

NAG C Library Function Document

nag_zsyrc (f16zuc)

1 Purpose

nag_zsyrc (f16zuc) performs a rank- k update on a complex symmetric matrix.

2 Specification

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

```
void nag_zsyrc (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
               Integer n, Integer k, Complex alpha, const Complex a[], Integer pda,
               Complex beta, Complex c[], Integer pdcc, NagError *fail)
```

3 Description

nag_zsyrc (f16zuc) performs one of the symmetric rank- k update operations

$$C \leftarrow \alpha AA^T + \beta C \quad \text{or} \quad C \leftarrow \alpha A^T A + \beta C$$

where A is a complex matrix, C is an n by n complex symmetric matrix, 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: **uplo** – Nag_UploType *Input*

On entry: specifies whether the upper or lower triangular part of C is stored.

uplo = **Nag_Upper**

The upper triangular part of C is stored.

uplo = **Nag_Lower**

The lower triangular part of C is stored.

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

3: **trans** – Nag_TransType *Input*

On entry: specifies the operation to be performed.

trans = **Nag_NoTrans**

$$C \leftarrow \alpha AA^T + \beta C.$$

trans = Nag_Trans

$$C \leftarrow \alpha A^T A + \beta C.$$

Constraint: **trans** = Nag_NoTrans or Nag_Trans.

4: **n** – Integer *Input*

On entry: *n*, the order of the matrix *C*; the number of rows of *A* if **trans** = Nag_NoTrans, or the number of columns of *A* otherwise.

Constraint: **n** ≥ 0.

5: **k** – Integer *Input*

On entry: *k*, the number of columns of *A* if **trans** = Nag_NoTrans, or the number of rows of *A* otherwise.

Constraint: **k** ≥ 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

$\max(1, \mathbf{pda} \times \mathbf{k})$ when **trans** = Nag_NoTrans and **order** = Nag_ColMajor;
 $\max(1, \mathbf{n} \times \mathbf{pda})$ when **trans** = Nag_NoTrans and **order** = Nag_RowMajor;
 $\max(1, \mathbf{pda} \times \mathbf{n})$ when **trans** = Nag_Trans or Nag_ConjTrans and
order = Nag_ColMajor;
 $\max(1, \mathbf{k} \times \mathbf{pda})$ when **trans** = Nag_Trans or Nag_ConjTrans and
order = Nag_RowMajor.

If **order** = Nag_ColMajor, the (*i*,*j*)th element of the matrix *A* is stored in **a**[(*j* – 1) × **pda** + *i* – 1].

If **order** = Nag_RowMajor, the (*i*,*j*)th element of the matrix *A* is stored in **a**[(*i* – 1) × **pda** + *j* – 1].

On entry: the matrix *A*; *A* is *n* by *k* if **trans** = Nag_NoTrans, or *k* by *n* otherwise.

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 **order** = Nag_ColMajor,
 if **trans** = Nag_NoTrans, **pda** ≥ max(1, **n**);
 if **trans** = Nag_Trans or Nag_ConjTrans, **pda** ≥ max(1, **k**);
 if **order** = Nag_RowMajor,
 if **trans** = Nag_NoTrans, **pda** ≥ max(1, **k**);
 if **trans** = Nag_Trans or Nag_ConjTrans, **pda** ≥ max(1, **n**).

9: **beta** – Complex *Input*

On entry: the scalar β .

10: **c**[*dim*] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **c** must be at least max(1, **pdc** × **n**).

If **order** = Nag_ColMajor, the (*i*,*j*)th element of the matrix *C* is stored in **c**[(*j* – 1) × **pdc** + *i* – 1].

If **order** = Nag_RowMajor, the (*i*,*j*)th element of the matrix *C* is stored in **c**[(*i* – 1) × **pdc** + *j* – 1].

On entry: the *n* by *n* symmetric matrix *C*.

If **uplo** = **Nag_Upper**, the upper triangle of C must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = **Nag_Lower**, the lower triangle of C must be stored and the elements of the array above the diagonal are not referenced.

On exit: the updated matrix C .

11: **pdc** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **c**.

Constraint: **pdc** \geq $\max(1, \mathbf{n})$.

12: **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 $\langle value \rangle$ had an illegal value.

NE_ENUM_INT_2

On entry, **trans** = $\langle value \rangle$, **k** = $\langle value \rangle$, **pda** = $\langle value \rangle$.

Constraint: if **trans** = **Nag_NoTrans**, **pda** \geq $\max(1, \mathbf{k})$.

On entry, **trans** = $\langle value \rangle$, **k** = $\langle value \rangle$, **pda** = $\langle value \rangle$.

Constraint: if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pda** \geq $\max(1, \mathbf{k})$.

On entry, **trans** = $\langle value \rangle$, **n** = $\langle value \rangle$, **pda** = $\langle value \rangle$.

Constraint: if **trans** = **Nag_NoTrans**, **pda** \geq $\max(1, \mathbf{n})$.

On entry, **trans** = $\langle value \rangle$, **n** = $\langle value \rangle$, **pda** = $\langle value \rangle$.

