NAG Library Function Document

nag_dsfrk (f16yqc)

1 Purpose
nag_dsfrk (f16yqc) 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 real matrix, $C$ is an $n$ by $n$ real symmetric matrix stored in Rectangular Full Packed (RFP) format, and $\alpha$ and $\beta$ are real scalars.

2 Specification

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

void nag_dsfrk (Nag_OrderType order, Nag_RFP_Store transr,
                Nag_UploType uplo, Nag_TransType trans, Integer n, Integer k,
                double alpha, const double a[], Integer pda, double beta, double cr[],
                NagError *fail)
```

3 Description

nag_dsfrk (f16yqc) 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 real matrix, $C$ is an $n$ by $n$ real symmetric matrix stored in Rectangular Full Packed (RFP) format, and $\alpha$ and $\beta$ are real scalars. The RFP storage format is described in Section 3.3.3 in the f07 Chapter Introduction.

If $n = 0$ or if $\beta = 1.0$ and either $k = 0$ or $\alpha = 0.0$ then nag_dsfrk (f16yqc) returns immediately. If $\beta=0.0$ and either $k=0$ or $\alpha = 0.0$ then $C$ is set to the zero matrix.

4 References


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:  `transr` – Nag_RFP_Store  
`Input`  
On entry: specifies whether the RFP representation of $C$ is normal or transposed.  
`transr = Nag_RFP_Normal`  
The matrix $C$ is stored in normal RFP format.
transr = Nag_RFP_Trans
The matrix C is stored in transposed RFP format.

Constraint: transr = Nag_RFP_Normal or Nag_RFP_Trans.

3: uplo – Nag_UploType
Input

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

uplo = Nag_Upper
The upper triangular part of C is stored in RFP format.

uplo = Nag_Lower
The lower triangular part of C is stored in RFP format.

Constraint: uplo = Nag_Upper or Nag_Lower.

4: trans – Nag_TransType
Input

On entry: specifies the operation to be performed.

trans = Nag_NoTrans
\[ C \leftarrow \alpha A^T + \beta C. \]

trans = Nag_Trans
\[ C \leftarrow \alpha A^T A + \beta C. \]

Constraint: trans = Nag_NoTrans or Nag_Trans.

5: n – Integer
Input

On entry: n, the order of the matrix C.

Constraint: n \geq 0.

6: k – Integer
Input

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

Constraint: k \geq 0.

7: alpha – double
Input

On entry: the scalar \( \alpha \).

8: a[dim] – const double
Input

Note: the dimension, dim, of the array a must be at least
\[
\max(1, pdA \times k) \text{ when } trans = Nag\_NoTrans \text{ and } order = Nag\_ColMajor; \\
\max(1, n \times pdA) \text{ when } trans = Nag\_NoTrans \text{ and } order = Nag\_RowMajor; \\
\max(1, pdA \times n) \text{ when } trans = Nag\_Trans \text{ and } order = Nag\_ColMajor; \\
\max(1, k \times pdA) \text{ when } trans = Nag\_Trans \text{ and } order = Nag\_RowMajor.
\]

On entry: the matrix A; A is n by k if trans = Nag_NoTrans, or k by n if trans = Nag_Trans. If alpha = 0.0, a is not referenced.

9: 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 ≥ max(1, n);
  if trans = Nag_Trans, pda ≥ max(1, k);
if order = Nag_RowMajor,
  if trans = Nag_NoTrans, pda ≥ max(1, k);
  if trans = Nag_Trans, pda ≥ max(1, n).

10: beta – double
   Input
   On entry: the scalar β.

11: cr[n × (n + 1)/2] – double
    Input/Output
   On entry: the upper or lower triangular part (as specified by uplo) of the n by n symmetric matrix C, stored in RFP format (as specified by transr). The storage format is described in detail in Section 3.3.3 in the f07 Chapter Introduction. If β = 0.0, cr need not be set on entry.
   On exit: the updated matrix C, that is its upper or lower triangular part stored in RFP format.

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 ⟨value⟩ had an illegal value.

NE_ENUM_INT_2
   On entry, trans = ⟨value⟩, k = ⟨value⟩, pda = ⟨value⟩.
   Constraint: if trans = Nag_NoTrans, pda ≥ max(1, k).
   On entry, trans = ⟨value⟩, k = ⟨value⟩, pda = ⟨value⟩.
   Constraint: if trans = Nag_Trans, pda ≥ max(1, k).
   On entry, trans = ⟨value⟩, n = ⟨value⟩, pda = ⟨value⟩.
   Constraint: if trans = Nag_NoTrans, pda ≥ max(1, n).
   On entry, trans = ⟨value⟩, n = ⟨value⟩, pda = ⟨value⟩.
   Constraint: if trans = Nag_Trans, pda ≥ max(1, n).

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

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

Not applicable.

### 8 Parallelism and Performance

nag_dsfrk (f16yqc) is not threaded by NAG in any implementation.

nag_dsfrk (f16yqc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

### 9 Further Comments

None.

### 10 Example

This example reads in the lower triangular part of a symmetric matrix $C$ which it converts to RFP format. It also reads in $\alpha$, $\beta$ and a 6 by 4 matrix $A$ and then performs the symmetric rank-4 update $C \leftarrow \alpha AA^T + \beta C$.

#### 10.1 Program Text

