NAG Library Function Document

nag_airy_bi_deriv_vector (s17axc)

1 Purpose

nag_airy_bi_deriv_vector (s17axc) returns an array of values for the derivative of the Airy function $B_i(x)$.

2 Specification

```c
#include <nag.h>
#include <nags.h>
void nag_airy_bi_deriv_vector (Integer n, const double x[], double f[],
                             Integer ivalid[], NagError *fail)
```

3 Description

nag_airy_bi_deriv_vector (s17axc) calculates an approximate value for the derivative of the Airy function $B_i(x_i)$ for an array of arguments $x_i$, for $i = 1, 2, \ldots, n$. It is based on a number of Chebyshev expansions.

For $x < -5$,

$$B_i'(x) = \sqrt{-x} \left[ -a(t) \sin z + \frac{b(t)}{\zeta} \cos z \right],$$

where $z = \frac{\pi}{4} + \zeta$, $\zeta = \frac{2}{3} \sqrt{-x^3}$ and $a(t)$ and $b(t)$ are expansions in the variable $t = -2 \left( \frac{\zeta}{x} \right)^3 - 1$. For $-5 \leq x \leq 0$,

$$B_i'(x) = \sqrt{3} (x^2 f(t) + g(t)),$$

where $f$ and $g$ are expansions in $t = -2 \left( \frac{x}{3} \right)^3 - 1$. For $0 < x < 4.5$,

$$B_i'(x) = e^{3x/2} y(t),$$

where $y(t)$ is an expansion in $t = 4x/9 - 1$. For $4.5 \leq x < 9$,

$$B_i'(x) = e^{21x/8} u(t),$$

where $u(t)$ is an expansion in $t = 4x/9 - 3$. For $x \geq 9$,

$$B_i'(x) = \sqrt{x} e^{z} v(t),$$

where $z = \frac{2}{3} \sqrt{x^3}$ and $v(t)$ is an expansion in $t = 2 \left( \frac{18}{z} \right) - 1$. For $|x| < \text{the square of the machine precision}$, the result is set directly to $B_i'(0)$. This saves time and avoids possible underflows in calculation.
For large negative arguments, it becomes impossible to calculate a result for the oscillating function with any accuracy so the function must fail. This occurs for \( x < -\left(\frac{\sqrt{\pi}}{\epsilon}\right)^{4/7} \), where \( \epsilon \) is the machine precision.

For large positive arguments, where \( B_i \) grows in an essentially exponential manner, there is a danger of overflow so the function must fail.

4 References


5 Arguments

1: \( n \) – Integer

   \( n \), the number of points.

   \( n \geq 0 \).

2: \( x[n] \) – const double

   \( x_i \) of the function, for \( i = 1, 2, \ldots, n \).

3: \( f[n] \) – double

   \( B_i(x_i) \), the function values.

4: \( ivalid[n] \) – Integer

   \( i - 1 \) contains the error code for \( x_i \), for \( i = 1, 2, \ldots, n \).

   \( ivalid[i - 1] = 0 \)

   No error.

   \( ivalid[i - 1] = 1 \)

   \( x_i \) is too large and positive. \( f[i - 1] \) contains zero. The threshold value is the same as for fail.code = NE_REAL_ARG_GT in nag_airy_bi_deriv (s17akc), as defined in the Users’ Note for your implementation.

   \( ivalid[i - 1] = 2 \)

   \( x_i \) is too large and negative. \( f[i - 1] \) contains zero. The threshold value is the same as for fail.code = NE_REAL_ARG_LT in nag_airy_bi_deriv (s17akc), as defined in the Users’ Note for your implementation.

5: \( fail \) – NagError *

   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

   Dynamic memory allocation failed.

   See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

   On entry, argument \( \langle value \rangle \) had an illegal value.
On entry, \( n = \{\text{value}\} \).
Constraint: \( n \geq 0 \).

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

On entry, at least one value of \( x \) was invalid. Check ivalid for more information.

