NAG Library Function Document

nag_ztpttr (f01vdc)

1 Purpose

nag_ztpttr (f01vdc) unpacks a complex triangular matrix, stored in a standard packed format array, to a full format array.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_ztpttr (Nag_OrderType order, Nag_UploType uplo, Integer n,
     const Complex ap[], Complex a[], Integer pda, NagError *fail)
```

3 Description

nag_ztpttr (f01vdc) unpacks a complex n by n triangular matrix A, stored in an array of length $n(n+1)/2$, to conventional storage in a full format array. This function is intended for possible use in conjunction with functions from Chapters f06, f07, f08 and f16 where some functions use triangular matrices stored in the packed form. Packed storage format is described in Section 3.3.2 in the f07 Chapter Introduction.

4 References

None.

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 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.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **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 **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 $i \geq j$.

5: **a**[dim] – Complex *Output*

Note: the dimension, *dim*, of the array **a** must be at least **pda** × **n**.

On exit: the triangular matrix A.

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

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

If **uplo** = Nag_Upper, A is upper triangular and the elements of the array below the diagonal are not set.

If **uplo** = Nag_Lower, A is lower triangular and the elements of the array above the diagonal are not set.

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

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: **n** ≥ 0.

NE_INT_2

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **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 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 packed by columns and unpacks it to full format.

10.1 Program Text

```
/* nag_ztpttr (f01vdc) 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, i, j, pda;
  Integer        lenap, mx, n, nx;
  Complex        *a = 0, *ap = 0;
  /* Arrays */
  char           nag_enum_arg[40], form[] = "%5.2f";
  /* Nag Types */
  Nag_OrderType  order;
  Nag_UploType   uplo;
  Nag_MatrixType matrix;
  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_ztpttr (f01vdc) Example Program Results\n\n");
  /* Skip heading in data file*/
#ifdef _WIN32
  scanf_s("%*[^\n] ");
  scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
  scanf_s("%39s  %*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%*[^\n] ");
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
  scanf("%39s  %*[^\n] ", nag_enum_arg);
#endif
  pda = n;
  lenap = (n * (n + 1))/2;
  if (!(a = NAG_ALLOC(pda*n, Complex)) || !(ap = NAG_ALLOC(lenap, Complex))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  /* 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
    }
  }

  /* 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");

  /* Convert to triangular matrix from packed vector to full form using
   * nag_ztpttr (f01vdc).
   */
```

```
  nag_ztpttr(order, uplo, n, ap, a, pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztpttr (f01vdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the unpacked matrix. */
  matrix = (uplo == Nag_Upper ? Nag_UpperMatrix : Nag_LowerMatrix);

  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive).
   */
  nag_gen_complx_mat_print_comp(order, matrix, Nag_NonUnitDiag, n, n, a, pda,
                                Nag_BracketForm, form, "Unpacked Matrix A:",
                                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(a);
  NAG_FREE(ap);
  return exit_status;
}
```

10.2 Program Data

```
nag_ztpttr (f01vdc) Example Program Data
 4                                      : n
 Nag_Upper                              : 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_ztpttr (f01vdc) Example Program Results

 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)

 Unpacked Matrix A:
                1              2              3              4
 1  ( 1.10, 1.10)  ( 1.20, 1.20)  ( 1.30, 1.30)  ( 1.40, 1.40)
 2                 ( 2.20, 2.20)  ( 2.30, 2.30)  ( 2.40, 2.40)
 3                                ( 3.30, 3.30)  ( 3.40, 3.40)
 4                                               ( 4.40, 4.40)
```