Module 3.2: nag_gamma_fun
Gamma Functions

nag_gamma_fun contains procedures for approximating the gamma function and closely related real-valued functions.

Contents

Introduction 3.2.3

Procedures

nag_gamma 3.2.5
Gamma function

nag_log_gamma 3.2.9
Log gamma function

nag_polygamma 3.2.13
Polygamma functions

nag_incompl_gamma 3.2.17
Incomplete gamma functions

Examples

Example 1: A Simple Use of nag_gamma, nag_log_gamma and nag_incompl_gamma 3.2.19

Example 2: A Simple Use of nag_polygamma 3.2.21

Additional Examples 3.2.23

References 3.2.24

Introduction

This module contains procedures for approximating the gamma function and closely related real-valued functions.

- `nag_gamma` calculates the gamma (or generalized factorial) function

  $$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,dt, \quad x > 0,$$

  extended to non-integral negative x using the identity

  $$\Gamma(x) = \frac{\Gamma(x+1)}{x}.$$

- `nag_log_gamma` calculates the logarithm of $\Gamma(x)$.
- `nag_polygamma` returns a sequence of values of scaled derivatives of the psi function.
- `nag_incompl_gamma` computes values for the incomplete gamma functions

  $$P(a,x) = \frac{1}{\Gamma(a)}\int_0^x t^{a-1}e^{-t}\,dt,$$

  $$Q(a,x) = \frac{1}{\Gamma(a)}\int_x^\infty t^{a-1}e^{-t}\,dt.$$

Further details of these functions may be found in Abramowitz and Stegun [1], Chapter 6.

In general the approximations are based on expansions in terms of Chebyshev polynomials $T_r(t) = \cos(r \arccos t)$. Further details appear in Section 6.1 of the individual procedure documents.

Procedure: nag_gamma

1 Description

nag_gamma returns an approximation to the gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt,\ x > 0,$$

extended to non-integral negative x using the identity

$$\Gamma(x) = \frac{\Gamma(x+1)}{x}.$$

2 Usage

```
USE nag_gamma_fun
[value =] nag_gamma(x  [, optional arguments])
```

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

3 Arguments

3.1 Mandatory Argument

x — real(kind=*wp*), intent(in)

Input: the argument x of the function.

Constraints: x must not be a negative integer.

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 large. An approximate value of $\Gamma(x)$ is returned by this procedure at the nearest valid argument.

202 Possibility of overflow.

Argument `x` is too close to zero. An approximate value of $\Gamma(x)$ is returned by this procedure at the nearest valid argument.

203 Possibility of underflow.

Argument `x` is too large and negative. There is a danger of underflow. Zero is returned.

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

The procedure is based on the Chebyshev expansion:

$$\Gamma(1+u) = \sum_{r=0}^{'} a_r T_r(t) \quad \text{where} \quad 0 \leq u < 1\ ,\ t = 2u-1,$$

and uses the property $\Gamma(1+x) = x\Gamma(x)$. If $x = N+1+u$ where N is integral and $0 \leq u < 1$ then it follows that:

$$\Gamma(x) = \begin{cases} \dfrac{\Gamma(1+u)}{x(x+1)(x+2)...(x-N-1)}, & \text{for } N < 0, \\ \Gamma(1+u), & \text{for } N = 0, \\ (x-1)(x-2)...(x-N)\,\Gamma(1+u), & \text{for } N > 0. \end{cases}$$

There are three possible failures for this procedure:

(i) if x is too large, there is a danger of overflow since $\Gamma(x)$ could become too large to be represented in the machine;

(ii) if x is equal to a negative integer, $\Gamma(x)$ would overflow since it has poles at such points;

(iii) if x is too near zero, there is again the danger of overflow on some machines. For small x, $\Gamma(x) \simeq 1/x$, and on some machines there exists a range of non-zero but small values of x for which $1/x$ is larger than the greatest representable value.

If x is too large and negative, such that there is an unavoidable danger of setting underflow, this procedure returns zero.

6.2 Accuracy

Let δ and ε be the relative errors in the argument and the 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:

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

(provided ε is also greater than the representation error). Here $\psi(x)$ is the digamma function $\dfrac{\Gamma'(x)}{\Gamma(x)}$. The behaviour of the error amplification factor $|\theta|$ is shown in Figure 1.

