Module 3.4: nag_bessel_fun
Bessel Functions

nag_bessel_fun contains procedures for approximating Bessel functions and modified Bessel functions for real arguments x or complex arguments z.

Contents

Introduction

This module contains procedures for approximating Bessel functions and modified Bessel functions of the first and second kinds, for real or complex arguments.

The Bessel functions $J_\nu(z)$ and $Y_\nu(z)$ are linearly independent solutions of the differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0,$$

such that $J_\nu(z)$ is bounded as $z \to 0$. $J_\nu(z)$ and $Y_\nu(z)$ are known as Bessel functions of the first and second kinds respectively. For complex arguments the procedures `nag_bessel_j` and `nag_bessel_y` approximate the values of the functions $J_\nu(z)$ and $Y_\nu(z)$ respectively. The procedures `nag_bessel_j0`, `nag_bessel_j1`, `nag_bessel_y0` and `nag_bessel_y1` are for real arguments and approximate the values of the functions $J_0(x)$, $J_1(x)$, $Y_0(x)$ and $Y_1(x)$ respectively.

Similarly, the modified Bessel functions $I_\nu(z)$ and $K_\nu(z)$ are linearly independent solutions of the differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0,$$

such that $I_\nu(z)$ is bounded as $z \to 0$. $I_\nu(z)$ and $K_\nu(z)$ are known as modified Bessel functions of the first and second kinds respectively. For complex arguments the procedures `nag_bessel_i` and `nag_bessel_k` approximate the values of the functions $I_\nu(z)$ and $K_\nu(z)$ respectively. The procedures `nag_bessel_i0`, `nag_bessel_i1`, `nag_bessel_k0` and `nag_bessel_k1` are for real arguments and approximate the values of the functions $I_0(x)$, $I_1(x)$, $K_0(x)$ and $K_1(x)$ respectively.

The procedures for functions of a real argument are in general based on expansions in terms of Chebyshev polynomials $T_r(t) = \cos(r \arccos t)$, where $t = t(x)$ is a mapping from the region of interest to the interval $[-1, 1]$, on which the Chebyshev polynomials are defined. Further details appear in Section 6.1 of the individual procedure documents.

The procedures for functions of a complex argument relate all functions to the modified Bessel functions I_ν and K_ν computed in the right-hand half complex plane, including their analytic continuations. I_ν and K_ν are computed by different methods according to the values of z and ν. The methods include power series, asymptotic expansions and Wronskian evaluations.

For further details of Bessel and modified Bessel functions, see Abramowitz and Stegun [1], Chapter 9.

Procedure: nag_bessel_j0

1 Description

nag_bessel_j0 evaluates an approximation to the Bessel function of the first kind $J_0(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_j0(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $J_0(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)
Input: the argument x of the function.

3.2 Optional Argument

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	The result is not accurate.
	Argument x is too large for meaningful accuracy. Phase cannot be calculated accurately. This procedure returns the amplitude of the J_0 oscillation, $\sqrt{2/(\pi\|x\|)}$.

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

Since $J_0(-x) = J_0(x)$, we need only consider the case $x \geq 0$.

- For $0 < x \leq 8$, the procedure uses a Chebyshev expansion of the form

$$J_0(x) = {\sum_{r=0}}' a_r T_r(t), \quad \text{with} \quad t = 2\left(\frac{x}{8}\right)^2 - 1.$$

- For $x > 8$, it uses

$$J_0(x) = \sqrt{\frac{2}{\pi x}} \left[P_0(x) \cos\left(x - \frac{\pi}{4}\right) - Q_0(x) \sin\left(x - \frac{\pi}{4}\right) \right]$$

where $P_0(x) = \sum_{r=0}{}' b_r T_r(t)$, and $Q_0(x) = \frac{8}{x} \sum_{r=0}{}' c_r T_r(t)$, with $t = 2\left(\frac{8}{x}\right)^2 - 1$.

- For x near zero, $J_0(x) \simeq 1$. This approximation is used when x is sufficiently small for the result to be correct to `EPSILON(1.0_`*wp*`)`.
- For very large x, it becomes impossible to provide results with any reasonable accuracy (see Section 6.2), hence the procedure fails. Such arguments contain insufficient information to determine the phase of oscillation of $J_0(x)$; only the amplitude, $\sqrt{2/(\pi|x|)}$, can be determined and this is returned if `error%code` = 201 on exit. The range for which this occurs is roughly related to `EPSILON(1.0_`*wp*`)`; the procedure will fail if $|x| \gtrsim 1/$`EPSILON(1.0_`*wp*`)`.

6.2 Accuracy

Let δ be the relative error in the argument and E be the absolute error in the result. (Since $J_0(x)$ oscillates about zero, absolute error and not relative error is significant.)

If δ is somewhat larger than `EPSILON(1.0_`*wp*`)` (e.g., if δ is due to data errors etc.), then E and δ are approximately related by

$$E \simeq |\theta|\delta, \quad \text{where} \quad \theta = xJ_1(x)$$

(provided E is also within machine bounds). The behaviour of the amplification factor $|\theta|$ is shown in Figure 1.

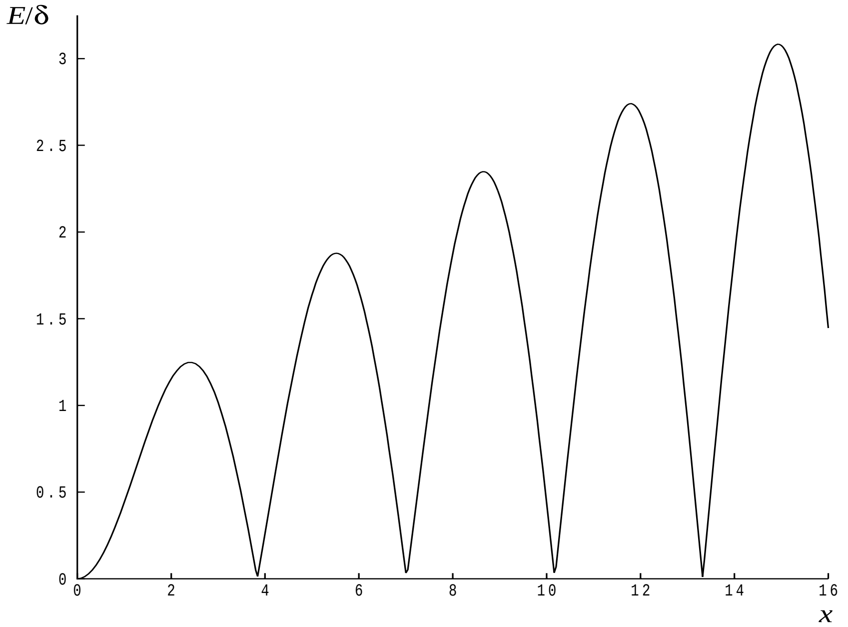

Figure 1: The error amplification factor $|\theta|$.

However, if δ is of the same order as `EPSILON(1.0_`*wp*`)`, then rounding errors could make E slightly larger than the above relation predicts.

For very large x, the above relation ceases to apply. In this region,

$$J_0(x) \simeq \sqrt{\frac{2}{\pi|x|}} \cos\left(x - \frac{\pi}{4}\right).$$

The amplitude $\sqrt{2/(\pi|x|)}$ can be calculated with reasonable accuracy for all x, but $\cos\left(x - \frac{\pi}{4}\right)$ cannot. If $x - \frac{\pi}{4}$ is written as $2N\pi + \phi$ where N is an integer and $0 \le \phi < 2\pi$, then $\cos\left(x - \frac{\pi}{4}\right)$ is determined by ϕ only. If $x \gtrsim \delta^{-1}$, ϕ cannot be determined with any accuracy at all. Thus if x is greater than, or of the order of, the inverse of `EPSILON(1.0_`wp`)`, it is impossible to calculate the phase of $J_0(x)$ and the procedure must fail.

Procedure: nag_bessel_j1

1 Description

nag_bessel_j1 evaluates an approximation to the Bessel function of the first kind $J_1(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_j1(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $J_1(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)
Input: the argument x of the function.

3.2 Optional Argument

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	The result is not accurate.
	Argument x is too large for meaningful accuracy. Phase cannot be calculated accurately. This procedure returns the amplitude of the J_1 oscillation, $\sqrt{2/(\pi\|x\|)}$.

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

Since $J_1(-x) = -J_1(x)$, we need only consider the case $x \geq 0$.

