

NAG Library Function Document

nag_dspr (f16pqc)

1 Purpose

nag_dspr (f16pqc) performs a rank-1 update on a real symmetric matrix stored in packed form.

2 Specification

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

void nag_dspr (Nag_OrderType order, Nag_UploType uplo, Integer n,
              double alpha, const double x[], Integer incx, double beta, double ap[],
              NagError *fail)
```

3 Description

nag_dspr (f16pqc) performs the symmetric rank-1 update operation

$$A \leftarrow \alpha x x^T + \beta A,$$

where A is an n by n real symmetric matrix, stored in packed form, x is an n -element real vector, while α and β are real 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 A is stored.
uplo = Nag_Upper
 The upper triangular part of A is stored.
uplo = Nag_Lower
 The lower triangular part of A is stored.
Constraint: **uplo** = Nag_Upper or Nag_Lower.
- 3: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: **n** \geq 0.

- 4: **alpha** – double *Input*
On entry: the scalar α .
- 5: **x**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$.
On entry: the *n*-element vector *x*.
 If $\mathbf{incx} > 0$, x_i must be stored in $\mathbf{x}[(i - 1) \times \mathbf{incx}]$, for $i = 1, 2, \dots, \mathbf{n}$.
 If $\mathbf{incx} < 0$, x_i must be stored in $\mathbf{x}[(\mathbf{n} - i) \times |\mathbf{incx}|]$, for $i = 1, 2, \dots, \mathbf{n}$.
 Intermediate elements of **x** are not referenced. If $\mathbf{n} = 0$, **x** is not referenced and may be **NULL**.
- 6: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of *x*.
Constraint: $\mathbf{incx} \neq 0$.
- 7: **beta** – double *Input*
On entry: the scalar β .
- 8: **ap**[*dim*] – double *Input/Output*
Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$.
On entry: the *n* by *n* symmetric matrix *A*, packed by rows or columns.
 The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:
 if **order** = Nag_ColMajor and **uplo** = Nag_Upper,
 A_{ij} is stored in $\mathbf{ap}[(j - 1) \times j/2 + i - 1]$, for $i \leq j$;
 if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
 A_{ij} is stored in $\mathbf{ap}[(2n - j) \times (j - 1)/2 + i - 1]$, for $i \geq j$;
 if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
 A_{ij} is stored in $\mathbf{ap}[(2n - i) \times (i - 1)/2 + j - 1]$, for $i \leq j$;
 if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
 A_{ij} is stored in $\mathbf{ap}[(i - 1) \times i/2 + j - 1]$, for $i \geq j$.
On exit: the updated matrix *A*.
- 9: **fail** – NagError * *Input/Output*
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument *<value>* had an illegal value.

NE_INT

On entry, $\mathbf{incx} = \langle \text{value} \rangle$.

Constraint: $\mathbf{incx} \neq 0$.

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

NE_INTERNAL_ERROR

An unexpected error has been triggered by this function. Please contact NAG.
 See Section 3.6.6 in How to Use the NAG Library and its Documentation 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 How to Use the NAG Library and its Documentation 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

nag_dspr (f16pqc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

Perform rank-1 update of real symmetric matrix A , stored in packed storage format, using vector x :

$$A \leftarrow A - xx^T,$$

where A is the 4 by 4 matrix given by

$$A = \begin{pmatrix} 4.30 & 4.00 & 0.40 & -0.28 \\ 4.00 & -4.87 & 0.31 & 0.07 \\ 0.40 & 0.31 & -8.02 & -5.95 \\ -0.28 & 0.07 & -5.95 & 0.12 \end{pmatrix}$$

and

$$x = (2.0, 2.0, 0.2, -0.14)^T.$$

10.1 Program Text

```
/* nag_dspr (f16pqc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
```

```

double alpha, beta;
Integer ap_len, exit_status, i, incx, j, n, xlen;

/* Arrays */
double *ap = 0, *x = 0;
char nag_enum_arg[40];

/* Nag Types */
NagError fail;
Nag_OrderType order;
Nag_UploType uplo;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
    order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_dspr (f16pqc) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    /* Read the problem dimension */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &n);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &n);
#endif

    /* Read the uplo storage parameter */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, (unsigned)_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 scalar parameters */
#ifdef _WIN32
    scanf_s("%lf%lf%*[\n] ", &alpha, &beta);
#else
    scanf("%lf%lf%*[\n] ", &alpha, &beta);
#endif

    /* Read increment parameter */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &incx);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &incx);
#endif

    ap_len = n * (n + 1) / 2;
    xlen = MAX(1, 1 + (n - 1) * ABS(incx));

    if (n > 0) {
        /* Allocate memory */
        if (!(ap = NAG_ALLOC(ap_len, double)) || !(x = NAG_ALLOC(xlen, double)))

```

```

    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else {
    printf("Invalid n\n");
    exit_status = 1;
    return exit_status;
}

/* Input matrix A and vector x */

if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i) {
        for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A_UPPER(i, j));
#else
            scanf("%lf", &A_UPPER(i, j));
#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", &A_LOWER(i, j));
#else
            scanf("%lf", &A_LOWER(i, j));
#endif
#ifdef _WIN32
            scanf_s("%*[\n] ");
#else
            scanf("%*[\n] ");
#endif
        }
    }
    for (i = 0; i < xlen; ++i)
#ifdef _WIN32
        scanf_s("%lf%*[\n] ", &x[i]);
#else
        scanf("%lf%*[\n] ", &x[i]);
#endif

/* nag_dspr (f16pqc).
 * Rank one update of real symmetric matrix,
 * packed storage.
 */
nag_dspr(order, uplo, n, alpha, x, incx, beta, ap, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dspr.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print updated matrix A */
/* nag_pack_real_mat_print (x04ccc).
 * Print real packed triangular matrix (easy-to-use)
 */
fflush(stdout);
nag_pack_real_mat_print(order, uplo, Nag_NonUnitDiag, n, ap,
    "Updated Matrix A", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_pack_real_mat_print (x04ccc).\n%s\n",

```

```

        fail.message);
    exit_status = 1;
    goto END;
}

END:
    NAG_FREE(ap);
    NAG_FREE(x);

    return exit_status;
}

```

10.2 Program Data

```

nag_dspr (f16pqc) Example Program Data
  4                               :Value of n
  Nag_Lower                       :Storage of A
-1.0    1.0                       :Values of alpha and beta
  1                               :Value of incx
  4.30
  4.00  -4.87
  0.40  0.31  -8.02
-0.28  0.07  -5.95  0.12  :End of matrix A
  2.00
  2.00
  0.20
-0.14                               :End of vector x

```

10.3 Program Results

```

nag_dspr (f16pqc) Example Program Results

Updated Matrix A

```

	1	2	3	4
1	0.3000			
2	0.0000	-8.8700		
3	0.0000	-0.0900	-8.0600	
4	0.0000	0.3500	-5.9220	0.1004