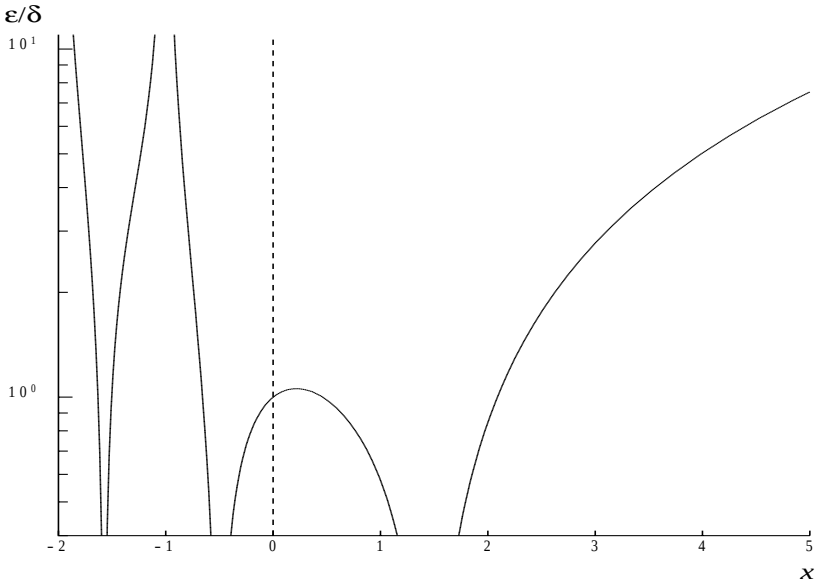

Figure 1: The error amplification factor $|\theta|$.

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.

There is clearly a severe, but unavoidable, loss of accuracy for arguments close to the poles of $\Gamma(x)$ at negative integers. However, relative accuracy is preserved near the pole at $x = 0$ right up to the point of failure arising from the danger of overflow. Also accuracy will necessarily be lost as x becomes large since in this region $\varepsilon \simeq \delta x \ln x$.

However, since $\Gamma(x)$ increases rapidly with x, the procedure must fail due to the danger of setting overflow before this loss of accuracy is too great. (For example, for $x = 20$, the amplification factor $\simeq 60$.)

Procedure: nag_log_gamma

1 Description

nag_log_gamma returns an approximation to the logarithm of the gamma function, $\ln\Gamma(x)$.

2 Usage

```
USE nag_gamma_fun
[value =] nag_log_gamma(x  [, optional arguments])
```

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

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 large and the function would overflow. This procedure returns the value at the largest permissible argument.

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

The procedure is based on Chebyshev expansions, and the result is computed for x in various intervals as follows.

- For $0 < x \leq x_{\text{small}}$, $\ln\Gamma(x) = -\ln x$ to within machine accuracy.
- For $x_{\text{small}} < x \leq 15$, the recursive relation $\Gamma(1+x) = x\Gamma(x)$ is used to reduce the calculation to one involving $\Gamma(1+u)$, $0 \leq u < 1$ which is evaluated as:

  $$\Gamma(1+u) = \sum_{r=0}^{\prime} a_r T_r(t), \quad t = 2u - 1.$$

  Once $\Gamma(x)$ has been calculated, the required result is produced by taking the logarithm.
- For $15 < x \leq x_{\text{big}}$,

  $$\ln\Gamma(x) = \left(x - \frac{1}{2}\right)\ln x - x + \frac{1}{2}\ln 2\pi + \frac{y(x)}{x}$$

  where $y(x) = \sum_{r=0}^{\prime} b_r T_r(t)$, $t = 2\left(\frac{15}{x}\right)^2 - 1$.
- For $x_{\text{big}} < x \leq x_{\text{vbig}}$, the term $y(x)/x$ is negligible and so its calculation is omitted.
- For $x > x_{\text{vbig}}$, there is a danger of setting overflow so the procedure must fail.
- For $x \leq 0$, the function is not defined and the procedure fails.

The parameters x_{small}, x_{big} and x_{vbig} are calculated such that:

if $x < x_{\text{small}}$, $\Gamma(x) = 1/x$ to within machine accuracy;

if $x > x_{\text{big}}$,

$$\ln\Gamma(x) = \left(x - \frac{1}{2}\right)\ln x - x + \frac{1}{2}\ln 2\pi$$

to within machine accuracy;

$\ln\Gamma(x_{\text{vbig}})$ is close to the value returned by `HUGE(1.0_wp)`.