- For $0 < x \leq 8$, the procedure uses a Chebyshev expansion of the form

$$J_1(x) = \frac{x}{8} \sum_{r=0}{}' a_r T_r(t), \quad \text{with} \quad t = 2\left(\frac{x}{8}\right)^2 - 1.$$

- For $x > 8$, it uses

$$J_1(x) = \sqrt{\frac{2}{\pi x}} \left[P_1(x) \cos\left(x - \frac{3\pi}{4}\right) - Q_1(x) \sin\left(x - \frac{3\pi}{4}\right) \right]$$

  where $P_1(x) = \sum_{r=0}{}' b_r T_r(t)$, and $Q_1(x) = \frac{8}{x} \sum_{r=0}{}' c_r T_r(t)$, with $t = 2\left(\frac{8}{x}\right)^2 - 1$.

- For x near zero, $J_1(x) \simeq x/2$. This approximation is used when x is sufficiently small for the result to be correct to `EPSILON(1.0_wp)`.

- For very large x, it becomes impossible to provide results with any reasonable accuracy (see Section 6.2), hence the procedure fails. Such arguments contain insufficient information to determine the phase of oscillation of $J_1(x)$; only the amplitude, $\sqrt{2/(\pi|x|)}$, can be determined and this is returned if `error%code` = 201 on exit. The range for which this occurs is roughly related to `EPSILON(1.0_wp)`; the procedure will fail if $|x| \gtrsim 1/$`EPSILON(1.0_wp)`.

6.2 Accuracy

Let δ be the relative error in the argument and E be the absolute error in the result. (Since $J_1(x)$ oscillates about zero, absolute error and not relative error is significant.)

If δ is somewhat larger than `EPSILON(1.0_wp)` (e.g., if δ is due to data errors etc.), then E and δ are approximately related by

$$E \simeq |\theta|\delta, \quad \text{where} \quad \theta = xJ_0(x) - J_1(x)$$

(provided E is also within machine bounds). The behaviour of the amplification factor $|\theta|$ is shown in Figure 2.

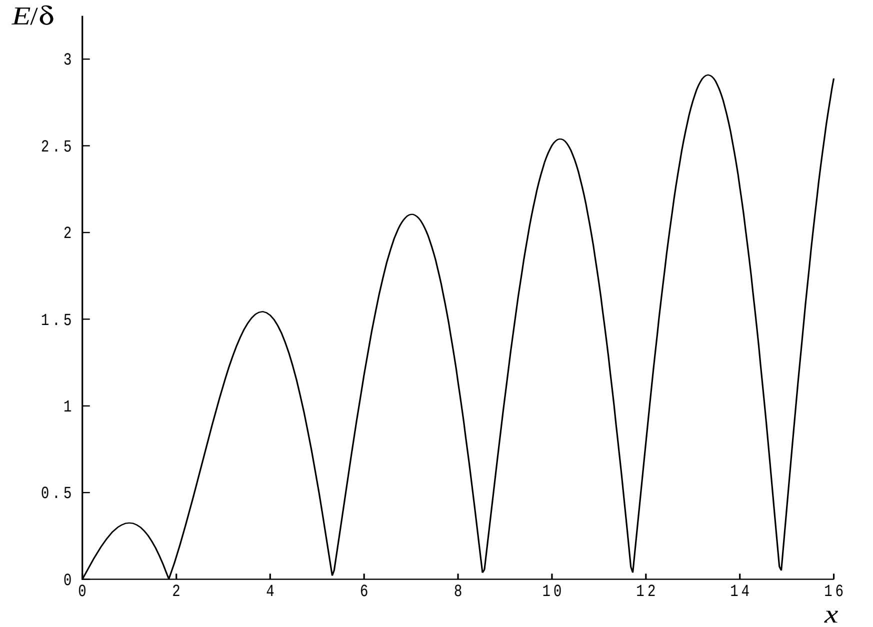

Figure 2: The error amplification factor $|\theta|$.

However, if δ is of the same order as `EPSILON(1.0_wp)`, then rounding errors could make E slightly larger than the above relation predicts.

For very large x, the above relation ceases to apply. In this region,

$$J_1(x) \simeq \sqrt{\frac{2}{\pi|x|}} \cos\left(x - \frac{3\pi}{4}\right).$$

The amplitude $\sqrt{2/(\pi|x|)}$ can be calculated with reasonable accuracy for all x, but $\cos\left(x - \frac{3\pi}{4}\right)$ cannot. If $x - \frac{3\pi}{4}$ is written as $2N\pi + \phi$ where N is an integer and $0 \le \phi < 2\pi$, then $\cos\left(x - \frac{3\pi}{4}\right)$ is determined by ϕ only. If $x \gtrsim \delta^{-1}$, ϕ cannot be determined with any accuracy at all. Thus if x is greater than, or of the order of the reciprocal of EPSILON(1.0_*wp*), it is impossible to calculate the phase of $J_1(x)$ and the procedure must fail.

Procedure: nag_bessel_j

1 Description

nag_bessel_j evaluates an approximation to either the Bessel function of the first kind $J_\nu(z)$, or the sequence of Bessel functions of the first kind $J_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$. The real non-negative order is given by ν (or $\nu + n$) and the complex argument z is such that $-\pi < \arg z \leq \pi$. There is also an option for scaling the result.

2 Usage

```
USE nag_bessel_fun
```

[*value* =] nag_bessel_j(z, nu [, *optional arguments*])

The function result is a scalar of type complex(kind=*wp*), or

[*value* =] nag_bessel_j(z, nu, n [, *optional arguments*])

The function returns an array-valued result of type complex(kind=*wp*) and dimension N.

3 Arguments

3.1 Mandatory Arguments

z — complex(kind=*wp*), intent(in)

Input: the argument z of the function.

nu — real(kind=*wp*), intent(in)

Input: the order, ν, of the first member of the sequence of functions.

Constraints: nu ≥ 0.0.

n — integer, intent(in)

Input: the number of terms, N, in the sequence of functions $J_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$.

Constraints: n ≥ 1.

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.

scale — logical, intent(in), optional

Input: determines whether or not the results are scaled.

If scale = .true., then the results are scaled by the factor $e^{-|\mathrm{Im}(\mathbf{z})|}$;

if scale = .false., then the results are returned unscaled.

This option can be used to prevent underflow or overflow from occurring, thus increasing the range of the valid arguments.

Default: scale = .false..

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow. Im(z) is too large. No computation has been performed due to the likelihood of overflow.
202	Total loss of accuracy. $|z|$ or nu + n − 1 is too large, so that errors due to argument reduction in elementary functions mean that all precision would be lost.
203	Partial loss of accuracy. $|z|$ or nu + n − 1 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	Termination condition has not been met. This error may occur because the arguments supplied would have caused overflow or underflow. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.
205	Possibility of underflow. All or some of the returned results have been set to zero because of underflow.

5 Examples of Usage

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

6 Further Comments

If the function required is $J_0(z)$ or $J_1(z)$, i.e., $\nu = 0.0$ or $\nu = 1.0$, where z is real and positive, and only a single unscaled function is required, then it is much cheaper to use the procedure `nag_bessel_j0` or `nag_bessel_j1` respectively.

6.1 Algorithmic Detail

The procedure is derived from the routine CBESJ in Amos [2]. It is based on the relation

$$J_\nu(z) = \begin{cases} e^{\nu\pi i/2} I_\nu(-iz), & \text{Im}(\mathtt{z}) \geq 0.0, \\ e^{-\nu\pi i/2} I_\nu(iz), & \text{Im}(\mathtt{z}) < 0.0. \end{cases}$$

The Bessel function $I_\nu(z)$ is computed using a variety of techniques depending on the region under consideration.

When $N > 1$, extra values of $J_\nu(z)$ are computed using recurrence relations.

Although the procedure may not be called with ν less than zero, for negative orders the formulae

$$J_{-\nu}(z) = J_\nu(z)\cos(\pi\nu) - Y_\nu(z)\sin(\pi\nu)$$

may be used (for the Bessel function $Y_\nu(z)$ see the procedure nag_bessel_y).

For very large $|z|$ or $(\nu + N - 1)$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$ or $(\nu + N - 1)$, the computation is performed but the results are accurate to less than half of machine precision. If Im(z) is large, there is the risk of overflow and so no computation is performed.

6.2 Accuracy

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 $s \approx \max(1, |\log_{10}|z||), |\log_{10}\nu|)$ represents the number of digits lost due to the argument reduction. Thus the larger the values of $|z|$ and ν, the less the precision in the result. If this procedure is called with $N > 1$, then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to this procedure with different base values of ν and different N, the computed values may not agree exactly. Empirical tests with modest values of ν and z have shown that the discrepancy is limited to the least significant 3–4 digits of precision.

Procedure: nag_bessel_y0

1 Description

nag_bessel_y0 evaluates an approximation to the Bessel function of the second kind $Y_0(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_y0(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $Y_0(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)

Input: the argument x of the function.

Constraints: x > 0.0.

3.2 Optional Argument

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	The result is not accurate. Argument x is too large for meaningful accuracy. Phase cannot be calculated accurately. This procedure returns the amplitude of the Y_0 oscillation, $\sqrt{2/(\pi x)}$.

