Module 3.8: nag_airy_fun

Airy Functions

nag_airy_fun contains procedures for approximating Airy functions, or their derivatives, with real or complex arguments.

Contents

Introduction

This module contains procedures for approximating Airy functions, or their derivatives, with real or complex arguments.

The Airy functions $\mathrm{Ai}(z)$ and $\mathrm{Bi}(z)$ are linearly independent solutions of the differential equation

$$\frac{d^2y}{dz^2} - z\,y = 0.$$

Given a real/complex value of the argument z, the procedures `nag_airy_ai` and `nag_airy_bi` approximate the values of $\mathrm{Ai}(z)$ and $\mathrm{Bi}(z)$; or their derivatives $\mathrm{Ai}'(z)$ and $\mathrm{Bi}'(z)$ respectively.

Airy functions are related to Bessel functions of fractional order by the equations:

$$\mathrm{Ai}(z) = \frac{\sqrt{z}K_{1/3}(w)}{\pi\sqrt{3}}, \qquad \mathrm{Ai}'(z) = \frac{-zK_{2/3}(w)}{\pi\sqrt{3}}$$

$$\mathrm{Bi}(z) = \frac{\sqrt{z}}{\sqrt{3}}(I_{-1/3}(w) + I_{1/3}(w)), \qquad \mathrm{Bi}'(z) = \frac{z}{\sqrt{3}}(I_{-2/3}(w) + I_{2/3}(w)),$$

where K_ν and I_ν are the modified Bessel functions and $w = 2z\sqrt{z}/3$.

In the case of real arguments, the algorithms are based on a number of Chebyshev expansions; while in the complex case the algorithms are based on an efficient recurrence relation used in the right half plane and analytically continued into the left half plane. Further details appear in Section 6.1 of the individual procedure documents.

For further details of Airy functions, see Abramowitz and Stegun [1], Chapter 10.

Procedure: nag_airy_ai

1 Description

nag_airy_ai evaluates an approximation to the Airy function $\mathrm{Ai}(z)$ or its derivative $\mathrm{Ai}'(z)$.

2 Usage

```
USE nag_airy_fun
[value =] nag_airy_ai(z  [, optional arguments])
```

The function result is a scalar, of the same type as z, containing $\mathrm{Ai}(z)$ or $\mathrm{Ai}'(z)$.

3 Arguments

3.1 Mandatory Argument

z — real(kind=*wp*)/complex(kind=*wp*), intent(in)

Input: the argument z of the function.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

deriv — logical, intent(in), optional

Input: specifies whether the function or its derivative is required.

If `deriv` = `.false.`, $\mathrm{Ai}(z)$ is returned;

if `deriv` = `.true.`, $\mathrm{Ai}'(z)$ is returned.

Default: `deriv` = `.false.`.

scale — logical, intent(in), optional

Input: specifies whether or not the result should be scaled when `z` is complex.

If `scale` = `.true.`, and z is complex the result is returned scaled by the factor $e^{2z\sqrt{z}/3}$;

if `scale` = `.false.`, the result is returned unscaled.

Default: `scale` = `.false.`.

Note: when `z` is real, `scale` is ignored.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Failures (error%level = 2):

error%code	Description
201	Possibility of underflow.
	z is real, large and positive. There is a danger of underflow since $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$ decay exponentially. The value zero is returned.
202	Impossible to calculate phase accurately.
	z is real, large and negative. It is impossible to calculate the phase of the oscillatory function with any precision. The value zero is returned.
203	Partial loss of accuracy.
	z is complex and $\lvert\mathtt{z}\rvert$ is too large, so that errors due to argument reduction in elementary functions make it likely that the result is accurate to less than half of machine precision.
204	Total loss of accuracy.
	z is complex and $\lvert\mathtt{z}\rvert$ is too large, so that errors due to argument reduction in elementary functions mean that all precision in the result would be lost. The value zero is returned.
205	Possibility of overflow.
	z is complex and the real part of $2.0\mathtt{z}\sqrt{\mathtt{z}}/3.0$ is too large, so that overflow may occur during the calculations. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.
206	Termination condition has not been met.
	This error may occur because z is complex and the arguments would have caused overflow or underflow. This problem may be avoided if the optional argument `scale` is used and set to `.true.`.
207	Possibility of underflow.
	The returned result is set to zero since there is a danger of underflow. This can only occur when z is complex and `scale` = `.false.`.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