6.2 Accuracy

Let δ and ε be the relative errors in the argument and result respectively, and E be the absolute error in the result.

If δ is somewhat larger than `EPSILON(1.0_wp)`, then

$$E \simeq |\theta_a|\delta \quad \text{with} \quad \theta_a = x\psi(x), \quad \text{and} \quad \varepsilon \simeq |\theta_r|\delta \quad \text{with} \quad \theta_r = \frac{x\psi(x)}{\ln\Gamma(x)},$$

where $\psi(x)$ is the digamma function $\dfrac{\Gamma'(x)}{\Gamma(x)}$. The behaviour of these error amplification factors is shown in Figures 2 and 3.

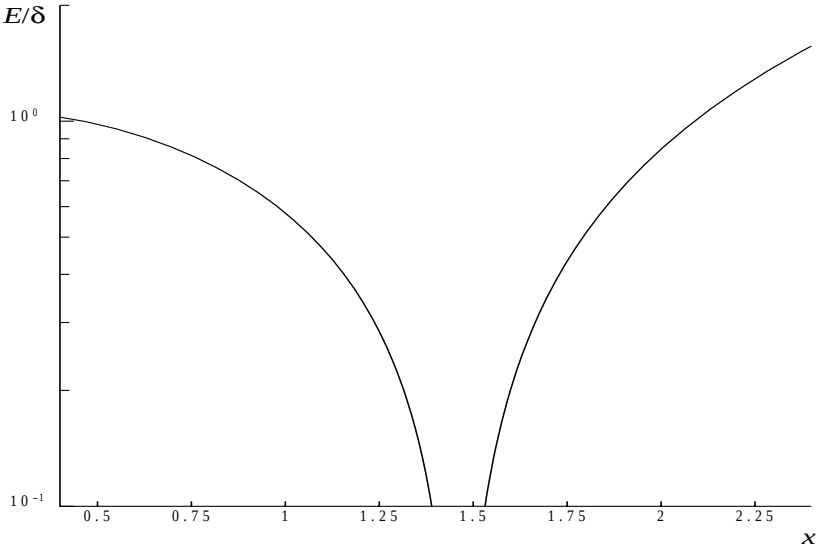

Figure 2: The absolute error amplification factor $|\theta_a|$.

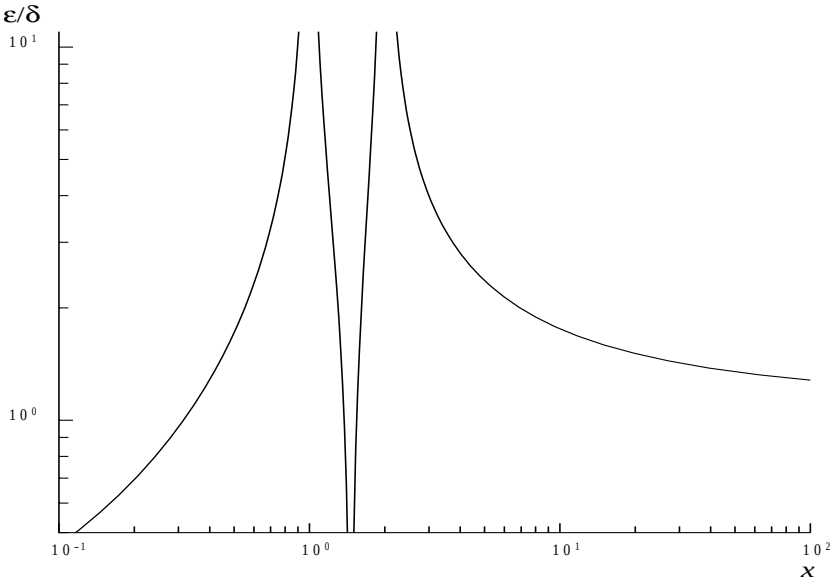

Figure 3: The relative error amplification factor $|\theta_r|$.

These show that relative error can be controlled, since except near $x = 1$ or 2 relative error is attenuated by the function or at least is not greatly amplified.

For large x, $\varepsilon \simeq \left(1 + \dfrac{1}{\ln x}\right)\delta$ and for small x, $\varepsilon \simeq \dfrac{1}{\ln x}\delta$.

