NAG Library Function Document

nag_ddot (f16eac)

1 Purpose

nag_ddot (f16eac) updates a scalar by a scaled dot product of two real vectors, by performing

$$r \leftarrow \beta r + \alpha x^{\mathrm{T}} y.$$

2 Specification

```
#include <nag.h>
#include <nagf16.h>
void nag_ddot (Nag_ConjType conj, Integer n, double alpha, const double x[],
     Integer incx, double beta, const double y[], Integer incy, double *r,
     NagError *fail)
```

3 Description

nag_ddot (f16eac) performs the operation

$$r \leftarrow \beta r + \alpha x^{\mathrm{T}} y$$

where x and y are n-element real vectors, and r, α and β real scalars. If n is less than zero, or, if β is equal to one and either α or n is equal to zero, this function returns immediately.

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: **conj** – Nag_ConjType *Input*

On entry: **conj** is not used. The presence of this argument in the BLAST standard is for consistency with the interface of the complex variant of this function.

Constraint: **conj** = Nag_NoConj or Nag_Conj.

2: **n** – Integer *Input*

On entry: n, the number of elements in x and y.

3: **alpha** – double *Input*

On entry: the scalar α.

4: $\mathbf{x}[\mathbf{1} + (\mathbf{n} - \mathbf{1}) \times |\mathbf{incx}|]$ – const double *Input*

On entry: the n-element vector x.

If **incx** > 0, x_i must be stored in $\mathbf{x}[(i-1) \times \mathbf{incx}]$, for $i = 1, 2, \ldots, \mathbf{n}$.

If **incx** < 0, x_i must be stored in $\mathbf{x}[(\mathbf{n} - i) \times |\mathbf{incx}|]$, for $i = 1, 2, \ldots, \mathbf{n}$.

Intermediate elements of **x** are not referenced. If $\alpha = 0.0$ or $\mathbf{n} = 0$, **x** is not referenced and may be **NULL**.

5: **incx** – Integer *Input*

On entry: the increment in the subscripts of **x** between successive elements of x.

Constraint: **incx** $\neq 0$.

6: **beta** – double *Input*

On entry: the scalar β.

7: **y**[**1** + (**n** − **1**) × |**incy**|] – const double *Input*

On entry: the n-element vector y.

If **incy** > 0, y_i must be stored in **y**$[(i-1) \times$ **incy**], for $i = 1, 2, \ldots,$ **n**.

If **incy** < 0, y_i must be stored in **y**[(**n** $- i) \times$ |**incy**|], for $i = 1, 2, \ldots,$ **n**.

Intermediate elements of **y** are not referenced. If $\alpha = 0.0$ or **n** $= 0$, **y** is not referenced and may be **NULL**.

8: **incy** – Integer *Input*

On entry: the increment in the subscripts of **y** between successive elements of y.

Constraint: **incy** $\neq 0$.

9: **r** – double * *Input/Output*

On entry: the initial value, r, to be updated. If $\beta = 0.0$, **r** need not be set on entry.

On exit: the value r, scaled by β and updated by the scaled dot product of x and y.

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** = ⟨*value*⟩.
Constraint: **incx** $\neq 0$.

On entry, **incy** = ⟨*value*⟩.
Constraint: **incy** $\neq 0$.

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 dot product $x^{\mathrm{T}}y$ is computed using the BLAS routine DDOT.

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_ddot (f16eac) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example computes the scaled sum of two dot products, $r = \alpha_1 x^{\mathrm{T}} y + \alpha_2 u^{\mathrm{T}} v$, where

$$\begin{array}{lll} \alpha_1 = 0.3, & x = (1, 2, 3, 4, 5), & y = (-5, -4, 3, 2, 1), \\ \alpha_2 = -7.0, & u = v = (0.4, 0.3, 0.2, 0.1). & \end{array}$$

y and v are stored in reverse order, and u is stored in reverse order in every other element of a real array.

10.1 Program Text

```
/* nag_ddot (f16eac) 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>

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  double alpha, beta, r;
  Integer call, i, incx, incy, ix, iy, n;

  /* Arrays */
  double *x = 0, *y = 0;

  /* Nag Types */
  Nag_ConjType conj = Nag_NoConj;
  NagError fail;

  INIT_FAIL(fail);
  printf("nag_ddot (f16eac) Example Program Results\n\n");

  /* Skip heading in data file. */
#ifdef _WIN32
```

```
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Accumulate two dot products, set beta=zero initially. */
  beta = 0.0;
  for (call = 1; call <= 2; call++) {
    /* Read data for dot product. */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
#else
    scanf("%" NAG_IFMT "%*[^\n] ", &n);
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &incx, &incy);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &incx, &incy);
#endif
    if (!(x = NAG_ALLOC(1 + (n - 1) * ABS(incx), double)) ||
        !(y = NAG_ALLOC(1 + (n - 1) * ABS(incy), double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

#ifdef _WIN32
    scanf_s("%lf%*[^\n] ", &alpha);
#else
    scanf("%lf%*[^\n] ", &alpha);
#endif

    /* Read the vectors x and y and store forwards or backwards
     * as determined by incx (resp. incy). */
    for (i = 0, ix = (incx > 0 ? 0 : (1-n)*incx); i < n; i++, ix += incx)
#ifdef _WIN32
      scanf_s("%lf", &x[ix]);
#else
      scanf("%lf", &x[ix]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    for (i = 0, iy = (incy > 0 ? 0 : (1-n)*incy); i < n; i++, iy += incy)
#ifdef _WIN32
      scanf_s("%lf", &y[iy]);
#else
      scanf("%lf", &y[iy]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* nag_ddot computes r = beta*r + alpha*(x^T*y). */
    nag_ddot(conj, n, alpha, x, incx, beta, y, incy, &r, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ddot (f16eac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
    /* Reset beta for accumulation and deallocate x, y. */
    beta = 1.0;
    NAG_FREE(x);
    NAG_FREE(y);
  }
```

```
  printf("Accumulated dot product, r = %9.4f\n", r);
END:
  NAG_FREE(x);
  NAG_FREE(y);
  return exit_status;
}
```

10.2 Program Data

```
nag_ddot (f16eac) Example Program Data
   5                                         : first dot product, n
   1    -1                                   : incx and incy
   0.3                                       : alpha
   1.0   2.0   3.0   4.0   5.0               : Vector x
  -5.0  -4.0   3.0   2.0   1.0               : Vector y

   4                                         : second dot product, n
  -2    -1                                   : incx and incy
  -7.0                                       : alpha
   0.4   0.3   0.2   0.1                     : Vector x
   0.4   0.3   0.2   0.1                     : Vector y
```

10.3 Program Results

```
nag_ddot (f16eac) Example Program Results

Accumulated dot product, r =    0.6000
```