6 Special Functions

Functions:

gammln: logarithm of the gamma function
factrl: factorial function
bico: binomial coefficient function
factln: logarithm of the factorial function
beta: beta function
gammp: incomplete gamma function
gammq: incomplete complementary gamma function
gser: series evaluation of the incomplete gamma function
gcf: continued fraction evaluation of the incomplete gamma function
erf: error function
erfc: complementary error function
erfcc: complementary error function, consise routine
betai: incomplete beta function
betacf: incomplete beta function, continued fraction evaluation
bessj0: bessel function J0
bessy0: bessel function Y0
bessj1: bessel function J1
bessy1: bessel function Y1
bessj: bessel function J of integer order
bessy: bessel function Y of integer order
bessi0: modified bessel function I0
bessk0: modified bessel function K0
bessi1: modified bessel function I1
bessk1: modified bessel function K1
bessi: modified bessel function I of integer order
bessk: modified bessel function K of integer order
plgndr: associated Legendre polynomial
el2: elliptic integrals of the first and second kind
sncndn: Jacobian elliptic function

gammln

Description:

This function returns the logarithm of the gamma function $ln(\Gamma(xx))$ for $xx > 0$. Full accuracy is obtained for $xx > 1$. For $0 < xx < 1$ the reflection formula

$$\Gamma(1-z) = \frac{\pi z}{\Gamma(1+z)sin(\pi z)}$$

can be used first.

gammln(xx) $\longrightarrow$ value

Where:

xx is a real number with $xx > 0$,
value is a real number being an approximation to the logarithm of the gamma function at xx.

Example:

Three values of the logarithm of the gamma function are computed using `gammln` and compared with standard values.

```
-> gammln(1.0)?
-0.0000000
-> ln(1.0)?
0.0000000
-> gammln(1.8)?
-0.0710839
-> ln(0.931384)?
-0.0710836
-> gammln(0.2)?
1.5240638
-> ln(4.5908436)?
1.5240638
```

factrl

Description:

This function computes a floating point form of the factorial function.

factrl(n) $\longrightarrow$ **value**

Where:

n is a positive integer,
value is a floating point number approximating the value of the factorial function at n.

Example:

Three values of the factorial function are computed using `factrl`.

```
-> factrl(1)?
1.0000000
-> factrl(4)?
24.0000000
-> factrl(6)?
720.0000000
```

---

bico

Description:

This function returns the binomial coefficient $\binom{n}{k}$ as a floating point number.

bico(n, k) $\longrightarrow$ **value**

Where:

n is a positive integer,
k is a positive integer with $n \geq k$,
value is a floating point number approximating the binomial coefficient $\binom{n}{k}$.

Example:

Three binomial coefficients are computed using `bico`.

```
-> bico(6,1)?
6.0000000
-> bico(6,3)?
20.0000000
-> bico(15,7)?
6435.0000011
```

---

factln

Description:

This function computes the logarithm of the factorial function $ln(n!)$.

factln(n) $\longrightarrow$ value

Where:

n is a positive integer,
value is a floating point number approximating $ln(n!)$.

Example:

Two values of the logarithm of the factorial function are computed using `factln` and compared with known values.

```
-> factln(1)?
-0.0000000
-> ln(1.0)?
0.0000000
-> factln(4)?
3.1780538
-> ln(24.0)?
3.1780539
```

---

beta

Description:

This function computes the beta function $\beta(z, w)$.

beta(z, w) $\longrightarrow$ value

Where:

z is a real number,
w is a real number,
value is a floating point number being an approximation to $\beta(z, w)$.

Example:

Three values of the beta function are computed.

```
-> beta(1.0,1.0)?
1.0000000
-> beta(0.4,1.0)?
2.5000000
-> beta(0.8,1.0)?
1.2500000
```

---

gammp

Description:

This function computes the incomplete gamma function $P(a, x)$.

gammp(a, x) $\longrightarrow$ value

Where:

a is a non-negative real number,
x is a non-negative real number,
value is a floating point number being an approximation to $P(a, x)$.