Real Arguments

For real arguments the following expansions are used to evaluate $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$:

- For $z < -5$,

$$\begin{aligned}\mathrm{Ai}(z) &= \sqrt[4]{-z}\left[a_1(t)\sin y - b_1(t)\cos y\right], \\ \mathrm{Ai}'(z) &= \sqrt[4]{-z}\left[a_2(t)\cos y + \frac{b_2(t)}{\zeta}\sin y\right],\end{aligned}$$

where $y = \pi/4 + \zeta$, $\zeta = 2\sqrt{-z^3}/3$ and $a_1(t)$, $b_1(t)$, $a_2(t)$ and $b_2(t)$ are expansions in $t = -2(5/z)^3 - 1$.

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

$$\mathrm{Ai}(z) = f_1(t) - zg_1(t), \quad \mathrm{Ai}'(z) = z^2 f_2(t) - g_2(t),$$

  where f_1, g_1, f_2 and g_2 are expansions in $t = -2(z/5)^3 - 1$.

- For $0 < z < 4.5$,

$$\mathrm{Ai}(z) = e^{-3z/2} s_1(t), \quad \mathrm{Ai}'(z) = e^{-11z/8} s_2(t),$$

  where s_1 and s_2 are expansions in $t = 4z/9 - 1$.

- For $4.5 \leq z < 9$,

$$\mathrm{Ai}(z) = e^{-5z/2} u_1(t), \quad \mathrm{Ai}'(z) = e^{-5z/2} u_2(t),$$

  where u_1 and u_2 are expansions in $t = 4z/9 - 3$.

- For $z \geq 9$,

$$\mathrm{Ai}(z) = \sqrt[4]{z} e^{-y} v_1(t), \quad \mathrm{Ai}'(z) = \sqrt[4]{-z} e^{-y} v_2(t),$$

  where $y = 2\sqrt{z^3}/3$ and v_1 and v_2 are expansions in $t = 36/y - 1$.

- For $|z| <$ `EPSILON(1.0_wp)`, the results are set directly to $\mathrm{Ai}(0)$ and $\mathrm{Ai}'(0)$ respectively. This saves time and guards against underflow in intermediate calculations.

- For large negative arguments it becomes impossible to calculate the phase of the oscillatory function with any accuracy and the procedure fails. This occurs when

$$z < -\left(\frac{3}{2 \times \mathtt{EPSILON}(1.0\_wp)}\right)^{2/3}$$

  for evaluation of $\mathrm{Ai}(z)$ and when

$$z < -\left(\frac{\sqrt{\pi}}{\mathtt{EPSILON}(1.0\_wp)}\right)^{4/7}$$

  for evaluation of $\mathrm{Ai}'(z)$.

- For large positive arguments, where Ai and Ai′ decay in an essentially exponential manner, there is a danger of underflow so the procedure fails.

Complex Arguments

For complex arguments the procedure is derived from the routine CAIRY in Amos [2]. It is based on the relations $\mathrm{Ai}(z) = \dfrac{\sqrt{z} K_{1/3}(w)}{\pi\sqrt{3}}$, and $\mathrm{Ai}'(z) = \dfrac{-z K_{2/3}(w)}{\pi\sqrt{3}}$, where K_ν is the modified Bessel function and $w = 2z\sqrt{z}/3$.

For very large $|z|$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$, the computation is performed but the results are accurate to less than half of the machine precision. If the real part of w is too large and an unscaled function is required, there is a risk of overflow and no computation is performed.

6.2 Accuracy

Real Arguments

For a real argument z, the accuracy in calculating $\mathrm{Ai}(z)$ or $\mathrm{Ai}'(z)$ depends on the value of z.

