NAG_ALLOC(n * nrhs, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A and B from data file */
  scanf("%39s%*[^\n] ", nag_enum_arg);
  /* 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)
            scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
                  &A_UPPER(i, j).im);
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
                  &A_LOWER(i, j).im);
        }
      scanf("%*[^\n] ");
    }
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }
  scanf("%*[^\n] ");

  /* Factorize A */
  /* nag_zhptrf (f07prc).
   * Bunch-Kaufman factorization of complex Hermitian
   * indefinite matrix, packed storage
   */
  nag_zhptrf(order, uplo, n, ap, ipiv, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhptrf (f07prc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Compute solution */
  /* nag_zhptrs (f07psc).
   * Solution of complex Hermitian indefinite system of linear
   * equations, multiple right-hand sides, matrix already
   * factorized by nag_zhptrf (f07prc), packed storage
   */
  nag_zhptrs(order, uplo, n, nrhs, ap, ipiv, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhptrs (f07psc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print solution */
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                nrhs, b, pdb, Nag_BracketForm, "%7.4f",
                                "Solution(s)", Nag_IntegerLabels,
                                0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(ipiv);
  NAG_FREE(ap);
```

```
  NAG_FREE(b);
  return exit_status;
}
```

10.2 Program Data

```
nag_zhptrf (f07prc) Example Program Data
  4  2                                                        :Values of n and nrhs
  Nag_Lower                                                   :Value of uplo
 (-1.36, 0.00)
 ( 1.58,-0.90) (-8.87, 0.00)
 ( 2.21, 0.21) (-1.84, 0.03) (-4.63, 0.00)
 ( 3.91,-1.50) (-1.78,-1.18) ( 0.11,-0.11) (-1.84, 0.00)  :End of matrix A
 ( 7.79,  5.48) (-35.39, 18.01)
 (-0.77,-16.05) (  4.23,-70.02)
 (-9.58,  3.88) (-24.79, -8.40)
 ( 2.98,-10.18) ( 28.68,-39.89)                               :End of matrix B
```

10.3 Program Results

```
nag_zhptrf (f07prc) Example Program Results

 Solution(s)
                     1                   2
 1  ( 1.0000,-1.0000)  ( 3.0000,-4.0000)
 2  (-1.0000, 2.0000)  (-1.0000, 5.0000)
 3  ( 3.0000,-2.0000)  ( 7.0000,-2.0000)
 4  ( 2.0000, 1.0000)  (-8.0000, 6.0000)
```