

NAG C Library Function Document

nag_zhemm (f16zcc)

1 Purpose

nag_zhemm (f16zcc) performs matrix-matrix multiplication for a complex Hermitian matrix.

2 Specification

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

```
void nag_zhemm (Nag_OrderType order, Nag_SideType side, Nag_UploType uplo,
               Integer m, Integer n, Complex alpha, const Complex a[], Integer pda,
               const Complex b[], Integer pdb, Complex beta, Complex c[], Integer pd,
               NagError *fail)
```

3 Description

nag_zhemm (f16zcc) performs one of the matrix-matrix operations

$$C \leftarrow \alpha AB + \beta C \quad \text{or} \quad C \leftarrow \alpha BA + \beta C$$

where A is a complex Hermitian matrix, B and C are m by n complex matrices, and α and β are complex scalars.

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: **side** – Nag_SideType *Input*
On entry: specifies whether B is operated on from the left or the right.
side = Nag_LeftSide
 B is pre-multiplied from the left.
side = Nag_RightSide
 B is post-multiplied from the right.
Constraint: **side = Nag_LeftSide** or **Nag_RightSide**.
- 3: **uplo** – Nag_UploType *Input*
On entry: specifies whether the upper or lower triangular part of A is stored.
uplo = Nag_Upper
The upper triangular part of A is stored.

uplo = Nag_Lower

The lower triangular part of A is stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

- 4: **m** – Integer *Input*
On entry: m , the number of rows of the matrices B and C ; the order of A if **side** = Nag_LeftSide.
Constraint: $m \geq 0$.
- 5: **n** – Integer *Input*
On entry: n , the number of columns of the matrices B and C ; the order of A if **side** = Nag_RightSide.
Constraint: $n \geq 0$.
- 6: **alpha** – Complex *Input*
On entry: the scalar α .
- 7: **a**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **a** must be at least
 $\max(1, \mathbf{pda} \times \mathbf{m})$ when **side** = Nag_LeftSide;
 $\max(1, \mathbf{pda} \times \mathbf{n})$ when **side** = Nag_RightSide.
If **order** = Nag_ColMajor, the (i, j) th element of the matrix A is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.
If **order** = Nag_RowMajor, the (i, j) th element of the matrix A is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.
On entry: the Hermitian matrix A ; A is m by m if **side** = Nag_LeftSide, or n by n if **side** = Nag_RightSide.
If **uplo** = Nag_Upper, the upper triangle of A must be stored and the elements of the array below the diagonal are not referenced.
If **uplo** = Nag_Lower, the lower triangle of A must be stored and the elements of the array above the diagonal are not referenced.
- 8: **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 **side** = Nag_LeftSide, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **side** = Nag_RightSide, $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 9: **b**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **b** must be at least
 $\max(1, \mathbf{pdb} \times \mathbf{n})$ when **order** = Nag_ColMajor;
 $\max(1, \mathbf{pdb} \times \mathbf{m})$ when **order** = Nag_RowMajor.
If **order** = Nag_ColMajor, the (i, j) th element of the matrix B is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.
If **order** = Nag_RowMajor, the (i, j) th element of the matrix B is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.
On entry: the m by n matrix B .
- 10: **pdb** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **b**.

Constraints:

if **order** = **Nag_ColMajor**, **pdb** \geq $\max(1, \mathbf{m})$;
 if **order** = **Nag_RowMajor**, **pdb** \geq $\max(1, \mathbf{n})$.

- 11: **beta** – Complex *Input*
On entry: the scalar β .
- 12: **c**[*dim*] – Complex *Input/Output*
Note: the dimension, *dim*, of the array **c** must be at least
 $\max(1, \mathbf{pdc} \times \mathbf{n})$ when **order** = **Nag_ColMajor**;
 $\max(1, \mathbf{pdc} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.
 If **order** = **Nag_ColMajor**, the (*i,j*)th element of the matrix *C* is stored in $\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$.
 If **order** = **Nag_RowMajor**, the (*i,j*)th element of the matrix *C* is stored in $\mathbf{c}[(i-1) \times \mathbf{pdc} + j - 1]$.
On entry: the *m* by *n* matrix *C*.
 If **beta** = 0, **c** need not be set.
On exit: the updated matrix *C*.
- 13: **pdc** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **c**.
Constraints:
 if **order** = **Nag_ColMajor**, **pdc** \geq $\max(1, \mathbf{m})$;
 if **order** = **Nag_RowMajor**, **pdc** \geq $\max(1, \mathbf{n})$.
- 14: **fail** – NagError * *Input/Output*
 The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

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

NE_ENUM_INT_2

On entry, **side** = $\langle \text{value} \rangle$, **m** = $\langle \text{value} \rangle$, **pda** = $\langle \text{value} \rangle$.
 Constraint: if **side** = **Nag_LeftSide**, **pda** \geq $\max(1, \mathbf{m})$.

On entry, **side** = $\langle \text{value} \rangle$, **n** = $\langle \text{value} \rangle$, **pda** = $\langle \text{value} \rangle$.
 Constraint: if **side** = **Nag_RightSide**, **pda** \geq $\max(1, \mathbf{n})$.

NE_INT

On entry, **m** = $\langle \text{value} \rangle$.
 Constraint: **m** \geq 0.