For negative arguments the function is oscillatory and hence absolute error is appropriate. In the positive region the function has essentially exponential behaviour and hence relative error is needed. The absolute error, \( E \), and the relative error \( \epsilon \), are related in principle to the relative error in the argument \( \delta \), by

\[
E \approx |x^2 \text{Bi}(x)|\delta \quad \epsilon \approx \left| \frac{x^2 \text{Bi}(x)}{\text{Bi}'(x)} \right| \delta.
\]

In practice, approximate equality is the best that can be expected. When \( \delta \), \( \epsilon \) or \( E \) is of the order of the machine precision, the errors in the result will be somewhat larger.

For small \( x \), positive or negative, errors are strongly attenuated by the function and hence will effectively be bounded by the machine precision.

For moderate to large negative \( x \), the error is, like the function, oscillatory. However, the amplitude of the absolute error grows like \( \frac{|x|^{7/4}}{\sqrt{\pi}} \). Therefore it becomes impossible to calculate the function with any accuracy if \( |x|^{7/4} > \frac{\sqrt{\pi}}{\delta} \).

For large positive \( x \), the relative error amplification is considerable: \( \frac{\epsilon}{\delta} \sim \sqrt{x^3} \). However, very large arguments are not possible due to the danger of overflow. Thus in practice the actual amplification that occurs is limited.

Not applicable.

None.

This example reads values of \( x \) from a file, evaluates the function at each value of \( x_i \) and prints the results.
10.1 Program Text

/* nag_airy_bi_deriv_vector (s17axc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nags.h>

int main(void)
{
    Integer exit_status = 0;
    Integer i, n;
    double *f = 0, *x = 0;
    Integer *invalid = 0;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_airy_bi_deriv_vector (s17axc) Example Program Results\n");
    printf(" x f invalid\n");
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
    scanf("%"NAG_IFMT", &n);
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif

    if (!(x = NAG_ALLOC(n, double)) ||
        !(f = NAG_ALLOC(n, double)) ||
        !(invalid = NAG_ALLOC(n, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    for (i=0; i<n; i++)
    {
        scanf_s("%lf", &x[i]);
        scanf("%lf", &x[i]);
        scanf("%*[\n]");
        scanf("%*[\n]");
    }

    /* nag_airy_bi_deriv_vector (s17axc). */
    nag_airy_bi_deriv_vector(n, x, f, invalid, &fail);
if (fail.code!=NE_NOERROR && fail.code!=NW_INVALID)
{
    printf("Error from nag_airy_bi_deriv_vector (s17axc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
for (i=0; i<n; i++)
    printf(" %11.3e %11.3e %4"NAG_IFMT"
", x[i], f[i], ivalid[i]);

END:
    NAG_FREE(f);
    NAG_FREE(x);
    NAG_FREE(ivalid);
    return exit_status;
}

10.2 Program Data

nag_airy_bi_deriv_vector (s17axc) Example Program Data

7
-10.0 -1.0 0.0 1.0 5.0 10.0 20.0

10.3 Program Results

nag_airy_bi_deriv_vector (s17axc) Example Program Results

<table>
<thead>
<tr>
<th>x</th>
<th>f</th>
<th>ivalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.000e+01</td>
<td>1.194e-01</td>
<td>0</td>
</tr>
<tr>
<td>-1.000e+00</td>
<td>5.924e-01</td>
<td>0</td>
</tr>
<tr>
<td>0.000e+00</td>
<td>4.483e-01</td>
<td>0</td>
</tr>
<tr>
<td>1.000e+00</td>
<td>9.324e-01</td>
<td>0</td>
</tr>
<tr>
<td>5.000e+00</td>
<td>1.436e+03</td>
<td>0</td>
</tr>
<tr>
<td>1.000e+01</td>
<td>1.429e+09</td>
<td>0</td>
</tr>
<tr>
<td>2.000e+01</td>
<td>9.382e+25</td>
<td>0</td>
</tr>
</tbody>
</table>