

# The AMath Special Functions

---

## Reference Manual and Implementation Notes

Draft Version 0.4

Wolfgang Ehrhardt

Feb. 2012

Copyright © 2009-2012 Wolfgang Ehrhardt

This electronic draft version is distributed under the terms and conditions of the "Creative Commons license Attribution - Noncommercial - No Derivative Works 3.0"<sup>1</sup>. This is not a license, but a human-readable summary of the full license from:

<http://creativecommons.org/licenses/by-nc-nd/3.0/>

You are free to copy, distribute and transmit this draft under the following conditions:

- Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work)
- Noncommercial. You may not use this work for commercial purposes.
- No Derivative Works. You may not alter, transform, or build upon this work

With the understanding that:

- Any of the above conditions can be waived if you get permission from the copyright holder.
- Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the URL given above.

---

<sup>1</sup>The license type may change in future versions of this text.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>AMath functions</b>	<b>6</b>
<b>3</b>	<b>Special functions</b>	<b>7</b>
3.1	Bessel functions and related . . . . .	8
3.1.1	Bessel functions of integer order . . . . .	8
3.1.2	Modified Bessel functions of integer order . . . . .	10
3.1.3	Bessel functions of real order . . . . .	12
3.1.4	Modified Bessel functions of real order . . . . .	14
3.1.5	Airy functions . . . . .	15
3.1.6	Kelvin functions . . . . .	17
3.1.7	Spherical Bessel functions . . . . .	19
3.1.8	Struve functions . . . . .	21
3.2	Elliptic integrals, elliptic and theta functions . . . . .	23
3.2.1	Complete elliptic integral of the 1st kind . . . . .	23
3.2.2	Complete elliptic integral of the 2nd kind . . . . .	23
3.2.3	Complete elliptic integral of the 3rd kind . . . . .	23
3.2.4	Legendre elliptic integral of the 1st kind . . . . .	24
3.2.5	Legendre elliptic integral of the 2nd kind . . . . .	24
3.2.6	Legendre elliptic integral of the 3rd kind . . . . .	25
3.2.7	Carlson style elliptic integrals . . . . .	25
3.2.8	Bulirsch style elliptic integrals . . . . .	26
3.2.9	Maple style elliptic integrals . . . . .	28
3.2.10	Heuman's Lambda function . . . . .	31
3.2.11	Jacobi Zeta function . . . . .	31
3.2.12	Elliptic modulus . . . . .	32
3.2.13	Elliptic nome . . . . .	32
3.2.14	Jacobi amplitude . . . . .	32
3.2.15	Jacobi elliptic functions . . . . .	33
3.2.16	Inverse Jacobi elliptic functions . . . . .	34
3.2.17	Jacobi theta functions . . . . .	34
3.2.18	Jacobi theta functions at zero . . . . .	35
3.3	Error function and related . . . . .	38
3.3.1	Dawson's integral . . . . .	38
3.3.2	Error function erf . . . . .	38
3.3.3	Complementary error function erfc . . . . .	38
3.3.4	Imaginary error function erfi . . . . .	38
3.3.5	Exponentially scaled complementary error function . . . . .	39
3.3.6	Inverse function of erf . . . . .	39

3.3.7	Inverse function of $\operatorname{erfc}$	39
3.3.8	$\operatorname{expint}3$	39
3.3.9	Goodwin-Staton integral	40
3.3.10	Fresnel integrals	40
3.4	Exponential integrals and related	41
3.4.1	Hyperbolic cosine integral $\operatorname{Chi}$	41
3.4.2	Cosine integral $\operatorname{Ci}$	41
3.4.3	Entire cosine integral $\operatorname{Cin}$	41
3.4.4	Entire hyperbolic cosine integral $\operatorname{Cinh}$	42
3.4.5	Exponential integral $E_1$	42
3.4.6	Exponential integral $Ei$	42
3.4.7	Entire exponential integral $\operatorname{Ein}$	43
3.4.8	Exponential integrals $E_n$	43
3.4.9	Logarithmic integral $\operatorname{li}$	44
3.4.10	Hyperbolic sine integral $\operatorname{Shi}$	44
3.4.11	Sine integral $\operatorname{Si}$	44
3.4.12	Shifted sine integral $\operatorname{si}$	45
3.5	Gamma function and related	46
3.5.1	Gamma functions	46
3.5.2	Incomplete gamma functions	48
3.5.3	Beta functions	50
3.5.4	Factorials, Pochhammer symbol, binomial coefficient	51
3.5.5	Psi and polygamma functions	54
3.6	Zeta functions and polylogarithms	56
3.6.1	Riemann zeta function	56
3.6.2	Prime zeta function	57
3.6.3	Dirichlet eta function	58
3.6.4	Dirichlet beta function	58
3.6.5	Dirichlet lambda function	59
3.6.6	Hurwitz zeta function	59
3.6.7	Legendre's Chi-function	59
3.6.8	Lerch's transcendent	60
3.6.9	Polylogarithms of integer order	61
3.6.10	Polylogarithms of real order	62
3.6.11	Dilogarithm function	62
3.6.12	Clausen function	62
3.6.13	Inverse-tangent integral	63
3.7	Orthogonal polynomials and related	64
3.7.1	Chebyshev polynomials of the first kind	64
3.7.2	Chebyshev polynomial of the second kind	64
3.7.3	Gegenbauer (ultraspherical) polynomials	65
3.7.4	Hermite polynomials	65
3.7.5	Jacobi polynomials	65
3.7.6	Generalized Laguerre polynomials	66
3.7.7	Laguerre polynomials	66
3.7.8	Associated Laguerre polynomials	66
3.7.9	Legendre polynomials	67
3.7.10	Associated Legendre polynomials	67
3.7.11	Legendre functions of the second kind	68
3.7.12	Spherical harmonic functions	68
3.7.13	Zernike radial polynomials	68
3.8	Statistical distributions	69

3.8.1	Beta distribution . . . . .	69
3.8.2	Binomial distribution . . . . .	69
3.8.3	Cauchy distribution . . . . .	70
3.8.4	Chi-square distribution . . . . .	70
3.8.5	Exponential distribution . . . . .	70
3.8.6	F-distribution . . . . .	71
3.8.7	Gamma distribution . . . . .	71
3.8.8	Hypergeometric distribution . . . . .	72
3.8.9	Laplace distribution . . . . .	72
3.8.10	Logistic distribution . . . . .	72
3.8.11	Lognormal distribution . . . . .	73
3.8.12	Negative binomial distribution . . . . .	73
3.8.13	Gaussian Normal distribution . . . . .	74
3.8.14	Standard Normal distribution . . . . .	74
3.8.15	Pareto distribution . . . . .	74
3.8.16	Poisson distribution . . . . .	75
3.8.17	t-distribution . . . . .	75
3.8.18	Triangular distribution . . . . .	75
3.8.19	Uniform distribution . . . . .	76
3.8.20	Weibull distribution . . . . .	76
3.9	Other special functions . . . . .	78
3.9.1	Arithmetic-geometric mean . . . . .	78
3.9.2	Bernoulli numbers . . . . .	78
3.9.3	Debye functions . . . . .	79
3.9.4	Lambert W functions . . . . .	79
3.9.5	Riemann prime counting function . . . . .	80
<b>A</b>	<b>Licenses</b> . . . . .	<b>81</b>
A.1	Boost . . . . .	81
A.2	Cephes . . . . .	81
A.3	FDLIBM . . . . .	82
A.4	SLATEC . . . . .	82
	<b>Bibliography</b> . . . . .	<b>83</b>

# Chapter 1

## Introduction

The **AMath** package contains Pascal/Delphi open source code for accurate mathematical methods without using multi precision arithmetic. Please note that the high accuracy can only be achieved with the `rmNearest` rounding mode; it decreases if other modes are used. The main parts of the package are the **AMath** and **AMTools** units and the Special Functions units.