On entry, **n** = $\langle \text{value} \rangle$.
 Constraint: **n** \geq 0.

NE_INT_2

On entry, **pdb** = $\langle \text{value} \rangle$, **m** = $\langle \text{value} \rangle$.
 Constraint: **pdb** \geq $\max(1, \mathbf{m})$.

On entry, **pdb** = *<value>*, **n** = *<value>*.

Constraint: **pdb** \geq max(1, **n**).

On entry, **pdc** = *<value>*, **m** = *<value>*.

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

On entry, **pdc** = *<value>*, **n** = *<value>*.

Constraint: **pdc** \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-matrix product

$$C = \alpha AB + \beta C$$

where

$$A = \begin{pmatrix} 1.0 + 0.0i & 1.0 + 2.0i & -2.0 + 3.0i \\ 1.0 - 2.0i & 2.0 + 0.0i & 1.0 + 2.0i \\ -2.0 - 3.0i & 1.0 - 2.0i & 3.0 + 0.0i \end{pmatrix},$$

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

$$C = \begin{pmatrix} -3.5 - 0.5i & 1.5 + 2.0i \\ -4.5 + 1.5i & -2.0 + 3.5i \\ -5.5 + 3.5i & 3.0 - 1.5i \end{pmatrix},$$

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

9.1 Program Text

```

/* nag_zhemm (f16zcc) 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, j, m, n, pda, pdb, pdc;

/* Arrays */
Complex *a=0, *b=0, *c=0;
char nag_enum_arg[40];

/* Nag Types */
NagError fail;
Nag_OrderType order;
Nag_SideType side;
Nag_UploType uplo;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
#define B(I,J) b[(J-1)*pdb + I - 1]
#define C(I,J) c[(J-1)*pdc + I - 1]
    order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
#define C(I,J) c[(I-1)*pdc + J - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

    Vprintf( "nag_zhemm (f16zcc) Example Program Results\n\n");

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

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

    /* Read the side parameter */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac).
     * Converts NAG enum member name to value
     */
    side = nag_enum_name_to_value(nag_enum_arg);
    /* Read uplo */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac).
     * Converts NAG enum member name to value
     */
    uplo = 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);

    if (side == Nag_LeftSide)
        pda = m;
    else
        pda = n;
#ifdef NAG_COLUMN_MAJOR
    pdb = m;
    pdc = m;
#else
    pdb = n;
    pdc = n;
#endif

    if (m > 0 && n > 0)
    {
        /* Allocate memory */
        if (side == Nag_LeftSide)

```

```

    {
        if ( !(a = NAG_ALLOC(m*m, Complex)) )
        {
            Vprintf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
else
    {
        if ( !(a = NAG_ALLOC(n*n, Complex)) )
        {
            Vprintf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
if ( !(b = NAG_ALLOC(m*n, Complex)) ||
    !(c = NAG_ALLOC(m*n, 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 */
if (uplo == Nag_Upper)
    {
        for (i = 1; i <= pda; ++i)
        {
            for (j = i; j <= pda; ++j)
                Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
            Vscanf("%*[\n] ");
        }
    }
else
    {
        for (i = 1; i <= pda; ++i)
        {
            for (j = 1; j <= i; ++j)
                Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
            Vscanf("%*[\n] ");
        }
    }

/* Input matrix B */
for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
        Vscanf("%*[\n] ");
    }

/* Input matrix C */
for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &C(i,j).re, &C(i,j).im);
        Vscanf("%*[\n] ");
    }

/* nag_zhemm(f16zcc).
 * Hermitian matrix-matrix multiply.
 *
 */
nag_zhemm(order, side, uplo, m, n, alpha, a, pda,

```

```

        b, pdb, beta, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from nag_zhemm.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print result */
/* nag_gen_complx_mat_print (x04dac).
 * Print Complex general matrix (easy-to-use)
 */
nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                        m, n, c, pdc, "Matrix Matrix Product",
                        0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

END:
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);
if (c) NAG_FREE(c);

return exit_status;
}

```

9.2 Program Data

```

nag_zhemm (f16zcc) Example Program Data
 3 2                :Values of m, n
Nag_LeftSide       : side
Nag_Lower          : uplo
( 1.0, 0.0) ( 2.0, 0.0) : alpha, beta
( 1.0, 0.0)
( 1.0,-2.0) ( 2.0, 0.0)
(-2.0,-3.0) ( 1.0,-2.0) ( 3.0, 0.0) : the end of matrix A
( 1.0,-1.0) ( 1.0, 2.0)
(-2.0, 1.0) ( 2.0,-2.0)
( 3.0,-1.0) (-3.0, 1.0) : the end of matrix B
(-3.5,-0.5) ( 1.5, 2.0)
(-4.5, 1.5) (-2.0, 3.5)
(-5.5, 3.5) ( 3.0,-1.5) : the end of matrix C

```

9.3 Program Results

nag_zhemm (f16zcc) Example Program Results

```

Matrix Matrix Product

```

	1	2
1	-13.0000	13.0000
	6.0000	-3.0000
2	-9.0000	0.0000
	7.0000	-2.0000
3	-7.0000	-1.0000
	8.0000	-13.0000