Example:

Three values of the incomplete gamma function are computed.

```
-> gammp(0.1,3.1622777E-02)?
0.7420263
-> gammp(0.5,7.0710678E-01)?
0.7656418
-> gammp(1.0,5.0000000)?
0.9932621
```

---

gammq

Description:

This function returns the incomplete gamma function $Q(a, x) = 1 - P(a, x)$.

gammq(a, x) $\longrightarrow$ value

Where:

a is a non-negative real number,
x is a non-negative real number,
value is a floating point number being an approximation to $Q(a, x)$.

Example:

Two values if the incomplete gamma function `gammq` are computed and compared with the results from `gammp`.

```
-> gammq(0.1,3.1622777E-02)?
0.2579737
-> 1 - 0.7420263?
0.2579737
-> gammq(0.5,7.0710678E-01)?
0.2343582
-> 1 - 0.7656418?
0.2343582
```

---

gser

Description:

This function computes the incomplete gamma function $P(a,x)$ evaluated by its series representation. It also computes the logarithm of the gamma function.

gser(a, x) $\longrightarrow$ [gamser, gln]

Where:

a is a non-negative real number,
x is a non-negative real number,
gamser is a floating point approximation to the incomplete gamma function,
gln is a floating point number approximating $ln(\Gamma(a))$.

Pre-variables:

gser_itmax [100] an integer being the number of series terms to sum,
gser_eps [3.0d-7] a floating point number giving the relative accuracy.

Example:

Two values of the incomplete gamma function are computed using `gser` and the results compared with those obtained using `gammp` and `gammln`.

```
-> gser(0.1,3.1622777E-02)?
[0.7420263, 2.2527126]
-> [gammp(0.1,3.1622777E-02),gammln(0.1)]?
[0.7420263, 2.2527126]
-> gser(0.5,7.0710678E-01)?
[0.7656418, 0.5723649]
-> [gammp(0.5,7.0710678E-01),gammln(0.5)]?
[0.7656418, 0.5723649]
```

gcf

Description:

Returns the incomplete gamma function $Q(a, x)$ using a continued fraction representation.

gcf(a, x) $\longrightarrow$ [gammcf, gln]

Where:

a is a non-negative real number,
x is a non-negative real number,
gammcf is a floating point approximation to the incomplete gamma function $Q(a, x)$.
gln is a floating point number approximating $ln(\Gamma(a))$.

Pre-variables:

gcf_itmax [100] an integer being the number of series terms to sum,
gcf_eps [3.0d-7] a floating point number giving the relative accuracy.

Example:

The incomplete gamma function is computed using `gcf`.

```
-> gcf(0.1,3.1622777E-02)?
Senpack error:  a too large , itmax too small in gcf
-> [gammq(0.1,3.1622777E-02),gammln(0.1)]?
[0.2579737, 2.2527126]
-> gcf(0.5,7.0710678E-01)?
[0.2343583, 0.5723649]
-> [gammq(0.5,7.0710678E-01),gammln(0.5)]?
[0.2343582, 0.5723649]
```

erf

Description:

This function computes the error function $erf(x)$.

erf(x) $\longrightarrow$ **value**

Where:

x is a real number,
value is a floating point number approximating $erf(x)$.

Example:

Three values of the error function `erf` are computed.

```
-> erf(0.0)?
0.0000000
-> erf(0.4)?
0.4283924
-> erf(0.8)?
0.7421010
```

---

erfc

Description:

This function computes the complementary error function $erfc(x)$.

erfc(x) $\longrightarrow$ **value**

Where:

x is a real number,
value is a floating point number approximating $erfc(x)$.

Example:

Three values of the complementary error function `erfc` are computed.

```
-> erfc(0.0)?
1.0000000
-> erfc(0.4)?
0.5716076
-> 1 - 0.4283924?
0.5716076
-> erfc(0.8)?
0.2578990
-> 1 - 0.7421010?
0.2578990
```

---

erfcc

Description:

This function computes the complementary error function $erfc(x)$ using a Chebyshev approximation to the functional form to obtain a fractional error less than 1.2d-7.

erfcc(x) ⟶ value

Where:

x is a real number,
value is a floating point number approximating $erfc(x)$.

Example:

Three values of the complementary error function `erfcc` are computed.

```
-> erfcc(0.0)?
1.0000010
-> erfcc(0.4)?
0.5716079
-> 1 - 0.4283924?
0.5716076
-> erfcc(0.8)?
0.2578992
-> 1 - 0.7421010?
0.2578990
```

---

betacf

Description:

This function computes the continued fraction expansion used by *betai*.

betacf(a, b, x) ⟶ value

Where:

a is a real number,
b is a real number,
x is a real number,
value is a floating point number representing the value of the continued fraction expansion.