The units and basic test programs can be compiled with the usual Pascal (TP 5, 5.5, 6; BP 7; VP 2.1; FPC 1.0, 2.0, 2.2, 2.4, 2.6) and Delphi (1..7, 9, 10, 12) versions.<sup>1</sup>

This draft describes the **AMath** Special Functions with additional implementation notes and formulas. For the other **AMath** functions see the **AMath** HTML introduction at [http://home.netsurf.de/wolfgang.ehrhardt/amath\\_functions.html](http://home.netsurf.de/wolfgang.ehrhardt/amath_functions.html).

The latest version of this document is available from <http://home.netsurf.de/wolfgang.ehrhardt>.

The **bibliography** lists the general references items, but there are some special additional references of limited scope in the Pascal units, see e.g. `procedure sincosPix2` in `SFBasic.pas`.

The  $\LaTeX$  text is written with  $\TeX$ studio V2.2 and  $\text{MiK}\TeX$  V2.4.

---

<sup>1</sup>The units can be compiled with TP5 but some functions generate invalid operations due to TP5's brain-damaged usage of the FPU: TP5 essentially evaluates expressions as if the FPU stack is virtually unlimited. Especially functions/expressions using Carlson's elliptic integrals may generate FPU exceptions.

## Chapter 2

# AMath functions

The **AMath** unit implements accurate mathematical functions, it makes many routines of Delphi's **math** unit available to other supported Pascal versions and fixes bugs and inaccuracies of Delphi.

The elementary mathematical functions include: exponential, logarithmic, trigonometric, hyperbolic, inverse circular and hyperbolic functions. Then there are polynomial, vector, statistic operations as well as floating point and FPU control functions.

All standard elementary transcendental functions with one argument have peak relative errors less than 2.2e-19, values for `power(x,y)` are 2.1e-19 (for  $|x|, |y| < 1000$ ) and 3.4e-19 (for  $|x|, |y| < 2000$ ).

The accurate implementation of special functions needs versions of certain elementary mathematical functions that are more accurate than those supplied by Delphi and Pascal, especially `sin`, `cos`, and `power`.

The RTL trigonometric functions are accurate for arguments  $x$  with  $|x| \leq \pi/4$ , the (Intel) FPU supports arguments  $|x| \leq 2^{62}\pi$  (but with varying accuracy). **AMath** uses two types of trigonometric range reductions. The Cody/Waite style type is used for  $|x| \leq 2^{39}$  and is based on the Cephes [7] routines in `sinl.c`.

The Payne/Hanek style range reduction is used for large arguments  $|x| > 2^{39}$ , or if the Cody/Waite reduced argument is very close to a multiple of  $\pi/2$ . The Payne/Hanek reduction is described e.g. in [6] and the **AMath** implementation is a Pascal translation of the FDLIBM [5] C function `_kernel_rem_pio2`.

The power function `power(x,y) = xy = exp(y ln x)` is based on [7, `powl.c`]: Following Cody and Waite, the Cephes function uses a lookup table of 32 entries and pseudo extended precision arithmetic to obtain several extra bits of accuracy in both the logarithm and the exponential. The **AMath** routine uses a table size of 512 entries resulting in about four additional bits of accuracy; the tables are calculated with `MPArith`.<sup>1</sup>

The following less common notations from the **AMath** unit are used in this manual: The Euler constant  $\gamma = 0.5772156649\dots$  and some accuracy improved functions: `expm1(x) = exp(x) - 1`, `expx2(x) = exp(x · |x|)`, `ln1p(x) = ln(1 + x)`, and `powm1(x,y) = xy - 1`.

---

<sup>1</sup>The table with 1024 entries gives nearly full accuracy, but requires about 15 KB.

## Chapter 3

# Special functions

The units **SpecFun** and **SpecFunX** interface special functions for extended and double precision; the following function groups are available: Bessel functions and related, elliptic integrals/functions and theta functions, gamma function and related, error function and related, exponential integrals and related, zeta functions and polylogarithms, orthogonal polynomials and related, statistical distributions, and other special functions.

The interface units **SpecFun** and **SpecFunX** actually use common functions located in more special units roughly representing the above function groups.

Currently all functions have double and extended precision versions (with name suffix `x`), eg. `erfc` vs. `erfcx`. Generally the extended versions have larger relative errors (measured in the corresponding machine epsilon `eps_x` or `eps_d`) than their double counterparts, especially `gammax` and `betax` for large arguments. Note that some functions are sensitive to small changes in the argument; therefore in high precision comparisons argument values should be used, that are representable in both calculations.

Many function implementations have special code for very small or very large arguments (i.e. for  $x < \text{eps}_x$ ,  $x < \text{eps}_x^{1/2}$ , or  $x > 1/\text{eps}_x$ ), where accurate results can be achieved with only one or two terms of a Maclaurin series or asymptotic expression. These special branches are normally not mentioned in the following descriptions.

## 3.1 Bessel functions and related

The unit `sfBessel` contains the common code for the Bessel and related functions.

### 3.1.1 Bessel functions of integer order

There are some internal routines for the Bessel functions of integer order:

```
procedure bess_m0p0(x: extended; var m0, p0: extended);
```

The procedure returns the modulus  $M_0(x)$  and phase  $\theta_0(x)$  of  $J_0(x)$  and  $Y_0(x)$ , for  $|x| \geq 9$  (c.f. Abramowitz and Stegun [1, 9.2.17 – 9.2.31]), the results are computed with two rational approximations from Cephes [7, file j0l.c].

```
procedure bess_m1p1(x: extended; var m1, p1: extended);
```

The procedure returns the modulus  $M_1(x)$  and phase  $\theta_1(x)$  of  $J_1(x)$  and  $Y_1(x)$ , for  $|x| \geq 9$  (c.f. Abramowitz and Stegun [1, 9.2.17 – 9.2.31]), the results are computed with two rational approximations from Cephes [7, file j1l.c].

#### $\mathbf{J_0(x)}$

---

```
function bessel_j0(x: double): double;  
function bessel_j0x(x: extended): extended;
```

---

These functions return  $J_0(x)$ , the Bessel function of the 1st kind, order zero. If  $|x| < 9$  a rational approximation from Cephes [7, file j0l.c] is used, for  $9 \leq |x| < 500$  the common function calls `bess_m0p0` and returns  $J_0(x) = M_0 \cos \theta_0$ , otherwise the result is computed with the internal asymptotic function for the real order `bessj_large(0, x)`.

#### $\mathbf{J_1(x)}$

---

```
function bessel_j1(x: double): double;  
function bessel_j1x(x: extended): extended;
```

---

These functions return  $J_1(x)$ , the Bessel function of the 1st kind, order one. If  $|x| < 9$  a rational approximation from Cephes [7, file j1l.c] is used, for  $9 \leq |x| < 500$  the common function calls `bess_m1p1` and returns  $J_1(x) = M_1 \cos \theta_1$ , otherwise the result is computed with the internal asymptotic function for the real order `bessj_large(1, x)`.

#### $\mathbf{J_n(x)}$

---

```
function bessel_jn(n: integer; x: double): double;  
function bessel_jnx(n: integer; x: extended): extended;
```

---

These functions return  $J_n(x)$ , the Bessel function of the 1st kind, order  $n$ :

$$J_n(x) = \left(\frac{1}{2}x\right)^n \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{1}{4}x^2\right)^k}{k!(n+k)!}, \quad J_{-n}(x) = (-1)^n J_n(x),$$

see [30, 10.2.2]. **Note:** This routine is **not** suitable for large  $n$  or  $x$ , in these cases the real order function  $J_\nu(x)$  with  $\nu = n$  should be called. For  $n = 0, 1$  the basic functions  $J_0(x)$  or  $J_1(x)$  are returned. For  $x > n$  the forward recurrence formula

$$J_{n+1} = \frac{2n}{x} J_n(x) - J_{n-1}(x)$$

is stable and the result is calculated with the starting values  $J_0(x)$  and  $J_1(x)$ . For  $n \geq x$  the formula is unstable and the recurrence is applied in the backward direction. Starting with  $J_{n+1}(x)/J_n(x)$ , which is computed by a continued fraction algorithm<sup>1</sup> (c.f. Boost[19, `bessel_jy.hpp`]), the final result is normalized with the true value of  $J_0(x)$ .