Constraint: if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pda** \geq $\max(1, \mathbf{n})$.

NE_INT

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

Constraint: **k** \geq 0.

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

Constraint: **n** \geq 0.

NE_INT_2

On entry, **pdc** = $\langle value \rangle$, **n** = $\langle value \rangle$.

Constraint: **pdc** \geq $\max(1, \mathbf{n})$.

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

Perform rank- k update of complex symmetric 4 by 4 matrix C using 4 by 2 matrix A ($k = 2$), $C = C - (1.0 - 1.0i)AA^T$, where

$$C = \begin{pmatrix} 4.78 + 1.03i & 2.00 - 0.30i & 2.89 - 1.34i & -1.89 + 1.15i \\ 2.00 - 0.30i & -4.11 - 2.30i & 2.36 - 4.25i & 0.04 - 3.69i \\ 2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.57i & -0.02 + 0.46i \\ -1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 1.91i \end{pmatrix}$$

and

$$A = \begin{pmatrix} 1.7 - 2.3i & -1.8 + 2.4i \\ 2.9 - 2.1i & 1.2 + 1.4i \\ -2.9 + 1.0i & 0.6 + 0.8i \\ 1.5 + 0.9i & -1.4 - 1.7i \end{pmatrix}.$$

9.1 Program Text

```

/* nag_zsyrc (f16zuc) 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 adim1, adim2, exit_status, i, j, k, n, pda, pdc;

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

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_UploType uplo;
    Nag_TransType trans;
    Nag_MatrixType matrix;

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

    exit_status = 0;
    INIT_FAIL(fail);

    Vprintf( "nag_zsyrc (f16zuc) Example Program Results\n\n");

```

```

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

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

/* Read the uplo parameter */
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 the transpose parameter */
Vscanf("%s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value(x04nac), see above. */
trans = nag_enum_name_to_value(nag_enum_arg);
/* Read scalar parameters */
Vscanf(" ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);
Vscanf(" ( %lf , %lf )%*[\n] ", &beta.re, &beta.im);

if (trans == Nag_NoTrans) {
    adim1 = n;
    adim2 = k;
} else {
    adim1 = k;
    adim2 = n;
}

#ifdef NAG_COLUMN_MAJOR
    pda = adim1;
#else
    pda = adim2;
#endif
pdc = n;
if (k > 0 && n > 0)
{
    /* Allocate memory */
    if ( !(a = NAG_ALLOC(k*n, Complex)) ||
        !(c = NAG_ALLOC(n*n, Complex)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    Vprintf("Invalid k or n\n");
    exit_status = 1;
    return exit_status;
}

/* Input matrix A. */
for (i = 1; i <= adim1; ++i)
{
    for (j = 1; j <= adim2; ++j)
        Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
    Vscanf("%*[\n] ");
}
/* Input matrix C. */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &C(i,j).re, &C(i,j).im);
    }
    Vscanf("%*[\n] ");
}
else

```

```

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

/* nag_zsyrc(f16zuc).
 * Rank k update of complex symmetric matrix.
 *
 */
nag_zsyrc(order, uplo, trans, n, k, alpha, a, pda, beta, c, pdc,
          &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from nag_zsyrc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
if (uplo == Nag_Upper)
    {
        matrix = Nag_UpperMatrix;
    }
else
    {
        matrix = Nag_LowerMatrix;
    }
/* Print updated matrix C */
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
nag_gen_complx_mat_print_comp(order, matrix, Nag_NonUnitDiag, n, n, c,
                              pdc, Nag_BracketForm, "%6.2f",
                              "Updated Matrix C", Nag_IntegerLabels,
                              0, Nag_IntegerLabels, 0, 80, 0, 0,
                              &fail);

if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s"
                "\n", fail.message);
        exit_status = 1;
        goto END;
    }
END:
if (a) NAG_FREE(a);
if (c) NAG_FREE(c);

return exit_status;
}

```

9.2 Program Data

```

nag_zsyrc (f16zuc) Example Program Data
 4 2 :Values of n and k
Nag_Lower :Value of uplo
Nag_NoTrans :Value of trans
( -1.0, 1.0) :Value of alpha
( 1.0, 0.0) :Value of beta
( 1.7, -2.3) ( -1.8, 2.4)
( 2.9, -2.1) ( 1.2, 1.4)
( -2.9, 1.0) ( 0.6, 0.8)
( 1.5, 0.9) ( -1.4, -1.7) :End of matrix A
( 4.78, 1.03)
( 2.00,-0.30) (-4.11,-2.30)
( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.57)
(-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-1.91) :End of matrix C

```

9.3 Program Results

nag_zsyrk (f16zuc) Example Program Results

Updated Matrix C

	1	2	3	4
1	(26.16, 12.57)			
2	(17.30, 4.16)	(1.23, 10.00)		
3	(0.15, -15.34)	(-1.72, -21.75)	(1.86, 12.54)	
4	(-10.89, 14.59)	(-2.36, 7.79)	(7.96, -1.02)	(-7.64, -8.86)
