

NAG Library Function Document

nag_dsyrr (f16ppc)

1 Purpose

nag_dsyrr (f16ppc) performs a rank-1 update on a real symmetric matrix.

2 Specification

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

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

3 Description

nag_dsyrr (f16ppc) 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, 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: **a**[*dim*] – double *Input/Output*
Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.
On entry: the *n* by *n* symmetric matrix *A*.
 If **order** = Nag_ColMajor, A_{ij} is stored in $\mathbf{a}[(j - 1) \times \mathbf{pda} + i - 1]$.
 If **order** = Nag_RowMajor, A_{ij} is stored in $\mathbf{a}[(i - 1) \times \mathbf{pda} + j - 1]$.
 If **uplo** = Nag_Upper, the upper triangular part of *A* must be stored and the elements of the array below the diagonal are not referenced.
 If **uplo** = Nag_Lower, the lower triangular part of *A* must be stored and the elements of the array above the diagonal are not referenced.
On exit: the updated matrix *A*.
- 9: **pda** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix *A* in the array **a**.
Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 10: **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 2.3.1.2 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, **incx** = $\langle value \rangle$.

Constraint: **incx** \neq 0.

On entry, **n** = $\langle value \rangle$.

Constraint: **n** \geq 0.

NE_INT_2

On entry, **pda** = $\langle value \rangle$, **n** = $\langle value \rangle$.

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

NE_INTERNAL_ERROR

An unexpected error has been triggered by this function. Please contact NAG.

See Section 2.7.6 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 2.7.5 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_dsyrr (f16ppc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

Perform rank-1 update of real symmetric matrix A 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_dsyrr (f16ppc) 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 exit_status, i, incx, j, n, pda, xlen;

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

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

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_dsyrr (f16ppc) 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

pda = n;

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

if (n > 0) {
  /* Allocate memory */
  if (!(a = NAG_ALLOC(pda * n, 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(i, j));
#else
      scanf("%lf", &A(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(i, j));
#else
      scanf("%lf", &A(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_dsyr (f16ppc).
 * Rank one update of real symmetric matrix.
 */
nag_dsyr(order, uplo, n, alpha, x, incx, beta, a, pda, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_dsyr.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

```

```

if (uplo == Nag_Upper) {
    matrix = Nag_UpperMatrix;
}
else {
    matrix = Nag_LowerMatrix;
}
/* Print updated matrix A */
/* nag_gen_real_mat_print (x04cac).
 * Print real general matrix (easy-to-use)
 */
fflush(stdout);
nag_gen_real_mat_print(order, matrix, Nag_NonUnitDiag, n,
                      n, a, pda, "Updated Matrix A", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
}

END:
NAG_FREE(a);
NAG_FREE(x);

return exit_status;
}

```

10.2 Program Data

```

nag_dsyrc (f16ppc) 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_dsyrc (f16ppc) 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

```