The function $\ln\Gamma(x)$ has zeros at $x = 1$ and 2 and hence relative accuracy is not maintainable near those points. However, absolute accuracy can still be provided near those zeros as is shown above.

If, however, δ is of the order of `EPSILON(1.0_wp)`, then rounding errors in the procedure's internal arithmetic may result in errors which are slightly larger than those predicted by the equalities. It should be noted that even in areas where strong attenuation of errors is predicted the relative precision is bounded by `EPSILON(1.0_wp)`.

Procedure: nag_polygamma

1 Description

nag_polygamma returns approximate values of scaled derivatives

$$w(k,x) = \frac{(-1)^{k+1}\psi^{(k)}(x)}{k!}$$

of the psi function

$$\psi(x) = \frac{d}{dx}\ln\Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)},$$

for $x > 0$ and $k = n,\ n+1,\ldots,n+m-1$, where $\psi^{(k)}$ denotes the kth derivative of ψ.

2 Usage

```
USE nag_gamma_fun
CALL nag_polygamma(x, psi  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$m > 0$ — the number of members, m = SIZE(psi), required in the sequence $w(k,x)$, for $k = n, n+1,\ldots,n+m-1$

3.1 Mandatory Arguments

x — real(kind=*wp*), intent(in)

Input: the argument x of the function.

Constraints: x > 0.0.

psi / psi(*m*) — real(kind=*wp*), intent(out)

Output: a value of the scaled derivative or a sequence of values of scaled derivatives of the psi function $\psi(x)$.

If psi is a scalar variable, then psi contains the required value $w(k,x)$, for $k = n$;

if psi is a rank 1 array of size m, then its elements contain the required values $w(k,x)$, for $k = n, n+1,\ldots,n+m-1$.

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.

n — integer, intent(in), optional

Input: the first member n of the sequence of functions.

Constraints: n ≥ 0.

Default: n = 0.

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.
399	An unexpected Library error.
	`n + SIZE(psi) − 1` may be too large. If possible, a reduction in the value of `SIZE(psi)` should be considered.

Failures (error%level = 2):

error%code	Description
201	Possibility of overflow.
	Either `x` is too small, or `n + SIZE(psi) − 1` is too large. If possible, a reduction in the value of `SIZE(psi)` should be considered.
202	Possibility of underflow.
	Either `x` or `n + SIZE(psi) − 1` is too large. If possible, a reduction in the value of `SIZE(psi)` should be considered.

5 Examples of Usage

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

If an error occurs with `error%code` = 399, you could replace the following call:

```
CALL nag_polygamma(x,psi,n=n)
```

by:

```
CALL nag_polygamma(x,psi(1:size(psi)/2),n=n)
CALL nag_polygamma(x,psi(1+size(psi)/2:),n=n+size(psi)/2)
```

6 Further Comments

6.1 Algorithmic Detail

The procedure is derived from the routine PSIFN in Amos [2]. The basic method of evaluation of $w(k,x)$ uses the asymptotic series

$$w(k,x) \sim \varepsilon(k,x) + \frac{1}{2x^{k+1}} + \frac{1}{x^k}\sum_{j=1}^{\infty} B_{2j}\frac{(2j+k-1)!}{(2j)!k!x^{2j}}$$

for large x greater than a machine-dependent value $x_{\min}$, followed by backward recurrence using

$$w(k,x) = w(k,x+1) + x^{-k-1}$$

for smaller values of x, where $\varepsilon(k,x) = -\ln x$ when $k = 0$, $\varepsilon(k,x) = 1/kx^k$ when $k > 0$, and B_{2j}, $j = 1,2,\ldots$, are the Bernoulli numbers.

When k is large, the above procedure may be inefficient, and the expansion

$$w(k,x) = \sum_{j=1}^{\infty} \frac{1}{(x+j)^{k+1}},$$

which converges rapidly for large k, is used instead.

6.2 Accuracy

All constants in 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, then clearly the maximum number of correct digits in the results obtained is limited by $p = \min(t, 18)$. Empirical tests of this procedure, taking values of x in the range $0.0 < x < 50.0$, and n in the range $1 \leq n \leq 50$, have shown that the maximum relative error is a loss of approximately two decimal places of precision. Tests with $n = 0$, i.e., testing the function $-\psi(x)$, have shown somewhat better accuracy, except at points close to the zero of $\psi(x)$, $x \simeq 1.461632$, where only absolute accuracy can be obtained.

6.3 Timing

The time taken for a call of this procedure is approximately proportional to m, plus a constant. In general, it is much cheaper to call this procedure with m greater than 1 to evaluate the function $w(k,x)$, for $k = n, n+1, \ldots, n+m-1$, rather than to make m separate calls of this procedure.

Procedure: nag_incompl_gamma

1 Description

nag_incompl_gamma approximates the values of one of the incomplete gamma functions $P(a,x)$ or $Q(a,x)$ in the normalised form

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} e^{-t} dt,$$

$$Q(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty t^{a-1} e^{-t} dt,$$

with $x \geq 0$ and $a > 0$, to a user-specified accuracy. With this normalisation, $P(a,x) + Q(a,x) = 1$.

2 Usage

