NAG Library Function Document

nag_ztrttf (f01vfc)

1 Purpose

nag_ztrttf (f01vfc) copies a complex 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_ztrttf (Nag_OrderType order, Nag_RFP_Store transr,
                 Nag_UploType uplo, Integer n, const Complex a[], Integer pda,
                 Complex ar[], NagError *fail)
```

3 Description

nag_ztrttf (f01vfc) packs a complex $n$ 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
       
       *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:  
   **transr** – Nag_RFP_Store
       
       *Input*

       *On entry:* specifies whether the normal RFP representation of $A$ or its conjugate transpose is stored.

       **transr** = Nag_RFP_Normal
               The RFP representation of the matrix $A$ is stored.

       **transr** = Nag_RFP_ConjTrans
               The conjugate transpose of the RFP representation of the matrix $A$ is stored.

       *Constraint:* **transr** = Nag_RFP_Normal or Nag_RFP_ConjTrans.

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

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

Constraint: n ≥ 0.

5: a[dim] – const Complex

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.

6: pda – Integer

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).

7: ar[n × (n + 1)/2] – Complex

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.

8: 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_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 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_ztrttf (f01vfc) 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;
    Integer i, j, k, pda, pdar, q, lar1, lar2, lenar, mx, n, nx;
    /* Arrays */
    Complex *a = 0, *ar = 0;
    char nag_enum_transr[40], nag_enum_uplo[40], form[] = "%5.2f";
    /* Nag Types */
    Nag_MatrixType matrix;
    Nag_OrderType order;
    Nag_RFP_Store transr;
    Nag_UploType uplo;
    NagError fail;
    
    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[J*pda + I]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[I*pda + J]
    order = Nag_RowMajor;
    #endif

    /*...*/
#endif

INIT_FAIL(fail);

printf("nag_ztrttf (f01vfc) Example Program Results\n\n");
/* Skip heading in data file*/
#endif

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, Complex)) || !(ar = NAG_ALLOC(lenar, Complex))) {
    printf("Allocation failure
");
    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. */
for (i = 0; i < n; i++) {
    #ifdef _WIN32
    for (j = 0; j < n; j++) scanf_s(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
    #else
    for (j = 0; j < n; j++) scanf(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
    #endif
}
/* Print the unpacked array. */
matrix = (uplo == Nag_Upper ? Nag_UpperMatrix : Nag_LowerMatrix);
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive).
 */
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
#endif

/* Convert complex triangular matrix from full format, a, to
 * Rectangular Full Packed form, ar, using nag_ztrttf (f01vfc).
 */
if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztrttf (f01vfc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print the packed Rectangular Full Packed array */
if (order==Nag_RowMajor) {
    mx = inc1;
    nx = lenar;
} else {
    mx = lenar;
    nx = inc1;
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mx,
        nx, ar, lenar, Nag_BracketForm, 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_complx_mat_print_comp (x04dbc).\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_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive).
 */
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix,
        Nag_NonUnitDiag, lar1, lar2, ar, pdar,
        Nag_BracketForm, form,
        "RFP Packed Array AR "
        "(graphical representation):",
        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(ar);
return exit_status;

10.2 Program Data

nag_ztrttf (f01vfc) Example Program Data

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>Nag_RFP_Normal Nag_Upper</td>
</tr>
<tr>
<td>(1.1,1.1) (1.2,1.2) (1.3,1.3) (1.4,1.4)</td>
</tr>
<tr>
<td>(0.0,0.0) (2.2,2.2) (2.3,2.3) (2.4,2.4)</td>
</tr>
<tr>
<td>(0.0,0.0) (0.0,0.0) (3.3,3.3) (3.4,3.4)</td>
</tr>
<tr>
<td>(0.0,0.0) (0.0,0.0) (0.0,0.0) (4.4,4.4)</td>
</tr>
</tbody>
</table>

Unpacked Matrix A:

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>( 1.10, 1.10) ( 1.20, 1.20) ( 1.30, 1.30) ( 1.40, 1.40)</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>( 2.20, 2.20) ( 2.30, 2.30) ( 2.40, 2.40)</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_ztrttf (f01vfc) Example Program Results

Unpacked Matrix A:

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>( 1.10, 1.10) ( 1.20, 1.20) ( 1.30, 1.30) ( 1.40, 1.40)</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>( 2.20, 2.20) ( 2.30, 2.30) ( 2.40, 2.40)</td>
</tr>
</tbody>
</table>
RFP Packed Array AR:
1 ( 1.30, 1.30) ( 1.40, 1.40) ( 2.30, 2.30) ( 2.40, 2.40) ( 3.30, 3.30)
1 ( 3.40, 3.40) ( 1.10,-1.10) ( 4.40, 4.40) ( 1.20,-1.20) ( 2.20,-2.20)

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