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

$$A \leftarrow \alpha x y^{\mathrm{T}} + \beta A,$$

or

$$A \leftarrow \alpha x y^{\mathrm{H}} + \beta A,$$

where A is an m by n complex matrix, x is an m element complex vector, y is an n-element complex vector, and α and β 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 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: **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: **m** ≥ 0.

4: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

5: **alpha** – Complex *Input*

On entry: the scalar α.

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 vector x.

7: **incx** – Integer *Input*

On entry: the increment in the subscripts of **x** between successive elements of x.

Constraint: **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 vector y.

9: **incy** – Integer *Input*

On entry: the increment in the subscripts of **y** between successive elements of y.

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

10: **beta** – Complex *Input*

On entry: the scalar β.

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

If **order** = 'Nag_RowMajor', A_{ij} is stored in **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** $\geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, **pda** $\geq \mathbf{n}$.

13: **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.

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** ≥ 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** $\geq$ **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.

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

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

$$A \leftarrow A - xy^{\mathrm{H}},$$

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 **y** (**incy** $= 2$).

10.1 Program Text

```
/* nag_zger (f16smc) 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 */
  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 */
  scanf("%*[^\n] ");

  /* Read the problem dimensions */
  scanf("%ld%ld%*[^\n] ", &m, &n);
```

```
  /* Read scalar parameters */
  scanf(" ( %lf , %lf )%*[^\n] ", &alpha.re, &alpha.im);
  scanf(" ( %lf , %lf )%*[^\n] ", &beta.re, &beta.im);
  /* Read increment parameters */
  scanf("%ld%ld%*[^\n] ", &incx, &incy);

#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)
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
      scanf("%*[^\n] ");
    }
  for (i = 0; i < xlen; ++i)
    scanf(" ( %lf , %lf )%*[^\n] ", &x[i].re, &x[i].im);
  for (i = 0; i < ylen; ++i)
    scanf(" ( %lf , %lf )%*[^\n] ", &y[i].re, &y[i].im);

  /* 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)
```