For negative arguments the functions are oscillatory and hence absolute error is the appropriate measure. In the positive region the function is essentially exponential in character and here relative error is appropriate. The absolute error E_1 and the relative error ε_1 in the computed value of $\mathrm{Ai}(z)$ are related in principle to the relative error in the argument, δ, by

$$E_1 \simeq |z\,\mathrm{Ai}'(z)|\delta, \quad \varepsilon_1 \simeq \left|\frac{z\,\mathrm{Ai}'(z)}{\mathrm{Ai}(z)}\right|\delta.$$

Similarly, the absolute error E_2 and relative error ε_2 in the computed value of $\mathrm{Ai}'(z)$ satisfy

$$E_2 \simeq |z^2\,\mathrm{Ai}(z)|\delta, \quad \varepsilon_2 \simeq \left|\frac{z^2\,\mathrm{Ai}(z)}{\mathrm{Ai}'(z)}\right|\delta$$

in principle. In practice, approximate equality is the best that can be expected. When δ, E_1, E_2, ε_1 or ε_2 is of the order of `EPSILON(1.0_wp)`, the errors in the result will be somewhat larger.

For small z, errors are strongly damped by the function and hence will be bounded essentially by the value of `EPSILON(1.0_wp)`.

For moderate to large negative z, the error in $\mathrm{Ai}(z)$ is oscillatory, and the amplitude E_1/δ of the error grows like $|z|^{5/4}/\sqrt{\pi}$. However the phase error will be growing roughly like $2|z|^{3/2}/3$ and hence all accuracy will be lost for large negative arguments due to the impossibility of calculating sin and cos to any accuracy if $2|z|^{3/2}/3 > 1/\delta$. Similarly, the amplitude E_2/δ of the error in $\mathrm{Ai}'(z)$ grows like $|z|^{7/4}/\sqrt{\pi}$. Therefore it becomes impossible to calculate the function with any accuracy if $|z|^{7/4} > \sqrt{\pi}/\delta$.

For large positive arguments, the relative error amplification for computation of both $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$ is considerable:

$$\frac{\varepsilon_1}{\delta} \sim \sqrt{z^3}, \quad \frac{\varepsilon_2}{\delta} \sim \sqrt{z^3}.$$

However, very large arguments are not possible due to the danger of underflow. Thus in practice error amplification is limited.

Complex Arguments

For complex z, all constants used by this procedure are given to approximately 18 digits of precision. Let t denote the number of digits of precision in the floating-point arithmetic being used. Clearly the maximum number of correct digits in the results obtained is limited by $p = \min(t, 18)$. Because of errors in argument reduction occurring during the evaluation of elementary functions by this procedure, the actual number of correct digits is limited, in general, by $p - s$, where $s \approx \max(1, |\log_{10}|z||)$ represents the number of digits lost due to the argument reduction. Thus the larger the value of $|z|$, the less the precision in the result.

Empirical tests with modest values of z, checking relations between Airy functions $\mathrm{Ai}(z)$, $\mathrm{Ai}'(z)$, $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$, have shown errors limited to the least significant 3–4 digits of precision.

Procedure: nag_airy_bi

1 Description

nag_airy_bi evaluates an approximation to the Airy function $\mathrm{Bi}(z)$ or its derivative $\mathrm{Bi}'(z)$.

2 Usage

```
USE nag_airy_fun
[value =] nag_airy_bi(z  [, optional arguments])
```

The function result is a scalar, of the same type as z, containing $\mathrm{Bi}(z)$ or $\mathrm{Bi}'(z)$.

3 Arguments

3.1 Mandatory Argument

z — real(kind=*wp*)/complex(kind=*wp*), intent(in)

Input: the argument z of the function.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

deriv — logical, intent(in), optional

Input: specifies whether the function or its derivative is required.

If deriv = .false., $\mathrm{Bi}(z)$ is returned;

if deriv = .true., $\mathrm{Bi}'(z)$ is returned.

Default: deriv = .false..

scale — logical, intent(in), optional

Input: specifies whether or not the result is scaled when z is complex.

If scale = .true., and z is complex the result is returned scaled by the factor $e^{|\mathrm{Re}(2z\sqrt{z}/3)|}$;

if scale = .false., the result is returned unscaled.

Default: scale = .false..

