NAG Library Function Document

nag_zpstrf (f07krc)

1 Purpose

nag_zpstrf (f07krc) computes the Cholesky factorization with complete pivoting of a complex Hermitian positive semidefinite matrix.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_zpstrf (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Complex a[], Integer pda, Integer piv[], Integer *rank, double tol,
     NagError *fail)
```

3 Description

nag_zpstrf (f07krc) forms the Cholesky factorization of a complex Hermitian positive semidefinite matrix A either as $P^{\mathrm{T}}AP = U^{\mathrm{H}}U$ if **uplo** = Nag_Upper or $P^{\mathrm{T}}AP = LL^{\mathrm{H}}$ if **uplo** = Nag_Lower, where P is a permutation matrix, U is an upper triangular matrix and L is lower triangular.

This algorithm does not attempt to check that A is positive semidefinite.

4 References

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

Lucas C (2004) LAPACK-style codes for Level 2 and 3 pivoted Cholesky factorizations *LAPACK Working Note No. 161. Technical Report CS-04-522* Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA http://www.netlib.org/lapack/lawnspdf/lawn161.pdf

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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 $U^{\mathrm{H}}U$, where U is upper triangular.

uplo = Nag_Lower
The lower triangular part of A is stored and A is factorized as LL^{H}, 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: **a**[dim] – Complex *Input/Output*

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

On entry: the n by n Hermitian positive semidefinite matrix A.

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

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

If **uplo** = Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = Nag_Lower, the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

On exit: if **uplo** = Nag_Upper, the first **rank** rows of the upper triangle of A are overwritten with the nonzero elements of the Cholesky factor U, and the remaining rows of the triangle are destroyed.

If **uplo** = Nag_Lower, the first **rank** columns of the lower triangle of A are overwritten with the nonzero elements of the Cholesky factor L, and the remaining columns of the triangle are destroyed.

5: **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: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

6: **piv**[**n**] – Integer *Output*

On exit: **piv** is such that the nonzero entries of P are $P(\mathbf{piv}[k-1], k) = 1$, for $k = 1, 2, \ldots, n$.

7: **rank** – Integer * *Output*

On exit: the computed rank of A given by the number of steps the algorithm completed.

8: **tol** – double *Input*

On entry: user defined tolerance. If $\mathbf{tol} < 0$, then $n \times \max_{k=1,n} |A_{kk}| \times$ ***machine precision*** will be used. The algorithm terminates at the rth step if the $(r+1)$th step pivot $<$ **tol**.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

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_INT_2

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

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NW_NOT_POS_DEF

The matrix A is not positive definite. It is either positive semidefinite with computed rank as returned in **rank** and less than n, or it may be indefinite, see Section 9.

7 Accuracy

If **uplo** = Nag_Lower and **rank** $= r$, the computed Cholesky factor L and permutation matrix P satisfy the following upper bound

$$\frac{\|A - PLL^{\mathrm{H}}P^{\mathrm{T}}\|_2}{\|A\|_2} \leq 2rc(r)\epsilon\big(\|W\|_2 + 1\big)^2 + O\big(\epsilon^2\big),$$

where

$$W = L_{11}^{-1}L_{12}, \quad L = \begin{pmatrix} L_{11} & 0 \\ L_{12} & 0 \end{pmatrix}, \quad L_{11} \in \mathbb{C}^{r \times r},$$

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

$$\|W\|_2 \leq \sqrt{\frac{1}{3}(n-r)(4^r - 1)}.$$

So there is no guarantee of stability of the algorithm for large n and r, although $\|W\|_2$ is generally small in practice.

8 Parallelism and Performance

nag_zpstrf (f07krc) 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 $4nr^2 - 8/3r^3$, where r is the computed rank of A.

This algorithm does not attempt to check that A is positive semidefinite, and in particular the rank detection criterion in the algorithm is based on A being positive semidefinite. If there is doubt over semidefiniteness then you should use the indefinite factorization nag_zhetrf (f07mrc). See Lucas (2004) for further information.

The real analogue of this function is nag_dpstrf (f07kdc).

10 Example

This example computes the Cholesky factorization of the matrix A, where

$$A = \begin{pmatrix} 12.40 + 0.00i & 2.39 + 0.00i & 5.50 + 0.05i & 4.47 + 0.00i & 11.89 + 0.00i \\ 2.39 + 0.00i & 1.63 + 0.00i & 1.04 + 0.10i & 1.14 + 0.00i & 1.81 + 0.00i \\ 5.50 + 0.05i & 1.04 + 0.10i & 2.45 + 0.00i & 1.98 - 0.03i & 5.28 - 0.02i \\ 4.47 + 0.00i & 1.14 + 0.00i & 1.98 - 0.03i & 1.71 + 0.00i & 4.14 + 0.00i \\ 11.89 + 0.00i & 1.81 + 0.00i & 5.28 - 0.02i & 4.14 + 0.00i & 11.63 + 0.00i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zpstrf (f07krc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  Integer i, j, n, pda, rank;
  double tol;
  /* Arrays */
  Complex *a = 0;
  Integer *piv = 0;
  char nag_enum_arg[40];
  /* Nag Types */
  Nag_UploType uplo;
  Nag_OrderType order;
  Nag_MatrixType matrix;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_zpstrf (f07krc) Example Program Results\n");
  /* Skip heading in data file and retrieve data */
#ifdef _WIN32
  scanf_s("%*[^\n]%" NAG_IFMT "%39s%*[^\n]", &n, nag_enum_arg,
          (unsigned)_countof(nag_enum_arg));
#else
  scanf("%*[^\n]%" NAG_IFMT "%39s%*[^\n]", &n, nag_enum_arg);
#endif
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
  if (!(a = NAG_ALLOC(n * n, Complex)) || !(piv = NAG_ALLOC(n, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
```

