NAG Library Function Document

nag_dtfttp (f01vlc)

1 Purpose

nag_dtfttp (f01vlc) copies a real triangular matrix, stored in a Rectangular Full Packed (RFP) format array, to a standard packed format array.

2 Specification

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

3 Description

nag_dtfttp (f01vlc) packs a real \( n \times n \) 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


5 Arguments

1: \textbf{order} \hspace{1cm} \textbf{Input}

\textit{On entry:} the \texttt{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 \texttt{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \texttt{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{transr} \hspace{1cm} \textbf{Input}

\textit{On entry:} specifies whether the normal RFP representation of \( A \) or its transpose is stored.

\texttt{transr} = Nag_RFP_Normal

The RFP representation of the matrix \( A \) is stored.

\texttt{transr} = Nag_RFP_Trans

The transpose of the RFP representation of the matrix \( A \) is stored.

\textit{Constraint:} \texttt{transr} = Nag_RFP_Normal or Nag_RFP_Trans.

3: \textbf{uplo} \hspace{1cm} \textbf{Input}

\textit{On entry:} specifies whether \( A \) is upper or lower triangular.

\texttt{uplo} = Nag_Upper

\( A \) is upper triangular.
uplo = Nag_Lower
    A is lower triangular.

Constraint: uplo = Nag_Upper or Nag_Lower.

4: n – Integer

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

Constraint: n ≥ 0.

5: ar[n × (n + 1)/2] – const double

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] – double

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 ≤ j;
if order = Nag_ColMajor and uplo = Nag_Lower,
    A_{ij} is stored in ap[(2n - j) × (j - 1)/2 + i - 1], for i ≥ j;
if order = Nag_RowMajor and uplo = Nag_Upper,
    A_{ij} is stored in ap[(2n - i) × (i - 1)/2 + j - 1], for i ≤ j;
if order = Nag_RowMajor and uplo = Nag_Lower,
    A_{ij} is stored in ap[(i - 1) × i/2 + j - 1], for i ≥ j.

7: fail – NagError *

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_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.
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_dtfttp (f01vlc) 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, incl = 1, indent = 0, ncols = 80;
    Integer i, j, k, lar1, lar2, lenap, lenar, mx, n, nx, 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 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*(N-1) - I*(I+1)/2)
    #define KU(I,J,N) (I*(N-1) - I*(I-1)/2 + J)
    #endif
    INIT_FAIL(fail);
    printf("nag_dtfttp (f01vlc) 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, double)) || !(ar = NAG_ALLOC(lenar, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Nag_RFP_Store */
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++) {
#ifdef _WIN32
    for (j = 0; j < lar2; j++) scanf_s("%lf ", &AR(i,j));
#else
    for (j = 0; j < lar2; j++) scanf("%lf ", &AR(i,j));
#endif
}

/* Print the packed Rectangular Full Packed array */
if (order==Nag_RowMajor) {
    nx = lenar;
} else {
    mx = lenar;
    nx = incl1;
}
nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mx, nx, ar, lenar, 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_real_mat_print_comp (x04dbc).
    fail.message); exit_status = 1;
}

/* Convert real triangular matrix from Rectangular Full Packed to *
 * packed vector form using nag_dfttp (f01vlc).
 */
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dfttp (f01vlc).\n\n", fail.message);
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_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mx, nx,
                            ap, lenap, form, "Packed Array 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");

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

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

10.2 Program Data
nag_dtfttp (f01vlc) Example Program Data

4
Nag_RFP_Normal Nag_Upper : transr, uplo
1.30 1.40
2.30 2.40
3.30 3.40
1.10 4.40
1.20 2.20 : RFP array ar[]

10.3 Program Results
nag_dtfttp (f01vlc) Example Program Results

RFP Packed Array AR:
1 1.30 1.40 2.30 2.40 3.30 3.40 1.10 4.40 1.20 2.20

Packed Array AP:
1 1.10 1.20 1.30 1.40 2.20 2.30 2.40 3.30 3.40 4.40

Packed Matrix AP (printed using KL/KU macros):
<table>
<thead>
<tr>
<th>1.10</th>
<th>1.20</th>
<th>1.30</th>
<th>1.40</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.20</td>
<td>2.30</td>
<td>2.40</td>
<td></td>
</tr>
<tr>
<td>3.30</td>
<td>3.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.40</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>