

# NAG Library Function Document

## nag\_zsyr2k (f16zwc)

### 1 Purpose

nag\_zsyr2k (f16zwc) performs a rank- $2k$  update on a complex symmetric matrix.

### 2 Specification

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

```
void nag_zsyr2k (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
                Integer n, Integer k, Complex alpha, const Complex a[], Integer pda,
                Complex b[], Integer pdb, Complex beta, Complex c[], Integer pdc,
                NagError *fail)
```

### 3 Description

nag\_zsyr2k (f16zwc) performs one of the symmetric rank- $2k$  update operations

$$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C \quad \text{or} \quad C \leftarrow \alpha A^T B + \alpha B^T A + \beta C,$$

where  $A$  and  $B$  are complex matrices,  $C$  is an  $n$  by  $n$  complex symmetric matrix, and  $\alpha$  and  $\beta$  are complex scalars.

### 4 References

The BLAS Technical Forum Standard (2001) <http://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 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  
 $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C.$

**trans** = Nag\_Trans

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

*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  $A$  and  $B$  if **trans** = Nag\_NoTrans, or the number of columns of  $A$  and  $B$  otherwise.

*Constraint:*  $n \geq 0$ .

5: **k** – Integer *Input*

*On entry:*  $k$ , the number of columns of  $A$  and  $B$  if **trans** = Nag\_NoTrans, or the number of rows of  $A$  and  $B$  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 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,  $A_{ij}$  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 matrix row or column elements (depending on the value of **order**) in **a**.

*Constraints:*

if **order** = Nag\_ColMajor,

if **trans** = Nag\_NoTrans, **pda**  $\geq$   $\max(1, \mathbf{n})$ ;

if **trans** = Nag\_Trans or Nag\_ConjTrans, **pda**  $\geq$   $\max(1, \mathbf{k})$ ;

if **order** = Nag\_RowMajor,

if **trans** = Nag\_NoTrans, **pda**  $\geq$   $\max(1, \mathbf{k})$ ;

if **trans** = Nag\_Trans or Nag\_ConjTrans, **pda**  $\geq$   $\max(1, \mathbf{n})$ .

9: **b**[*dim*] – Complex *Output*

**Note:** the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{k})$  when **trans** = Nag\_NoTrans and **order** = Nag\_ColMajor;

$\max(1, \mathbf{n} \times \mathbf{pdb})$  when **trans** = Nag\_NoTrans and **order** = Nag\_RowMajor;

$\max(1, \mathbf{pdb} \times \mathbf{n})$  when **trans** = Nag\_Trans or Nag\_ConjTrans and **order** = Nag\_ColMajor;

$\max(1, \mathbf{k} \times \mathbf{pdb})$  when **trans** = Nag\_Trans or Nag\_ConjTrans and **order** = Nag\_RowMajor.

If **order** = Nag\_ColMajor,  $B_{ij}$  is stored in **b**[( $j - 1$ )  $\times$  **pdb** +  $i - 1$ ].

If **order** = Nag\_RowMajor,  $B_{ij}$  is stored in **b**[( $i - 1$ )  $\times$  **pdb** +  $j - 1$ ].

*On exit:* the matrix  $B$ ;  $B$  is  $n$  by  $k$  if **trans** = Nag\_NoTrans, or  $k$  by  $n$  otherwise.

- 10: **pdb** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **b**.  
*Constraints:*  
 if **order** = Nag\_ColMajor,  
   if **trans** = Nag\_NoTrans, **pdb**  $\geq$  max(1, **n**);  
   if **trans** = Nag\_Trans or Nag\_ConjTrans, **pdb**  $\geq$  max(1, **k**);  
 if **order** = Nag\_RowMajor,  
   if **trans** = Nag\_NoTrans, **pdb**  $\geq$  max(1, **k**);  
   if **trans** = Nag\_Trans or Nag\_ConjTrans, **pdb**  $\geq$  max(1, **n**).
- 11: **beta** – Complex *Input*  
*On entry:* the scalar  $\beta$ .
- 12: **c[dim]** – Complex *Input/Output*  
**Note:** the dimension, *dim*, of the array **c** must be at least max(1, **pdc**  $\times$  **n**).  
*On entry:* the *n* by *n* symmetric matrix *C*.  
 If **order** = Nag\_ColMajor,  $C_{ij}$  is stored in **c**[(*j* – 1)  $\times$  **pdc** + *i* – 1].  
 If **order** = Nag\_RowMajor,  $C_{ij}$  is stored in **c**[(*i* – 1)  $\times$  **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*.
- 13: **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 **c**.  
*Constraint:* **pdc**  $\geq$  max(1, **n**).
- 14: **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  $\langle value \rangle$  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, **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, **k**).

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

Constraint: if **trans** = Nag\_NoTrans, **pdb**  $\geq$  max(1, **k**).

On entry, **trans** =  $\langle value \rangle$ , **k** =  $\langle value \rangle$ , **pdb** =  $\langle value \rangle$ .  
 Constraint: if **trans** = Nag\_Trans or Nag\_ConjTrans, **pdb**  $\geq$  max(1, **k**).