### **Y<sub>0</sub>(x)**

---

```
function bessel_y0(x: double): double;
function bessel_y0x(x: extended): extended;
```

---

These functions return  $Y_0(x)$ , the Bessel function of the 2nd kind, order zero for  $x > 0$ . If  $x < 9$  two rational approximations from Cephes [7, file `j0l.c`] are used, for  $9 \leq x < 1600$  the common function calls `bess_m0p0` and returns  $Y_0(x) = M_0 \sin \theta_0$ , otherwise the result is computed with the internal asymptotic function for the real order `bessy_large(0, x)`.

### **Y<sub>1</sub>(x)**

---

```
function bessel_y1(x: double): double;
function bessel_y1x(x: extended): extended;
```

---

These functions return  $Y_1(x)$ , the Bessel function of the 2nd kind, order one for  $x > 0$ . If  $x < 9$  two rational approximations from Cephes [7, file `j1l.c`] are used, for  $9 \leq x < 1600$  the common function calls `bess_m1p1` and returns  $Y_1(x) = M_1 \sin \theta_1$ , otherwise the result is computed with the internal asymptotic function for the real order `bessy_large(1, x)`.

### **Y<sub>n</sub>(x)**

---

```
function bessel_yn(n: integer; x: double): double;
function bessel_ynx(n: integer; x: extended): extended;
```

---

These functions return  $Y_n(x)$ , the Bessel function of the 2nd kind, order  $n$  for  $x > 0$ , expressible for  $n \geq 0$  as (see [30, 10.8.1]):

$$Y_n(x) = -\frac{(\frac{1}{2}x)^{-n}}{\pi} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!} (\frac{1}{4}x^2)^k + \frac{2}{\pi} \ln(\frac{1}{2}x) J_n(x) - \frac{(\frac{1}{2}x)^n}{\pi} \sum_{k=0}^{\infty} (\psi(k+1) + \psi(n+k+1)) \frac{(-\frac{1}{4}x^2)^k}{k!(n+k)!}$$

and with  $Y_{-n}(x) = (-1)^n Y_n(x)$  for negative  $n$ . **Note:** This routine is **not** suitable for large  $n$  or  $x$ , in these cases the real order function  $Y_\nu(x)$  with  $\nu = n$  should be called. For  $n = 0, 1$  the basic functions  $Y_0(x)$  or  $Y_1(x)$  are returned.

The forward recurrence formula is stable

$$Y_{n+1} = \frac{2n}{x} Y_n(x) - Y_{n-1}(x)$$

and is used to compute the result with the starting values  $Y_0(x)$  and  $Y_1(x)$ .

---

<sup>1</sup> See Press et al.[13, Ch. 5.5 and 6.5] for basic information.

### 3.1.2 Modified Bessel functions of integer order

All of the four basic modified Bessel functions of integer order  $I_0, I_1, K_0, K_1$  have internal versions for small arguments.

```
function bess_i0_small(x: extended): extended;
```

This routine computes the modified Bessel function  $I_0(x)$  for  $|x| \leq 3$  using a Chebyshev approximation from Fullerton [20, 14] (files dbesi0.f and dbsi0e.f).

```
function bess_i1_small(x: extended): extended;
```

This routine computes the modified Bessel function  $I_1(x)$  for  $|x| \leq 3$  using a Chebyshev approximation from Fullerton [20, 14] (files dbesi1.f and dbsi1e.f).

```
function bess_k0_small(x: extended): extended;
```

This routine computes the modified Bessel function  $K_0(x)$  for  $0 < x \leq 2$  using a Chebyshev approximation from Fullerton [20, 14] (files dbesk0.f and dbsk0e.f).

```
function bess_k1_small(x: extended): extended;
```

This routine computes the modified Bessel function  $K_1(x)$  for  $0 < x \leq 2$  using a Chebyshev approximation from Fullerton [20, 14] (files dbesk1.f and dbsk1e.f).

#### $I_0(x)$

---

```
function bessel_i0(x: double): double;  
function bessel_i0x(x: extended): extended;
```

---

These functions compute  $I_0(x)$ , the modified Bessel function of the 1st kind, order 0. If  $|x| \leq 3$  the result is `bess_i0_small(x)`, otherwise  $e^{|x|}I_{0,e}(x)$  is returned.

#### Exponentially scaled $I_{0,e}(x)$

---

```
function bessel_i0e(x: double): double;  
function bessel_i0ex(x: extended): extended;
```

---

These functions compute  $I_{0,e}(x) = e^{-|x|}I_0(x)$ , the exponentially scaled modified Bessel function of the 1st kind, order 0. If  $|x| \leq 3$  the result is `bess_i0_small(x)e^{-|x|}`, otherwise two Chebyshev approximations from Fullerton [20, 14] (file `dbsi0e.f`) are used.

#### $I_1(x)$

---

```
function bessel_i1(x: double): double;  
function bessel_i1x(x: extended): extended;
```

---

These functions compute  $I_1(x)$ , the modified Bessel function of the 1st kind, order 1. If  $|x| \leq 3$  the result is `bess_i1_small(x)`, otherwise  $e^{|x|}I_{1,e}(x)$  is returned.

#### Exponentially scaled $I_{1,e}(x)$

---

```
function bessel_i1e(x: double): double;  
function bessel_i1ex(x: extended): extended;
```

---

These functions compute  $I_{1,e}(x) = e^{-|x|}I_1(x)$ , the exponentially scaled modified Bessel function of the 1st kind, order 1. If  $|x| \leq 3$  the result is `bess_i1_small(x)e^{-|x|}`, otherwise two Chebyshev approximations from Fullerton [20, 14] (file `dbsi1e.f`) are used.

## **I<sub>n</sub>(x)**

---

```
function bessel_in(n: integer; x: double): double;  
function bessel_inx(n: integer; x: extended): extended;
```

---

These functions calculate  $I_n(x)$ , the modified Bessel function of the 1st kind, order  $n$ , see [30, 10.25.2]:

$$I_n(x) = \left(\frac{1}{2}x\right)^n \sum_{k=0}^{\infty} \frac{\left(\frac{1}{4}x^2\right)^k}{k!(n+k)!}, \quad I_{-n}(x) = I_n(x).$$

**Note:** This routine is **not** suitable for large  $n$  or  $x$ , in these cases the real order function  $I_\nu(x)$  with  $\nu = n$  should be called. For  $n = 0, 1$  the basic functions  $I_0(x)$  or  $I_1(x)$  are returned. Except for very small  $x$ , the result is computed with the value of  $I_0(x)$  and Miller's algorithm [30, 3.6(iii)] using the backward recurrence

$$I_{n-1}(x) = \frac{2n}{x}I_n(x) + I_{n+1}(x).$$

## **K<sub>0</sub>(x)**

---

```
function bessel_k0(x: double): double;  
function bessel_k0x(x: extended): extended;
```

---

These functions calculate  $K_0(x)$ , the modified Bessel function of the 2nd kind, order zero,  $x > 0$ . If  $x \leq 2$  the result is `bess_k0_small(x)`, otherwise  $e^{-x}K_{0,e}(x)$  is returned.

## **Exponentially scaled K<sub>0,e</sub>(x)**

---

```
function bessel_k0e(x: double): double;  
function bessel_k0ex(x: extended): extended;
```

---

These functions compute  $K_{0,e}(x) = e^x K_0(x)$ , the exponentially scaled modified Bessel function of the 2nd kind, order 0,  $x > 0$ . If  $x \leq 2$  the common function returns `bess_k0_small(x)ex`, otherwise two Chebyshev approximations from Fullerton [20, 14] (file `dbsk0e.f`) are used.

## **K<sub>1</sub>(x)**

---

```
function bessel_k1(x: double): double;  
function bessel_k1x(x: extended): extended;
```

---

These functions calculate  $K_1(x)$ , the modified Bessel function of the 2nd kind, order one,  $x > 0$ . If  $x \leq 2$  the result is `bess_k1_small(x)`, otherwise  $e^{-x}K_{1,e}(x)$  is returned.

