

NAG Library Function Document

nag_daxpby (f16ecc)

1 Purpose

nag_daxpby (f16ecc) computes the sum of two scaled vectors, for real vectors and scalars.

2 Specification

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

void nag_daxpby (Integer n, double alpha, const double x[], Integer incx,
                 double beta, double y[], Integer incy, NagError *fail)
```

3 Description

nag_daxpby (f16ecc) performs the operation

$$y \leftarrow \alpha x + \beta y$$

where x and y are n -element real vectors, and α and β real scalars. If n is equal to zero, or if α is equal to zero and β is equal to 1, 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: **n** – Integer *Input*
On entry: n , the number of elements in x and y .
Constraint: $n \geq 0$.
- 2: **alpha** – double *Input*
On entry: the scalar α .
- 3: **x[dim]** – const double *Input*
Note: the dimension, dim , of the array **x** must be at least $\max(1, 1 + (n - 1) \times |incx|)$.
On entry: the n -element vector x .
 If $incx > 0$, x_i must be stored in $x[(i - 1) \times incx]$, for $i = 1, 2, \dots, n$.
 If $incx < 0$, x_i must be stored in $x[(n - i) \times |incx|]$, for $i = 1, 2, \dots, n$.
 Intermediate elements of **x** are not referenced. If $n = 0$, **x** is not referenced and may be **NULL**.
- 4: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: $incx \neq 0$.

- 5: **beta** – double *Input*
On entry: the scalar β .
- 6: **y**[*dim*] – double *Input/Output*
Note: the dimension, *dim*, of the array **y** must be at least $\max(1, 1 + (\mathbf{n} - 1) \times |\mathbf{incy}|)$.
On entry: the *n*-element vector *y*.
 If **incy** > 0, y_i must be stored in **y**[(*i* − 1) × **incy**], for $i = 1, 2, \dots, \mathbf{n}$.
 If **incy** < 0, y_i must be stored in **y**[(**n** − *i*) × |**incy**|], for $i = 1, 2, \dots, \mathbf{n}$.
 Intermediate elements of **y** are not referenced.
On exit: the updated vector *y* stored in the array elements used to supply the original vector *y*.
 Intermediate elements of **y** are unchanged.
- 7: **incy** – Integer *Input*
On entry: the increment in the subscripts of **y** between successive elements of *y*.
Constraint: **incy** ≠ 0.
- 8: **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** ≠ 0.

On entry, **incy** = *⟨value⟩*.

Constraint: **incy** ≠ 0.

On entry, **n** = *⟨value⟩*.

Constraint: **n** ≥ 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 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_daxpby` (f16ecc) 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 computes the result of a scaled vector accumulation for

$$\begin{aligned}\alpha &= 3, & x &= (-6, 4.5, 3.7, 2.1, -4)^T, \\ \beta &= -1, & y &= (-5.1, -5, 6.4, -2.4, -3)^T.\end{aligned}$$

x and y are stored in reverse order.

10.1 Program Text

```
/* nag_daxpby (f16ecc) 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, i, incx, incy, ix, iy, n;
    double alpha, beta;
    /* Arrays */
    double *x = 0, *y = 0;
    /* Nag Types */
    NagError fail;

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_daxpby (f16ecc) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%s[^\n] ");
#else
    scanf("%s[^\n] ");
#endif
    /* Read number of elements */
```

```

#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &n);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &n);
#endif
/* Read increments */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy);
#endif
/* Read factors alpha and beta */
#ifdef _WIN32
    scanf_s("%lf%lf%*[\n] ", &alpha, &beta);
#else
    scanf("%lf%lf%*[\n] ", &alpha, &beta);
#endif

    if (n > 0) {
        /* Allocate memory */
        if (!(x = NAG_ALLOC(MAX(1, 1 + (n - 1) * ABS(incx)), double)) ||
            !(y = NAG_ALLOC(MAX(1, 1 + (n - 1) * ABS(incy)), double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else {
        printf("Invalid n\n");
        exit_status = 1;
        goto END;
    }

    /* 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_daxpby (f16ecc).
     * Performs y := alpha*x + beta*y */
    nag_daxpby(n, alpha, x, incx, beta, y, incy, &fail);

    if (fail.code != NE_NOERROR) {
        printf("Error from nag_daxpby (f16ecc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Display the vector y forwards or backwards

```

```

    * as determined by incy. */
    printf("Result of the scaled vector accumulation is\n");
    printf("y = (");
    for (i = 0, iy = (incy > 0 ? 0 : (1-n)*incy); i < n; i++, iy += incy)
        printf("%9.4f%s", y[iy], (i < n-1 ? ", " : ")\n"));

END:
    NAG_FREE(x);
    NAG_FREE(y);

    return exit_status;
}

```

10.2 Program Data

```

nag_daxpby (f16ecc) Example Program Data
  5          : n
 -1         -1 : incx and incy
 3.0      -1.0 : alpha and beta
-6.0      4.5   3.7   2.1   -4.0 : Vector x
-5.1     -5.0   6.4  -2.4   -3.0 : Vector y

```

10.3 Program Results

```

nag_daxpby (f16ecc) Example Program Results

```

```

Result of the scaled vector accumulation is
y = ( -12.9000,  18.5000,   4.7000,   8.7000,  -9.0000)

```