```
USE nag_gamma_fun
[value =] nag_incompl_gamma(a, x  [, optional arguments])
```

The function result is a scalar, of type real(kind=*wp*), containing the incomplete gamma function $P(a,x)$ unless the optional argument p_fun is supplied and set to .false., in which case the function result will contain the incomplete gamma function $Q(a,x)$.

3 Arguments

3.1 Mandatory Arguments

a — real(kind=*wp*), intent(in)
Input: the argument a of the functions.
Constraints: a > 0.0.

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

tol — real(kind=*wp*), intent(in), optional
Input: the relative accuracy required.
Constraints: EPSILON(1.0_*wp*) $\leq$ tol < 1.0.
Default: tol = EPSILON(1.0_*wp*).

p_fun — logical, intent(in), optional
Input: specifies which value of the function $P(a,x)$ or $Q(a,x)$ is to be returned:
if p_fun = .true., the value of the function $P(a,x)$ is returned;
if p_fun = .false., the value of the function $Q(a,x)$ is returned.
Default: p_fun = .true..

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	No convergence. Convergence of the Taylor series or Legendre's continued fraction fails within 600 iterations. This error is extremely unlikely to occur; if it does, contact NAG.

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

Several methods are used to evaluate the functions depending on the arguments a and x. The methods include the Taylor expansion for $P(a,x)$, Legendre's continued fraction for $Q(a,x)$, and power series for $Q(a,x)$. When both a and x are large, and $a \simeq x$, the uniform asymptotic expansion of Temme [5] is employed for greater efficiency; specifically, this expansion is used when $a \geq 20$ and $0.7a \leq x \leq 1.4a$.

This procedure is derived from subroutine GAM in Gautschi [4].

6.2 Accuracy

There are rare occasions when the relative accuracy attained is somewhat less than that specified by the optional argument `tol`. However, the error should never exceed more than one or two decimal places. Note also that there is a limit of 18 decimal places on the achievable accuracy, because constants in the procedure are given to this precision.

6.3 Timing

The time taken for a call of this procedure depends on the precision requested through `tol`, and also varies slightly with the input arguments a and x.

Example 1: A Simple Use of nag_gamma, nag_log_gamma and nag_incompl_gamma

This example program evaluates the functions nag_gamma and nag_log_gamma at a set of values of the argument x. It also evaluates the incomplete gamma functions at a set of values of the two arguments a and 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_gamma_fun_ex01

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_gamma_fun, ONLY : nag_gamma, nag_log_gamma, nag_incompl_gamma
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: m = 5, n = 9
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  REAL (wp) :: p, q, y
  ! .. Local Arrays ..
  REAL (wp) :: a(m), x(n)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_gamma_fun_ex01'

  ! evaluates gamma function
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '      x        gamma(x)'
  x = (/ -1.5_wp, 1.0_wp, 1.25_wp, 1.5_wp, 1.75_wp, 2.0_wp, 5.0_wp, &
   8.0_wp, 10.0_wp/)

  DO i = 1, n

    y = nag_gamma(x(i))

    WRITE (nag_std_out,fmt='(1X,1P,2E12.3)') x(i), y
  END DO

  ! evaluates log gamma function
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '      x      log gamma(x)'
  x = (/ 1.0_wp, 1.25_wp, 1.5_wp, 1.75_wp, 2.0_wp, 5.0_wp, 10.0_wp, &
   20.0_wp, 1000.0_wp/)

  DO i = 1, n

    y = nag_log_gamma(x(i))

    WRITE (nag_std_out,fmt='(1X,1P,2E12.3)') x(i), y
  END DO
```

```
    ! evaluates incomplete gamma function
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '         a           x        p(a,x)      q(a,x)'
    a = (/ 2.0_wp, 7.0_wp, 0.5_wp, 20.0_wp, 21.0_wp/)
    x(1:m) = (/ 3.0_wp, 1.0_wp, 99.0_wp, 21.0_wp, 20.0_wp/)

    DO i = 1, m

      p = nag_incompl_gamma(a(i),x(i))
      q = 1.0_wp - p

      WRITE (nag_std_out,fmt='(1X,4F12.4)') a(i), x(i), p, q
    END DO

  END PROGRAM nag_gamma_fun_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_gamma_fun_ex01

      x        gamma(x)
  -1.500E+00   2.363E+00
   1.000E+00   1.000E+00
   1.250E+00   9.064E-01
   1.500E+00   8.862E-01
   1.750E+00   9.191E-01
   2.000E+00   1.000E+00
   5.000E+00   2.400E+01
   8.000E+00   5.040E+03
   1.000E+01   3.629E+05

      x      log gamma(x)
   1.000E+00   0.000E+00
   1.250E+00  -9.827E-02
   1.500E+00  -1.208E-01
   1.750E+00  -8.440E-02
   2.000E+00   0.000E+00
   5.000E+00   3.178E+00
   1.000E+01   1.280E+01
   2.000E+01   3.934E+01
   1.000E+03   5.905E+03

        a           x        p(a,x)      q(a,x)
     2.0000      3.0000      0.8009      0.1991
     7.0000      1.0000      0.0001      0.9999
     0.5000     99.0000      1.0000      0.0000
    20.0000     21.0000      0.6157      0.3843
    21.0000     20.0000      0.4409      0.5591
```