```c
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    double alpha, beta;
    Integer i, j, k, n, pda, pdc;
    /* Arrays */
    double *a = 0, *c = 0, *cr = 0;
    char nag_enum_arg[40];
    /* Nag Types */
    Nag_OrderType order;
    Nag_RFP_Store transr;
    Nag_UploType uplo;
    Nag_MatrixType matrix;
    Nag_TransType trans;
```
NagError fail;

INIT_FAIL(fail);

printf("nag_dsfrk (f16yqc) Example Program Results\n");
/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[
\n "");
#else
    scanf("%*[
\n "");
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT "%"NAG_IFMT "%*[
\n "", &n, &k);
#else
    scanf("%"NAG_IFMT "%"NAG_IFMT "%*[
\n "", &n, &k);
#endif
pdc = n;
#ifdef NAG_COLUMN_MAJOR
    order = Nag_ColMajor;
    pda = n;
#define C(I, J) c[(J-1)*pdc + I-1]
#define A(I, J) a[(J-1)*pda + I-1]
#else
    order = Nag_RowMajor;
    pda = k;
#define C(I, J) c[(I-1)*pdc + J-1]
#define A(I, J) a[(I-1)*pda + J-1]
#endif
if (!(c = NAG_ALLOC(pdc*n, double)) ||
    !(cr = NAG_ALLOC((n * (n + 1))/2, double)) ||
    !(a = NAG_ALLOC(n*k, double))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Nag_RFP_Store */
#ifdef _WIN32
    scanf_s("%39s ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s ", nag_enum_arg);
#endif
transr = (Nag_RFP_Store) nag_enum_name_to_value (nag_enum_arg);
/* Nag_UploType */
#ifdef _WIN32
    scanf_s("%39s %*[
\n "", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s ", nag_enum_arg);
#endif
uplo = (Nag_UploType) nag_enum_name_to_value (nag_enum_arg);
/* Nag_TransType */
#ifdef _WIN32
    scanf_s("%39s %*[
\n "", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s ", nag_enum_arg);
#endif
trans = (Nag_TransType) nag_enum_name_to_value (nag_enum_arg);
#ifdef _WIN32
    scanf_s("%lf%lf%*[
\n "", &alpha, &beta);
#else
    scanf("%lf%lf%*[
\n "", &alpha, &beta);
#endif
/* Read upper or lower triangle of matrix C from data file */
if (uplo == Nag_Lower) {
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++) {
            #ifdef _WIN32
                scanf_s("%lf", &C(i, j));
            #else
                scanf("%lf", &C(i, j));
            #endif
        }
    }
}
```c
scanf("%lf", &C(i, j));
#endif
}
} else {
    for (i = 1; i <= n; i++) {
        for (j = i; j <= n; j++) {
            #ifdef _WIN32
                scanf_s("%lf", &C(i, j));
            #else
                scanf("%lf", &C(i, j));
            #endif
        }
    }
#else
    scanf("%*[\n] ");
#endif
/* Read matrix A from data file */
for (i = 1; i <= n; i++) {
    for (j = 1; j <= k; j++) {
        #ifdef _WIN32
            scanf_s("%lf", &A(i, j));
        #else
            scanf("%lf", &A(i, j));
        #endif
    }
}
/* Convert symmetric matrix C from full triangular storage to rectangular full *
* packed storage (in cr) using nag_dtrttf (f01vec).
*/
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrttf (f01vec).\n", fail.message);
    exit_status = 1;
    goto END;
}
printf("\n");
/* Perform the rank-k update of real symmetric matrix C by real matrix A *
* using nag_dsfrk (f16yqc).
*/
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dsfrk (f16yqc).\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Convert C back from rectangular full packed (cr) to standard triangular *
* storage format (c) using nag_dtfttr (f01vgc).
*/
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtfttr (f01vgc).\n", fail.message);
    exit_status = 1;
    goto END;
}
matrix = (uplo == Nag_Upper ? Nag_UpperMatrix : Nag_LowerMatrix);
/* Print out the result, stored in the lower triangle of matrix C using *
* the easy-to-use print routine nag_gen_real_mat_print (x04cac).
*/
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
    exit_status = 1;
    goto END;
}
```

fail.message);
    exit_status = 1;
}

END:
NAG_FREE(a);
NAG_FREE(c);
NAG_FREE(cr);
return exit_status;
}

10.2 Program Data

nag_dsfrk (f16yqc) Example Program Data

6 4
    : n, k
Nag_RFP_Normal Nag_Lower Nag_NoTrans : transr, uplo, trans
    : alpha, beta
4.21 0.89

1.0
2.0 2.0
3.0 3.0 3.0
4.0 4.0 4.0 4.0
5.0 5.0 5.0 5.0 5.0
6.0 6.0 6.0 6.0 6.0 6.0 : matrix C

3.21 1.32 2.31 0.25
1.65 1.87 0.32 -1.54
1.80 2.88 2.05 -0.89
5.25 -2.95 -0.95 -3.80
1.58 -2.69 -2.90 -1.04
-1.11 -0.66 -0.59 0.80 : matrix A

10.3 Program Results

nag_dsfrk (f16yqc) Example Program Results

The Solution

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74.3339</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>35.9614</td>
<td>38.3792</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>61.9998</td>
<td>46.3791</td>
<td>72.2571</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>44.8769</td>
<td>40.1617</td>
<td>13.6156</td>
<td>220.8276</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-18.4440</td>
<td>-2.9162</td>
<td>-37.3241</td>
<td>101.0169</td>
<td>85.3835</td>
<td></td>
</tr>
</tbody>
</table>