

# NAG Library Function Document

## nag\_dtrttp (f01vac)

### 1 Purpose

nag\_dtrttp (f01vac) copies a real triangular matrix, stored in a full format array, to a standard packed format array.

### 2 Specification

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

void nag_dtrttp (Nag_OrderType order, Nag_UploType uplo, Integer n,
                const double a[], Integer pda, double ap[], NagError *fail)
```

### 3 Description

nag\_dtrttp (f01vac) packs a real  $n$  by  $n$  triangular matrix  $A$ , stored conventionally in a full format array, into an array of length  $n(n+1)/2$ . The matrix is packed by rows or columns depending on **order**. 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 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: **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:*  $n \geq 0$ .

- 4: **a**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least **pda** × **n**.  
*On entry:* the triangular matrix *A*.  
If **order** = Nag\_ColMajor,  $A_{ij}$  is stored in **a**[(*j* – 1) × **pda** + *i* – 1].  
If **order** = Nag\_RowMajor,  $A_{ij}$  is stored in **a**[(*i* – 1) × **pda** + *j* – 1].  
If **uplo** = Nag\_Upper, the upper triangular part of *A* must be stored and the elements of the array below the diagonal are not referenced.  
If **uplo** = Nag\_Lower, the lower triangular part of *A* must be stored and the elements of the array above the diagonal are not referenced.
- 5: **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** ≥ max(1, **n**).
- 6: **ap**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **ap** must be at least **n** × (**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) × *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  $i \geq j$ .
- 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\_INT\_2

On entry, **pda** = *<value>* and **n** = *<value>*.

Constraint: **pda** ≥ max(1, **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 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\_dtrttp (f01vac) is not threaded in any implementation.

**9 Further Comments**

None.

**10 Example**

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

**10.1 Program Text**

```

/* nag_dtrttp (f01vac) 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, i, j, pda;
    Integer lenap, n;
    /* Arrays */
    double *a = 0, *ap = 0;
    char nag_enum_arg[40], form[] = "%5.2f";
    /* Nag Types */
    Nag_MatrixType matrix;
    Nag_OrderType order;
    Nag_UploType uplo;
    NagError fail;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[J*pda + I]
    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
#define A(I, J) a[I*pda + J]
    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_dtrttp (f01vac) 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, (unsigned)_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, double)) || !(ap = NAG_ALLOC(lenap, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
    /* Read a triangular matrix of order n. */
    for (i = 0; i < n; i++) {
        if (uplo == Nag_Upper) {
#ifdef _WIN32
            for (j = i; j < n; j++)
                scanf_s("%lf", &A(i, j));
            scanf_s("%*[\n] ");
#else
            for (j = i; j < n; j++)
                scanf("%lf", &A(i, j));
            scanf("%*[\n] ");
#endif
        }
        else {
#ifdef _WIN32
            for (j = 0; j <= i; j++)
                scanf_s("%lf", &A(i, j));
            scanf_s("%*[\n] ");
#else
            for (j = 0; j <= i; j++)
                scanf("%lf", &A(i, j));
            scanf("%*[\n] ");
#endif
        }
    }

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

    /* nag_gen_real_mat_print_comp (x04cbc).
     * Print real general matrix (comprehensive).
     */
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, matrix, Nag_NonUnitDiag, n, n, a, pda,
                                form, "Unpacked Matrix A:", 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;
        goto END;
    }

```

```

}
printf("\n");

/* Convert to triangular matrix from full to packed vector form using
 * nag_dtrttp (f01vac).
 */
nag_dtrttp(order, uplo, n, a, pda, ap, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrttp (f01vac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the packed vector using macros KL or KU. */
printf(" Packed Array 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("%6s", " ");
        for (j = i; j < n; j++)
            printf("%6.2f", ap[KU(i, j)]);
    }
    else {
        for (j = 0; j <= i; j++)
            printf("%6.2f", ap[KL(i, j)]);
    }
    printf("\n");
}

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

```

## 10.2 Program Data

```

nag_dtrttp (f01vac) Example Program Data
4                : n
Nag_Upper       : uplo
1.1  1.2  1.3  1.4
      2.2  2.3  2.4
           3.3  3.4
              4.4      : Unpacked Matrix A

```

## 10.3 Program Results

```

nag_dtrttp (f01vac) Example Program Results

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

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

```

---