

# NAG Library Function Document

## nag\_zsyrk (f16zuc)

### 1 Purpose

nag\_zsyrk (f16zuc) performs a rank- $k$  update on a complex symmetric matrix.

### 2 Specification

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

void nag_zsyrk (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 pdc, NagError *fail)
```

### 3 Description

nag\_zsyrk (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  $\alpha$  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 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: **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:*  $n \geq 0$ .
- 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:*  $k \geq 0$ .
- 6: **alpha** – Complex *Input*  
*On entry:* the scalar  $\alpha$ .
- 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 and **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{k} \times \mathbf{pda})$  when **trans** = Nag\_Trans and **order** = Nag\_RowMajor.  
If **order** = Nag\_ColMajor,  $A_{ij}$  is stored in **a**[( $j - 1$ )  $\times$  **pda** +  $i - 1$ ].  
If **order** = Nag\_RowMajor,  $A_{ij}$  is stored in **a**[( $i - 1$ )  $\times$  **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 row or column elements (depending on the value of **order**) in the array **a**.  
*Constraints:*  
if **order** = Nag\_ColMajor,  
if **trans** = Nag\_NoTrans, **pda**  $\geq$   $\max(1, \mathbf{n})$ ;  
if **trans** = Nag\_Trans, **pda**  $\geq$   $\max(1, \mathbf{k})$ .;  
if **order** = Nag\_RowMajor,  
if **trans** = Nag\_NoTrans, **pda**  $\geq$   $\max(1, \mathbf{k})$ ;  
if **trans** = Nag\_Trans, **pda**  $\geq$   $\max(1, \mathbf{n})$ .
- 9: **beta** – Complex *Input*  
*On entry:* the scalar  $\beta$ .
- 10: **c**[*dim*] – Complex *Input/Output*  
**Note:** the dimension, *dim*, of the array **c** must be at least  $\max(1, \mathbf{pdc} \times \mathbf{n})$ .

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

If **order** = Nag\_ColMajor,  $C_{ij}$  is stored in  $\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$ .

If **order** = Nag\_RowMajor,  $C_{ij}$  is stored in  $\mathbf{c}[(i-1) \times \mathbf{pdc} + j - 1]$ .

If **uplo** = Nag\_Upper, the upper triangular part of  $C$  must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = Nag\_Lower, the lower triangular part 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 row or column elements (depending on the value of **order**) of the matrix  $C$  in the array  $\mathbf{c}$ .

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

12: **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.

See Section 3.2.1.2 in the Essential Introduction for further information.

### 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,  $\mathbf{pda} \geq \max(1, \mathbf{k})$ .

On entry, **trans** =  $\langle value \rangle$ , **k** =  $\langle value \rangle$ , **pda** =  $\langle value \rangle$ .  
Constraint: if **trans** = Nag\_Trans,  $\mathbf{pda} \geq \max(1, \mathbf{k})$ .

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

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

### NE\_INT

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

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

### NE\_INT\_2

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

### 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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.  
See Section 3.6.6 in the Essential Introduction for further information.

## NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.  
See Section 3.6.5 in the Essential Introduction 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

Not applicable.

## 9 Further Comments

None.

## 10 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

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

### 10.1 Program Text

```

/* nag_zsyrk (f16zuc) Example Program.
 *
 * Copyright 2014 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);

    printf("nag_zsyrc (f16zuc) 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] ", &n, &k);
#else
    scanf("%"NAG_IFMT%"NAG_IFMT"%*[\n] ", &n, &k);
#endif

    /* Read the uplo parameter */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s%*[\n] ", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
    /* Read the transpose parameter */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s%*[\n] ", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac), see above. */
    trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
    /* 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

    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)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("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)
#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
}
/* Input matrix C. */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &C(i, j).re, &C(i, j).im);
#else
            scanf(" ( %lf , %lf )", &C(i, j).re, &C(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &C(i, j).re, &C(i, j).im);
#else

```

```

        scanf(" ( %lf , %lf )", &C(i, j).re, &C(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
    }

/* 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)
{
    printf("Error from nag_zsyrc (f16zuc).\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)
 */
fflush(stdout);
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)
{
    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(c);

return exit_status;
}

```

## 10.2 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

```

### 10.3 Program Results

nag\_zsyrc (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)
```

---