

NAG Library Function Document

nag_dger (f16pmc)

1 Purpose

nag_dger (f16pmc) performs a rank-1 update on a real general matrix.

2 Specification

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

void nag_dger (Nag_OrderType order, Nag_ConjType conj, Integer m, Integer n,
              double alpha, const double x[], Integer incx, const double y[],
              Integer incy, double beta, double a[], Integer pda, NagError *fail)
```

3 Description

nag_dger (f16pmc) performs the rank-1 update operation

$$A \leftarrow \alpha xy^T + \beta A,$$

where A is an m by n real matrix, x is an m element real vector, y is an n -element real vector, and α and β are real scalars. If m or n is equal to zero or if β is equal to one and α is equal to zero, this function returns immediately.

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: **conj** – Nag_ConjType *Input*
On entry: the argument **conj** is not referenced if x and y are real vectors. It is suggested that you set **conj** = Nag_NoConj where the elements y_i are not conjugated.
Constraint: **conj** = Nag_NoConj.
- 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: $\mathbf{n} \geq 0$.
- 5: **alpha** – double *Input*
On entry: the scalar α .
- 6: **x**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$.
On entry: the n -element vector x .
If $\mathbf{incx} > 0$, x_i must be stored in $\mathbf{x}[(i - 1) \times \mathbf{incx}]$, for $i = 1, 2, \dots, \mathbf{m}$.
If $\mathbf{incx} < 0$, x_i must be stored in $\mathbf{x}[(\mathbf{m} - i) \times |\mathbf{incx}|]$, for $i = 1, 2, \dots, \mathbf{m}$.
Intermediate elements of **x** are not referenced. If $\mathbf{m} = 0$, **x** is not referenced and may be **NULL**.
- 7: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: $\mathbf{incx} \neq 0$.
- 8: **y**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **y** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incy}|)$.
On entry: the n -element vector y .
If $\mathbf{incy} > 0$, y_i must be stored in $\mathbf{y}[(i - 1) \times \mathbf{incy}]$, for $i = 1, 2, \dots, \mathbf{n}$.
If $\mathbf{incy} < 0$, y_i must be stored in $\mathbf{y}[(\mathbf{n} - i) \times |\mathbf{incy}|]$, for $i = 1, 2, \dots, \mathbf{n}$.
Intermediate elements of **y** are not referenced. If $\alpha = 0.0$ or $\mathbf{n} = 0$, **y** is not referenced and may be **NULL**.
- 9: **incy** – Integer *Input*
On entry: the increment in the subscripts of **y** between successive elements of y .
Constraint: $\mathbf{incy} \neq 0$.
- 10: **beta** – double *Input*
On entry: the scalar β .
- 11: **a**[*dim*] – double *Input/Output*
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 matrix A .
On exit: the updated matrix A .
- 12: **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, **pda** \geq max(1, **m**);
 if **order** = Nag_RowMajor, **pda** \geq **n**.

13: **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 *⟨value⟩* had an illegal value.

NE_INT

On entry, **incx** = *⟨value⟩*.

Constraint: **incx** \neq 0.

On entry, **incy** = *⟨value⟩*.

Constraint: **incy** \neq 0.

On entry, **m** = *⟨value⟩*.

Constraint: **m** \geq 0.

On entry, **n** = *⟨value⟩*.

Constraint: **n** \geq 0.

NE_INT_2

On entry, **pda** = *⟨value⟩*, **m** = *⟨value⟩*.

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

On entry, **pda** = *⟨value⟩* and **n** = *⟨value⟩*.

Constraint: **pda** \geq **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 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_dger (f16pmc) is not threaded in any implementation.

9 Further Comments

The argument `conj` is not referenced in this case where x and y are real vectors.

10 Example

Perform rank-1 update of real matrix A using vectors x and y :

$$A \leftarrow A - xy^T,$$

where A is the 3 by 2 matrix given by

$$A = \begin{pmatrix} 3.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 9.0 \end{pmatrix},$$

$$x = (2.0, 3.0, 5.0)^T \quad \text{and} \quad y = (0.0, 1.0, 0.0)^T.$$

10.1 Program Text

```

/* nag_dger (f16pmc) 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 */
    double alpha, beta;
    Integer exit_status, i, incx, incy, j, m, n, pda, xlen, ylen;

    /* Arrays */
    double *a = 0, *x = 0, *y = 0;

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_ConjType conj;

#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;
    conj = Nag_NoConj;
    INIT_FAIL(fail);

    printf("nag_dger (f16pmc) 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 scalar parameters */
#ifdef _WIN32
    scanf_s("%lf%lf%*[\n] ", &alpha, &beta);
#else
    scanf("%lf%lf%*[\n] ", &alpha, &beta);
#endif
/* Read increment parameters */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy);
#endif

#ifdef NAG_COLUMN_MAJOR
    pda = m;
#else
    pda = n;
#endif

xlen = MAX(1, 1 + (m - 1) * ABS(incx));
ylen = MAX(1, 1 + (n - 1) * ABS(incy));

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

/* Input matrix A and vectors x and y */

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
}
for (i = 0; i < xlen; ++i)
#ifdef _WIN32
    scanf_s("%lf%*[\n] ", &x[i]);

```

```

#else
    scanf("%lf%*[\n] ", &x[i]);
#endif
    for (i = 0; i < ylen; ++i)
#ifdef _WIN32
        scanf_s("%lf%*[\n] ", &y[i]);
#else
        scanf("%lf%*[\n] ", &y[i]);
#endif

    /* nag_dger (f16pmc).
     * Rank one update of real matrix.
     */
    nag_dger(order, conj, m, n, alpha, x, incx, y, incy, beta, a, pda, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dger.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print updated matrix A */
    /* 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, m,
                           n, a, pda, "Updated Matrix A", 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(x);
    NAG_FREE(y);

    return exit_status;
}

```

10.2 Program Data

```

nag_dger (f16pmc) Example Program Data
  3 2                : m, n the dimensions of matrix A
-1.0 1.0            : alpha, beta
  1 2                : incx, incy
  3.0 2.0
  3.0 4.0
  5.0 9.0           : the end of matrix A
  2.0
  3.0
  5.0                : the end of vector x
  1.0
  0.0
  1.0
  0.0                : the end of vector y

```

10.3 Program Results

nag_dger (f16pmc) Example Program Results

```

Updated Matrix A
      1      2
  1  1.0000  0.0000
  2  0.0000  1.0000
  3  0.0000  4.0000

```