## **Exponentially scaled K<sub>1,e</sub>(x)**

---

```
function bessel_k1e(x: double): double;  
function bessel_k1ex(x: extended): extended;
```

---

These functions compute  $K_{1,e}(x) = e^x K_1(x)$ , the exponentially scaled modified Bessel function of the 2nd kind, order one,  $x > 0$ . If  $x \leq 2$  the common function returns `bess_k1_small(x)ex`, otherwise two Chebyshev approximations from Fullerton [20, 14] (file `dbsk1e.f`) are used.

**K<sub>n</sub>(x)**

---

```
function bessel_kn(n: integer; x: double): double;  
function bessel_knx(n: integer; x: extended): extended;
```

---

These functions return  $K_n(x)$ , the modified Bessel function of the 2nd kind, order  $n$  for  $x > 0$ , expressible for  $n \geq 0$  as (see [30, 10.31.1]):

$$K_n(x) = \frac{1}{2} \left(\frac{1}{2}x\right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!} \left(-\frac{1}{4}x^2\right)^k + (-1)^{n+1} \ln\left(\frac{1}{2}x\right) I_n(x) \\ + (-1)^{n+1} \frac{1}{2} \left(\frac{1}{2}x\right)^n \sum_{k=0}^{\infty} (\psi(k+1) + \psi(n+k+1)) \frac{\left(\frac{1}{4}x^2\right)^k}{k!(n+k)!}$$

and with  $K_{-n}(x) = K_n(x)$  for negative  $n$ . **Note:** This routine is **not** suitable for large  $n$  or  $x$ , in these cases the real order function  $K_\nu(x)$  with  $\nu = n$  should be called. For  $n = 0, 1$  the basic functions  $K_0(x)$  or  $K_1(x)$  are returned. The forward recurrence formula is stable

$$K_{n+1} = \frac{2n}{x} K_n(x) + K_{n-1}(x)$$

and is used to compute the result with the starting values  $K_0(x)$  and  $K_1(x)$ .

### 3.1.3 Bessel functions of real order

First the internal functions are described;  $\nu$  is written as  $v$  in the source code.

```
procedure h1v_large(v, x: extended; var mv, tmx: extended);
```

The procedure calculates the modulus  $M_\nu(x)$  and phase  $\theta_\nu(x)$  of the Hankel function  $H_\nu^{(1)}(x) = J_\nu(x) + iY_\nu(x)$  for large  $x > 0$  using the asymptotic expansions from Abramowitz and Stegun [1, 9.2.28/9.2.29] with  $\mu = 4\nu^2$ :

$$M_\nu^2 \sim \frac{2}{\pi x} \left( 1 + \frac{1}{2} \frac{\mu-1}{(2x)^2} + \frac{1 \cdot 3}{2 \cdot 4} \frac{(\mu-1)(\mu-9)}{(2x)^4} + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \frac{(\mu-1)(\mu-9)(\mu-25)}{(2x)^6} + \dots \right) \\ \theta_\nu \sim x - \left(\frac{1}{2}\nu + \frac{1}{4}\right) + \frac{\mu-1}{2(4x)} + \frac{(\mu-1)(\mu-25)}{6(4x)^3} + \frac{(\mu-1)(\mu^2-114\mu+1073)}{5(4x)^5} + \dots$$

The returned values are  $mv = M_\nu(x)$  and  $tmx = \theta_\nu(x) - x$ .

```
function bessj_large(v, x: extended): extended;
```

This function returns  $J_\nu(x)$  for large  $x > 0$  with the modulus/phase asymptotic expansion `h1v_large` and  $J_\nu(x) = M_\nu \cos \theta_\nu$ .

```
function bessy_large(v, x: extended): extended;
```

This function returns  $Y_\nu(x)$  for large  $x > 0$  with the modulus/phase asymptotic expansion `h1v_large` and  $Y_\nu(x) = M_\nu \sin \theta_\nu$ .

```
procedure bessel_jy(v, x: extended; BT: byte; var Jv, Yv: extended);
```

This procedure returns  $J_\nu(x)$  and/or  $Y_\nu(x)$  depending on `BT`, it assumes  $x > 0$  and  $|\nu| < \text{MaxLongint}$ . (For  $x < 0$  the functions  $J_\nu$  and  $Y_\nu$  are in general complex; and  $Y_\nu$  is singular for  $x = 0$ .)

`bessel_jy` is a bit complicated and this description shows only the main implementation topics, for the subtle parts the source code is the final reference.

If  $x$  is 'large' (this depends on  $\nu$ ), the asymptotic expansions are used, and for  $\nu < 0$  the functions are computed with the reflection formulas [13, 6.7.19]

$$\begin{aligned} J_{-\nu}(x) &= \cos(\pi\nu)J_\nu(x) - \sin(\pi\nu)Y_\nu(x), \\ Y_{-\nu}(x) &= \sin(\pi\nu)J_\nu(x) + \cos(\pi\nu)Y_\nu(x). \end{aligned}$$

The essential calculation method is described in Ch. 6.7 and the C function `bessjy` in Press et al.[13], some ideas are from the Boost [19] code in file `bessel_jy.hpp`, function `bessel_jy`. *Steed's method* for  $x \geq 2$  uses two continued fractions and the Wronskian relation

$$J_\nu(x)Y'_\nu(x) - Y_\nu(x)J'_\nu(x) = \frac{2}{\pi x}$$

to compute  $J_\nu$ ,  $J'_\nu$ ,  $Y_\nu$ , and  $Y'_\nu$  simultaneously. The first continued fraction CF1 ([13, 6.7.2] and [1, 9.1.73]) evaluates

$$\frac{J'_\nu}{J_\nu} = \frac{\nu}{x} - \frac{J_{\nu+1}}{J_\nu} = \frac{\nu}{x} - \frac{1}{2(\nu+1)/x - \frac{1}{2(\nu+2)/x - \dots}}$$

while the second CF2 from [13, 6.7.3] is complex

$$\frac{J'_\nu + iY'_\nu}{J_\nu + iY_\nu} = -\frac{1}{2x} + i + \frac{i}{x} \frac{(1/2)^2 - \nu^2}{2(x+i) + \frac{(2/2)^2 - \nu^2}{2(x+2i) + \dots}}$$

If  $x < 2$  Temme's method [Temme76a] for evaluating  $Y_{\nu'}$  and  $Y_{\nu'+1}$  (with  $|\nu'| \leq 1/2$ ) is used together with the forward recurrence relation to get  $Y_\nu(x)$ . If  $J_\nu(x)$  is needed, it is computed with CF1 and the Wronskian.

### $\mathbf{J}_\nu(\mathbf{x})$

---

```
function bessel_jv(v, x: double): double;
function bessel_jvx(v, x: extended): extended;
```

---

These functions return  $J_\nu(x)$ , the Bessel function of the 1st kind, real order  $\nu$ :

$$J_\nu(x) = \left(\frac{1}{2}x\right)^\nu \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{1}{4}x^2\right)^k}{k!\Gamma(\nu+k+1)},$$

see [30, 10.2.2], if  $x < 0$  then  $\nu$  must be an integer. The formula is used only for  $x < 1$  or  $\nu > x^2/4$ . If  $\nu = n$  is an integer smaller than 200 then  $J_n(x)$  is returned. Otherwise the result is computed with general procedure `bessel_jy`.

### $\mathbf{Y}_\nu(\mathbf{x})$

---

```
function bessel_yv(v, x: double): double;
function bessel_yvx(v, x: extended): extended;
```

---

These functions return  $Y_\nu(x)$ , the Bessel function of the 2nd kind, real order  $\nu$ , and  $x > 0$ ; see [30, 10.2.3]:

$$Y_\nu(x) = \frac{J_\nu(x) \cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}.$$

The result is computed with general procedure `bessel_jy`, except in two special cases: When  $\nu = n$  is an integer smaller than 2000 then  $Y_n(x)$  is returned, and if  $\nu$  is a negative half-integer then  $Y_\nu(x) = -J_{-\nu}(x) \sin \nu\pi$  (this may avoid the reflection formulas with both functions, note  $|\sin \nu\pi| = 1$ ).