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

  /* Read triangular part of A from data file */
  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
  }
  else if (uplo == Nag_Lower) {
    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
  }
  else {
    printf("Invalid uplo.\n");
    exit_status = 1;
    goto END;
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  tol = -1.0;

  /* Factorize A using nag_zpstrf (f07krc) which performs a Cholesky
   * factorization of complex Hermitian positive semidefinite matrix.
   */
  nag_zpstrf(order, uplo, n, a, pda, piv, &rank, tol, &fail);

  if (fail.code == NW_NOT_POS_DEF) {
    /* A is not of full rank.
     * Zero out columns rank+1 to n.
     */
    if (uplo == Nag_Upper)
      for (j = rank + 1; j <= n; j++)
        for (i = rank + 1; i <= j; i++)
          A(i, j) = nag_complex(0.0, 0.0);
    else if (uplo == Nag_Lower)
      for (j = rank + 1; j <= n; j++)
        for (i = j; i <= n; i++)
          A(i, j) = nag_complex(0.0, 0.0);
  }
  else if (fail.code != NE_NOERROR) {
    printf("Error from nag_zpstrf (f07krc)\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print rank of A. */
  printf("\nComputed rank: %" NAG_IFMT "\n\n", rank);

  /* Print factorization using
```

```
   * 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, n, Nag_BracketForm, "%5.2f", "Factor",
                                Nag_IntegerLabels, NULL, Nag_IntegerLabels,
                                NULL, 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;
  }

  /* Print pivot indices. */
  printf("\nPivots:\n");
  for (i = 0; i < n; i++)
    printf("%11" NAG_IFMT "", piv[i]);
  printf("\n");

END:
  NAG_FREE(a);
  NAG_FREE(piv);
  return exit_status;
}
```

10.2 Program Data

```
nag_zpstrf (f07krc) Example Program Data
  5            Nag_Lower                                                 : n, uplo
(12.40, 0.00)
( 2.39, 0.00) ( 1.63, 0.00)
( 5.50, 0.05) ( 1.04, 0.10) ( 2.45, 0.00)
( 4.47, 0.00) ( 1.14, 0.00) ( 1.98,-0.03) ( 1.71, 0.00)
(11.89, 0.00) ( 1.81, 0.00) ( 5.28,-0.02) ( 4.14, 0.00) (11.63, 0.00) : A
```

10.3 Program Results

```
nag_zpstrf (f07krc) Example Program Results

Computed rank: 3

 Factor
                1              2              3              4              5
 1  ( 3.52, 0.00)
 2  ( 0.68, 0.00)  ( 1.08, 0.00)
 3  ( 1.27, 0.00)  ( 0.26, 0.00)  ( 0.18, 0.00)
 4  ( 1.56, 0.01)  (-0.02, 0.08)  ( 0.01,-0.05)  ( 0.00, 0.00)
 5  ( 3.38, 0.00)  (-0.45, 0.00)  (-0.17, 0.00)  ( 0.00, 0.00)  ( 0.00, 0.00)

Pivots:
          1          2          4          3          5
```