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

On entry, **trans** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ , **pdb** =  $\langle value \rangle$ .  
 Constraint: if **trans** = Nag\_NoTrans, **pdb**  $\geq$  max(1, **n**);  
 .

On entry, **trans** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ , **pdb** =  $\langle value \rangle$ .  
 Constraint: if **trans** = Nag\_Trans or Nag\_ConjTrans, **pdb**  $\geq$  max(1, **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, **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 The BLAS Technical Forum Standard (2001)).

**8 Further Comments**

None.

**9 Example**

Perform rank- $2k$  update of complex symmetric 4 by 4 matrix  $C$  using 4 by 2 matrices  $A$  and  $B$ ,  $C = C + (-0.5 + 0.5i)AB^T + (-0.5 - 0.5i)BA^T$ , where

$$C = \begin{pmatrix} 4.78 + 0.00i & 2.00 - 0.30i & 2.89 - 1.34i & -1.89 + 1.15i \\ 2.00 - 0.30i & -4.11 + 0.00i & 2.36 - 4.25i & 0.04 - 3.69i \\ 2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.00i & -0.02 + 0.46i \\ -1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 + 0.00i \end{pmatrix},$$

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

and

$$B = \begin{pmatrix} -0.3 - 1.9i & 2.1 - 1.1i \\ -2.4 + 1.4i & 0.6 - 2.9i \\ -0.2 - 2.9i & -1.5 + 0.1i \\ 3.5 + 0.8i & 2.2 + 3.7i \end{pmatrix}.$$

## 9.1 Program Text

```

/* nag_zsyr2k (f16zwc) 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(int argc, char *argv[])
{
    FILE          *fpin, *fpout;
    char          *outfile = 0;

    /* Scalars */
    Complex       alpha, beta;
    Integer       adim1, adim2, exit_status, i, j, k, 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_UploType  uplo;
    Nag_TransType trans;
    Nag_MatrixType matrix;

#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);

    /* Check for command-line IO options */
    fpin = nag_example_file_io(argc, argv, "-data", NULL);
    fpout = nag_example_file_io(argc, argv, "-results", NULL);
    (void) nag_example_file_io(argc, argv, "-nag_write", &outfile);

    fprintf(fpout, "nag_zsyr2k (f16zwc) Example Program Results\n\n");

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

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

```

```

/* Read the uplo parameter */
fscanf(fpin, "%s%*[\n] ", nag_enum_arg);
/* 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 */
fscanf(fpin, "%s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value(x04nac), see above. */
trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
/* Read scalar parameters */
fscanf(fpin, " ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);
fscanf(fpin, " ( %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
pdb = pda;
pdc = n;
if (k > 0 && n > 0)
{
    /* Allocate memory */
    if (!(a = NAG_ALLOC(k*n, Complex)) ||
        !(b = NAG_ALLOC(k*n, Complex)) ||
        !(c = NAG_ALLOC(n*n, Complex)))
    {
        fprintf(fpout, "Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    fprintf(fpout, "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)
        fscanf(fpin, " ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
    fscanf(fpin, "%*[\n] ");
}
/* Input matrix B. */
for (i = 1; i <= adim1; ++i)
{
    for (j = 1; j <= adim2; ++j)
        fscanf(fpin, " ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    fscanf(fpin, "%*[\n] ");
}
/* Input matrix C. */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)

```

```

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

/* nag_zsyr2k(f16zwc).
 * Rank 2k update of complex symmetric matrix.
 *
 */
nag_zsyr2k(order, uplo, trans, n, k, alpha, a, pda, b, pdb, beta,
           c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_zsyr2k.\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)
 */
if (outfile) fclose(fpout);
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, outfile,
                              &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s"
           "\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);
if (c) NAG_FREE(c);

return exit_status;
}

```

## 9.2 Program Data

```
nag_zsyr2k (f16zwc) Example Program Data
  4  2                               :Values of n and k
  Nag_Lower                           :Value of uplo
  Nag_NoTrans                          :Value of trans
  ( -0.5, 0.5)                         :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
  ( -0.3, -1.9) (  2.1, -1.1)
  ( -2.4,  1.4) (  0.6, -2.9)
  ( -0.2, -2.9) ( -1.5,  0.1)
  (  3.5,  0.8) (  2.2,  3.7)           :End of matrix B
  ( 4.78, 0.00)
  ( 2.00,-0.30) (-4.11, 0.00)
  ( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.00)
  (-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33, 0.00) :End of matrix C
```

## 9.3 Program Results

nag\_zsyr2k (f16zwc) Example Program Results

Updated Matrix C

	1	2	3	4
1	( 6.32, -10.50)			
2	( -5.76, -3.84)	(-11.33, -5.70)		
3	( 3.72, -0.15)	( 11.39, 4.41)	( -5.42, -4.57)	
4	( 9.02, 3.46)	( -2.03, -7.09)	( 2.47, -5.06)	( -2.84, 12.31)

---