Note: when z is real, scale is ignored.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow. z is real, large and positive. No computation has been performed due to the likelihood of overflow since $\mathrm{Bi}(z)$ grows in an exponential manner. The value zero is returned.
202	Impossible to calculate phase accurately. z is real, large and negative. It is impossible to calculate the phase of the oscillatory function with any precision. The value zero is returned.
203	Partial loss of accuracy. z is complex and |z| is too large, so that errors due to argument reduction in elementary functions make it likely that the result is accurate to less than half of machine precision.
204	Total loss of accuracy. z is complex and |z| is too large, so that errors due to argument reduction in elementary functions mean that all precision in the result would be lost. The value zero is returned.
205	Possibility of overflow. z is complex and the real part of (z) is too large, so that overflow may occur during the calculations. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.
206	Termination condition has not been met. This error may occur because z is complex and the arguments would have caused overflow or underflow. This problem may be avoided if the optional argument `scale` is used and set to `.true.`.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

Real Arguments

For real arguments the following expansions are used to evaluate $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$:

- For $z < -5$,

$$\begin{aligned}\mathrm{Bi}(z) &= \sqrt[4]{-z}\left[a_1(t)\cos y + b_1(t)\sin y\right], \\ \mathrm{Bi}'(z) &= \sqrt[4]{-z}\left[-a_2(t)\sin y + \frac{b_2(t)}{\zeta}\cos y\right],\end{aligned}$$

  where $y = \pi/4+\zeta$, $\zeta = 2\sqrt{-z^3}/3$ and $a_1(t)$, $b_1(t)$, $a_2(t)$ and $b_2(t)$ are expansions in $t = -2(5/z)^3-1$.

- For $-5 \le z \le 0$,

$$\mathrm{Bi}(z) = \sqrt{3}(f_1(t) + zg_1(t)), \quad \mathrm{Bi}'(z) = \sqrt{3}(z^2 f_2(t) + g_2(t)),$$

  where f_1, g_1, f_2 and g_2 are expansions in $t = -2(z/5)^3 - 1$.

- For $0 < z < 4.5$,

$$\mathrm{Bi}(z) = e^{11z/8} s_1(t), \quad \mathrm{Bi}'(z) = e^{3z/2} s_2(t),$$

  where s_1 and s_2 are expansions in $t = 4z/9 - 1$.

- For $4.5 \le z < 9$,

$$\mathrm{Bi}(z) = e^{5z/2} u_1(t), \quad \mathrm{Bi}'(z) = e^{21z/8} u_2(t),$$

  where u_1 and u_2 are expansions in $t = 4z/9 - 3$.

- For $z \ge 9$,

$$\mathrm{Bi}(z) = \sqrt[4]{z} e^y v_1(t), \quad \mathrm{Bi}'(z) = \sqrt[4]{z} e^y v_2(t),$$

  where $y = 2\sqrt{z^3}/3$ and v_1 and v_2 are expansions in $t = 36/y - 1$.

- For $|z| <$ `EPSILON(1.0_`*wp*`)`, the results are set directly to $\mathrm{Bi}(0)$ and $\mathrm{Bi}'(0)$ respectively. This saves time and guards against underflow in intermediate calculations.

- For large negative arguments it becomes impossible to calculate the phase of the oscillatory function with any accuracy and the procedure fails. This occurs when

$$z < -\left(\frac{3}{2 \times \mathtt{EPSILON}(1.0\_wp)}\right)^{2/3}$$

  for evaluation of $\mathrm{Bi}(z)$ and when

$$z < -\left(\frac{\sqrt{\pi}}{\mathtt{EPSILON}(1.0\_wp)}\right)^{4/7}$$

  for evaluation of $\mathrm{Bi}'(z)$.

- For large positive arguments, where Bi and Bi$'$ grow in an essentially exponential manner, there is a danger of overflow so the procedure fails.

Complex Arguments

For complex arguments the procedure is derived from the routine CBIRY in Amos [2]. It is based on the relations $\mathrm{Bi}(z) = \frac{\sqrt{z}}{\sqrt{3}}(I_{-1/3}(w) + I_{1/3}(w))$, and $\mathrm{Bi}'(z) = \frac{z}{\sqrt{3}}(I_{-2/3}(w) + I_{2/3}(w))$, where I_ν is the modified Bessel function and $w = 2z\sqrt{z}/3$.