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

- For $x \leq 0$, the result $Y_0(x)$ is undefined and the procedure will fail for such arguments.
- For $0 < x \leq 8$, the procedure uses a Chebyshev expansion of the form

$$Y_0(x) = \frac{2}{\pi} \ln x \sum_{r=0}{}' a_r T_r(t) + \sum_{r=0}{}' b_r T_r(t), \quad \text{with} \quad t = 2\left(\frac{x}{8}\right)^2 - 1,$$

- For $x > 8$, it uses

$$Y_0(x) = \sqrt{\frac{2}{\pi x}} \left[P_0(x) \sin\left(x - \frac{\pi}{4}\right) + Q_0(x) \cos\left(x - \frac{\pi}{4}\right) \right]$$

  where $P_0(x) = \sum_{r=0}{}' c_r T_r(t)$, and $Q_0(x) = \frac{8}{x} \sum_{r=0}{}' d_r T_r(t)$, with $t = 2\left(\frac{8}{x}\right)^2 - 1$.

- For x near zero,

$$Y_0(x) \simeq \frac{2}{\pi} \left(\ln\left(\frac{x}{2}\right) + \gamma \right),$$

  where γ denotes Euler's constant. This approximation is used when x is sufficiently small for the result to be correct to `EPSILON(1.0_wp)`.

- For very large x, it becomes impossible to provide results with any reasonable accuracy (see Section 6.2), hence the procedure fails. Such arguments contain insufficient information to determine the phase of oscillation of $Y_0(x)$; only the amplitude, $\sqrt{2/(\pi x)}$, can be determined and this is returned if `error%code` = 201 on exit. The range for which this occurs is roughly related to `EPSILON(1.0_wp)`: the procedure will fail if $x \gtrsim 1/$`EPSILON(1.0_wp)`.

6.2 Accuracy

Let δ be the relative error in the argument and E be the absolute error in the result. (Since $Y_0(x)$ oscillates about zero, absolute error and not relative error is significant, except for very small x.)

If δ is somewhat larger than the machine representation error (e.g., if δ is due to data errors etc.), then E and δ are approximately related by

$$E \simeq |\theta|\delta \quad \text{where} \quad \theta = xY_1(x)$$

(provided E is also within machine bounds). The behaviour of the amplification factor $|\theta|$ is shown in Figure 3.
However, if δ is of the same order as the machine representation errors, then rounding errors could make E slightly larger than the above relation predicts.

For very small x, the errors are essentially independent of δ and the procedure should provide relative accuracy bounded by `EPSILON(1.0_wp)`.

For very large x, the above relation ceases to apply. In this region,

$$Y_0(x) \simeq \sqrt{\frac{2}{\pi x}} \sin\left(x - \frac{\pi}{4}\right).$$

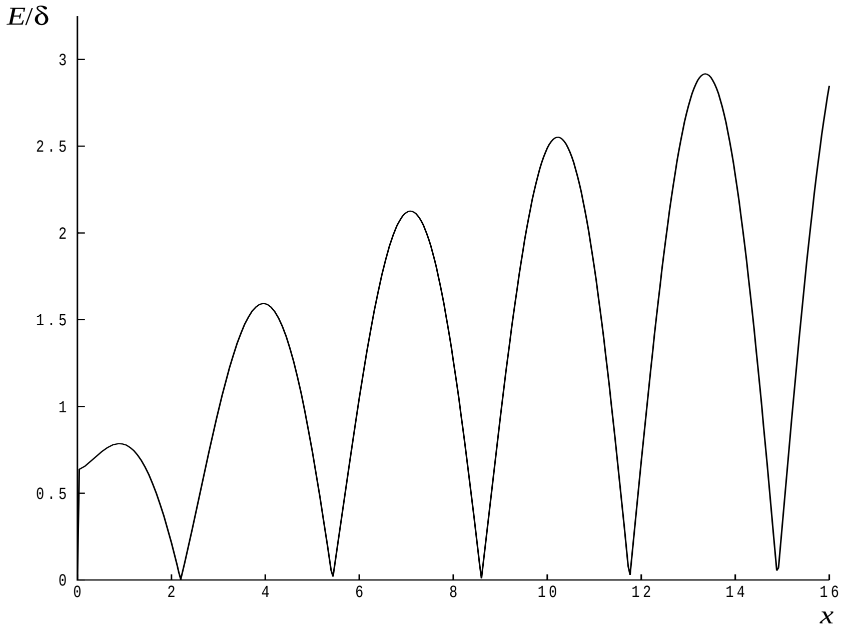

Figure 3: The error amplification factor $|\theta|$.

The amplitude $\sqrt{2/(\pi x)}$ can be calculated with reasonable accuracy for all x, but $\sin\left(x - \frac{\pi}{4}\right)$ cannot. If $x - \frac{\pi}{4}$ is written as $2N\pi + \phi$ where N is an integer and $0 \le \phi < 2\pi$, then $\sin\left(x - \frac{\pi}{4}\right)$ is determined by ϕ only. If $x \gtrsim \delta^{-1}$, ϕ cannot be determined with any accuracy at all. Thus if x is greater than, or of the order of the inverse of `EPSILON(1.0_`*wp*`)`, it is impossible to calculate the phase of $Y_0(x)$ and the procedure must fail.

Procedure: nag_bessel_y1

1 Description

nag_bessel_y1 evaluates an approximation to the Bessel function of the second kind $Y_1(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_y1(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $Y_1(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)

Input: the argument x of the function.

Constraints: x > 0.0.

3.2 Optional Argument

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow. Argument x is too close to zero. This procedure returns the value of $Y_1(x)$ at the nearest valid argument.
202	The result is not accurate. Argument x is too large for meaningful accuracy. Phase cannot be calculated accurately. This procedure returns the amplitude of the Y_1 oscillation, $\sqrt{2/(\pi x)}$.

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

- For $x \leq 0$, the result $Y_1(x)$ is undefined and the procedure will fail for such arguments.
- For $0 < x \leq 8$, the procedure uses a Chebyshev expansion of the form

$$Y_1(x) = \frac{2}{\pi} \ln x \frac{x}{8} \sum_{r=0}{}' a_r T_r(t) - \frac{2}{\pi x} + \frac{x}{8} \sum_{r=0}{}' b_r T_r(t), \quad \text{with} \quad t = 2\left(\frac{x}{8}\right)^2 - 1.$$

- For $x > 8$, it uses

$$Y_1(x) = \sqrt{\frac{2}{\pi x}} \left[P_1(x) \sin\left(x - \frac{3\pi}{4}\right) + Q_1(x) \cos\left(x - \frac{3\pi}{4}\right) \right]$$

  where $P_1(x) = \sum_{r=0}{}' c_r T_r(t)$, and $Q_1(x) = \frac{8}{x} \sum_{r=0}{}' d_r T_r(t)$, with $t = 2\left(\frac{8}{x}\right)^2 - 1$.

- For x near zero, $Y_1(x) \simeq -2/(\pi x)$. This approximation is used when x is sufficiently small for the result to be correct to EPSILON(1.0_*wp*). For extremely small x, there is a danger of overflow in calculating $-2/(\pi x)$ and for such arguments the procedure will fail.
- For very large x, it becomes impossible to provide results with any reasonable accuracy (see Section 6.2), hence the procedure fails. Such arguments contain insufficient information to determine the phase of oscillation of $Y_1(x)$, only the amplitude, $\sqrt{2/(\pi x)}$, can be determined and this is returned if error%code = 202 on exit. The range for which this occurs is roughly related to EPSILON(1.0_*wp*); the procedure will fail if $x \gtrsim 1/$EPSILON(1.0_*wp*).

6.2 Accuracy

Let δ be the relative error in the argument and E be the absolute error in the result. (Since $Y_1(x)$ oscillates about zero, absolute error and not relative error is significant, except for very small x.)

If δ is somewhat larger than EPSILON(1.0_*wp*) (e.g., if δ is due to data errors etc.), then E and δ are approximately related by

$$E \simeq |\theta|\delta, \quad \text{where} \quad \theta = xY_0(x) - Y_1(x)$$

(provided E is also within machine bounds). The behaviour of the amplification factor $|\theta|$ is shown in Figure 4.

