NAG Library Function Document

nag_airy_bi_vector (s17avc)

1 Purpose

nag_airy_bi_vector (s17avc) returns an array of values of the Airy function, $\mathrm{Bi}(x)$.

2 Specification

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

3 Description

nag_airy_bi_vector (s17avc) evaluates an approximation to the Airy function $\mathrm{Bi}(x_i)$ for an array of arguments x_i, for $i = 1, 2, \dots, n$. It is based on a number of Chebyshev expansions.

For $x < -5$,

$$\mathrm{Bi}(x) = \frac{a(t)\cos z + b(t)\sin z}{(-x)^{1/4}},$$

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

For $-5 \leq x \leq 0$,

$$\mathrm{Bi}(x) = \sqrt{3}(f(t) + xg(t)),$$

where f and g are expansions in $t = -2\left(\frac{x}{5}\right)^3 - 1$.

For $0 < x < 4.5$,

$$\mathrm{Bi}(x) = e^{11x/8}y(t),$$

where y is an expansion in $t = 4x/9 - 1$.

For $4.5 \leq x < 9$,

$$\mathrm{Bi}(x) = e^{5x/2}v(t),$$

where v is an expansion in $t = 4x/9 - 3$.

For $x \geq 9$,

$$\mathrm{Bi}(x) = \frac{e^z u(t)}{x^{1/4}},$$

where $z = \frac{2}{3}\sqrt{x^3}$ and u is an expansion in $t = 2\left(\frac{18}{z}\right) - 1$.

For $|x| <$ ***machine precision***, the result is set directly to $\mathrm{Bi}(0)$. This both saves time and avoids possible intermediate underflows.

For large negative arguments, it becomes impossible to calculate the phase of the oscillating function with any accuracy so the function must fail. This occurs if $x < -\left(\frac{3}{2\epsilon}\right)^{2/3}$, where ϵ is the ***machine precision***.

For large positive arguments, there is a danger of causing overflow since Bi grows in an essentially exponential manner, so the function must fail.

4 References

Abramowitz M and Stegun I A (1972) *Handbook of Mathematical Functions* (3rd Edition) Dover Publications

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of points.

Constraint: $\mathbf{n} \geq 0$.

2: $\mathbf{x}[\mathbf{n}]$ – const double *Input*

On entry: the argument x_i of the function, for $i = 1, 2, \ldots, \mathbf{n}$.

3: $\mathbf{f}[\mathbf{n}]$ – double *Output*

On exit: $\mathrm{Bi}(x_i)$, the function values.

4: $\mathbf{ivalid}[\mathbf{n}]$ – Integer *Output*

On exit: $\mathbf{ivalid}[i-1]$ contains the error code for x_i, for $i = 1, 2, \ldots, \mathbf{n}$.

$\mathbf{ivalid}[i-1] = 0$
: No error.

$\mathbf{ivalid}[i-1] = 1$
: x_i is too large and positive. $\mathbf{f}[i-1]$ contains zero. The threshold value is the same as for **fail.code** = NE_REAL_ARG_GT in nag_airy_bi (s17ahc), as defined in the Users' Note for your implementation.

$\mathbf{ivalid}[i-1] = 2$
: x_i is too large and negative. $\mathbf{f}[i-1]$ contains zero. The threshold value is the same as for **fail.code** = NE_REAL_ARG_LT in nag_airy_bi (s17ahc), as defined in the Users' Note for your implementation.

5: **fail** – NagError * *Input/Output*

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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

NE_INTERNAL_ERROR

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.

NE_NO_LICENCE

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.

NW_IVALID

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

7 Accuracy

For negative arguments the function is oscillatory and hence absolute error is the appropriate measure. In the positive region the function is essentially exponential-like and here relative error is appropriate. The absolute error, E, and the relative error, ϵ, are related in principle to the relative error in the argument, δ, by

$$E \simeq |x\mathrm{Bi}'(x)|\delta, \epsilon \simeq \left|\frac{x\mathrm{Bi}'(x)}{\mathrm{Bi}(x)}\right|\delta.$$

In practice, approximate equality is the best that can be expected. When δ, ϵ or E is of the order of the ***machine precision***, the errors in the result will be somewhat larger.

For small x, errors are strongly damped and hence will be bounded essentially by the ***machine precision***.

For moderate to large negative x, the error behaviour is clearly oscillatory but the amplitude of the error grows like amplitude $\left(\frac{E}{\delta}\right) \sim \frac{|x|^{5/4}}{\sqrt{\pi}}$.

However the phase error will be growing roughly as $\frac{2}{3}\sqrt{|x|^3}$ and hence all accuracy will be lost for large negative arguments. This is due to the impossibility of calculating sin and cos to any accuracy if $\frac{2}{3}\sqrt{|x|^3} > \frac{1}{\delta}$.

For large positive arguments, the relative error amplification is considerable:

$$\frac{\epsilon}{\delta} \sim \sqrt{x^3}.$$

This means a loss of roughly two decimal places accuracy for arguments in the region of 20. However very large arguments are not possible due to the danger of causing overflow and errors are therefore limited in practice.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

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_vector (s17avc) 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  *ivalid = 0;
  NagError fail;

  INIT_FAIL(fail);

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

  printf("nag_airy_bi_vector (s17avc) Example Program Results\n");
  printf("\n");
  printf("     x           f          ivalid\n");
  printf("\n");
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &n);
#else
  scanf("%"NAG_IFMT"", &n);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

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

  for (i=0; i<n; i++)
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
```

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

  /* nag_airy_bi_vector (s17avc).
   * Airy function Bi(x)
   */
  nag_airy_bi_vector(n, x, f, ivalid, &fail);
  if (fail.code!=NE_NOERROR && fail.code!=NW_IVALID)
    {
      printf("Error from nag_airy_bi_vector (s17avc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  for (i=0; i<n; i++)
    printf(" %11.3e %11.3e %4"NAG_IFMT"\n", 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_vector (s17avc) 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_vector (s17avc) Example Program Results

     x           f          ivalid

  -1.000e+01  -3.147e-01    0
  -1.000e+00   1.040e-01    0
   0.000e+00   6.149e-01    0
   1.000e+00   1.207e+00    0
   5.000e+00   6.578e+02    0
   1.000e+01   4.556e+08    0
   2.000e+01   2.104e+25    0
```