

NAG C Library Function Document

nag_zgemv (f16sac)

1 Purpose

nag_zgemv (f16sac) performs matrix-vector multiplication for a complex general matrix.

2 Specification

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

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

3 Description

nag_zgemv (f16sac) performs one of the matrix-vector operations

$$y \leftarrow \alpha Ax + \beta y, \quad y \leftarrow \alpha A^T x + \beta y \quad \text{or} \quad y \leftarrow \alpha A^H x + \beta y$$

where A is an m by n complex matrix, x and y are complex vectors, and α and β are complex scalars.

If $m = 0$ or $n = 0$, no operation is performed.

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

$$y \leftarrow \alpha Ax + \beta y.$$

trans = Nag_Trans

$$y \leftarrow \alpha A^T x + \beta y.$$

trans = Nag_ConjTrans

$$y \leftarrow \alpha A^H x + \beta y.$$

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: **alpha** – Complex *Input*
On entry: the scalar α .
- 6: **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 **a**[($j - 1$) \times **pda** + $i - 1$].
If **order** = **Nag_RowMajor**, the (i, j) th element of the matrix A is stored in **a**[($i - 1$) \times **pda** + $j - 1$].
On entry: the m by n matrix A .
- 7: **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**, $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 8: **x**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **x** must be at least
 $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$ when **trans** = **Nag_NoTrans**;
 $\max(1, 1 + (\mathbf{m} - 1)|\mathbf{incx}|)$ when **trans** = **Nag_Trans** or **Nag_ConjTrans**.
On entry: the vector x .
- 9: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: $\mathbf{incx} \neq 0$.
- 10: **beta** – Complex *Input*
On entry: the scalar β .
- 11: **y**[*dim*] – Complex *Input/Output*
Note: the dimension, *dim*, of the array **y** must be at least
 $\max(1, 1 + (\mathbf{m} - 1)|\mathbf{incy}|)$ when **trans** = **Nag_NoTrans**;
 $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incy}|)$ when **trans** = **Nag_Trans** or **Nag_ConjTrans**.
On entry: the vector y .
If **beta** = 0, **y** need not be set.
On exit: the updated vector y .

- 12: **incy** – Integer *Input*
On entry: the increment in the subscripts of **y** between successive elements of **y**.
Constraint: **incy** \neq 0.
- 13: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

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

NE_INT

On entry, **inex** = $\langle value \rangle$.
Constraint: **inex** \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** = $\langle value \rangle$, **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

To compute the matrix-vector product

$$y = \alpha Ax + \beta y$$

where

$$A = \begin{pmatrix} 1.0 + 1.0i & 1.0 + 2.0i \\ 2.0 + 1.0i & 2.0 + 2.0i \\ 3.0 + 1.0i & 3.0 + 2.0i \end{pmatrix},$$

$$x = \begin{pmatrix} 1.0 - 1.0i \\ 2.0 - 2.0i \end{pmatrix},$$

$$y = \begin{pmatrix} -3.5 - 0.5i \\ -4.5 + 1.5i \\ -5.5 + 3.5i \end{pmatrix},$$

$$\alpha = 1.0 + 0.0i \quad \text{and} \quad \beta = 2.0 + 0.0i.$$

9.1 Program Text

```

/* nag_zgemv (f16sac) 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;
    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);

    Vprintf( "nag_zgemv (f16sac) Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^\\n] ");

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

    /* Read the transpose parameter */
    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);
    /* Read scalar parameters */
    Vscanf(" ( %lf , %lf ) ( %lf , %lf )%*[\n] ",
           &alpha.re, &alpha.im, &beta.re, &beta.im);
    /* Read increment parameters */
    Vscanf("%ld%ld%*[\n] ", &incx, &incy);

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

    if (trans == Nag_NoTrans)
    {
        xlen = MAX(1, 1 + (n - 1)*ABS(incx));
        ylen = MAX(1, 1 + (m - 1)*ABS(incy));
    }
    else
    {
        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)) )
        {
            Vprintf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else
    {
        Vprintf("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)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
        Vscanf("%*[\n] ");
    }
    for (i = 1; i <= xlen; ++i)
        Vscanf(" ( %lf , %lf )%*[\n] ", &x[i - 1].re, &x[i - 1].im);
    for (i = 1; i <= ylen; ++i)
        Vscanf(" ( %lf , %lf )%*[\n] ", &y[i - 1].re, &y[i - 1].im);

    /* nag_zgemv(f16sac).
    * Complex valued matrix-vector multiply.
    */
    nag_zgemv(order, trans, m, n, alpha, a, pda, x, incx,
              beta, y, incy, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from nag_zgemv.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

```

```

    }

    /* Print output vector y */
    Vprintf("%s\n", " y");
    for (i = 1; i <= ylen; ++i)
    {
        Vprintf("(%11f,%11f)\n", y[i-1].re, y[i-1].im);
    }

    END:
    if (a) NAG_FREE(a);
    if (x) NAG_FREE(x);
    if (y) NAG_FREE(y);

    return exit_status;
}

```

9.2 Program Data

```

nag_zgemv (f16sac) Example Program Data
  3 2          : m, n the dimensions of matrix A
  Nag_NoTrans : trans
  ( 1.0, 0.0) ( 2.0, 0.0) : alpha, beta
  1 1          : incx, incy
  ( 1.0, 1.0) ( 1.0, 2.0)
  ( 2.0, 1.0) ( 2.0, 2.0)
  ( 3.0, 1.0) ( 3.0, 2.0) : the end of matrix A
  ( 1.0,-1.0)
  ( 2.0,-2.0)             : the end of vector x
  (-3.5,-0.5)
  (-4.5, 1.5)
  (-5.5, 3.5)           : the end of vector y

```

9.3 Program Results

```

nag_zgemv (f16sac) Example Program Results

  Y
  ( 1.000000, 1.000000)
  ( 2.000000, 2.000000)
  ( 3.000000, 3.000000)

```