### 3.1.4 Modified Bessel functions of real order

The internal functions are described first, they are similar to the unmodified.

```
procedure bessel_ik(v, x: extended; CalcI, escale: boolean;
                   var Iv, Kv: extended);
```

This procedure returns  $I_\nu(x)$  and/or  $K_\nu(x)$  depending on CalcI for  $x > 0$ , and  $|\nu| < \text{MaxLongint}$ . If escale=true the values are exponentially scaled. For  $\nu < 0$  the functions are computed with the reflection formulas [13, 6.7.40]

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

$$K_{-\nu}(x) = K_\nu(x).$$

The Wronskian relation for the modified functions is ([13, 6.7.20])

$$I_\nu(x)K'_\nu(x) - K_\nu(x)I'_\nu(x) = -\frac{1}{x},$$

and the first continued fraction CF1 becomes (see [13, 6.7.21] and [30, 10.33.1]):

$$\frac{I'_\nu}{I_\nu} = \frac{\nu}{x} + \frac{1}{2(\nu+1)/x + \frac{1}{2(\nu+2)/x + \dots}}$$

CF2 is a bit complicated and described in [13, 6.7.22–6.7.27].

The forward recurrence for  $K_\nu$  is used, if  $x < 2$  with a corresponding Temme normalisation [Temme76b] or CF2 otherwise.

When  $I_\nu(x)$  shall be computed, this is done with an asymptotic expansion from [1, 9.7.1] or [30, 10.40.4] if  $x > 100$  and  $2\nu < x^{1/2}$ ; otherwise CF1 and the Wronskian are used.

**I<sub>ν</sub>(x)**

---

```
function bessel_iv(v, x: double): double;
function bessel_ivx(v, x: extended): extended;
```

---

These functions return  $I_\nu(x)$ , the modified Bessel function of the 1st kind, real order  $\nu$ ,  $x \geq 0$  if  $\nu$  is not an integer, defined as [30, 10.25.2]:

$$I_\nu(x) = \left(\frac{1}{2}x\right)^\nu \sum_{k=0}^{\infty} \frac{\left(\frac{1}{4}x^2\right)^k}{k!\Gamma(\nu+k+1)!}$$

The common routine **sfci\_iv** handles the four special  $\nu$  cases

$$I_\nu(x) = \begin{cases} I_0(x) & \nu = 0, \\ I_1(x) & |\nu| = 1, \\ \sinh(x)/(x\pi/2)^{1/2} & \nu = 1/2, \\ \cosh(x)/(x\pi/2)^{1/2} & \nu = -1/2. \end{cases}$$

For other arguments the general procedure **bessel\_ik** is used.

### Exponentially scaled $I_{\nu,e}(x)$

---

```
function bessell_ive(v, x: double): double;  
function bessell_ivex(v, x: extended): extended;
```

---

The functions return  $I_{\nu,e}(x) = I_{\nu}(x) \exp(-|x|)$  the exponentially scaled modified Bessel function of the 1st kind, real order  $\nu$ ,  $x \geq 0$  if  $\nu$  is not an integer. When  $\nu = 0$  or  $|\nu| = 1$  the integer order routines  $I_{0,e}(x)$  or  $I_{1,e}(x)$  are used, otherwise the result is computed with the general procedure `bessell_ik`.

### $K_{\nu}(x)$

---

```
function bessell_kv(v, x: double): double;  
function bessell_kvex(v, x: extended): extended;
```

---

These functions return  $K_{\nu}(x)$ , the modified Bessel function of the 2nd kind, real order  $\nu$ ,  $x > 0$ , defined as [30, 10.27.4]:

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

If  $\nu = n$  is an integer<sup>2</sup> the integer order function  $K_n$  is used. For  $|\nu| = 1/2$  there is the special case  $\exp(-x)(\pi/(2x))^{1/2}$ , and otherwise the result is computed with the general procedure `bessell_ik`.

### Exponentially scaled $K_{\nu,e}(x)$

---

```
function bessell_kve(v, x: double): double;  
function bessell_kvex(v, x: extended): extended;
```

---

These functions return  $e^x K_{\nu}(x)$ , the exponentially scaled modified Bessel function of the 2nd kind, real order  $\nu$ ,  $x > 0$ . When  $\nu = 0$  or  $|\nu| = 1$  the integer order routines  $K_{0,e}(x)$  or  $K_{1,e}(x)$  are used, for  $|\nu| = 1/2$  the value  $(\pi/(2x))^{1/2}$  is returned, otherwise the result is computed with the general procedure `bessell_ik`.

## 3.1.5 Airy functions

In this section let  $z = (2/3)|x|^{3/2}$ . For  $x \geq 0$  the Airy functions can be defined by

$$\begin{aligned} \text{Ai}(x) &= \frac{1}{\pi} \sqrt{\frac{x}{3}} K_{1/3}(z), \\ \text{Bi}(x) &= \sqrt{\frac{x}{3}} (I_{1/3}(z) + I_{-1/3}(z)), \end{aligned}$$

There are two parametrised internal functions that compute the Airy functions for small arguments.

```
function Airy_small(x, f0, f1: extended): extended;
```

This function returns  $\text{Ai}(x)$  or  $\text{Bi}(x)$  using Maclaurin series for 'small'  $x$ . It evaluates the two series expansions from [30, 9.4.1/9.4.3]:

$$f_0 \cdot \left(1 + \frac{1}{3!}x^3 + \frac{1 \cdot 4}{6!}x^6 + \frac{1 \cdot 4 \cdot 7}{9!}x^9 + \dots\right) + f_1 \cdot \left(x + \frac{2}{4!}x^4 + \frac{2 \cdot 5}{7!}x^7 + \frac{2 \cdot 5 \cdot 8}{10!}x^{10} + \dots\right)$$

---

<sup>2</sup> and less than `MaxLongint`

**function** `AiryP_small(x, f0, f1: extended): extended;`

This function returns  $\text{Ai}'(x)$  or  $\text{Bi}'(x)$  using Maclaurin series for 'small'  $x$ . It evaluates the two series expansions from Olver et al. [30, 9.4.2/9.4.4]:

$$f_0 \cdot \left(1 + \frac{2}{3!}x^3 + \frac{2 \cdot 5}{6!}x^6 + \frac{2 \cdot 5 \cdot 8}{9!}x^9 + \dots\right) + f_1 \cdot \left(\frac{1}{2!}x^2 + \frac{1 \cdot 4}{5!}x^5 + \frac{1 \cdot 4 \cdot 7}{8!}x^8 + \dots\right)$$

### Airy Ai(x)

---

**function** `airy_ai(x: double): double;`  
**function** `airy_aix(x: extended): extended;`

---

These functions calculate the Airy function  $\text{Ai}(x)$ . `Airy_small(x, f0, f1)` is returned for  $-2 \leq x \leq 1$  with

$$f_0 = \text{Ai}(0) = 3^{-2/3} / \Gamma(2/3), \quad f_1 = \text{Ai}'(0) = -3^{-1/3} / \Gamma(1/3).$$

For  $x > 1$  the definition formula is evaluated

$$\text{Ai}(x) = \frac{1}{\pi} \sqrt{\frac{x}{3}} K_{1/3}(z),$$

and for  $x < -2$  the result is (cf. Press et al. [13, 6.7.46]):

$$\text{Ai}(x) = \frac{1}{2} \sqrt{-x} \left( J_{1/3}(z) - \frac{1}{\sqrt{3}} Y_{1/3}(z) \right).$$

### Airy Ai'(x)

---

**function** `airy_aip(x: double): double;`  
**function** `airy_aipx(x: extended): extended;`

---

These functions calculate the Airy function  $\text{Ai}'(x)$ . `AiryP_small(x, Ai(0), Ai'(0))` is returned if  $-1 \leq x \leq 1$ . For  $x > 1$  the formula from [13, 6.7.45] is used

$$\text{Ai}'(x) = -\frac{x}{\pi \sqrt{3}} K_{2/3}(z),$$

and for  $x < -1$  the result is [13, 6.7.46]):

$$\text{Ai}'(x) = -\frac{x}{2} \left( J_{2/3}(z) + \frac{1}{\sqrt{3}} Y_{2/3}(z) \right).$$

