

# NAG Library Function Document

## nag\_zger (f16smc)

### 1 Purpose

nag\_zger (f16smc) performs a rank-1 update on a complex general matrix.

### 2 Specification

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

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

### 3 Description

nag\_zger (f16smc) performs the rank-1 update operation

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

or

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

where  $A$  is an  $m$  by  $n$  complex matrix,  $x$  is an  $m$  element complex vector,  $y$  is an  $n$ -element complex vector, and  $\alpha$  and  $\beta$  are complex scalars.

### 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** specifies whether the elements  $y_i$  are used unconjugated or conjugated, as follows:

**conj** = Nag\_NoConj  
The elements  $y_i$  are not conjugated.

**conj** = Nag\_Conj  
The complex conjugate of the elements  $y_i$  are used.

*Constraint:* **conj** = Nag\_NoConj or Nag\_Conj.

- 3: **m** – Integer *Input*  
*On entry:*  $m$ , the number of rows of the matrix  $A$ .  
*Constraint:*  $\mathbf{m} \geq 0$ .
- 4: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $A$ .  
*Constraint:*  $\mathbf{n} \geq 0$ .
- 5: **alpha** – Complex *Input*  
*On entry:* the scalar  $\alpha$ .
- 6: **x**[*dim*] – const Complex *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 Complex *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** – Complex *Input*  
*On entry:* the scalar  $\beta$ .
- 11: **a**[*dim*] – Complex *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 2.3.1.2 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, **incx** =  $\langle value \rangle$ .

Constraint: **incx**  $\neq$  0.

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

Constraint: **incy**  $\neq$  0.

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** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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 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\_zger (f16smc) is not threaded in any implementation.

## 9 Further Comments

None.

## 10 Example

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

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

where  $A$  is the 3 by 2 complex matrix given by

$$A = \begin{pmatrix} 4.0 + 4.0i & 2.0 + 2.0i \\ 4.0 + 7.0i & 4.0 + 3.0i \\ 11.0 + 3.0i & 9.0 + 7.0i \end{pmatrix},$$

and the vectors  $x$  and  $y$  are

$$x = \begin{pmatrix} 2.0 + 1.0i \\ 3.0 + 2.0i \\ 5.0 - 1.0i \end{pmatrix}$$

and

$$y = \begin{pmatrix} 2.0 + 1.0i \\ 1.0 - 2.0i \end{pmatrix}.$$

The vector  $y$  is stored in every second element of array  $\mathbf{y}$  ( $\mathbf{incy} = 2$ ).

### 10.1 Program Text

```

/* nag_zger (f16smc) 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 */
    Complex alpha, beta;
    Integer exit_status, i, incx, incy, j, m, n, pda, xlen, ylen;

    /* Arrays */
    Complex *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_zger (f16smc) 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.re, &alpha.im);
#else
  scanf(" ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);
#endif
#ifdef _WIN32
  scanf_s(" ( %lf , %lf )%*[\n] ", &beta.re, &beta.im);
#else
  scanf(" ( %lf , %lf )%*[\n] ", &beta.re, &beta.im);
#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, Complex)) ||
        !(x = NAG_ALLOC(xlen, Complex)) || !(y = NAG_ALLOC(ylen, Complex)))
    {
      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 , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
        scanf_s("%*[\n] ");
#else
        scanf("%*[\n] ");
#endif
    }
    for (i = 0; i < xlen; ++i)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )%*[\n] ", &x[i].re, &x[i].im);
#else
        scanf(" ( %lf , %lf )%*[\n] ", &x[i].re, &x[i].im);
#endif
    for (i = 0; i < ylen; ++i)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )%*[\n] ", &y[i].re, &y[i].im);
#else
        scanf(" ( %lf , %lf )%*[\n] ", &y[i].re, &y[i].im);
#endif
}

/* nag_zger (f16smc).
 * Rank one update of complex matrix.
 */
nag_zger(order, conj, m, n, alpha, x, incx, y, incy, beta, a, pda, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zger.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print updated matrix A */
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix,
                              Nag_NonUnitDiag, m, n, a, pda,
                              Nag_BracketForm, "%7.4f",
                              "Updated Matrix A", Nag_IntegerLabels,
                              0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\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_zger (f16smc) Example Program Data
  3 2          : m, n the dimensions of matrix A
(-1.0, 0.0)   : alpha
( 1.0, 0.0)   : beta
  1 2          : incx, incy
( 4.0, 4.0) ( 2.0, 2.0)
( 4.0, 7.0) ( 4.0, 3.0)
(11.0, 3.0) ( 9.0, 7.0) : the end of matrix A
( 2.0, 1.0)
( 3.0, 2.0)
( 5.0,-1.0)   : the end of vector x
( 2.0, 1.0)
( 0.0, 0.0)
( 1.0,-2.0)
( 0.0, 0.0)   : the end of vector y
```

## 10.3 Program Results

nag\_zger (f16smc) Example Program Results

Updated Matrix A

	1	2
1	( 1.0000, 0.0000)	(-2.0000, 5.0000)
2	( 0.0000, 0.0000)	(-3.0000, 7.0000)
3	( 0.0000, 0.0000)	( 6.0000,18.0000)

---