Example 2: A Simple Use of nag_polygamma

This example program evaluates the function nag_polygamma 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_gamma_fun_ex02

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_gamma_fun, ONLY : nag_polygamma
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: m = 5, n = 4
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  ! .. Local Arrays ..
  REAL (wp) :: psi(0:n-1), x(m)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_gamma_fun_ex02'

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) &
   '      x          psi(0,x)      psi(1,x)      psi(2,x)      psi(3,x)'
  x = (/ 0.1_wp, 0.5_wp, 2.5_wp, 3.6_wp, 8.0_wp/)

  DO i = 1, m

    CALL nag_polygamma(x(i),psi)

    WRITE (nag_std_out,fmt='(1X,5(1PE12.4,2X))') x(i), psi
  END DO

END PROGRAM nag_gamma_fun_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_gamma_fun_ex02

       x          psi(0,x)      psi(1,x)      psi(2,x)      psi(3,x)
   1.0000E-01    1.0424E+01    1.0143E+02    1.0009E+03    1.0001E+04
   5.0000E-01    1.9635E+00    4.9348E+00    8.4144E+00    1.6235E+01
   2.5000E+00   -7.0316E-01    4.9036E-01    1.1810E-01    3.7318E-02
   3.6000E+00   -1.1357E+00    3.1988E-01    5.0750E-02    1.0653E-02
   8.0000E+00   -2.0156E+00    1.3314E-01    8.8498E-03    7.8321E-04
```

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_gamma_fun_ex03`

Evaluation of the gamma (or generalized factorial) function.

`nag_gamma_fun_ex04`

Evaluation of the incomplete gamma function.

`nag_gamma_fun_ex05`

Evaluation of the logarithm of the gamma function.

References

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

[2] Amos D E (1983) Algorithm 610: A portable FORTRAN subroutine for derivatives of the psi function *ACM Trans. Math. Software* **9** 494–502

[3] Gautschi W (1979) A computational procedure for incomplete gamma functions *ACM Trans. Math. Software* **5** 466–481

[4] Gautschi W (1979) Algorithm 542: Incomplete gamma functions *ACM Trans. Math. Software* **5** 482–489

[5] Temme N M (1987) On the computation of the incomplete gamma functions for large values of the parameters *Algorithms for Approximation* (ed J C Mason and M G Cox) Oxford University Press