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

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

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 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: **trans** – Nag_TransType *Input*

On entry: specifies the operation to be performed.

trans = Nag_NoTrans
$B \leftarrow A$.

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

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** ≥ 0.

4: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

Constraint: $\mathbf{n} \geq 0$.

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 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = Nag_RowMajor, A_{ij} 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 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] – 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 $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = Nag_RowMajor, B_{ij} 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 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 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_2

On entry, **trans** = ⟨*value*⟩, **m** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **trans** = Nag_Trans or Nag_ConjTrans, **pdb** $\geq \max(1, \mathbf{m})$.

On entry, **trans** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **trans** = Nag_NoTrans, **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **trans** = ⟨*value*⟩, **pdb** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: if **trans** = Nag_NoTrans, **pdb** $\geq \max(1, \mathbf{m})$.

On entry, **trans** = ⟨*value*⟩, **pdb** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: if **trans** = Nag_Trans or Nag_ConjTrans, **pdb** $\geq \max(1, \mathbf{n})$.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pda** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: **pda** $\geq \max(1, \mathbf{m})$.

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.

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 2.7.6 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 2.7.5 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_dge_copy (f16qfc) is not threaded in any implementation.

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.
 *
 * 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 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]
  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_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, (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);

  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
```