For very large $|z|$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$, the computation is performed but the results are accurate to less than half of the machine precision. If the real part of z is too large and an unscaled function is required, there is a risk of overflow and no computation is performed.

6.2 Accuracy

Real Arguments

For a real argument z, the accuracy in calculating $\mathrm{Bi}(z)$ or $\mathrm{Bi}'(z)$ depends on the value of z.

For negative arguments the functions are oscillatory and hence absolute error is the appropriate measure. In the positive region the function is essentially exponential in character and here relative error is appropriate. The absolute error E_1 and the relative error ε_1 are related in principle to the relative error in the argument, δ, by

$$E_1 \simeq |z\,\mathrm{Bi}'(z)|\delta, \quad \varepsilon_1 \simeq \left|\frac{z\,\mathrm{Bi}'(z)}{\mathrm{Bi}(z)}\right|\delta.$$

Similarly, the absolute error E_2 and relative error ε_2 in the computed value of $\mathrm{Bi}'(z)$ satisfy

$$E_2 \simeq |z^2 \,\mathrm{Bi}(z)|\delta, \quad \varepsilon_2 \simeq \left|\frac{z^2 \,\mathrm{Bi}(z)}{\mathrm{Bi}'(z)}\right|\delta$$

in principle. In practice, approximate equality is the best that can be expected. When δ, E_1, E_2, ε_1 or ε_2 is of the order of EPSILON(1.0_*wp*), the errors in the result will be somewhat larger.

For small z, errors are strongly damped and hence will be bounded essentially by the value of EPSILON(1.0_*wp*).

For moderate to large negative z, the error in $\mathrm{Bi}(z)$ is oscillatory, and the amplitude E_1/δ of the error grows like $|z|^{5/4}/\sqrt{\pi}$. However the phase error will be growing roughly like $2|z|^{3/2}/3$ and hence all accuracy will be lost for large negative arguments due to the impossibility of calculating sin and cos to any accuracy if $2|z|^{3/2}/3 > 1/\delta$. Similarly, the amplitude E_2/δ of the error in $\mathrm{Bi}'(z)$ grows like $|z|^{7/4}/\sqrt{\pi}$. Therefore it becomes impossible to calculate the function with any accuracy if $|z|^{7/4} > \sqrt{\pi}/\delta$.

For large positive arguments, the relative error amplification for computation of both $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$ is considerable:

$$\frac{\varepsilon_1}{\delta} \sim \sqrt{z^3}, \quad \frac{\varepsilon_2}{\delta} \sim \sqrt{z^3}.$$

However, very large arguments are not possible due to the danger of underflow. Thus in practice error amplification is limited.

Complex Arguments

For complex z, all constants used by this procedure are given to approximately 18 digits of precision. Let t denote the number of digits of precision in the floating-point arithmetic being used. Clearly the maximum number of correct digits in the results obtained is limited by $p = \min(t, 18)$. Because of errors in argument reduction occurring during the evaluation of elementary functions by this procedure, the actual number of correct digits is limited, in general, by $p - s$, where $s \approx \max(1, |\log_{10}|z||)$ represents the number of digits lost due to the argument reduction. Thus the larger the value of $|z|$, the less the precision in the result.

Empirical tests with modest values of z, checking relations between Airy functions $\mathrm{Ai}(z)$, $\mathrm{Ai}'(z)$, $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$, have shown errors limited to the least significant 3–4 digits of precision.

Example 1: A simple use of nag_airy_ai

This example program uses the function **nag_airy_ai** to evaluate $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$ for real and complex values.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_airy_fun_ex01

  ! Example Program Text for nag_airy_fun
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_airy_fun, ONLY : nag_airy_ai
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  REAL (wp) :: result_r
  COMPLEX (wp) :: result_c
  ! .. Local Arrays ..
  REAL (wp) :: x(7)
  COMPLEX (wp) :: z(4)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_airy_fun_ex01'

  x = (/ -10.0_wp, -1.0_wp, 0.0_wp, 1.0_wp, 5.0_wp, 10.0_wp, 20.0_wp/)

  ! Airy function Ai - real arguments
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '       x           Ai(x)'
  DO i = 1, 7

    result_r = nag_airy_ai(x(i))

    WRITE (nag_std_out,'(2(1x,1p,E12.3))') x(i), result_r
  END DO

  ! Derivative of Airy function Ai - real arguments
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '       x           Ai''(x)'
  DO i = 1, 7

    result_r = nag_airy_ai(x(i),deriv=.TRUE.)

