

## NAG Library Function Document

### nag\_dge\_copy (f16qfc)

#### 1 Purpose

nag\_dge\_copy (f16qfc) copies a real general matrix.

#### 2 Specification

```
#include <nag.h>
#include <nagf16.h>

void nag_dge_copy (Nag_OrderType order, Nag_TransType trans, Integer m,
                  Integer n, const double a[], Integer pda, double b[], Integer pdb,
                  NagError *fail)
```

#### 3 Description

nag\_dge\_copy (f16qfc) performs the matrix-copy operation

$$B \leftarrow A \quad \text{or} \quad B \leftarrow A^T$$

where  $A$  and  $B$  are  $m$  by  $n$  real rectangular 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 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: **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.
- 3: **m** – Integer *Input*  
*On entry:*  $m$ , the number of rows of the matrix  $A$ .  
*Constraint:*  $m \geq 0$ .

- 4: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $A$ .  
*Constraint:*  $n \geq 0$ .
- 5: **a**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least  
 $\max(1, \mathbf{pda} \times \mathbf{n})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{m} \times \mathbf{pda})$  when **order** = Nag\_RowMajor.  
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** +  $j - 1$ ].  
*On entry:* the  $m$  by  $n$  general matrix  $A$ .
- 6: **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:*  $\mathbf{pda} \geq \max(1, \mathbf{m})$ .
- 7: **b**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **b** must be at least  
 $\max(1, \mathbf{pdb} \times \mathbf{n})$  when **trans** = Nag\_NoTrans and **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{m} \times \mathbf{pdb})$  when **trans** = Nag\_NoTrans and **order** = Nag\_RowMajor;  
 $\max(1, \mathbf{pdb} \times \mathbf{m})$  when **trans** = Nag\_Trans or Nag\_ConjTrans and **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdb})$  when **trans** = Nag\_Trans or Nag\_ConjTrans and **order** = Nag\_RowMajor.  
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** +  $j - 1$ ].  
*On exit:* the matrix  $B$ ;  $B$  is  $n$  by  $k$  if **trans** = Nag\_NoTrans, or  $k$  by  $n$  otherwise.
- 8: **pdb** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) of the matrix  $B$  in the array **b**.  
*Constraints:*  
if **trans** = Nag\_NoTrans,  $\mathbf{pdb} \geq \max(1, \mathbf{m})$ ;  
if **trans** = Nag\_Trans or Nag\_ConjTrans,  $\mathbf{pdb} \geq \max(1, \mathbf{n})$ .
- 9: **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\_ENUM\_INT\_2**

On entry, **trans** =  $\langle value \rangle$ , **pdb** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$ .

Constraint: if **trans** = Nag\_NoTrans, **pdb**  $\geq$  max(1, **m**).

On entry, **trans** =  $\langle value \rangle$ , **pdb** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: if **trans** = Nag\_Trans or Nag\_ConjTrans, **pdb**  $\geq$  max(1, **n**).

**NE\_INT**

On entry, **m** =  $\langle value \rangle$ .

Constraint: **m**  $\geq$  0.

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

Constraint: **n**  $\geq$  0.

**NE\_INT\_2**

On entry, **pda** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$ .

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

**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**

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 a 4 by 3 real general matrix *A* to the matrix *B*.

**10.1 Program Text**

```

/* nag_dge_copy (f16qfc) 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      bdim1, bdim2, exit_status, i, j, m, n, pda, pdb;

    /* Arrays */
    double       *a = 0, *b = 0;
    char         nag_enum_arg[40];

    /* Nag Types */
    NagError     fail;
    Nag_OrderType order;
    Nag_TransType trans;

#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_dge_copy (f16qfc) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read the problem dimensions */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT%"NAG_IFMT"%*[\n] ", &m, &n);
#else
    scanf("%"NAG_IFMT%"NAG_IFMT"%*[\n] ", &m, &n);
#endif
    /* 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_enum_name_to_value(nag_enum_arg);

    if (order == Nag_ColMajor)
    {
        pda = m;
        if (trans == Nag_NoTrans)
        {
            {
                pdb = m;
                bdim1 = pdb;
                bdim2 = n;
            }
        }
        else
        {
            {
                pdb = n;
                bdim1 = pdb;
                bdim2 = m;
            }
        }
    }
}

```

```

else
{
    pda = n;
    if (trans == Nag_NoTrans)
    {
        pdb = n;
        bdim1 = m;
        bdim2 = pdb;
    }
    else
    {
        pdb = m;
        bdim1 = n;
        bdim2 = pdb;
    }
}

if (m > 0 && n > 0)
{
    /* Allocate memory */
    if (!(a = NAG_ALLOC(m*n, double)) ||
        !(b = NAG_ALLOC(m*n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid m or n\n");
    exit_status = 1;
    return exit_status;
}

/* Read A from data file */
for (i = 1; i <= m; ++i)
{
    for (j = 1; 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

/* nag_dge_copy (f16qfc).
 * General matrix copy.
 */
nag_dge_copy(order, trans, m, n, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_copy (f16qfc).\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)
 */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                        bdim1, bdim2, 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_dge_copy (f16qfc) Example Program Data
4 3                               :Values of m, n
Nag_NoTrans                       :Value of trans
1.1    1.2    1.3
2.1    2.2    2.3
3.1    3.2    3.3
4.1    4.2    4.3                :End of matrix A

```

## 10.3 Program Results

```

nag_dge_copy (f16qfc) Example Program Results

```

```

Copy of Input Matrix
      1      2      3
1    1.1000  1.2000  1.3000
2    2.1000  2.2000  2.3000
3    3.1000  3.2000  3.3000
4    4.1000  4.2000  4.3000

```

---