

NAG Library Function Document

nag_zwaxpby (f16ghc)

1 Purpose

nag_zwaxpby (f16ghc) computes the sum of two scaled vectors, preserving input, for complex scalars and vectors.

2 Specification

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

void nag_zwaxpby (Integer n, Complex alpha, const Complex x[], Integer incx,
                 Complex beta, const Complex y[], Integer incy, Complex w[],
                 Integer incw, NagError *fail)
```

3 Description

nag_zwaxpby (f16ghc) performs the operation

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

where x and y are n -element complex vectors, and α and β are complex 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: **n** – Integer *Input*
On entry: n , the number of elements in x , y and w .
Constraint: $n \geq 0$.
- 2: **alpha** – Complex *Input*
On entry: the scalar α .
- 3: **x**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (n - 1) \times |\mathbf{incx}|)$.
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 $i = 1, 2, \dots, n$.
 If $\mathbf{incx} < 0$, x_i must be stored in $\mathbf{x}[(n - i) \times |\mathbf{incx}|]$, 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: $\mathbf{incx} \neq 0$.

- 5: **beta** – Complex *Input*
On entry: the scalar β .
- 6: **y**[*dim*] – const Complex *Input*
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**[($\mathbf{n} - i$) × **incy**], for $i = 1, 2, \dots, \mathbf{n}$.
 Intermediate elements of **y** are not referenced. If $\beta = 0.0$ or $\mathbf{n} = 0$, **y** is not referenced and may be NULL.
- 7: **incy** – Integer *Input*
On entry: the increment in the subscripts of **y** between successive elements of *y*.
Constraint: **incy** ≠ 0.
- 8: **w**[*dim*] – Complex *Output*
Note: the dimension, *dim*, of the array **w** must be at least $\max(1, 1 + (\mathbf{n} - 1) \times |\mathbf{incw}|)$.
On exit: the *n*-element vector *w*.
 If **incw** > 0, w_i is in **w**[(*i* - 1) × **incw**], for $i = 1, 2, \dots, \mathbf{n}$.
 If **incw** < 0, w_i is in **w**[($\mathbf{n} - i$) × **incw**], for $i = 1, 2, \dots, \mathbf{n}$.
 Intermediate elements of **w** are not referenced.
- 9: **incw** – Integer *Input*
On entry: the increment in the subscripts of **w** between successive elements of *w*.
Constraint: **incw** ≠ 0.
- 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, **incw** = *<value>*.

Constraint: **incw** ≠ 0.

On entry, **incx** = *<value>*.

Constraint: **incx** ≠ 0.

On entry, **incy** = *<value>*.

Constraint: **incy** ≠ 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 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_zwaxpby (f16ghc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example computes the result of a scaled vector accumulation for

$$\begin{aligned} \alpha &= 3 + 2i, & x &= (-6 + 1.2i, 3.7 + 4.5i, -4 + 2.1i)^T, \\ \beta &= -i, & y &= (-5.1, 6.4 - 5i, -3 - 2.4i)^T. \end{aligned}$$

x and y , and also the sum vector w , are stored in reverse order.

10.1 Program Text

```
/* nag_zwaxpby (f16ghc) 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, incw, incx, incy, iw, ix, iy, n;
  Complex alpha, beta;
  /* Arrays */
  Complex *w = 0, *x = 0, *y = 0;
  /* Nag Types */
  NagError fail;

  exit_status = 0;
  INIT_FAIL(fail);
```

```

printf("nag_zwaxpby (f16ghc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\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 "%" NAG_IFMT "%*[\n] ", &incx, &incy,
&incw);
#else
scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy,
&incw);
#endif
/* Read factors alpha and beta */
#ifdef _WIN32
scanf_s(" ( %lf , %lf ) ", &alpha.re, &alpha.im);
#else
scanf(" ( %lf , %lf ) ", &alpha.re, &alpha.im);
#endif
#ifdef _WIN32
scanf_s(" ( %lf , %lf ) %*[\n] ", &beta.re, &beta.im);
#else
scanf(" ( %lf , %lf ) %*[\n] ", &beta.re, &beta.im);
#endif

if (n > 0) {
/* Allocate memory */
if (!(w = NAG_ALLOC(MAX(1, 1 + (n - 1) * ABS(incw)), Complex)) ||
!(x = NAG_ALLOC(MAX(1, 1 + (n - 1) * ABS(incx)), Complex)) ||
!(y = NAG_ALLOC(MAX(1, 1 + (n - 1) * ABS(incy)), Complex)))
{
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 , %lf ) ", &x[ix].re, &x[ix].im);
#else
scanf(" ( %lf , %lf ) ", &x[ix].re, &x[ix].im);
#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 , %lf ) ", &y[iy].re, &y[iy].im);
#else
scanf(" ( %lf , %lf ) ", &y[iy].re, &y[iy].im);
#endif
}

```

```

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

/* nag_zwaxpby (f16ghc).
 * Performs w := alpha*x + beta*y */
nag_zwaxpby(n, alpha, x, incx, beta, y, incy, w, incw, &fail);

if (fail.code != NE_NOERROR) {
    printf("Error from nag_zwaxpby (f16ghc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Display the vector w forwards or backwards
 * as determined by incw. */
printf("Result of the scaled vector addition is\n");
printf("w = (");
for (i = 0, iw = (incw > 0 ? 0 : (1-n)*incw); i < n; i++, iw += incw)
    printf("(%9.4f, %9.4f)%s", w[iw].re, w[iw].im, (i < n-1 ? ", " : ")\n"));

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

    return exit_status;
}

```

10.2 Program Data

```

nag_zwaxpby (f16ghc) Example Program Data
3                                     : n
1  1  1                             : incx, incy and incw
( 3., 2.) ( 0.,-1.)                 : alpha and beta
(-4., 2.1) ( 3.7, 4.5) (-6., 1.2)   : Vector x
(-3.,-2.4) ( 6.4,-5.) (-5.1,0.)     : Vector y

```

10.3 Program Results

nag_zwaxpby (f16ghc) Example Program Results

```

Result of the scaled vector addition is
w = (( -18.6000,  1.3000), ( -2.9000,  14.5000), ( -20.4000,  -3.3000))

```