    WRITE (nag_std_out,'(2(1x,1p,E12.3))') x(i), result_r
  END DO

  z(1) = (0.3_wp,0.4_wp)
  z(2) = (0.2_wp,0.0_wp)
  z(3) = (1.1_wp,-6.6_wp)
  z(4) = (-1.0_wp,0.0_wp)
```

```
    ! Airy function Ai - complex arguments, no scaling
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z                  Ai(z)'
    DO i = 1, 4

      result_c = nag_airy_ai(z(i))

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

    ! Airy function Ai - complex arguments, scaled
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z               scaled  Ai(z)'
    DO i = 1, 4

      result_c = nag_airy_ai(z(i),scale=.TRUE.)

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

    ! Derivative of Airy function Ai - complex argument, no scaling
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z                  Ai''(z)'
    DO i = 1, 4

      result_c = nag_airy_ai(z(i),deriv=.TRUE.)

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

    ! Derivative of Airy function Ai - complex argument, scaled
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z              scaled  Ai''(z)'
    DO i = 1, 4

      result_c = nag_airy_ai(z(i),scale=.TRUE.,deriv=.TRUE.)

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

  END PROGRAM nag_airy_fun_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_airy_fun_ex01

       x          Ai(x)
   -1.000E+01    4.024E-02
   -1.000E+00    5.356E-01
    0.000E+00    3.550E-01
    1.000E+00    1.353E-01
    5.000E+00    1.083E-04
    1.000E+01    1.105E-10
    2.000E+01    1.692E-27

       x          Ai'(x)
   -1.000E+01    9.963E-01
```

```
 -1.000E+00   -1.016E-02
  0.000E+00   -2.588E-01
  1.000E+00   -1.591E-01
  5.000E+00   -2.474E-04
  1.000E+01   -3.521E-10
  2.000E+01   -7.586E-27

          z                  Ai(z)
(  0.3000,  0.4000)  (  0.2716, -0.1002)
(  0.2000,  0.0000)  (  0.3037,  0.0000)
(  1.1000, -6.6000)  (-43.6632,-47.9030)
( -1.0000,  0.0000)  (  0.5356,  0.0000)

          z               scaled  Ai(z)
(  0.3000,  0.4000)  (  0.2998, -0.0366)
(  0.2000,  0.0000)  (  0.3224,  0.0000)
(  1.1000, -6.6000)  (  0.1655,  0.0597)
( -1.0000,  0.0000)  (  0.4209, -0.3312)

          z                  Ai'(z)
(  0.3000,  0.4000)  ( -0.2612,  0.0385)
(  0.2000,  0.0000)  ( -0.2524,  0.0000)
(  1.1000, -6.6000)  (164.8134, 23.5278)
( -1.0000,  0.0000)  ( -0.0102,  0.0000)

