

NAG Library Function Document

nag_dtr_copy (f16qec)

1 Purpose

nag_dtr_copy (f16qec) copies a real triangular matrix.

2 Specification

```
#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

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee <http://www.netlib.org/blas/blast-forum/blas-report.pdf>

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 the upper or lower triangular part of A is stored.
uplo = Nag_Upper
 The upper triangular part of A is stored.
uplo = Nag_Lower
 The lower triangular part of A is stored.
Constraint: **uplo** = Nag_Upper or Nag_Lower.
- 3: **trans** – Nag_TransType *Input*
On entry: specifies the operation to be performed.
trans = Nag_NoTrans
 $B \leftarrow A$.

trans = Nag_Trans or Nag_ConjTrans
 $B \leftarrow A^T$.

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: **diag** – Nag_DiagType *Input*

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 *Input*

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

Constraint: $n \geq 0$.

6: **a**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.

On entry: the n by n triangular matrix A .

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

If **order** = Nag_RowMajor, A_{ij} is stored in **a**[($i - 1$) \times **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 *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: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

8: **b**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **b** must be at least $\max(1, \mathbf{pdb} \times \mathbf{n})$.

On exit: the n by n triangular matrix B .

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

If **order** = Nag_RowMajor, B_{ij} is stored in **b**[($i - 1$) \times **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 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 $\langle value \rangle$ had an illegal value.

NE_INT

On entry, **n** = $\langle value \rangle$.
 Constraint: **n** \geq 0.

NE_INT_2

On entry, **pda** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: **pda** \geq max(1, **n**).
 On entry, **pdb** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 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 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

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

nag_dtr_copy (f16qec) is not threaded in any implementation.

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

```

/* nag_dtr_copy (f16qec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + 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, (unsigned)_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, (unsigned)_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, (unsigned)_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 {
        for (i = 1; i <= n; ++i) {
            for (j = 1; j <= i; ++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
    }

/* nag_dtr_copy (f16qec).
 * Triangular matrix copy.
 *
 */
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).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print output */
/* nag_gen_real_mat_print (x04cac).
 * Print real general matrix (easy-to-use)
 */
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).\n%s\n", 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

Copy of Input Matrix

	1	2	3	4
1	1.0000			
2	2.0000	2.0000		
3	3.0000	3.0000	3.0000	
4	4.0000	4.0000	4.0000	4.0000
