NAG Library Function Document  
nag_dtrttf (f01vec)

1 Purpose

nag_dtrttf (f01vec) copies a real triangular matrix, stored in a full format array, to a Rectangular Full Packed (RFP) format array.

2 Specification

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

void nag_dtrttf (Nag_OrderType order, Nag_RFP_Store transr,
                Nag_UploType uplo, Integer n, const double a[], Integer pda,
                double ar[], NagError *fail)
```

3 Description

nag_dtrttf (f01vec) packs a real $n$ by $n$ triangular matrix $A$, stored conventionally in a full format array, into 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.

4 References


5 Arguments

1: order – Nag_OrderType  
   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  
   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  
   On entry: specifies whether $A$ is upper or lower triangular.  
   uplo = Nag_Upper  
   $A$ is upper triangular.
\texttt{uplo} = \text{Nag\_Lower}

$A$ is lower triangular.

\textit{Constraint:} \texttt{uplo} = \text{Nag\_Upper} or \text{Nag\_Lower}.

4: \quad \texttt{n} – \text{Integer} \hspace{1cm} \textit{Input}

\textit{On entry:} $n$, the order of the matrix $A$.

\textit{Constraint:} $n \geq 0$.

5: \quad \texttt{a}[$\text{dim}$] – \text{const double} \hspace{1cm} \textit{Input}

\textit{Note:} the dimension, \textit{dim}, of the array \texttt{a} must be at least \texttt{pda} \times \texttt{n}.

\textit{On entry:} the triangular matrix $A$.

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

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

If \texttt{uplo} = \text{Nag\_Upper}, the upper triangular part of $A$ must be stored and the elements of the array below the diagonal are not referenced.

If \texttt{uplo} = \text{Nag\_Lower}, the lower triangular part of $A$ must be stored and the elements of the array above the diagonal are not referenced.

6: \quad \texttt{pda} – \text{Integer} \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) of the matrix $A$ in the array \texttt{a}.

\textit{Constraint:} $\texttt{pda} \geq \max(1, \texttt{n})$.

7: \quad \texttt{ar}[$\texttt{n} \times (\texttt{n} + 1)/2$] – \text{double} \hspace{1cm} \textit{Output}

\textit{On exit:} the upper or lower $n$ by $n$ triangular matrix $A$ (as specified by \texttt{uplo}) in either normal or transposed RFP format (as specified by \texttt{transr}). The storage format is described in Section 3.3.3 in the f07 Chapter Introduction.

8: \quad \texttt{fail} – \text{NagError*} \hspace{1cm} \textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \quad \textbf{Error Indicators and Warnings}

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}

On entry, argument \langle value \rangle had an illegal value.

\textbf{NE\_INT}

On entry, $n = \langle value \rangle$.

\textit{Constraint:} $n \geq 0$.

\textbf{NE\_INT\_2}

On entry, $\texttt{pda} = \langle value \rangle$ and $n = \langle value \rangle$.

\textit{Constraint:} $\texttt{pda} \geq \max(1, \texttt{n})$. 

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 and copies it to RFP format.

10.1 Program Text

/* nag_dtrttf (f01vec) Example Program. *
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 25, 2014. */

#include <stdio.h>
#include <nag.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, pda, pdar, q, lar1, lar2, lenar, mx, n, nx;
    /* Arrays */
    double *a = 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;
    Nag_MatrixType matrix;
    NagError fail;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[J*pda + I]
    #else
    #define A(I, J) a[I*pda + J]
    #endif

    /* Program text */
    exit_status = 0;
    i = 1;
    k = 1;
    pda = 40;
    pdar = 40;
    q = 40;
    lar1 = 40;
    lar2 = 40;
    lenar = 40;
    mx = 40;
    n = 40;
    nx = 40;
    a = 0;
    ar = 0;
    nag_enum_transr[40] = "L";
    nag_enum_uplo[40] = "U";
    form[] = "%5.2f";
    order = Nag_ColMajor;
    transr = Nag_RFP_Store;
    uplo = Nag_Upper;
    matrix = Nag_RFP;
    fail = Nag_Error;
    
    /* Program text */
}
/* Skip heading in data file*/
#if defined _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
pda = n;
lenar = (n * (n + 1))/2;
if (!(a = NAG_ALLOC(pda*n, 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);
/* Read a triangular matrix of order n into array A. */
for (i = 0; i < n; i++) {
    #ifdef _WIN32
        for (j = 0; j < n; j++) scanf_s("%lf", &A(i, j));
    #else
        for (j = 0; j < n; j++) scanf("%lf", &A(i, j));
    #endif
}
/* Print the unpacked matrix A. */
matrix = (uplo == Nag_Upper ? Nag_UpperMatrix : Nag_LowerMatrix);
/* nag_gen_real_mat_print_comp (x04cbc).
   * Print real general matrix (comprehensive).
   */
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 real triangular matrix from full format, a, to
   * Rectangular Full Packed form, ar using nag_dtrttf (f01vec).
   */
nag_dtrttf(order, transr, uplo, n, a, pda, ar, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrttf (f01vec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print the Rectangular Full Packed array */
if (order==Nag_RowMajor) {
    mx = incl;
    nx = lenar;
} else {
    mx = lenar;
}
nx = incl;
}
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,
cols, 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 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).
*/
nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, lar1,
lar2, ar, pdar, form,
"RFP Packed Array AR (graphical representation):",
Nag_IntegerLabels, NULL, Nag_IntegerLabels, NULL,
cols, 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(a);
NAG_FREE(ar);
return exit_status;

10.2 Program Data

nag_dtrttf (f01vec) Example Program Data
4 : n
Nag_RFP_Normal Nag_Upper : transr, uplo
1.1 1.2 1.3 1.4
0.0 2.2 2.3 2.4
0.0 0.0 3.3 3.4
0.0 0.0 0.0 4.4 : Unpacked Matrix A

10.3 Program Results

nag_dtrttf (f01vec) Example Program Results

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

RFP Packed Array AR (graphical representation):
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