NAG Library Function Document

nag_dtr_copy (f16qec)

1 Purpose

nag_dtr_copy (f16qec) copies a real triangular matrix.

2 Specification

```c
#include <nag.h>
#include <nagf16.h>
void nag_dtr_copy (Nag_OrderType order, Nag_UploType uplo,
   Nag_TransType trans, Nag_DiagType diag, Integer n, const double a[],
   Integer pda, double b[], Integer pdb, NagError *fail)
```

3 Description

nag_dtr_copy (f16qec) performs the triangular matrix copy operations

\[ B \leftarrow A \quad \text{or} \quad B \leftarrow A^T \]

where \( A \) and \( B \) are \( n \) by \( n \) real triangular matrices.

4 References


5 Arguments

1: \( \text{order} \) – Nag_OrderType

\textit{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: \( \text{uplo} \) – Nag_UploType

\textit{Input}

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

\texttt{uplo} = Nag_Upper

The upper triangular part of \( A \) is stored.

\texttt{uplo} = Nag_Lower

The lower triangular part of \( A \) is stored.

\textit{Constraint}: \texttt{uplo} = Nag_Upper or Nag_Lower.

3: \( \text{trans} \) – Nag_TransType

\textit{Input}

\textit{On entry}: specifies the operation to be performed.

\texttt{trans} = Nag_NoTrans

\[ B \leftarrow A. \]
trans = Nag_Trans or Nag_ConjTrans
B ← Aᵀ.

Constraint: trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: diag = Nag_DiagType

On entry: specifies whether A has nonunit or unit diagonal elements.

diag = Nag_NonUnitDiag
The diagonal elements are stored explicitly.

diag = Nag_UnitDiag
The diagonal elements are assumed to be 1 and are not referenced.

Constraint: diag = Nag_NonUnitDiag or Nag_UnitDiag.

5: n – Integer

On entry: n, the order of the matrices A and B.

Constraint: n ≥ 0.

6: a[dim] – const double

Note: the dimension, dim, of the array a must be at least max(1, pda × n).

On entry: the n by n 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.

If diag = Nag_UnitDiag, the diagonal elements of A are assumed to be 1, and are not referenced.

7: 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).

8: b[dim] – double

Note: the dimension, dim, of the array b must be at least max(1, pdb × n).

On exit: the n by n triangular matrix B.

If order = Nag_ColMajor, B_{ij} is stored in b[(j - 1) × pdb + i - 1].

If order = Nag_RowMajor, B_{ij} is stored in b[(i - 1) × pdb + j - 1].

If uplo = Nag_Upper and trans = Nag_NoTrans or if uplo = Nag_Lower and trans = Nag_Trans or trans = Nag_ConjTrans, B is upper triangular and the elements of the array below the diagonal are not set.

If uplo = Nag_Lower and trans = Nag_NoTrans or if uplo = Nag_Upper and trans = Nag_Trans or trans = Nag_ConjTrans, B is lower triangular and the elements of the array above the diagonal are not set.
9: **pdb** – Integer
   
   *Input*
   
   *On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **b**.
   
   *Constraint:* \( pdb \geq \max(1, n) \).

10: **fail** – NagError *
   
   *Input/Output*
   
   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, \( n = \langle \text{value} \rangle \).

*Constraint:* \( n \geq 0 \).

**NE_INT_2**

On entry, \( pda = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).

*Constraint:* \( pda \geq \max(1, n) \).

On entry, \( pdb = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).

*Constraint:* \( pdb \geq \max(1, n) \).

**NE_INTERNAL_ERROR**

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**

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001)).

8  **Parallelism and Performance**

Not applicable.

9  **Further Comments**

None.
10 Example

This example copies the lower triangular matrix $A$ to $B$ where

$$A = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 2.0 & 0.0 & 0.0 \\ 3.0 & 3.0 & 3.0 & 0.0 \\ 4.0 & 4.0 & 4.0 & 4.0 \end{pmatrix}. $$

10.1 Program Text

```c
/* nag_dtr_copy (f16qec) Example Program. * 
 * Copyright 2014 Numerical Algorithms Group. 
 * 
 * Mark 8, 2005. */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer exit_status, i, j, n, pda, pdb;
    /* Arrays */
    double *a = 0, *b = 0;
    char nag_enum_arg[40];
    /* Nag Types */
    NagError fail;
    Nag_DiagType diag;
    Nag_MatrixType matrix;
    Nag_OrderType order;
    Nag_TransType trans;
    Nag_UploType uplo;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J-1)*pda +I-1]
    #define B(I, J) b[(J-1)*pdb +I-1]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda+J-1]
    #define B(I, J) b[(I-1)*pdb +J-1]
    order = Nag_RowMajor;
    #endif

    exit_status = 0;
    INIT_FAIL(fail);
    printf("nag_dtr_copy (f16qec) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n ]");
    #else
    scanf("%*[\n ]");
    #endif
    /* Read the problem dimension */
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n ]", &n);
    #else
    scanf("%"NAG_IFMT"%*[\n ]", &n);
    #endif
    ...
/* Read uplo */
#ifdef _WIN32
    scanf_s("%39s*\n ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s*\n ", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
/* Read trans */
#ifdef _WIN32
    scanf_s("%39s*\n ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s*\n ", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value */
trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
/* Read diag */
#ifdef _WIN32
    scanf_s("%39s*\n ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s*\n ", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value */
diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);
pda = n;
pdb = n;
if (n > 0)
{
    /* Allocate memory */
    if (!(a = NAG_ALLOC(n*pda, double)) || (b = NAG_ALLOC(n*pdb, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid n\n");
    exit_status = 1;
    return exit_status;
}
/* Read A from data file */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
        #ifdef _WIN32
            scanf_s("%lf", &A(i, j));
        #else
            scanf("%lf", &A(i, j));
        #endif
    }
    #ifdef _WIN32
        scanf_s("%*\n ");
    #else
        scanf("%*\n ");
    #endif
}
else
{...}
```c
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= i; ++j)
        scanf("%lf", &A(i, j));
}
scanf("%*
[^
] ");
}

nag_dtr_copy(order, uplo, trans, diag, n, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dtr_copy (f16qec).
%s
", fail.message);
    exit_status = 1;
    goto END;
}

if (uplo == Nag_Upper)
    matrix = Nag_UpperMatrix;
else
    matrix = Nag_LowerMatrix;

fflush(stdout);
nag_gen_real_mat_print(order, matrix, Nag_NonUnitDiag, n, n, b, pdb, "Copy of Input Matrix", 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).
%s
", fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(a);
NAG_FREE(b);
return exit_status;
```

**10.2 Program Data**

nag_dtr_copy (f16qec) Example Program Data

4 :Value of n
Nag_Lower :Value of uplo
Nag_NoTrans :Value of trans
Nag_NonUnitDiag :Value of diag
1.0
2.0 2.0
3.0 3.0 3.0
4.0 4.0 4.0 4.0 :End of matrix A
## 10.3 Program Results

nag_dtr_copy (f16qec) Example Program Results

<table>
<thead>
<tr>
<th>Copy of Input Matrix</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.0000</td>
<td>2.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.0000</td>
<td>3.0000</td>
<td>3.0000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.0000</td>
<td>4.0000</td>
<td>4.0000</td>
<td>4.0000</td>
</tr>
</tbody>
</table>