Pre-variables:

betacf_itmax [100] an integer representing the maximum number of iterations,
betacf_eps [3.0d-7] a floating point number representing the accuracy.

---

betai

Description:

This function computes the incomplete gamma function $I_x(a, b)$

betai(a, b, x) $\longrightarrow$ **value**

Where:

a is a real number,
b is a real number,
x is a real number in the interval $[0, 1]$,
value is a floating point number approximating $I_x(a, b)$.

Example:

The incomplete gamma function `betai` is computed for three sets of input values.

```
-> betai(0.5, 0.5, 0.01)?
0.0637686
-> betai(1.0, 1.0, 0.5)?
0.5000000
-> betai(20.0, 5.0, 0.8)?
0.4598774
```

---

bessi

Description:

This function computes the modified bessel function I of integer order.

bessi(n, x) $\longrightarrow$ **value**

Where:

n is an integer with $n \geq 2$,
x is a real number,
value is a floating point number approximating $I_n(x)$.

Example:

The Bessel function `bessi` is computed for three sets of values.

```
-> bessi(2,0.2)?
0.0050167
-> bessi(2,10.0)?
2281.5189972
-> bessi(10,10.0)?
21.8917064
```

---

bessi0

Description:

This function computes the modified Bessel function I_0.

bessi0(x) $\longrightarrow$ **value**

Where:

x is a real number,
value is a floating point number approximating $I_0(x)$.

Example:

The Bessel function `bessi0` is computed for three sets of values.

```
-> bessi0(0.0)?
1.0000000
-> bessi0(0.8)?
1.1665149
-> bessi0(3.0)?
4.8807926
```

---

bessi1

Description:

This function computes the modified Bessel function I_1.

bessi1(x) $\longrightarrow$ **value**

Where:

x is a real number,
value is a floating point number approximating $I_1(x)$.

Example:

The Bessel function `bessi1` is computed for three sets of values.

```
-> bessi1(0.0)?
0.0000000
-> bessi1(0.8)?
0.4328648
-> bessi1(3.0)?
3.9533702
```

---

bessj

Description:

This function computes the Bessel function J of integer order.

bessj(n, x) $\longrightarrow$ value

Where:

n is an integer with $n \geq 2$,
x is a real number,
value is a floating point number approximating $J_n(x)$.

Example:

The Bessel function `bessj` is computed for three sets of values.

```
-> bessj(2,1.0)?
0.1149035
-> bessj(2,50.0)?
-0.0602025
-> bessj(10,2.0)?
0.0000003
```

---

bessj0

Description:

This function computes the Bessel function J_0.

bessj0(x) $\longrightarrow$ value

Where:

x is a real number,
value is a floating point number approximating $J_0(x)$.

Example:

The Bessel function `bessj0` is computed for three sets of values.

```
-> bessj0(-5.0)?
-0.1775968
-> bessj0(-1.0)?
0.7651977
-> bessj0(2.0)?
0.2238908
```

---

bessj1

Description:

This function computes the Bessel function J_1.

bessj1(x) $\longrightarrow$ value

Where:

x is a real number,
value is a floating point number approximating $J_1(x)$.

Example:

The Bessel function `bessj1` is computed for three sets of values.

```
-> bessj1(-5.0)?
0.3275791
-> bessj1(-2.0)?
-0.5767248
-> bessj1(3.0)?
0.3390590
```

---

bessk

Description:

This function computes the modified Bessel function K of integer order.

bessk(n, x) $\longrightarrow$ value

Where:

n is an integer with $n \geq 2$,
x is a positive real number,
value is a floating point approximation to $K_n(x)$.

Example:

The Bessel function `bessk` is computed for three sets of values.

```
-> bessk(2,0.2)?
49.5124278
-> bessk(2,3.0)?
0.0615105
-> bessk(5,1.0)?
360.9605868
```

---

bessk0

Description:

This function computes the modified Bessel function K_0.

bessk0(x) $\longrightarrow$ value

Where:

x is a possitive real number,
value is a floating point number approximating $K_0(x)$.