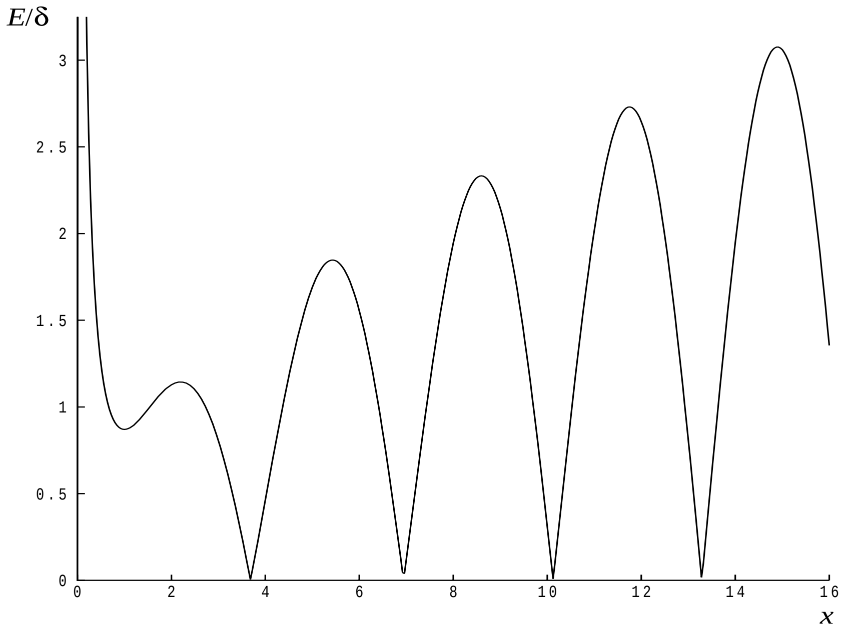

Figure 4: The error amplification factor $|\theta|$.

However, if δ is of the same order as `EPSILON(1.0_`wp`)`, then rounding errors could make E slightly larger than the above relation predicts.

For very small x, the absolute error becomes large, but the relative error in the result is of the same order as δ.

For very large x, the above relation ceases to apply. In this region,

$$Y_1(x) \simeq \frac{2}{\pi x} \sin\left(x - \frac{3\pi}{4}\right).$$

The amplitude $2/(\pi x)$ can be calculated with reasonable accuracy for all x, but $\sin\left(x - \frac{3\pi}{4}\right)$ cannot. If $x - \frac{3\pi}{4}$ is written as $2N\pi + \phi$ where N is an integer and $0 \leq \phi < 2\pi$, then $\sin\left(x - \frac{3\pi}{4}\right)$ is determined by ϕ only. If $x > \delta^{-1}$, ϕ cannot be determined with any accuracy at all. Thus if x is greater than, or of the order of, the inverse of `EPSILON(1.0_`wp`)`, it is impossible to calculate the phase of $Y_1(x)$ and the procedure must fail.

Procedure: nag_bessel_y

1 Description

nag_bessel_y evaluates an approximation to either the Bessel function of the second kind $Y_\nu(z)$, or the sequence of Bessel functions of the second kind $Y_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$. The real non-negative order is given by ν (or $\nu + n$) and the complex argument z is such that $-\pi < \arg z \le \pi$. There is also an option for scaling the result.

2 Usage

```
USE nag_bessel_fun
```

[*value* =] nag_bessel_y(z, nu [, *optional arguments*])

The function result is a scalar of type complex(kind=*wp*), or

[*value* =] nag_bessel_y(z, nu, n [, *optional arguments*])

The function returns an array-valued result of type complex(kind=*wp*) and dimension N.

3 Arguments

3.1 Mandatory Arguments

z — complex(kind=*wp*), intent(in)

Input: the argument z of the function.

Constraints: z $\neq$ (0.0, 0.0).

nu — real(kind=*wp*), intent(in)

Input: the order, ν, of the first member of the sequence of functions.

Constraints: nu $\geq$ 0.0.

n — integer, intent(in)

Input: the number of terms, N, in the sequence of functions $Y_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$.

Constraints: n $\geq$ 1.

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.

scale — logical, intent(in), optional

Input: determines whether or not the results are scaled.

If scale = .true., then the results are scaled by the factor $e^{-|\mathrm{Im}(\mathbf{z})|}$;

if scale = .false., then the results are returned unscaled.

This option can be used to prevent underflow or overflow from occurring, thus increasing the range of the valid arguments.

Default: scale = .false..

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow. |z| is too small. No computation has been performed due to the likelihood of overflow.
202	Total loss of accuracy. |z| or `nu` + `n` − 1 is too large, so that errors due to argument reduction in elementary functions mean that all precision would be lost.
203	Partial loss of accuracy. |z| or `nu` + `n` − 1 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	Termination condition has not been met. This error may occur because the arguments supplied would have caused overflow or underflow. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.
205	Possibility of underflow. All or some of the returned results have been set to zero because of underflow.
206	Possibility of overflow. `nu` + `n` −1 is too large for the given `z`. No computation has been performed due to the likelihood of overflow.

5 Examples of Usage

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

6 Further Comments

If the function required is $Y_0(z)$ or $Y_1(z)$, i.e., $\nu = 0.0$ or $\nu = 1.0$, where z is real and positive, and only a single unscaled function is required, then it is much cheaper to use the procedure `nag_bessel_y0` or `nag_bessel_y1` respectively.

6.1 Algorithmic Detail

The procedure is derived from the routine CBESY in Amos [2]. It is based on the relation

$$Y_\nu(z) = \frac{H_\nu^{(1)}(z) - H_\nu^{(2)}(z)}{2i},$$

where $H_\nu^{(1)}(z)$ and $H_\nu^{(2)}(z)$ are the Hankel functions of the first and second kinds respectively.

When $N > 1$ extra values of $Y_\nu(z)$ are computed using recurrence relations.

Although the procedure may not be called with ν less than zero, for negative orders the formulae

$$Y_{-\nu}(z) = Y_\nu(z)\cos(\pi\nu) + J_\nu(z)\sin(\pi\nu)$$

may be used (for the Bessel function $J_\nu(z)$ see the procedure `nag_bessel_j`).

For very large $|z|$ or $(\nu + N - 1)$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$ or $(\nu + N - 1)$, the computation is performed but the results are accurate to less than half of machine precision. If $|z|$ is very small, near the machine underflow threshold, or $(\nu + N - 1)$ is too large, there is the risk of overflow and so no computation is performed.

6.2 Accuracy

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 $s \approx \max(1, |\log_{10}|z||, |\log_{10}\nu|)$ represents the number of digits lost due to the argument reduction. Thus the larger the values of $|z|$ and ν, the less the precision in the result. If this procedure is called with $N > 1$, then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to this procedure with different base values of ν and different N, the computed values may not agree exactly. Empirical tests with modest values of ν and z have shown that the discrepancy is limited to the least significant 3–4 digits of precision.

Procedure: nag_bessel_i0

1 Description

nag_bessel_i0 evaluates an approximation to the modified Bessel function of the first kind $I_0(x)$ or to the exponentially scaled value $e^{-|x|}I_0(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_i0(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $I_0(x)$ or $e^{-|x|}I_0(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)

Input: the argument x 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.

scale — logical, intent(in), optional

Input: determines whether or not the result is scaled.

If `scale` = `.true.`, then the result is scaled by the factor $e^{-|x|}$;

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

This option can be used to prevent overflow from occurring, thus increasing the range of the valid arguments.

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

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. Argument `x` is too large. This procedure returns the approximate value of $I_0(x)$ at the nearest valid argument. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.

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

Since $I_0(-x) = I_0(x)$, we need only consider the case $x \geq 0$.

- For $0 < x \leq 4$, the procedure uses a Chebyshev expansion of the form

$$I_0(x) = e^x \sum_{r=0}{}' a_r T_r(t) \quad \text{where} \quad t = 2\left(\frac{x}{4}\right) - 1.$$

- For $4 < x \leq 12$, it uses

$$I_0(x) = e^x \sum_{r=0}{}' b_r T_r(t) \quad \text{where} \quad t = \frac{x-8}{4}.$$

- For $x > 12$,

$$I_0(x) = \frac{e^x}{\sqrt{x}} \sum_{r=0}{}' c_r T_r(t) \quad \text{where} \quad t = 2\left(\frac{12}{x}\right) - 1.$$

- For small x, $I_0(x) \simeq 1$. This approximation is used when x is sufficiently small for the result to be correct to EPSILON(1.0_*wp*).
- For large x, the procedure must fail because of the danger of overflow in calculating e^x. To avoid overflow you could calculate the scaled value $e^{-|x|}I_0(x)$ (see the optional argument scale).

6.2 Accuracy

Let δ and ε be the relative errors in the argument and result respectively.

If δ is somewhat larger than EPSILON(1.0_*wp*) (i.e., if δ is due to data errors etc.), then ε and δ are approximately related by

$$\varepsilon \simeq |\theta|\delta, \quad \text{where} \quad \theta = \frac{xI_1(x)}{I_0(x)}.$$

The behaviour of the error amplification factor $|\theta|$ is shown in Figure 5.