### Airy Bi(x)

---

**function** `airy_bi(x: double): double;`  
**function** `airy_bix(x: extended): extended;`

---

These functions calculate the Airy function  $\text{Bi}(x)$ . `Airy_small(x, f0, f1)` is returned for  $-1 \leq x \leq 1$  with

$$f_0 = \text{Bi}(0) = 3^{-1/6} / \Gamma(2/3), \quad f_1 = \text{Bi}'(0) = 3^{1/6} / \Gamma(1/3).$$

For  $x > 1$  the formula from [13, 6.7.44] is used

$$\text{Bi}(x) = \sqrt{x} \left( \frac{2}{\sqrt{3}} I_{1/3}(z) + \frac{1}{\pi} K_{1/3}(z) \right),$$

and for  $x < -1$  the result is [13, 6.7.46]):

$$\text{Bi}(x) = -\frac{1}{2} \sqrt{-x} \left( \frac{1}{\sqrt{3}} J_{1/3}(z) + Y_{1/3}(z) \right).$$

## Airy Bi'(x)

---

```
function airy_bip(x: double): double;  
function airy_bipx(x: extended): extended;
```

---

These functions calculate the Airy function  $\text{Bi}'(x)$ . `AiryP_small(x, Bi(0), Bi'(0))` is returned if  $-1 \leq x \leq 1$ . For  $x > 1$  the formula from [13, 6.7.45] is used

$$\text{Bi}'(x) = x \left( \frac{2}{\sqrt{3}} I_{2/3}(z) + \frac{1}{\pi} K_{2/3}(z) \right),$$

and for  $x < -1$  the result is [13, 6.7.46]:

$$\text{Bi}'(x) = -\frac{x}{2} \left( \frac{1}{\sqrt{3}} J_{2/3}(z) - Y_{2/3}(z) \right).$$

### 3.1.6 Kelvin functions

The Kelvin functions of general order  $\nu$  can be defined as

$$\begin{aligned} \text{ber}_\nu(x) + i \text{bei}_\nu(x) &= e^{+\nu\pi i/2} I_\nu(xe^{\pi i/4}) \\ \text{ker}_\nu(x) + i \text{kei}_\nu(x) &= e^{-\nu\pi i/2} K_\nu(xe^{\pi i/4}) \end{aligned}$$

see Abramowitz and Stegun [1, 9.9.1 and 9.9.2]. When  $\nu = 0$ , suffices are suppressed. These definitions show that the four functions `ber`, `bei`, `ker`, and `kei` are tightly related, e.g. in the implemented asymptotic expansion for say `ber` all four functions are computed. Consequently there are some internal routines common to the Kelvin functions:

```
function ber_sm(x: extended): extended;
```

This function computes the Kelvin function `ber`  $x$  for  $0 \leq x < 20$  with the series expansion [1, 9.9.10]:

$$\text{ber } x = \sum_{k=0}^{\infty} (-1)^k \frac{(\frac{1}{4}x^2)^{2k}}{((2k)!)^2}$$

```
function bei_sm(x: extended): extended;
```

This function computes the Kelvin function `bei`  $x$  for  $0 \leq x < 20$  with the series expansion [1, 9.9.10]:

$$\text{bei } x = \sum_{k=0}^{\infty} (-1)^k \frac{(\frac{1}{4}x^2)^{2k+1}}{((2k+1)!)^2}$$

```
function ker_sm(x: extended): extended;
```

This function computes the Kelvin function `ker`  $x$  for  $0 < x < 3$  with the series expansion [1, 9.9.12]:

$$\text{ker } x = -\ln(\frac{1}{2}x) \text{ber } x + \frac{1}{4}\pi \text{bei } x + \sum_{k=0}^{\infty} (-1)^k \frac{\psi(2k+1)}{((2k)!)^2} (\frac{1}{4}x^2)^{2k}$$

or with the harmonic numbers  $H_k = \psi(k+1) + \gamma$

$$\text{ker } x = -(\gamma + \ln(\frac{1}{2}x)) \text{ber } x + \frac{1}{4}\pi \text{bei } x + \sum_{k=0}^{\infty} (-1)^k \frac{H_{2k}}{((2k)!)^2} (\frac{1}{4}x^2)^{2k}$$

**function kei\_sm(x: extended): extended;**

This function computes the Kelvin function  $\text{kei } x$  for  $0 < x < 3$  with the series expansion [1, 9.9.12]:

$$\begin{aligned}\text{kei } x &= -\ln\left(\frac{1}{2}x\right) \text{ber } x - \frac{1}{4}\pi \text{ber } x + \sum_{k=0}^{\infty} (-1)^k \frac{\psi(2k+2)}{((2k+1)!)^2} \left(\frac{1}{4}x^2\right)^{2k+1} \\ &= -(\gamma + \ln\left(\frac{1}{2}x\right)) \text{ber } x - \frac{1}{4}\pi \text{ber } x + \sum_{k=0}^{\infty} (-1)^k \frac{H_{2k+1}}{((2k+1)!)^2} \left(\frac{1}{4}x^2\right)^{2k+1}\end{aligned}$$

**procedure kelvin\_large(x: extended; cb: boolean;  
var br, bi, kr, ki: extended);**

This procedure computes all four Kelvin functions for  $x \geq 20$  using asymptotic expansions from [1, 9.10.1 – 9.10.7]:

$$\begin{aligned}\text{ker } x &= \sqrt{\pi/(2x)} e^{-x/\sqrt{2}} \left( + f_0(-x) \cos \beta - g_0(-x) \sin \beta \right), \\ \text{kei } x &= \sqrt{\pi/(2x)} e^{-x/\sqrt{2}} \left( - f_0(-x) \sin \beta - g_0(-x) \cos \beta \right), \\ \text{ber } x &= \frac{e^{x/\sqrt{2}}}{\sqrt{2\pi x}} \left( f_0(x) \cos \alpha + g_0(x) \sin \alpha \right) - \frac{1}{\pi} \text{kei } x, \\ \text{bei } x &= \frac{e^{x/\sqrt{2}}}{\sqrt{2\pi x}} \left( f_0(x) \sin \alpha - g_0(x) \cos \alpha \right) + \frac{1}{\pi} \text{ker } x,\end{aligned}$$

with  $\alpha = x/\sqrt{2} - \frac{1}{8}\pi$  and  $\beta = \alpha + \frac{1}{4}\pi$  and the auxiliary functions

$$\begin{aligned}f_0(\pm x) &\sim 1 + \sum_{k=1}^{\infty} (\mp 1)^k \frac{(-1)(-9)\cdots(-(2k-1)^2)}{k!(8x)^k} \cos(k\pi/4) \\ g_0(\pm x) &\sim + \sum_{k=1}^{\infty} (\mp 1)^k \frac{(-1)(-9)\cdots(-(2k-1)^2)}{k!(8x)^k} \sin(k\pi/4)\end{aligned}$$

$\text{ker}$  and  $\text{kei}$  are always calculated,  $\text{ber}$  and  $\text{bei}$  only if  $\text{cb}=\text{true}$ .

**procedure ker\_kei\_med(x: extended; var kr, ki: extended);**

This procedure bridges the gap  $3 \leq x \leq 20$  for  $\text{ker}$  and  $\text{kei}$ , basically it uses the asymptotic form but each of the functions  $f_0$  and  $g_0$  are evaluated with a Chebyshev approximation.

### Kelvin functions ber(x) and bei(x)

---

**procedure kelvin\_berbei(x: double; var br, bi: double);**  
**procedure kelvin\_berbeix(x: extended; var br, bi: extended);**

---

These procedures calculate both Kelvin functions  $\text{br} = \text{ber}(x)$ , and  $\text{bi} = \text{bei}(x)$ . If  $|x| < 20$  the values are computed with the separate small functions  $\text{ber\_sm}$  and  $\text{bei\_sm}$ , and otherwise  $\text{kelvin\_large}$  is used.

