NAG Library Function Document

nag_zpftrf (f07wrc)

1 Purpose

nag_zpftrf (f07wrc) computes the Cholesky factorization of a complex Hermitian positive definite matrix stored in Rectangular Full Packed (RFP) format.

2 Specification

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

3 Description

nag_zpftrf (f07wrc) forms the Cholesky factorization of a complex Hermitian positive definite matrix A either as $A = U^{\mathrm{H}}U$ if **uplo** = Nag_Upper or $A = LL^{\mathrm{H}}$ if **uplo** = Nag_Lower, where U is an upper triangular matrix and L is a lower triangular, stored in RFP format. The RFP storage format is described in Section 3.3.3 in the f07 Chapter Introduction.

4 References

Demmel J W (1989) On floating-point errors in Cholesky *LAPACK Working Note No. 14* University of Tennessee, Knoxville http://www.netlib.org/lapack/lawnspdf/lawn14.pdf

Gustavson F G, Waśniewski J, Dongarra J J and Langou J (2010) Rectangular full packed format for Cholesky's algorithm: factorization, solution, and inversion *ACM Trans. Math. Software* **37, 2**

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: **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: **uplo** – Nag_UploType *Input*

On entry: specifies whether the upper or lower triangular part of A is stored.

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.

4: **n** – Integer *Input*

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

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

5: **ar**[**n** × (**n** + **1**)/**2**] – Complex *Input/Output*

On entry: the upper or lower triangular part (as specified by **uplo**) of the n by n Hermitian matrix A, in either normal or transposed RFP format (as specified by **transr**). The storage format is described in detail in Section 3.3.3 in the f07 Chapter Introduction.

On exit: if **fail.code** = NE_NOERROR, the factor U or L from the Cholesky factorization $A = U^{\mathrm{H}}U$ or $A = LL^{\mathrm{H}}$, in the same storage format as A.

6: **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 3.2.1.2 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, **n** = ⟨*value*⟩.
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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_MAT_NOT_POS_DEF

The leading minor of order ⟨*value*⟩ 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 factorize a Hermitian matrix stored in

RFP format which is not positive definite; the matrix must be treated as a full Hermitian matrix, by calling nag_zhetrf (f07mrc).

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

If **uplo** = Nag_Upper, the computed factor U is the exact factor of a perturbed matrix $A + E$, where

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

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

If **uplo** = Nag_Lower, a similar statement holds for the computed factor L. It follows that $\left|e_{ij}\right| \leq c(n)\epsilon\sqrt{a_{ii}a_{jj}}$.

8 Parallelism and Performance

nag_zpftrf (f07wrc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zpftrf (f07wrc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

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

A call to nag_zpftrf (f07wrc) may be followed by calls to the functions:

nag_zpftrs (f07wsc) to solve $AX = B$;

nag_zpftri (f07wwc) to compute the inverse of A.

The real analogue of this function is nag_dpftrf (f07wdc).

10 Example

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

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

and is stored using RFP format.

10.1 Program Text

```
/* nag_zpftrf (f07wrc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
```

```
 */

#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;
#define AR(I,J) ar[J*pdar + I]
#else
  order = Nag_RowMajor;
#define AR(I,J) ar[I*pdar + J]
#endif

  INIT_FAIL(fail);
  printf("nag_zpftrf (f07wrc) 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);
#else
  scanf("%" NAG_IFMT "", &n);
#endif
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  transr = (Nag_RFP_Store) nag_enum_name_to_value(nag_enum_arg);

  lenar = (n * (n + 1)) / 2;
  pda = n;
  if (!(ar = NAG_ALLOC(lenar, Complex)) || !(a = NAG_ALLOC(pda * n, Complex)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* 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
    }
  }

  /* Factorize A using nag_zpftrf (f07wrc) which performs a Cholesky
   * factorization of a complex Hermitian positive definite matrix in
   * Rectangular Full Packed format
   */
  nag_zpftrf(order, transr, uplo, n, ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zpftrf (f07wrc)\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Convert factorized matrix to full matrix format using
   * nag_ztfttr (f01vhc)
   */
  nag_ztfttr(order, transr, uplo, n, ar, a, pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from  nag_ztfttr (f01vhc)\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }

  matrix = (uplo == Nag_Lower ? Nag_LowerMatrix : Nag_UpperMatrix);

  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, matrix, diag, n, n, a, pda,
                                Nag_BracketForm, "%7.4f", "Factor",
                                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 = 3;
  }

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

10.2 Program Data

```
nag_zpftrf (f07wrc) 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_zpftrf (f07wrc) Example Program Results

 Factor
                     1                   2                   3                   4
 1  ( 1.7972, 0.0000)
 2  ( 0.8402, 1.0683)  ( 1.3164, 0.0000)
 3  ( 1.0572,-0.4674)  (-0.4702, 0.3131)  ( 1.5604,-0.0000)
 4  ( 0.2337,-1.3910)  ( 0.0834, 0.0368)  ( 0.9360, 0.8105)  ( 0.8713,-0.0000)
```