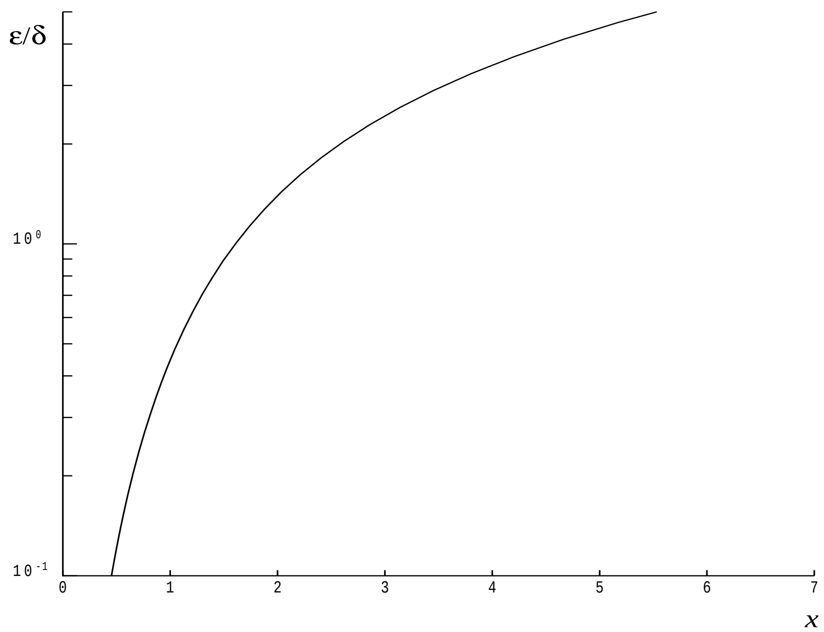

Figure 5: The error amplification factor $|\theta|$.

However, if δ is of the same order as EPSILON(1.0_*wp*), then rounding errors could make ε slightly larger than the above relation predicts.

For small x, the amplification factor is approximately $x^2/2$, which implies strong attenuation of the error, but in general ε can never be less than EPSILON(1.0_*wp*).

For large x, $\varepsilon \simeq x\delta$ and we have strong amplification of errors. However, the procedure must fail for quite moderate values of x, because $I_0(x)$ would overflow; hence in practice the loss of accuracy for large x is not excessive. Note that for large x the errors will be dominated by those of the Fortran intrinsic function EXP.

Procedure: nag_bessel_i1

1 Description

nag_bessel_i1 evaluates an approximation to the modified Bessel function of the first kind $I_1(x)$ or to the exponentially scaled value $e^{-|x|}I_1(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_i1(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $I_1(x)$ or $e^{-|x|}I_1(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)
Input: the argument x 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.

scale — logical, intent(in), optional
Input: determines whether or not the result is scaled.
If `scale` = `.true.`, then the result is scaled by the factor $e^{-|x|}$;
if `scale` = `.false.`, then the result is returned unscaled.
This option can be used to prevent overflow from occurring, thus increasing the range of the valid arguments.
Default: `scale` = `.false.`.

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. Argument `x` is too large. This procedure returns the approximate value of $I_1(x)$ at the nearest valid argument. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.

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

Since $I_1(-x) = -I_1(x)$, we need only consider the case $x \geq 0$.

- For $0 < x \leq 4$, the procedure uses a Chebyshev expansion of the form

$$I_1(x) = x \sum_{r=0}{}' a_r T_r(t), \quad \text{where} \quad t = 2\left(\frac{x}{4}\right)^2 - 1.$$

- For $4 < x \leq 12$, it uses

$$I_1(x) = e^x \sum_{r=0}{}' b_r T_r(t), \quad \text{where} \quad t = \frac{x-8}{4}.$$

- For $x > 12$,

$$I_1(x) = \frac{e^x}{\sqrt{x}} \sum_{r=0}{}' c_r T_r(t), \quad \text{where} \quad t = 2\left(\frac{12}{x}\right) - 1.$$

- For small x, $I_1(x) \simeq x$. This approximation is used when x is sufficiently small for the result to be correct to `EPSILON(1.0_`*wp*`)`.
- For large x, the procedure must fail because $I_1(x)$ cannot be represented without overflow. To avoid overflow you could calculate the scaled value $e^{-|x|} I_1(x)$, (see the optional argument `scale`).

6.2 Accuracy

Let δ and ε be the relative errors in the argument and result respectively.
If δ is somewhat larger than `EPSILON(1.0_`*wp*`)` (i.e., if δ is due to data errors etc.), then ε and δ are approximately related by

$$\varepsilon \simeq |\theta|\delta, \quad \text{where} \quad \theta = \frac{x I_0(x) - I_1(x)}{I_1(x)}.$$

The behaviour of the error amplification factor $|\theta|$ is shown in Figure 6.

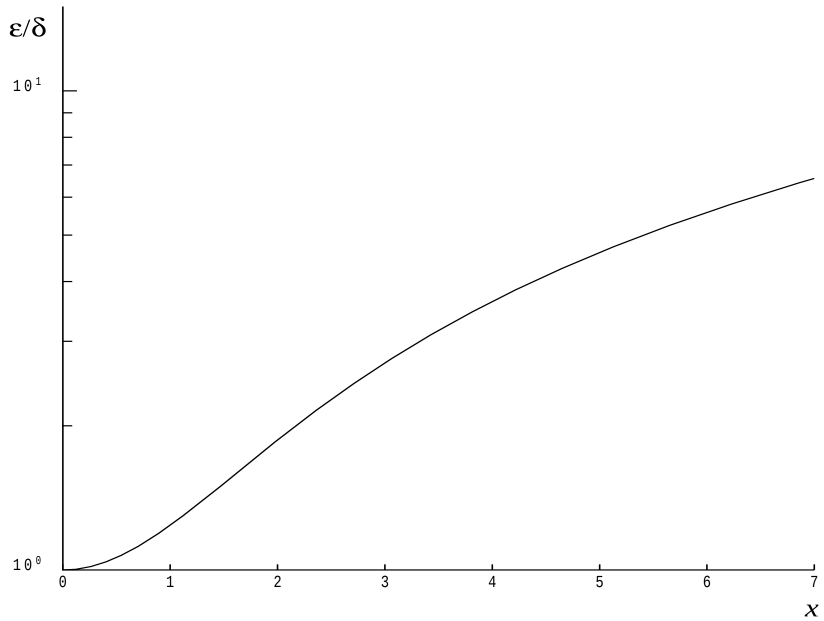

Figure 6: The error amplification factor $|\theta|$.

However, if δ is of the same order as `EPSILON(1.0_`*wp*`)`, then rounding errors could make ε slightly larger than the above relation predicts.
For small x, $\varepsilon \simeq \delta$ and there is no amplification of errors.
For large x, $\varepsilon \simeq x\delta$ and we have strong amplification of errors. However, the procedure must fail for quite moderate values of x because $I_1(x)$ would overflow; hence in practice the loss of accuracy for large x is not excessive. Note that for large x, the errors will be dominated by those of the Fortran intrinsic function `EXP`.

Procedure: nag_bessel_i

1 Description

nag_bessel_i evaluates an approximation to either the modified Bessel function of the first kind $I_\nu(z)$, or the sequence of modified Bessel functions of the first kind $I_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$. The real non-negative order is given by ν (or $\nu + n$) and the complex argument z is such that $-\pi < \arg z \leq \pi$. There is also an option for scaling the result.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_i(z, nu  [, optional arguments])
```

The function result is a scalar of type complex(kind=*wp*), or