Example:

The Bessel function `bessk0` is computed for three sets of values.

```
-> bessk0(0.1)?
2.4270690
-> bessk0(0.8)?
0.5653471
-> bessk0(3.0)?
0.0347395
```

---

bessk1

Description:

This function computes the modified Bessel function K_1.

bessk1(x) $\longrightarrow$ value

Where:

x is a positive real number,
value is a floating point number approximating $K_1(x)$.

Example:

The Bessel function `bessk1` is computed for three sets of values.

```
-> bessk1(0.1)?
9.8538446
-> bessk1(0.6)?
1.3028349
-> bessk1(1.6)?
0.2406339
```

---

bessy

Description:

This function computes the Bessel function Y of integer order.

bess(n, x) $\longrightarrow$ value

Where:

n is an integer with $n \geq 2$,
x is a positive real number,
value is a floating point number approximating $Y_n(x)$.

Example:

The Bessel function `bessy` is computed for five sets of values.

```
-> bessy(2,1.0)?
-1.6506831
-> bessy(2,50.0)?
0.0957932
-> bessy(5,10.0)?
0.1354030
```

---

bessy0

Description:

This function computes the Bessel function Y_0.

bessy0(x) $\longrightarrow$ value

Where:

x is a positive real number,
value is a floating point approximation to $Y_0(x)$

Example:

The Bessel function `bessy0` is computed for three sets of values.

```
-> bessy0(0.1)?
-1.5342392
-> bessy0(4.0)?
-0.0169409
-> bessy0(8.0)?
0.2235216
```

---

bessy1

Description:

This function computes the Bessel function Y_1.

$$\textbf{bessy1(x)} \longrightarrow \textbf{value}$$

Where:

x is a positive real number,
value is a floating point number approximating $Y_1(x)$.

Example:

The Bessel function `bessy1` is computed for three sets of values.

```
-> bessy1(0.1)?
-6.4589533
-> bessy1(4.0)?
0.3979251
-> bessy1(7.0)?
-0.3026727
```

---

plgndr

Description:

This function evaluates the associated Legrendre polynomials (spherical harmonics).

$$\textbf{plgndr(l, m, x)} \longrightarrow \textbf{value}$$

Where:

l is a positive integer,
m is positive integer with $0 \leq m \leq l$,
x is a real number such that $-1 \leq x \leq 1$,
value is a floating point number approximating $P_l^m(x)$.

Example:

Three values of `plgndr` are computed.

```
-> plgndr(1,0,1.0)?
1.0000000
-> plgndr(20,0,0.7071067)?
-0.1930653
-> plgndr(10,2,0.0)?
27.0703125
```

---

el2

Description:

This function evaluates elliptic integrals of the first and second kind.

el2(x, qqc, aa, bb) $\longrightarrow$ value

Where:

x is a positive real number,
qqc is a real number,
aa is a real number,
bb is a real number,
value is a floating point number approximating $el2(x, k_c, a, b)$ with $x \geq 0$, $qqc = k_c$, $aa = a$ and $bb = b$.

Pre-variables:

el2_ca [3.0d-4] a floating point number such that its square is the desired accuracy,
el2_cb [1.0d-9] a floating point number being 0.01 times the desired accuracy.

Example:

The elliptic integral `el2` is computed over a set of domains and the results compared with those obtained performing the required integrations explicitly using `qsimp`.

```
-> qqcrand_list : [0.2,0.3,0.8,0.7,0.9,0.4,0.7,0.3,0.3,0.9,0.5,
0.1,0.5,0.6,0.9,0.5,0.1,0.2,0.8,0.4];
-> arand_list : map(float,[7,3,5,9,1,2,5,8,7,2,3,6,2,9,1,6,7,5,6,2]);
-> brand_list : map(float,[3,2,1,9,2,8,7,6,4,5,9,1,6,6,9,5,6,5,7,4]);
-> xrand_list : map(float,[3,1,4,7,5,5,8,6,5,8,3,3,5,5,9,3,7,1,8,1]);
-> astop_list : map(atan,xrand_list)?
[1.2490458, 0.7853982, 1.3258177, 1.4288993, 1.3734008, 1.3734008, 1.4464413,
1.4056476, 1.3734008, 1.4464413, 1.2490458, 1.2490458, 1.3734008, 1.3734008,
1.4601391, 1.2490458, 1.4288993, 0.7853982, 1.4464413, 0.7853982]
-> func(x) := (a + b*(tan(x))^2)/sqrt((1d0+(tan(x))^2)*(1d0+qqc^2*(tan(x))^2));
-> x : 3.0; qqc : 0.2; a : 7.0; b : 3.0; astop : first(astop_list)?
1.2490458
-> el2(x,qqc,a,b)?
8.9842972
-> Compare results with those from using Simpsons rule: #
-> qsimp(func,[0.0,astop])?
8.9842974
-> x : 1.0; qqc : 0.3; a : 3.0; b : 2.0; astop : proj(astop_list,2)?
0.7853982
-> el2(x,qqc,a,b)?
2.4391885
-> Compare results with those from using Simpsons rule: #
-> qsimp(func,[0.0,astop])?
2.4391886
```

```
-> for n from 3 by 1 to 20 do sprog(
  x : proj(xrand_list,n), qqc : proj(qqcrand_list,n),
  a : proj(arand_list,n), b : proj(brand_list,n),
  astop : proj(astop_list,n),
  print([x,qqc,a,b,el2(x,qqc,a,b),qsimp(func,[0.0,astop])]));
[4.0000000, 0.8000000, 5.0000000, 1.0000000, 4.7289108, 4.7289110]
[7.0000000, 0.7000000, 9.0000000, 9.0000000, 14.9456644, 14.9456639]
[5.0000000, 0.9000000, 1.0000000, 2.0000000, 2.0685264, 2.0685263]
[5.0000000, 0.4000000, 2.0000000, 8.0000000, 9.5622760, 9.5622759]
[8.0000000, 0.7000000, 5.0000000, 7.0000000, 10.1024145, 10.1024140]
[6.0000000, 0.3000000, 8.0000000, 6.0000000, 14.4794347, 14.4794347]
[5.0000000, 0.3000000, 7.0000000, 4.0000000, 10.8382174, 10.8382169]
[8.0000000, 0.9000000, 2.0000000, 5.0000000, 5.1681773, 5.1681772]
[3.0000000, 0.5000000, 3.0000000, 9.0000000, 8.6216991, 8.6216989]
[3.0000000, 0.1000000, 6.0000000, 1.0000000, 6.5221401, 6.5221404]
[5.0000000, 0.5000000, 2.0000000, 6.0000000, 7.0499818, 7.0499816]
[5.0000000, 0.6000000, 9.0000000, 6.0000000, 12.6231530, 12.6231526]
[9.0000000, 0.9000000, 1.0000000, 9.0000000, 7.3452711, 7.3452709]
[3.0000000, 0.5000000, 6.0000000, 5.0000000, 8.5852049, 8.5852049]
[7.0000000, 0.1000000, 7.0000000, 6.0000000, 16.2491950, 16.2491955]
[1.0000000, 0.2000000, 5.0000000, 5.0000000, 4.3806767, 4.3806768]
[8.0000000, 0.8000000, 6.0000000, 7.0000000, 10.3428643, 10.3428640]
[1.0000000, 0.4000000, 2.0000000, 4.0000000, 2.0576884, 2.0576884]
```

cel

Description:

This function evaluates complete elliptic integrals of all three kinds.

cel(qqc, pp, aa, bb) $\longrightarrow$ value

Where:

qqc is a real number,
p is a real number,
aa is a real number,
bb is a real number,
value is a floating point number approximating $cel(k_c, p, a, b)$ with $qqc = k_c$, $pp = p$, $aa = a$ and $bb = b$.

Pre-variables:

cel_ca [3.0d-4] a floating point number such that its square is the desired accuracy,

Example:

A set of values of the elliptic integral `cel` are computed and the results compared with those obtained using an explicit numerical integration using `qsimp`.

```
-> qqcrand_list : map(float,[7,8,9,8,8,2,8,2,4,4,1,8,2,7,4,7,8,5,7,9]);
-> arand_list : map(float,[8,6,4,9,3,2,0,4,6,9,5,6,5,7,1,9,1,1,1,6]);
-> brand_list : map(float,[4,4,3,1,3,4,5,3,3,1,7,6,5,2,3,9,3,1,6,8]);
-> prand_list : map(float,[5,8,6,8,4,7,6,4,2,1,5,8,1,2,2,5,5,5,5,8]);
-> astop : float(%pi)/2;
-> func(x) := (a*(cos(x))^2 + b*(sin(x))^2)/
               (((cos(x))^2+p*(sin(x))^2)*sqrt((cos(x))^2+qqc^2*(sin(x))^2));
-> p : 5.0; qqc : 7.0; a : 8.0; b : 4.0;
-> cel(qqc,p,a,b)?
2.2317268
-> Compare results with those from using Simpsons rule: #
-> qsimp(func,[0.0,astop])?
2.2317268
-> p : 8.0; qqc : 8.0; a : 6.0; b : 4.0;
-> cel(qqc,p,a,b)?
1.4119005
-> Compare results with those from using Simpsons rule: #
-> qsimp(func,[0.0,astop])?
1.4119005
```

```
-> for n from 3 by 1 to 20 do sprog(
  p : proj(prand_list,n), qqc : proj(qqcrand_list,n),
  a : proj(arand_list,n), b : proj(brand_list,n),
 print([qqc,p,a,b,cel(qqc,p,a,b),qsimp(func,[0.0,astop])]));
[9.0000000, 6.0000000, 4.0000000, 3.0000000, 0.9596187, 0.9596187]
[8.0000000, 8.0000000, 9.0000000, 1.0000000, 1.9820911, 1.9820911]
[8.0000000, 4.0000000, 3.0000000, 3.0000000, 0.8890205, 0.8890205]
[2.0000000, 7.0000000, 2.0000000, 4.0000000, 1.1558350, 1.1558350]
[8.0000000, 6.0000000, 0.0000000, 5.0000000, 0.1695024, 0.1695024]
[2.0000000, 4.0000000, 4.0000000, 3.0000000, 2.2645355, 2.2645355]
[4.0000000, 2.0000000, 6.0000000, 3.0000000, 2.8818388, 2.8818388]
[4.0000000, 1.0000000, 9.0000000, 1.0000000, 4.3886287, 4.3886287]
[1.0000000, 5.0000000, 5.0000000, 7.0000000, 3.9465648, 3.9465648]
[8.0000000, 8.0000000, 6.0000000, 6.0000000, 1.4662044, 1.4662044]
[2.0000000, 1.0000000, 5.0000000, 5.0000000, 5.3912891, 5.3912891]
[7.0000000, 2.0000000, 7.0000000, 2.0000000, 2.3047328, 2.3047328]
[4.0000000, 2.0000000, 1.0000000, 3.0000000, 0.8469649, 0.8469649]
[7.0000000, 5.0000000, 9.0000000, 9.0000000, 2.7094742, 2.7094742]
[8.0000000, 5.0000000, 1.0000000, 3.0000000, 0.3565537, 0.3565537]
[5.0000000, 5.0000000, 1.0000000, 1.0000000, 0.3619335, 0.3619335]
[7.0000000, 5.0000000, 1.0000000, 6.0000000, 0.5219211, 0.5219211]
[9.0000000, 8.0000000, 6.0000000, 8.0000000, 1.4223820, 1.4223820]
```

sncndn

Description:

This function evaluates Jacobian elliptic functions.

sncndn(uu, emmc) ⟶ [sn, cn, dn]

Where:

uu is a real number such that $uu = u$,
emmc is a real number such that $emmc = k_c^2$,
sn is a floating point approximation to $sn(u, k_c)$,
cn is a floating point approximation to $dn(u, k_c)$,
dn is a floating point approximation to $cn(u, k_c)$.

Pre-variables:

sncndn_ca [3.0d-4] a floating point number such that the desired accuracy is its square.

Example:

Five sets of values of the functions `sn`, `cn` and `dn` are computed and a number of identities verified.

```
-> res : sncndn(0.4,1-0.1)?
[0.3884673, 0.9214625, 0.9924260]
-> sn : first(res); cn : proj(res,2); dn : last(res);
-> sn^2+cn^2?
1.0000000
-> 0.9*sn^2 + dn^2?
1.1207255
-> res : sncndn(0.5,1-0.25)?
[0.4750829, 0.8799410, 0.9713774]
-> sn : first(res); cn : proj(res,2); dn : last(res);
-> sn^2+cn^2?
1.0000000
-> 0.25*sn^2 + dn^2?
1.0000000
```