# NAG Library Function Document

# nag_ztfttp (f01vmc)

## 1 Purpose

nag_ztfttp (f01vmc) copies a complex triangular matrix, stored in a Rectangular Full Packed (RFP) format array, to a standard packed format array.

## 2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_ztfttp (Nag_OrderType order, Nag_RFP_Store transr,
     Nag_UploType uplo, Integer n, const Complex ar[], Complex ap[],
     NagError *fail)
```

## 3 Description

nag_ztfttp (f01vmc) packs a complex $n$ by $n$ triangular matrix $A$, stored in RFP format, to packed format. This function is intended for possible use in conjunction with functions from Chapters f06, f07 and f16 where some functions that use triangular matrices store them in RFP format. The RFP storage format is described in Section 3.3.3 in the f07 Chapter Introduction and the packed storage format is described in Section 3.3.2 in the f07 Chapter Introduction.

## 4 References

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 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: **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 RFP representation of the matrix $A$ is stored.

**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 $A$ is upper or lower triangular.

**uplo** = Nag_Upper
$A$ is upper triangular.

**uplo** = Nag_Lower
$A$ 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**$[\mathbf{n} \times (\mathbf{n}+\mathbf{1})/\mathbf{2}]$ – const Complex *Input*

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

6: **ap**$[dim]$ – Complex *Output*

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

*On exit*: the $n$ by $n$ triangular matrix $A$, packed by rows or columns depending on **order**.

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 **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 **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 **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 **ap**$[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

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

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

## 6 Error Indicators and Warnings

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction 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 the Essential Introduction for further information.

**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

Not applicable.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

None.

## 10 Example

This example reads in a triangular matrix in RFP format and copies it to packed format.

### 10.1 Program Text

```
/* nag_ztfttp (f01vmc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 25, 2014.
 */

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

int main(void)
{
  /* Scalars */
  Integer       exit_status = 0, inc1 = 1, indent = 0, ncols = 80;
  Integer       i, j, k, lar1, lar2, lenap, lenar, mx, n, nx, pdar, q;
  /* Arrays */
  Complex       *ap = 0, *ar = 0;
  char          nag_enum_transr[40], nag_enum_uplo[40], form[] = "%5.2f";
  /* Nag Types */
  Nag_OrderType order;
  Nag_RFP_Store transr;
  Nag_UploType  uplo;
  NagError      fail;

#ifdef NAG_COLUMN_MAJOR
  order = Nag_ColMajor;
#define AR(I,J) ar[J*pdar + I]
#define KU(I,J,N) (I + J*(J+1)/2)
#define KL(I,J,N) (J*(N-1) - J*(J-1)/2 + I)
#else
  order = Nag_RowMajor;
#define AR(I,J) ar[I*pdar + J]
#define KL(I,J,N) (J + I*(I+1)/2)
#define KU(I,J,N) (I*(N-1) - I*(I-1)/2 + J)
#endif
```

```
  INIT_FAIL(fail);

  printf("nag_ztfttp (f01vmc) Example Program Results\n\n");
  /* Skip heading in data file*/
#ifdef _WIN32
  scanf_s("%*[^\n] ");
  scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
  scanf_s("%39s ", nag_enum_transr, _countof(nag_enum_transr));
  scanf_s("%39s  %*[^\n] ", nag_enum_uplo, _countof(nag_enum_uplo));
#else
  scanf("%*[^\n] ");
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
  scanf("%39s ", nag_enum_transr);
  scanf("%39s  %*[^\n] ", nag_enum_uplo);
#endif
  lenap = (n * (n + 1))/2;
  lenar = lenap;
  if (!(ap = NAG_ALLOC(lenap, Complex)) || !(ar = NAG_ALLOC(lenar, Complex))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  transr = (Nag_RFP_Store) nag_enum_name_to_value(nag_enum_transr);
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_uplo);

  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 an RFP matrix into 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
  }

  /* Print the packed Rectangular Full Packed array */
  if (order==Nag_RowMajor) {
    mx = inc1;
    nx = lenar;
  } else {
    mx = lenar;
    nx = inc1;
  }
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mx,
                                nx, ar, lenar, Nag_BracketForm, form,
                                "RFP Packed Array AR:", Nag_IntegerLabels, NULL,
                                Nag_NoLabels, NULL, ncols, indent, NULL,
                                &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
            fail.message);
    exit_status = 1;
  }
  printf("\n");

  /* Convert real triangular matrix from Rectangular Full Packed to
   * packed vector form using nag_ztfttp (f01vmc).
```

```
   */
  nag_ztfttp(order, transr, uplo, n, ar, ap, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from  nag_ztfttp (f01vmc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the packed vector */
  if (order==Nag_RowMajor) {
    mx = inc1;
    nx = lenap;
  } else {
    mx = lenap;
    nx = inc1;
  }
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mx,
                                nx, ap, lenap, Nag_BracketForm, form,
                                "Packed Array AP:", Nag_IntegerLabels, NULL,
                                Nag_NoLabels, NULL, ncols, indent, NULL,
                                &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 1;
  }
  printf("\n");

  /* Print the packed vector using macros KL or KU. */
  printf(" Packed Matrix AP (printed using KL/KU macros):\n\n");
  for (i = 0; i < n; i++) {
    printf("  ");
    if (uplo==Nag_Upper) {
      for (j = 0; j < i; j++) printf("%15s"," ");
      for (j = i; j < n; j++)
        printf("  (%5.2f,%5.2f)", ap[KU(i,j,n)].re, ap[KU(i,j,n)].im);
    } else {
      for (j = 0; j <= i; j++)
        printf("  (%5.2f,%5.2f)", ap[KU(i,j,n)].re, ap[KU(i,j,n)].im);
    }
    printf("\n");
  }

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

## 10.2 Program Data

```
nag_ztfttp (f01vmc) Example Program Data
 4                              : n
 Nag_RFP_Normal Nag_Upper       : transr, uplo
 ( 1.30, 1.30) ( 1.40, 1.40)
 ( 2.30, 2.30) ( 2.40, 2.40)
 ( 3.30, 3.30) ( 3.40, 3.40)
 ( 1.10,-1.10) ( 4.40, 4.40)
 ( 1.20,-1.20) ( 2.20,-2.20)    : RFP array ar[]
```

## 10.3 Program Results

```
nag_ztfttp (f01vmc) Example Program Results

 RFP Packed Array AR:
 1  ( 1.30, 1.30)  ( 1.40, 1.40)  ( 2.30, 2.30)  ( 2.40, 2.40)  ( 3.30, 3.30)

 1  ( 3.40, 3.40)  ( 1.10,-1.10)  ( 4.40, 4.40)  ( 1.20,-1.20)  ( 2.20,-2.20)

 Packed Array AP:
```

```
 1  ( 1.10, 1.10)  ( 1.20, 1.20)  ( 1.30, 1.30)  ( 1.40, 1.40)  ( 2.20, 2.20)

 1  ( 2.30, 2.30)  ( 2.40, 2.40)  ( 3.30, 3.30)  ( 3.40, 3.40)  ( 4.40, 4.40)

 Packed Matrix AP (printed using KL/KU macros):

    ( 1.10, 1.10)  ( 1.20, 1.20)  ( 1.30, 1.30)  ( 1.40, 1.40)
                   ( 2.20, 2.20)  ( 2.30, 2.30)  ( 2.40, 2.40)
                                  ( 3.30, 3.30)  ( 3.40, 3.40)
                                                 ( 4.40, 4.40)
```