```
[value =] nag_bessel_i(z, nu, n  [, optional arguments])
```

The function returns an array-valued result of type complex(kind=*wp*) and dimension N.

3 Arguments

3.1 Mandatory Arguments

z — complex(kind=*wp*), intent(in)

Input: the argument z of the function.

nu — real(kind=*wp*), intent(in)

Input: the order, ν, of the first member of the sequence of functions.

Constraints: nu ≥ 0.0.

n — integer, intent(in)

Input: the number of terms, N, in the sequence of functions $I_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$.

Constraints: n ≥ 1.

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.

scale — logical, intent(in), optional

Input: determines whether or not the results are scaled.

If scale = .true., then the results are scaled by the factor $e^{-|\mathrm{Re}(\mathbf{z})|}$;

if scale = .false., then the results are returned unscaled.

This option can be used to prevent underflow or overflow from occurring, thus increasing the range of the valid arguments.

Default: scale = .false..

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow. Re(z) is too large. No computation has been performed due to the likelihood of overflow.
202	Total loss of accuracy. |z| or nu + n − 1 is too large, so that errors due to argument reduction in elementary functions mean that all precision would be lost.
203	Partial loss of accuracy. |z| or nu + n − 1 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	Termination condition has not been met. This error may occur because the arguments supplied would have caused overflow or underflow. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.
205	Possibility of underflow. All or some of the returned results have been set to zero because of underflow.

5 Examples of Usage

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

6 Further Comments

If the function required is $I_0(z)$ or $I_1(z)$, i.e., $\nu = 0.0$ or $\nu = 1.0$, where z is real and positive, and only a single unscaled function is required, then it is much cheaper to use the procedure `nag_bessel_i0` or `nag_bessel_i1` respectively.

6.1 Algorithmic Detail

The procedure is derived from the routine CBESI in Amos [2].

When $N > 1$ extra values of $I_\nu(z)$ are computed using recurrence relations.

Although the procedure may not be called with ν less than zero, for negative orders the formulae

$$I_{-\nu}(z) = I_\nu(z) + \frac{2}{\pi}\sin(\pi\nu)K_\nu(z)$$

may be used (for the Bessel function $K_\nu(z)$ see the procedure `nag_bessel_k`).

For very large $|z|$ or $(\nu + N - 1)$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$ or $(\nu + N - 1)$, the computation is performed but the results are accurate to less than half of machine precision. If Re(z) is too large and the unscaled function is required, there is the risk of overflow and so no computation is performed.

6.2 Accuracy

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 $s \approx \max(1, |\log_{10}|z||, |\log_{10}\nu|)$ represents the number of digits lost due to the argument reduction. Thus the larger the values of $|z|$ and ν, the less the precision in the result. If this procedure is called with $N > 1$, then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to this procedure with different base values of ν and different N, the computed values may not agree exactly. Empirical tests with modest values of ν and z have shown that the discrepancy is limited to the least significant 3–4 digits of precision.

Procedure: nag_bessel_k0

1 Description

nag_bessel_k0 evaluates an approximation to the modified Bessel function of the second kind $K_0(x)$ or to the exponentially scaled value $e^x K_0(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_k0(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $K_0(x)$ or $e^x K_0(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)

Input: the argument x of the function.

Constraints: x > 0.0.

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.

scale — logical, intent(in), optional

Input: determines whether or not the result is scaled.

If scale = .true., then the result is scaled by the factor e^x;

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

This option can be used to prevent underflow from occurring, thus increasing the range of the valid arguments.

Default: scale = .false..

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

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

- For $x \leq 0$, the result $K_0(x)$ is undefined and the procedure will fail for such arguments.
- For $0 < x \leq 1$, the procedure uses a Chebyshev expansion of the form

$$K_0(x) = -\ln x \sum_{r=0}{}' a_r T_r(t) + \sum_{r=0}{}' b_r T_r(t), \quad \text{where} \quad t = 2x^2 - 1.$$

- For $1 < x \leq 2$, it uses

$$K_0(x) = e^{-x} \sum_{r=0}{}' c_r T_r(t), \quad \text{where} \quad t = 2x - 3.$$

- For $2 < x \leq 4$,

$$K_0(x) = e^{-x} \sum_{r=0}{}' d_r T_r(t), \quad \text{where} \quad t = x - 3.$$

- For $x > 4$,

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{r=0}{}' e_r T_r(t), \quad \text{where} \quad t = \frac{9-x}{1+x}.$$

- For x near zero, $K_0(x) \simeq -\gamma - \ln(x/2)$, where γ denotes Euler's constant. This approximation is used when x is sufficiently small for the result to be correct to EPSILON(1.0_*wp*).
- For large x, where there is a danger of underflow due to the smallness of K_0, the result is set exactly to zero.

6.2 Accuracy

Let δ and ε be the relative errors in the argument and result respectively.

If δ is somewhat larger than EPSILON(1.0_*wp*) (i.e., if δ is due to data errors etc.), then ε and δ are approximately related by

$$\varepsilon \simeq |\theta|\delta, \quad \text{where} \quad \theta = \frac{xK_1(x)}{K_0(x)}.$$

The behaviour of the error amplification factor $|\theta|$ is shown in Figure 7.

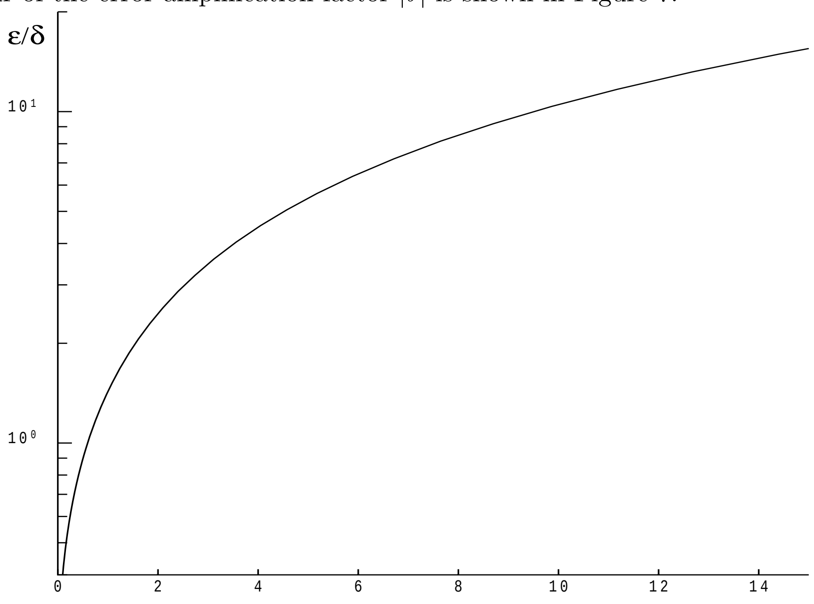

Figure 7: The error amplification factor $|\theta|$.

However, if δ is of the same order as EPSILON(1.0_wp), then rounding errors could make ε slightly larger than the above relation predicts.

For small x, the amplification factor is approximately $1/|\ln x|$, which implies strong attenuation of the error, but in general ε can never be less than EPSILON(1.0_wp).

For large x, $\varepsilon \simeq x\delta$ and we have strong amplification of the relative error. Eventually K_0, which is asymptotically given by $e^{-x}/\sqrt{x}$, becomes so small that it cannot be calculated without underflow and hence the procedure will return zero. Note that for large x the errors will be dominated by those of the Fortran intrinsic function EXP.

Procedure: nag_bessel_k1

1 Description

nag_bessel_k1 evaluates an approximation to the modified Bessel function of the second kind $K_1(x)$ or to the exponentially scaled value $e^x K_1(x)$.

2 Usage

```
USE nag_bessel_fun
[value =] nag_bessel_k1(x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing $K_1(x)$ or $e^x K_1(x)$.

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)

Input: the argument x of the function.

Constraints: x > 0.0.

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.

scale — logical, intent(in), optional

Input: determines whether or not the result is scaled.

If scale = .true., then the result is scaled by the factor e^x;

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

This option can be used to prevent underflow or overflow from occurring, thus increasing the range of the valid arguments.

Default: scale = .false..

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow.
	Argument `x` is too close to zero. This procedure returns approximately the largest representable value.

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

- For $x \leq 0$, the result $K_1(x)$ is undefined and the procedure will fail for such arguments.
- For $0 < x \leq 1$, the procedure uses a Chebyshev expansion of the form
$$K_1(x) = \frac{1}{x} + x \ln x \sum_{r=0}{}' a_r T_r(t) - x \sum_{r=0}{}' b_r T_r(t),$$
where $t = 2x^2 - 1$.
- For $1 < x \leq 2$, it uses
$$K_1(x) = e^{-x} \sum_{r=0}{}' c_r T_r(t),$$
where $t = 2x - 3$.
- For $2 < x \leq 4$,
$$K_1(x) = e^{-x} \sum_{r=0}{}' d_r T_r(t),$$
where $t = x - 3$.
- For $x > 4$,
$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{r=0}{}' e_r T_r(t),$$
where $t = \frac{9-x}{1+x}$.
- For x near zero, $K_1(x) \simeq 1/x$. This approximation is used when x is sufficiently small for the result to be correct to `EPSILON(1.0_wp)`. For very small x, on some machines it is impossible to calculate $1/x$ without overflow and the procedure must fail.
- For large x, where there is a danger of underflow due to the smallness of K_1, the result is set exactly to zero.

6.2 Accuracy

Let δ and ε be the relative errors in the argument and result respectively.

If δ is somewhat larger than `EPSILON(1.0_wp)` (i.e., if δ is due to data errors etc.), then ε and δ are approximately related by

$$\varepsilon \simeq |\theta|\delta, \quad \text{where} \quad \theta = \frac{xK_0(x) - K_1(x)}{K_1(x)}.$$

The behaviour of the error amplification factor $|\theta|$ is shown in Figure 8.