          z              scaled  Ai'(z)
(  0.3000,  0.4000)  ( -0.2744, -0.0236)
(  0.2000,  0.0000)  ( -0.2679,  0.0000)
(  1.1000, -6.6000)  ( -0.4254,  0.1523)
( -1.0000,  0.0000)  ( -0.0080,  0.0063)
```

Example 2: A simple use of nag_airy_bi

This example program uses the function **nag_airy_bi** to evaluate $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$ for real and complex values.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_airy_fun_ex02

  ! Example Program Text for nag_airy_fun
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_airy_fun, ONLY : nag_airy_bi
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  REAL (wp) :: result_r
  COMPLEX (wp) :: result_c
  ! .. Local Arrays ..
  REAL (wp) :: x(7)
  COMPLEX (wp) :: z(4)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_airy_fun_ex02'

  x = (/ -10.0_wp, -1.0_wp, 0.0_wp, 1.0_wp, 5.0_wp, 10.0_wp, 20.0_wp/)

  ! Airy function Bi - real arguments
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '       x          Bi(x)'
  DO i = 1, 7

    result_r = nag_airy_bi(x(i))

    WRITE (nag_std_out,'(2(1x,1p,E12.3))') x(i), result_r
  END DO

  ! Derivative of Airy function Bi - real arguments
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '       x          Bi''(x)'
  DO i = 1, 7

    result_r = nag_airy_bi(x(i),deriv=.TRUE.)

    WRITE (nag_std_out,'(2(1x,1p,E12.3))') x(i), result_r
  END DO

  z(1) = (0.3_wp,0.4_wp)
  z(2) = (0.2_wp,0.0_wp)
  z(3) = (1.1_wp,-6.6_wp)
  z(4) = (-1.0_wp,0.0_wp)
```

```
    ! Airy function Bi - complex arguments, no scaling
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z                  Bi(z)'
    DO i = 1, 4

      result_c = nag_airy_bi(z(i))

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

    ! Airy function Bi - complex arguments, scaled
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z               scaled  Bi(z)'
    DO i = 1, 4

      result_c = nag_airy_bi(z(i),scale=.TRUE.)

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

    ! Derivative of Airy function Bi - complex argument, no scaling
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z                  Bi''(z)'
    DO i = 1, 4

      result_c = nag_airy_bi(z(i),deriv=.TRUE.)

      WRITE (nag_std_out,'(2(2x,''('',F8.3,'','',F8.3,'')''))') z(i), result_c
    END DO

    ! Derivative of Airy function Bi - complex argument, scaled
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '          z              scaled  Bi''(z)'
    DO i = 1, 4

      result_c = nag_airy_bi(z(i),scale=.TRUE.,deriv=.TRUE.)

      WRITE (nag_std_out,'(2(2x,''('',F8.4,'','',F8.4,'')''))') z(i), result_c
    END DO

  END PROGRAM nag_airy_fun_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_airy_fun_ex02

       x          Bi(x)
  -1.000E+01   -3.147E-01
  -1.000E+00    1.040E-01
   0.000E+00    6.149E-01
   1.000E+00    1.207E+00
   5.000E+00    6.578E+02
   1.000E+01    4.556E+08
   2.000E+01    2.104E+25

       x          Bi'(x)
  -1.000E+01    1.194E-01
```

```
 -1.000E+00    5.924E-01
  0.000E+00    4.483E-01
  1.000E+00    9.324E-01
  5.000E+00    1.436E+03
  1.000E+01    1.429E+09
  2.000E+01    9.382E+25

        z                 Bi(z)
(  0.3000,  0.4000)  (  0.7355,  0.1825)
(  0.2000,  0.0000)  (  0.7055,  0.0000)
(  1.1000, -6.6000)  (-47.9039, 43.6634)
( -1.0000,  0.0000)  (  0.1040,  0.0000)

        z              scaled  Bi(z)
(  0.3000,  0.4000)  (  0.7051,  0.1750)
(  0.2000,  0.0000)  (  0.6646,  0.0000)
(  1.1000, -6.6000)  ( -0.1300,  0.1185)
( -1.0000,  0.0000)  (  0.1040,  0.0000)

        z                 Bi'(z)
(   0.300,   0.400)  (   0.409,   0.080)
(   0.200,   0.000)  (   0.462,   0.000)
(   1.100,  -6.600)  (  23.526,-164.812)
(  -1.000,   0.000)  (   0.592,   0.000)

        z             scaled  Bi'(z)
(  0.3000,  0.4000)  (  0.3924,  0.0764)
(  0.2000,  0.0000)  (  0.4351,  0.0000)
(  1.1000, -6.6000)  (  0.0638, -0.4473)
( -1.0000,  0.0000)  (  0.5924,  0.0000)
```

References

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

[2] Amos D E (1986) Algorithm 644: A portable package for Bessel functions of a complex argument and nonnegative order *ACM Trans. Math. Software* **12** 265–273