NAG C Library Function Document

nag_dsyr2k (f16yrc)

1 Purpose

nag_dsyr2k (f16yrc) performs a rank-2k update on a real symmetric matrix.

2 Specification

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

```
void nag_dsyr2k (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
     Integer n, Integer k, double alpha, const double a[], Integer pda, double b[],
     Integer pdb, double beta, double c[], Integer pdc, NagError *fail)
```

3 Description

nag_dsyr2k (f16yrc) performs one of the symmetric rank-2k update operations

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

where A and B are real matrices, C is an n by n real symmetric matrix, and α and β are real 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**

$C \leftarrow \alpha AB^{\mathrm{T}} + \alpha BA^{\mathrm{T}} + \beta C$.

trans = **Nag_Trans** or **Nag_ConjTrans**

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

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

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: $\mathbf{n} \geq 0$.

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: $\mathbf{k} \geq 0$.

6: **alpha** – double *Input*

On entry: the scalar α.

7: **a**[dim] – const double *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 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

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 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** $\geq \max(1, \mathbf{n})$;
  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] – double *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**, 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 **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 the array **b**.

Constraints:

if **order** = **Nag_ColMajor**,
if **trans** = **Nag_NoTrans**, **pdb** $\geq \max(1, \mathbf{n})$;
if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pdb** $\geq \max(1, \mathbf{k})$;

if **order** = **Nag_RowMajor**,
if **trans** = **Nag_NoTrans**, **pdb** $\geq \max(1, \mathbf{k})$;
if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pdb** $\geq \max(1, \mathbf{n})$.

11: **beta** – double *Input*

On entry: the scalar β.

12: **c**$[dim]$ – double *Input/Output*

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

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

If **order** = **Nag_RowMajor**, the (i,j)th element of the matrix C is stored in **c**$[(i-1) \times$ **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.

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

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 ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_2

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

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

On entry, **trans** = ⟨*value*⟩, **k** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **trans** = **Nag_NoTrans**, **pdb** $\geq \max(1, \mathbf{k})$.

On entry, **trans** = ⟨*value*⟩, **k** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pdb** $\geq \max(1, \mathbf{k})$.

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

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

On entry, **trans** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **trans** = **Nag_NoTrans**, **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **trans** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **trans** = **Nag_Trans** or **Nag_ConjTrans**, **pdb** $\geq \max(1, \mathbf{n})$.

NE_INT

On entry, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pdc** = ⟨*value*⟩, **n** = ⟨*value*⟩.
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-2k update of real symmetric 4 by 4 matrix C using 4 by 2 matrices A and B, $C = C - AB^{\mathrm{T}} - BA^{\mathrm{T}}$, where

$$C = \begin{pmatrix} 4.30 & -3.96 & 0.40 & -0.27 \\ -3.96 & -4.87 & 0.31 & 0.07 \\ 0.40 & 0.31 & -8.02 & -5.95 \\ -0.27 & 0.07 & -5.95 & 0.12 \end{pmatrix},$$

$$A = \begin{pmatrix} -3.0 & -5.0 \\ -1.0 & 1.0 \\ 2.0 & -1.0 \\ 1.0 & 1.0 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 3.0 & -2.0 \\ -1.0 & 1.0 \\ 2.0 & -1.0 \\ 1.0 & 0.0 \end{pmatrix}.$$

9.1 Program Text

```
/* nag_dsyr2k (f16yrc) 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 */
  double alpha, beta;
  Integer adim1, adim2, exit_status, i, j, k, n, pda, pdb, pdc;

  /* Arrays */
  double *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);

  Vprintf( "nag_dsyr2k (f16yrc) 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, &beta);

  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, double)) ||
           !(b = NAG_ALLOC(k*n, double)) ||
           !(c = NAG_ALLOC(n*n, double)) )
        {
          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", &A(i,j));
      Vscanf("%*[^\n] ");
    }
  /* Input matrix B. */
  for (i = 1; i <= adim1; ++i)
    {
      for (j = 1; j <= adim2; ++j)
        Vscanf("%lf", &B(i,j));
      Vscanf("%*[^\n] ");
    }
  /* Input matrix C. */
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            Vscanf("%lf", &C(i,j));
        }
      Vscanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            Vscanf("%lf", &C(i,j));
        }
      Vscanf("%*[^\n] ");
    }

  /* nag_dsyr2k(f16yrc).
   * Rank 2k update of symmetric matrix.
   *
   */
  nag_dsyr2k(order, uplo, trans, n, k, alpha, a, pda, b, pdb, beta,
```

```
            c, pdc, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_dsyr2k.\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_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  nag_gen_real_mat_print(order, matrix, Nag_NonUnitDiag, n,
                         n, c, pdc, "Updated Matrix C", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_gen_real_mat_print (x04cac).\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_dsyr2k (f16yrc) Example Program Data
  4  2                        :Values of n and k
  Nag_Lower                   :Value of uplo
  Nag_NoTrans                 :Value of trans
 -1.0  1.0                    :Values of alpha and beta
 -3.00  -5.00
 -1.00   1.00
  2.00  -1.00
  1.00   1.00                 :End of matrix A
  3.00  -2.00
 -1.00   1.00
  2.00  -1.00
  1.00   0.00                 :End of matrix B
  4.30
 -3.96  -4.87
  0.40   0.31  -8.02
 -0.27   0.07  -5.95   0.12   :End of matrix C
```

9.3 Program Results

```
nag_dsyr2k (f16yrc) Example Program Results

 Updated Matrix C

             1           2           3           4
 1      2.3000
 2      3.0400     -8.8700
 3     -6.6000      6.3100    -18.0200
 4      1.7300      1.0700     -8.9500     -1.8800
```