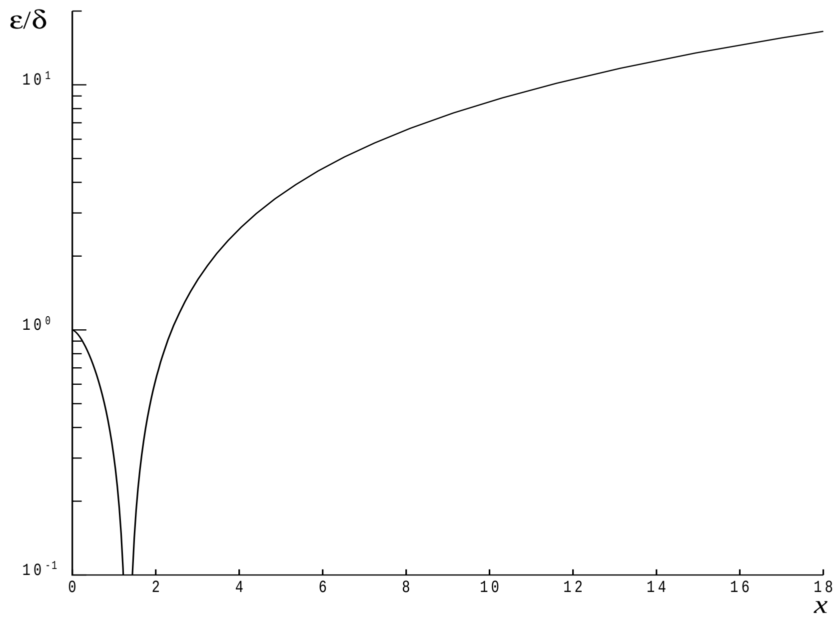

Figure 8: The error amplification factor $|\theta|$.

However, if δ is of the same order as EPSILON(1.0_wp), then rounding errors could make ε slightly larger than the above relation predicts.

For small x, $\varepsilon \simeq \delta$ and there is no amplification of errors.

For large x, $\varepsilon \simeq x\delta$ and we have strong amplification of the relative error. Eventually K_1, which is asymptotically given by $e^{-x}/\sqrt{x}$, becomes so small that it cannot be calculated without underflow and hence the procedure will return zero. Note that for large x the errors will be dominated by those of the Fortran intrinsic function EXP.

Procedure: nag_bessel_k

1 Description

nag_bessel_k evaluates an approximation to either the modified Bessel function of the second kind $K_\nu(z)$, or the sequence of modified Bessel functions of the second kind $K_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$. The real non-negative order is given by ν (or $\nu + n$) and the complex argument z is such that $-\pi <$ arg $z \leq \pi$. There is also an option for scaling the result.

2 Usage

```
USE nag_bessel_fun
```

[*value* =] nag_bessel_k(z, nu [, *optional arguments*])

The function result is a scalar of type complex(kind=*wp*), or

[*value* =] nag_bessel_k(z, nu, n [, *optional arguments*])

The function returns an array-valued result of type complex(kind=*wp*) and dimension N.

3 Arguments

3.1 Mandatory Arguments

z — complex(kind=*wp*), intent(in)

Input: the argument z of the function.

Constraints: z $\neq$ (0.0, 0.0).

nu — real(kind=*wp*), intent(in)

Input: the order, ν, of the first member of the sequence of functions.

Constraints: nu $\geq$ 0.0.

n — integer, intent(in)

Input: the number of terms, N, in the sequence of functions $K_{\nu+n}(z)$, $n = 0, 1, \ldots, N-1$.

Constraints: n $\geq$ 1.

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.

scale — logical, intent(in), optional

Input: determines whether or not the results are scaled.

If scale = .true., then the results are scaled by the factor e^z;

if scale = .false., then the results are returned unscaled.

This option can be used to prevent underflow or overflow from occurring, thus increasing the range of the valid arguments.

Default: scale = .false..

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

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow. |z| is too small. No computation has been performed due to the likelihood of overflow.
202	Total loss of accuracy. |z| or nu + n − 1 is too large, so that errors due to argument reduction in elementary functions mean that all precision would be lost.
203	Partial loss of accuracy. |z| or nu + n − 1 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	Termination condition has not been met. This error may occur because the arguments supplied would have caused overflow or underflow. This problem may be avoided by supplying the optional argument `scale` set to `.true.`.
205	Possibility of underflow. All or some of the returned results have been set to zero because of underflow.
206	Possibility of overflow. nu + n − 1 is too large for the given z. No computation has been performed due to the likelihood of overflow.

5 Examples of Usage

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

6 Further Comments

If the function required is $K_0(z)$ or $K_1(z)$, i.e., $\nu = 0.0$ or $\nu = 1.0$, where z is real and positive, and only a single unscaled function is required, then it is much cheaper to use the procedure `nag_bessel_k0` or `nag_bessel_k1` respectively.

6.1 Algorithmic Detail

The procedure is derived from the routine CBESK in Amos [2].

When $N > 1$ extra values of $J_\nu(z)$ are computed using recurrence relations.

Although the procedure may not be called with ν less than zero, for negative orders the formulae

$$K_{-\nu}(z) = K_\nu(z)$$

may be used.

For very large $|z|$ or $(\nu + N - 1)$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$ or $(\nu + N - 1)$, the computation is performed but the results are accurate to less than half of machine precision. If $|z|$ is very small, near the machine underflow threshold, or $(\nu + N - 1)$ is too large, there is the risk of overflow and so no computation is performed.

6.2 Accuracy

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 $s \approx \max(1, |\log_{10}|z||, |\log_{10}\nu|)$ represents the number of digits lost due to the argument reduction. Thus the larger the values of $|z|$ and ν, the less the precision in the result. If this procedure is called with $N > 1$, then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to this procedure with different base values of ν and different N, the computed values may not agree exactly. Empirical tests with modest values of ν and z have shown that the discrepancy is limited to the least significant 3–4 digits of precision.

Example 1: Evaluation of Real Bessel Functions

This example program evaluates the functions nag_bessel_y0, nag_bessel_y1, nag_bessel_j0, nag_bessel_j1, nag_bessel_k0, nag_bessel_k1, nag_bessel_i0 and nag_bessel_i1 at a set of values of the argument x.

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_bessel_fun_ex01

  ! Example Program Text for nag_bessel_fun
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_bessel_fun, ONLY : nag_bessel_y0, nag_bessel_y1, nag_bessel_i0, &
   nag_bessel_i1, nag_bessel_j0, nag_bessel_j1, nag_bessel_k0, &
   nag_bessel_k1
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 8
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  REAL (wp) :: i0, i1, j0, j1, k0, k1, y0, y1
  ! .. Local Arrays ..
  REAL (wp) :: x(n)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_bessel_fun_ex01'

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '       x         Y0(x)       Y1(x)'

  x = (/ 0.5_wp, 1.0_wp, 3.0_wp, 6.0_wp, 8.0_wp, 10.0_wp, 100.0_wp, &
   1000.0_wp/)
  DO i = 1, n

    y0 = nag_bessel_y0(x(i))
    y1 = nag_bessel_y1(x(i))

    WRITE (nag_std_out,'(1X,1P,3E12.3)') x(i), y0, y1
  END DO

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '       x         J0(x)       J1(x)'
  x = (/ -1.0_wp, 0.0_wp, 0.5_wp, 1.0_wp, 3.0_wp, 6.0_wp, 10.0_wp, &
   1000.0_wp/)

  DO i = 1, n

    j0 = nag_bessel_j0(x(i))
    j1 = nag_bessel_j1(x(i))

    WRITE (nag_std_out,'(1X,1P,3E12.3)') x(i), j0, j1
  END DO
```

```
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '       x         K0(x)       K1(x)'
    x = (/ 0.4_wp, 0.6_wp, 1.6_wp, 2.5_wp, 3.5_wp, 8.0_wp, 10.0_wp, &
     1000.0_wp/)

    DO i = 1, n

      k0 = nag_bessel_k0(x(i))
      k1 = nag_bessel_k1(x(i))

      WRITE (nag_std_out,'(1X,1P,3E12.3)') x(i), k0, k1
    END DO

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '                scaled      scaled'
    WRITE (nag_std_out,*) '       x         K0(x)       K1(x)'
    x = (/ 0.4_wp, 0.6_wp, 1.6_wp, 2.5_wp, 3.5_wp, 8.0_wp, 10.0_wp, &
     1000.0_wp/)

    DO i = 1, n

      k0 = nag_bessel_k0(x(i),scale=.TRUE.)
      k1 = nag_bessel_k1(x(i),scale=.TRUE.)

      WRITE (nag_std_out,'(1X,1P,3E12.3)') x(i), k0, k1
    END DO

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '       x         I0(x)       I1(x)'
    x = (/ -1.0_wp, 0.0_wp, 0.5_wp, 1.0_wp, 6.0_wp, 10.0_wp, 15.0_wp, &
     20.0_wp/)

    DO i = 1, n

      i0 = nag_bessel_i0(x(i))
      i1 = nag_bessel_i1(x(i))

      WRITE (nag_std_out,'(1X,1P,3E12.3)') x(i), i0, i1
    END DO

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '                scaled      scaled'
    WRITE (nag_std_out,*) '       x         I0(x)       I1(x)'
    x = (/ -1.0_wp, 0.0_wp, 0.5_wp, 1.0_wp, 6.0_wp, 10.0_wp, 20.0_wp, &
     1000.0_wp/)

    DO i = 1, n

      i0 = nag_bessel_i0(x(i),scale=.TRUE.)
      i1 = nag_bessel_i1(x(i),scale=.TRUE.)