### Kelvin functions `ker(x)` and `kei(x)`

---

```
procedure kelvin_kerkei(x: double; var kr, ki: double);  
procedure kelvin_kerkeix(x: extended; var kr, ki: extended);
```

---

These procedures calculate both Kelvin functions  $kr = \ker(x)$ , and  $ki = \text{kei}(x)$ , for  $x > 0$ . If  $x < 3$  the values are computed with the separate small functions `ker_sm` and `kei_sm`, in the range  $3 \leq x \leq 20$  the results are from `ker_kei_med`, and otherwise `kelvin_large` is used.

### Kelvin function `ber(x)`

---

```
function kelvin_ber(x: double): double;
```

---

These functions return the Kelvin function  $\text{ber}(x)$ . If  $|x| < 20$  the result is `ber_sm`, otherwise `kelvin_large` is used.

### Kelvin function `bei(x)`

---

```
function kelvin_bei(x: double): double;
```

---

These functions return the Kelvin function  $\text{bei}(x)$ . If  $|x| < 20$  the result is `bei_sm`, otherwise `kelvin_large` is used.

### Kelvin function `ker(x)`

---

```
function kelvin_ker(x: double): double;
```

---

These functions return the Kelvin function  $\ker(x)$  for  $x > 0$ . If  $x < 3$  the result is `ker_sm`, otherwise procedure `kelvin_kerkei` is used.

### Kelvin function `kei(x)`

---

```
function kelvin_kei(x: double): double;
```

---

These functions return the Kelvin function  $\text{kei}(x)$  for  $x > 0$ . If  $x < 3$  the result is `kei_sm`, otherwise procedure `kelvin_kerkei` is used.

## 3.1.7 Spherical Bessel functions

### Spherical Bessel function `jn(x)`

---

```
function sph_bessel_jn(n: integer; x: double): double;  
function sph_bessel_jnx(n: integer; x: extended): extended;
```

---

These functions return  $j_n(x)$ , the spherical Bessel function of the 1st kind, order  $n$ . Except for  $n = 0$  where the value `sinc(x)` is used, the result is calculated just from the definition [1, 10.1.1]:

$$j_n(x) = \sqrt{\frac{1}{2}\pi/x} J_{n+\frac{1}{2}}(x) \quad (x \geq 0), \quad \text{and} \quad j_n(-x) = (-1)^n j_n(x).$$

### Spherical Bessel function $y_n(x)$

---

```
function sph_bessel_yn(n: integer; x: double): double;  
function sph_bessel_ynx(n: integer; x: extended): extended;
```

---

These functions return  $y_n(x)$ , the spherical Bessel function of the 2nd kind, order  $n$ ,  $x \neq 0$ . The result is calculated using the definition [1, 10.1.1]:

$$y_n(x) = \sqrt{\frac{1}{2}\pi/x} Y_{n+\frac{1}{2}}(x) \quad (x > 0), \quad \text{and} \quad y_n(-x) = (-1)^{n+1} y_n(x).$$

### Modified spherical Bessel function $i_n(x)$

---

```
function sph_bessel_in(n: integer; x: double): double;  
function sph_bessel_inx(n: integer; x: extended): extended;
```

---

These functions compute  $i_n(x)$ , the modified spherical Bessel function of the 1st (and 2nd) kind, order  $n$ . Except for  $n = 0$  where the value  $\sinh(x)/x$  is returned, the result is calculated just from the definition [1, 10.2.2] or [30, 10.47.7]

$$i_n(x) = \sqrt{\frac{1}{2}\pi/x} I_{n+\frac{1}{2}}(x) \quad (x \geq 0), \quad \text{and} \quad i_n(-x) = (-1)^n i_n(x)$$

with the reflection formula from [30, 10.47.16]. Note that  $i_n$  is named  $i_n^{(1)}$  in the NIST handbook [30] and restricted to  $n \geq 0$ , the modified spherical Bessel function of the 2nd kind is then defined as

$$i_n^{(2)}(x) = \sqrt{\frac{1}{2}\pi/x} I_{-n-\frac{1}{2}}(x),$$

which can be expressed by  $i_n$ , i.e.

$$i_n^{(1)}(x) = i_n(x), \quad i_n^{(2)}(x) = i_{-n-1}(x), \quad (\text{for } n \geq 0).$$

### Exponentially scaled $i_{n,e}(x)$

---

```
function sph_bessel_ine(n: integer; x: double): double;  
function sph_bessel_inex(n: integer; x: extended): extended;
```

---

These functions return  $i_n(x) \exp(-|x|)$ , the exponentially scaled modified spherical Bessel function of the 1st/2nd kind, order  $n$ . For  $n = 0$  the result is

$$i_{0,e}(x) = -\frac{\expm1(-2|x|)}{2|x|},$$

otherwise the common function `sfc_sph_ine` uses  $I_{\nu,e}(x)$ , the exponentially scaled modified Bessel function of the 1st kind:

$$i_{n,e}(x) = \sqrt{\frac{1}{2}\pi/x} I_{n+\frac{1}{2},e}(x) \quad (x \geq 0), \quad \text{and} \quad i_{n,e}(-x) = (-1)^n i_{n,e}(x)$$

### Modified spherical Bessel function $k_n(x)$

---

```
function sph_bessel_kn(n: integer; x: double): double;
function sph_bessel_knx(n: integer; x: extended): extended;
```

---

These functions return  $k_n(x)$ , the modified spherical Bessel function of the 3rd kind, order  $n$ ,  $x > 0$ . Except for  $n = 0$  the result is calculated just from the definition [30, 10.47.9]:

$$k_0(x) = \frac{1}{2}\pi \frac{e^{-x}}{x} \quad \text{and} \quad k_n(x) = \sqrt{\frac{1}{2}\pi/x} K_{n+\frac{1}{2}}(x), \quad (x > 0).$$

### Exponentially scaled $k_{n,e}(x)$

---

```
function sph_bessel_kne(n: integer; x: double): double;
function sph_bessel_knex(n: integer; x: extended): extended;
```

---

These functions return  $k_n(x)e^x$ , the exponentially scaled modified spherical Bessel function of the 3rd kind, order  $n$ . For  $n = 0$  the result is  $\pi/(2x)$ , otherwise the common function `sfc_sph_kne` uses  $K_{\nu,e}(x)$

$$k_{n,e}(x) = \sqrt{\frac{1}{2}\pi/x} K_{n+\frac{1}{2},e}(x) \quad (x > 0).$$

### 3.1.8 Struve functions

The Struve functions  $\mathbf{H}_\nu(x)$  and the modified Struve functions  $\mathbf{L}_\nu(x)$  have the power series expansions (see Abramowitz and Stegun [1, 12.1.3 and 12.2.1]):

$$\mathbf{H}_\nu(x) = \left(\frac{1}{2}x\right)^{\nu+1} \sum_{k=0}^{\infty} \frac{(-1)^k \left(\frac{1}{2}x\right)^{2k}}{\Gamma(k + \frac{3}{2})\Gamma(k + \nu + \frac{3}{2})}$$
$$\mathbf{L}_\nu(x) = \left(\frac{1}{2}x\right)^{\nu+1} \sum_{k=0}^{\infty} \frac{\left(\frac{1}{2}x\right)^{2k}}{\Gamma(k + \frac{3}{2})\Gamma(k + \nu + \frac{3}{2})}$$

$\mathbf{L}_\nu(x)$  has the asymptotic expansion for large  $x > 0$  [1, 12.2.6]:

$$\mathbf{L}_\nu(x) \sim I_{-\nu}(x) + \frac{1}{\pi} \sum_{k=0}^{\infty} \frac{(-1)^{k+1} \Gamma(k + \frac{1}{2})}{\Gamma(\nu + \frac{1}{2} - k) \left(\frac{1}{2}x\right)^{2k-\nu+1}}$$

### Struve $\mathbf{H}_0(x)$

---

```
function struve_h0(x: double): double;
function struve_h0x(x: extended): extended;
```

---

These functions calculate  $\mathbf{H}_0(x)$ , the Struve function of order 0. For  $x < 0$  the result is  $-\mathbf{H}_0(|x|)$ . The common function `sfc_struve_h0` uses two Chebyshev approximations from [22, function STRVH0], one for  $x < 11$  and the second together with a call to  $Y_0(x)$  otherwise.<sup>3</sup>

---

<sup>3</sup> The function in [22] has a third approximation instead of  $Y_0$ .

