

# NAG Library Function Document

## nag\_dtpttf (f01vjc)

### 1 Purpose

nag\_dtpttf (f01vjc) copies a real 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_dtpttf (Nag_OrderType order, Nag_RFP_Store transr,
                Nag_UploType uplo, Integer n, const double ap[], double ar[],
                NagError *fail)
```

### 3 Description

nag\_dtpttf (f01vjc) copies a real  $n$  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 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 transpose is stored.  
**transr** = Nag\_RFP\_Normal  
 The RFP representation of the matrix  $A$  is stored.  
**transr** = Nag\_RFP\_Trans  
 The transpose of the RFP representation of the matrix  $A$  is stored.  
*Constraint:* **transr** = Nag\_RFP\_Normal or Nag\_RFP\_Trans.
- 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:* **n** ≥ 0.

5: **ap**[*dim*] – const double *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) × *j*/2 + *i* – 1], for  $i \leq j$ ;  
 if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap**[(2*n* – *j*) × (*j* – 1)/2 + *i* – 1], for  $i \geq j$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $A_{ij}$  is stored in **ap**[(2*n* – *i*) × (*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) × *i*/2 + *j* – 1], for  $i \geq j$ .

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

Not applicable.

**8 Parallelism and Performance**

nag\_dtpttf (f01vjc) is not threaded in any implementation.

**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_dtpttf (f01vjc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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;
    Integer i, j, k, lar1, lar2, lenap, lenar, n, pdar, q;
    /* Arrays */
    double *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) (I + J*(J+1)/2)
#define KL(I,J) (J*(n-1) - J*(J-1)/2 + I)
#else
    order = Nag_RowMajor;
#define KL(I,J) (J + I*(I+1)/2)
#define KU(I,J) (I*(n-1) - I*(I-1)/2 + J)
#endif

    INIT_FAIL(fail);

    printf("nag_dtpttf (f01vjc) 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, (unsigned)_countof(nag_enum_transr));
scanf_s("%39s %*[\n] ", nag_enum_uplo,
        (unsigned)_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, double)) || !(ar = NAG_ALLOC(lenar, double)))
{
    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 and print the packed vector ap using macros KL or KU. */
printf(" Packed Array ap (printed using KL/KU macros):\n\n");
for (i = 0; i < n; i++) {
    if (uplo == Nag_Upper) {
#ifdef _WIN32
        for (j = i; j < n; j++)
            scanf_s("%lf ", &ap[KU(i, j)]);
#else
        for (j = i; j < n; j++)
            scanf("%lf ", &ap[KU(i, j)]);
#endif
        for (j = 0; j < i; j++)
            printf("%6s", " ");
        for (j = i; j < n; j++)
            printf("%6.2f", ap[KU(i, j)]);
    }
    else {
#ifdef _WIN32
        for (j = 0; j <= i; j++)
            scanf_s("%lf ", &ap[KL(i, j)]);
#else
        for (j = 0; j <= i; j++)
            scanf("%lf ", &ap[KL(i, j)]);
#endif
        for (j = 0; j <= i; j++)
            printf("%6.2f", ap[KL(i, j)]);
    }
    printf("\n");
}
printf("\n");

/* Convert real triangular matrix from packed to Rectangular Full Packed
 * form using nag_dtpttf (f01vjc).
 */
nag_dtpttf(order, transr, uplo, n, ap, ar, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtpttf (f01vjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* 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_real_mat_print_comp (x04cbc).
 * Print real general matrix (comprehensive).
 */
fflush(stdout);
nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, lar1,
                             lar2, ar, pdar, 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_real_mat_print_comp (x04cbc).\n%s\n",
           fail.message);
    exit_status = 1;
}
}

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

```

## 10.2 Program Data

```

nag_dtpttf (f01vjc) Example Program Data
4 : n
Nag_RFP_Normal Nag_Upper : transr, uplo

1.1 1.2 1.3 1.4
   2.2 2.3 2.4
     3.3 3.4
       4.4 : ap[]

```

## 10.3 Program Results

```

nag_dtpttf (f01vjc) Example Program Results

Packed Array ap (printed using KL/KU macros):

1.10 1.20 1.30 1.40
     2.20 2.30 2.40
       3.30 3.40
         4.40

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

```

---