      WRITE (nag_std_out,'(1X,1P,3E12.3)') x(i), i0, i1
    END DO

  END PROGRAM nag_bessel_fun_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_bessel_fun_ex01

        x          Y0(x)        Y1(x)
    5.000E-01  -4.445E-01  -1.471E+00
    1.000E+00   8.826E-02  -7.812E-01
    3.000E+00   3.769E-01   3.247E-01
    6.000E+00  -2.882E-01  -1.750E-01
    8.000E+00   2.235E-01  -1.581E-01
    1.000E+01   5.567E-02   2.490E-01
    1.000E+02  -7.724E-02  -2.037E-02
    1.000E+03   4.716E-03  -2.478E-02

        x          J0(x)        J1(x)
   -1.000E+00   7.652E-01  -4.401E-01
    0.000E+00   1.000E+00   0.000E+00
    5.000E-01   9.385E-01   2.423E-01
    1.000E+00   7.652E-01   4.401E-01
    3.000E+00  -2.601E-01   3.391E-01
    6.000E+00   1.506E-01  -2.767E-01
    1.000E+01  -2.459E-01   4.347E-02
    1.000E+03   2.479E-02   4.728E-03

        x          K0(x)        K1(x)
    4.000E-01   1.115E+00   2.184E+00
    6.000E-01   7.775E-01   1.303E+00
    1.600E+00   1.880E-01   2.406E-01
    2.500E+00   6.235E-02   7.389E-02
    3.500E+00   1.960E-02   2.224E-02
    8.000E+00   1.465E-04   1.554E-04
    1.000E+01   1.778E-05   1.865E-05
    1.000E+03   0.000E+00   0.000E+00

                  scaled       scaled
        x          K0(x)        K1(x)
    4.000E-01   1.663E+00   3.259E+00
    6.000E-01   1.417E+00   2.374E+00
    1.600E+00   9.309E-01   1.192E+00
    2.500E+00   7.595E-01   9.002E-01
    3.500E+00   6.490E-01   7.365E-01
    8.000E+00   4.366E-01   4.631E-01
    1.000E+01   3.916E-01   4.108E-01
    1.000E+03   3.963E-02   3.965E-02

        x          I0(x)        I1(x)
   -1.000E+00   1.266E+00  -5.652E-01
    0.000E+00   1.000E+00   0.000E+00
    5.000E-01   1.063E+00   2.579E-01
    1.000E+00   1.266E+00   5.652E-01
    6.000E+00   6.723E+01   6.134E+01
    1.000E+01   2.816E+03   2.671E+03
    1.500E+01   3.396E+05   3.281E+05
    2.000E+01   4.356E+07   4.245E+07

                  scaled       scaled
        x          I0(x)        I1(x)
   -1.000E+00   4.658E-01  -2.079E-01
    0.000E+00   1.000E+00   0.000E+00
    5.000E-01   6.450E-01   1.564E-01
    1.000E+00   4.658E-01   2.079E-01
    6.000E+00   1.667E-01   1.521E-01
    1.000E+01   1.278E-01   1.213E-01
```

```
2.000E+01   8.978E-02   8.751E-02
1.000E+03   1.262E-02   1.261E-02
```

Example 2: Evaluation of Complex Bessel Functions

This example program evaluates the functions nag_bessel_i, nag_bessel_j, nag_bessel_k and nag_bessel_y given values of the arguments z, n and scale.

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_bessel_fun_ex02

  ! Example Program Text for nag_bessel_fun
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_bessel_fun, ONLY : nag_bessel_i, nag_bessel_j, nag_bessel_k, &
   nag_bessel_y
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC CMPLX, KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 4
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  CHARACTER (*), PARAMETER :: fmt1 = &
   '(I2, 4(2X,''('',F7.3,'','',F7.3,'')''))'
  CHARACTER (*), PARAMETER :: fmt2 = &
   '(1X,A,1X,''('',F7.3,'','',F7.3,'')'',A,F7.3,A)'
  ! .. Local Scalars ..
  INTEGER :: i
  REAL (wp) :: nu
  COMPLEX (wp) :: z
  ! .. Local Arrays ..
  COMPLEX (wp) :: bess_i(n), bess_j(n), bess_k(n), bess_y(n)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_bessel_fun_ex02'

  WRITE (nag_std_out,*)
  nu = 5.1_wp
  z = CMPLX(3.0_wp,2.0_wp,kind=wp)
  WRITE (nag_std_out,fmt2) 'Results for z = ', z, ',  nu = ', nu, &
   ',  scale = .FALSE.'
  WRITE (nag_std_out,*) 'n          I(z)                J(z) &
   &              K(z)                Y(z)'

  bess_i = nag_bessel_i(z,nu,n)
  bess_j = nag_bessel_j(z,nu,n)
  bess_k = nag_bessel_k(z,nu,n)
  bess_y = nag_bessel_y(z,nu,n)

  DO i = 1, n
    WRITE (nag_std_out,fmt1) i, bess_i(i), bess_j(i), bess_k(i), bess_y(i)
  END DO

  nu = 2.1_wp
  z = CMPLX(1.0_wp,-2.0_wp,kind=wp)
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,fmt2) 'Results for z = ', z, ',  nu = ', nu, &
   ',  scale = .TRUE.'
  WRITE (nag_std_out,*) 'n          I(z)                J(z) &
```

```
     &              K(z)               Y(z)'

    bess_i = nag_bessel_i(z,nu,n,scale=.TRUE.)
    bess_j = nag_bessel_j(z,nu,n,scale=.TRUE.)
    bess_k = nag_bessel_k(z,nu,n,scale=.TRUE.)
    bess_y = nag_bessel_y(z,nu,n,scale=.TRUE.)

    DO i = 1, n
      WRITE (nag_std_out,fmt1) i, bess_i(i), bess_j(i), bess_k(i), bess_y(i)
    END DO

  END PROGRAM nag_bessel_fun_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_bessel_fun_ex02

 Results for z =  (  3.000,  2.000),  nu =   5.100,  scale = .FALSE.
 n         I(z)              J(z)              K(z)              Y(z)
 1  ( -0.166, -0.058)  ( -0.094,  0.071)  ( -0.426,  0.243)  (  0.264,  0.337)
 2  ( -0.033, -0.038)  ( -0.035, -0.000)  ( -0.810,  1.255)  (  1.502,  0.220)
 3  ( -0.002, -0.012)  ( -0.007, -0.006)  ( -0.351,  5.298)  (  4.378, -2.539)
 4  (  0.001, -0.003)  ( -0.001, -0.002)  (  9.612, 19.384)  (  7.298,-18.103)

 Results for z =  (  1.000, -2.000),  nu =   2.100,  scale = .TRUE.
 n         I(z)              J(z)              K(z)              Y(z)
 1  ( -0.149, -0.076)  ( -0.043, -0.090)  (  0.486,  1.295)  ( -0.085,  0.024)
 2  ( -0.055,  0.033)  ( -0.034,  0.002)  ( -1.057,  2.515)  (  0.049,  0.029)
 3  (  0.001,  0.018)  ( -0.003,  0.008)  ( -7.061,  1.792)  (  0.074,  0.133)
 4  (  0.004,  0.002)  (  0.001,  0.001)  (-18.515,-17.705)  ( -0.365,  0.430)
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_bessel_fun_ex03`

Evaluation of the real Bessel functions $K_0(x)$ and $K_1(x)$ of the second kind.

`nag_bessel_fun_ex04`

Evaluation of the real Bessel functions $I_0(x)$ and $I_1(x)$ of the first kind.

`nag_bessel_fun_ex05`

Evaluation of the real Bessel functions $Y_0(x)$ and $Y_1(x)$ of the second kind.

`nag_bessel_fun_ex06`

Evaluation of the real Bessel functions $J_0(x)$ and $J_1(x)$ of the first kind.

`nag_bessel_fun_ex07`

Evaluation of the complex Bessel function $Y_\nu(z)$.

`nag_bessel_fun_ex08`

Evaluation of the complex Bessel function $J_\nu(z)$.

`nag_bessel_fun_ex09`

Evaluation of the complex Bessel function $I_\nu(z)$.

`nag_bessel_fun_ex10`

Evaluation of the complex Bessel function $K_\nu(z)$.

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

[3] Clenshaw C W (1962) Mathematical tables *Chebyshev Series for Mathematical Functions* HMSO