

NAG C Library Function Document

nag_zge_copy (f16tfc)

1 Purpose

nag_zge_copy (f16tfc) copies a complex general matrix.

2 Specification

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

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

3 Description

nag_zge_copy (f16tfc) performs the matrix-copy operation

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

where A and B are m by n complex general matrices.

4 References

The BLAS Technical Forum Standard (2001) www.netlib.org/blas/blast-forum

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 2.2.1.4 of 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**

$$B \leftarrow A^T.$$

trans = **Nag_ConjTrans**

$$B \leftarrow A^H.$$

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 Complex *Input*
Note: the dimension, *dim*, of the array **a** must be at least
 $\max(1, \mathbf{pda} \times \mathbf{n})$ when **order** = **Nag_ColMajor**;
 $\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.
If **order** = **Nag_ColMajor**, the (i,j) th element of the matrix A is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.
If **order** = **Nag_RowMajor**, the (i,j) th element of the matrix A is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.
On entry: the m by n general matrix A .
- 6: **pda** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **a**.
Constraints:
if **order** = **Nag_ColMajor**, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **order** = **Nag_RowMajor**, $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 7: **b**[*dim*] – Complex *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**, the (i,j) th element of the matrix B is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.
If **order** = **Nag_RowMajor**, the (i,j) th element of the matrix B is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.
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 matrix row or column elements (depending on the value of **order**) in the array **b**.
Constraints:
if **order** = **Nag_ColMajor**,
if **trans** = **Nag_NoTrans**, $\mathbf{pdb} \geq \max(1, \mathbf{m})$;
if **trans** = **Nag_Trans** or **Nag_ConjTrans**, $\mathbf{pdb} \geq \max(1, \mathbf{n})$;
if **order** = **Nag_RowMajor**,
if **trans** = **Nag_NoTrans**, $\mathbf{pdb} \geq \max(1, \mathbf{n})$;
if **trans** = **Nag_Trans** or **Nag_ConjTrans**, $\mathbf{pdb} \geq \max(1, \mathbf{m})$.
- 9: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

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

NE_ENUM_INT_2

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

Constraint: if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pdb** \geq $\max(1, m)$.

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

Constraint: if **trans** = **Nag_NoTrans**, **pdb** \geq $\max(1, n)$.

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 $\max(1, m)$.

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

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

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 consult NAG for assistance.

7 Accuracy

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of The BLAS Technical Forum Standard (2001)).

8 Further Comments

None.

9 Example

This example copies the transpose of a complex valued 4 by 3 matrix, A , to the matrix B .

9.1 Program Text

```

/* nag_zge_copy (f16tfc) Example Program.
 *
 * Copyright 2005 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 */
    Complex *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);
    Vprintf( "nag_zge_copy (f16tfc) Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[\n] ");
    /* Read the problem dimensions */
    Vscanf("%ld%ld%*[\n] ", &m, &n);
    /* Read trans */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* 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, Complex)) ||
      !(b = NAG_ALLOC(m*n, Complex)) )
  {
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
}
else
{
  Vprintf("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)
Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
}
Vscanf("%*[\n] ");

/* nag_zge_copy(f16tfc).
 * Complex valued general matrix copy.
 */
nag_zge_copy(order, trans, m, n, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from zge_copy.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print output */
/* nag_gen_complx_mat_print (x04dac).
 * Print Complex general matrix (easy-to-use)
 */
nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
  bdim1, bdim2, b, pdb,
  "Copy of Transposed Input Matrix", 0, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
    fail.message);
  exit_status = 1;
  goto END;
}

END:
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);

return exit_status;
}

```

9.2 Program Data

```
nag_zge_copy (f16tfc) Example Program Data
  4 3                               :Values of m, n
  Nag_Trans                          :Value of trans
  ( 1.0, 1.0) ( 1.0, 2.0) ( 1.0, 3.0)
  ( 2.0, 1.0) ( 2.0, 2.0) ( 2.0, 3.0)
  ( 3.0, 1.0) ( 3.0, 2.0) ( 3.0, 3.0)
  ( 4.0, 1.0) ( 4.0, 2.0) ( 4.0, 3.0) :End of matrix A
```

9.3 Program Results

```
nag_zge_copy (f16tfc) Example Program Results
```

```
Copy of Transposed Input Matrix
```

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