NAG Library Function Document

nag_ztpttf (f01vkc)

1 Purpose

nag_ztpttf (f01vkc) copies a complex triangular matrix, stored in a standard packed format array, to a Rectangular Full Packed (RFP) format array.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
```

```
void nag_ztpttf (Nag_OrderType order, Nag_RFP_Store transr,
     Nag_UploType uplo, Integer n, const Complex ap[], Complex ar[],
     NagError *fail)
```

3 Description

nag_ztpttf (f01vkc) copies a complex n by n triangular matrix, A, stored in packed format, to RFP 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: **ap**[*dim*] – const Complex *Input*

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

On entry: 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 $\mathbf{ap}[(j-1) \times j/2 + i - 1]$, 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 $i \geq j$.

6: **ar**[$\mathbf{n} \times (\mathbf{n}+\mathbf{1})/\mathbf{2}$] – Complex *Output*

On exit: 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.

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, $\mathbf{n}$ = ⟨*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 packed format and copies it to RFP format.

10.1 Program Text

```
/* nag_ztpttf (f01vkc) 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, indent = 0, ncols = 80, inc1 = 1;
  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 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 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_ztpttf (f01vkc) 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);

  /* Read the packed vector ap using macros KL or KU. */
  for (i = 0; i < n; i++) {
    if (uplo==Nag_Upper) {
#ifdef _WIN32
      for (j = i; j < n; j++) scanf_s(" ( %lf , %lf )", &ap[KU(i,j,n)].re,
                                      &ap[KU(i,j,n)].im);
#else
      for (j = i; j < n; j++) scanf(" ( %lf , %lf )", &ap[KU(i,j,n)].re,
                                    &ap[KU(i,j,n)].im);
#endif
    } else {
#ifdef _WIN32
      for (j = 0; j <= i; j++) scanf_s(" ( %lf , %lf )", &ap[KL(i,j,n)].re,
                                       &ap[KL(i,j,n)].im);
#else
      for (j = 0; j <= i; j++) scanf(" ( %lf , %lf )", &ap[KL(i,j,n)].re,
                                     &ap[KL(i,j,n)].im);
#endif
    }
  }

  if (order==Nag_RowMajor) {
    mx = inc1;
    nx = lenap;
  } else {
    mx = lenap;
    nx = inc1;
  }
  /* Print the packed vector */
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mx,
                                nx, ap, lenap, Nag_BracketForm, form,
                                "Packed Matrix AP:", Nag_IntegerLabels, NULL,
                                Nag_NoLabels, NULL, ncols, indent, NULL,
                                &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
           fail.message);
    exit_status = 1;
  }
  printf("\n");

  /* Convert complex triangular matrix from packed to Rectangular Full Packed
   * form using nag_ztpttf (f01vkc).
   */
  nag_ztpttf(order, transr, uplo, n, ap, ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from  nag_ztpttf (f01vkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* 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");

  /* Print the Rectangular Full Packed array
   * showing how the elements are arranged.
   */
  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;
  }

  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive).
   */
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                                  lar1, lar2, ar, pdar, Nag_BracketForm, form,
                                  "RFP Packed Array AR "
                                  "(structural representation):",
                                  Nag_IntegerLabels, NULL, Nag_IntegerLabels,
                                  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;
  }

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

10.2 Program Data

```
nag_ztpttf (f01vkc) Example Program Data
 4                                       : n
 Nag_RFP_Normal Nag_Upper               : transr, uplo

 (1.1,1.1) (1.2,1.2) (1.3,1.3) (1.4,1.4)
           (2.2,2.2) (2.3,2.3) (2.4,2.4)
                     (3.3,3.3) (3.4,3.4)
                               (4.4,4.4) : ap[]
```

10.3 Program Results

```
nag_ztpttf (f01vkc) Example Program Results

 Packed Matrix 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)

 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)

 RFP Packed Array AR (structural representation):
                1               2
 1  ( 1.30, 1.30)  ( 1.40, 1.40)
 2  ( 2.30, 2.30)  ( 2.40, 2.40)
 3  ( 3.30, 3.30)  ( 3.40, 3.40)
 4  ( 1.10,-1.10)  ( 4.40, 4.40)
 5  ( 1.20,-1.20)  ( 2.20,-2.20)
```