### Struve $\mathbf{H}_1(\mathbf{x})$

---

```
function struve_h1(x: double): double;  
function struve_h1x(x: extended): extended;
```

---

These functions calculate  $\mathbf{H}_1(x)$ , the Struve function of order 1. The common function `sfc_struve_h1` uses two Chebyshev approximations from [22, function STRVH1], one for  $|x| < 9$  and the second together with a call to  $Y_1(x)$  otherwise.

### Struve $\mathbf{L}_0(\mathbf{x})$

---

```
function struve_l0(x: double): double;  
function struve_l0x(x: extended): extended;
```

---

These functions calculate  $\mathbf{L}_0(x)$ , the modified Struve function of order 0. For  $x < 0$  the common function `sfc_struve_l0` returns  $-\mathbf{L}_0(|x|)$ . For  $x < 22$  the defining power series is used, otherwise the result is computed with the asymptotic expansion.

### Struve $\mathbf{L}_1(\mathbf{x})$

---

```
function struve_l1(x: double): double;  
function struve_l1x(x: extended): extended;
```

---

These functions calculate  $\mathbf{L}_1(x)$ , the modified Struve function of order 1. For  $|x| < 22$  the defining power series is used, otherwise the result is computed with the asymptotic expansion.

## 3.2 Elliptic integrals, elliptic and theta functions

The Pascal unit `sfEllInt` implements the common code for the elliptic integrals, elliptic and theta functions.

### 3.2.1 Complete elliptic integral of the 1st kind

---

```
function comp_ellint_1(k: double): double;  
function comp_ellint_1x(k: extended): extended;
```

---

These functions compute the value of the complete elliptic integral of the first kind  $K(k)$  with  $|k| < 1$

$$K(k) = \int_0^{\pi/2} \frac{dt}{\sqrt{1 - k^2 \sin^2 t}}.$$

The common function `sfc.EllipticK` returns

$$K(k) = \frac{\pi/2}{\text{agm}(1, \sqrt{1 - k^2})},$$

where the AGM is performed in-line.

### 3.2.2 Complete elliptic integral of the 2nd kind

---

```
function comp_ellint_2(k: double): double;  
function comp_ellint_2x(k: extended): extended;
```

---

These functions compute the value of the complete elliptic integral of the second kind  $E(k)$  with  $|k| \leq 1$

$$E(k) = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2 t} dt.$$

The common function `sfc.EllipticEC` returns

$$E(k) = \text{cel2}(k_c, 1, k_c^2)$$

with  $k_c = \sqrt{1 - k^2}$ , c.f. Bulirsch[10, p.80].

### 3.2.3 Complete elliptic integral of the 3rd kind

---

```
function comp_ellint_3(nu, k: double): double;  
function comp_ellint_3x(nu, k: extended): extended;
```

---

These functions compute the value of the complete elliptic integral of the third kind  $\Pi(\nu, k)$  with  $|k| < 1$ ,  $\nu \neq 1$

$$\Pi(\nu, k) = \int_0^{\pi/2} \frac{dt}{(1 - \nu \sin^2 t) \sqrt{1 - k^2 \sin^2 t}}.$$

The common function `sfc.EllipticPiC` returns

$$\Pi(\nu, k) = \text{cel}(k_c, 1 - \nu, 1, 1)$$

with  $k_c = \sqrt{1 - k^2}$ , c.f. Bulirsch[11, 1.2.2].

### 3.2.4 Legendre elliptic integral of the 1st kind

---

```
function ellint_1(phi,k: double): double;
function ellint_1x(phi,k: extended): extended;
```

---

These functions compute the incomplete Legendre elliptic integral of the first kind

$$F(\varphi, k) = \int_0^\varphi \frac{dt}{\sqrt{1 - k^2 \sin^2 t}}$$

with  $|k \sin \varphi| \leq 1$ . Obviously  $F(\varphi, k) = \varphi$  for  $k = 0$ ; and if  $|k| = 1$  and  $|\varphi| < \pi/2$  then  $F(\varphi, k) = \operatorname{arcd} \varphi$  (see [30, (19.6.8)]).

If  $|k| > 1$  the common function `sfc_ellint_1` performs a reciprocal-modulus transformation [30, (19.7.4)]

$$\begin{aligned} F(\varphi, k) &= F(\arcsin(k \sin \varphi), 1/k) / k \\ &= \operatorname{el1}(\tan(\arcsin(k \sin \varphi)), (1 - 1/k^2)^{1/2}) / k. \end{aligned}$$

For  $|k| < 1$  the argument  $\varphi$  is reduced mod  $\pi$ , i.e.  $\varphi = n\pi + \varphi'$  with  $|\varphi'| \leq \pi/2$ . If  $|\varphi'| = \pi/2$ , i.e.  $\varphi$  is an odd multiple  $m$  of  $\pi/2$ , then  $F = mK(k)$ . For  $|\varphi'| < \pi/2$  the integral is calculated with Bulirsch's [10] `el1`:

$$F(\varphi, k) = \operatorname{el1}(\tan \varphi', \sqrt{1 - k^2}).$$

And since  $F$  is quasi-periodic<sup>4</sup> the total result is

$$F(\varphi, k) = 2nK(k) + \operatorname{el1}(\tan \varphi', \sqrt{1 - k^2}).$$

### 3.2.5 Legendre elliptic integral of the 2nd kind

---

```
function ellint_2(phi,k: double): double;
function ellint_2x(phi,k: extended): extended;
```

---

These functions compute the incomplete Legendre elliptic integral of the second kind

$$E(\varphi, k) = \int_0^\varphi \sqrt{1 - k^2 \sin^2 t} dt$$

with  $|k \sin \varphi| \leq 1$ . Again  $E(\varphi, k) = \varphi$  for  $k = 0$ . If  $|k| > 1$  the common function `sfc_ellint_2` performs a reciprocal-modulus transformation [30, (19.7.4)]. The right hand side  $R = (E(\beta, k) - k'^2 F(\beta, k)) / k$  can be rewritten with the function

$$B(\beta, k) = \int_0^\beta \frac{\cos^2 x}{(1 - k^2 \sin^2 x)^{1/2}} dx$$

as  $R = kB(\beta, k)$ . The  $B$  integral can be evaluated by a single call to the function `el2` (or `cel2` if  $\beta = \pi/2$ ), see Bulirsch [10].

For  $|k| < 1$  the argument  $\varphi$  is reduced mod  $\pi$ , i.e.  $\varphi = n\pi + \varphi'$  with  $|\varphi'| \leq \pi/2$ . If  $|\varphi'| = \pi/2$ , i.e.  $\varphi$  is an odd multiple  $m$  of  $\pi/2$ , then  $E(\varphi, k) = mE(k)$ . For  $|\varphi'| < \pi/2$  the integral is calculated with Bulirsch's [10] `el2`:

$$E(\varphi, k) = \operatorname{el2}(\tan \varphi', k_c, 1, k_c^2), \quad k_c^2 = 1 - k^2.$$

And since  $E(\varphi, k)$  is quasi-periodic the total result is

$$E(\varphi, k) = 2nE(k) + \operatorname{el2}(\tan \varphi', k_c, 1, k_c^2).$$

---

<sup>4</sup> See e.g. [30, (19.2.10)]

### 3.2.6 Legendre elliptic integral of the 3rd kind

---

```
function ellint_3(phi, nu, k: double): double;
function ellint_3x(phi, nu, k: extended): extended;
```

---

These functions compute the incomplete Legendre elliptic integral of the third kind

$$\Pi(\varphi, \nu, k) = \int_0^\varphi \frac{dt}{(1 - \nu \sin^2 t) \sqrt{1 - k^2 \sin^2 t}}$$

with  $|k \sin \varphi| \leq 1$ . If  $\nu \sin^2 \varphi > 1$  the Cauchy principal value of the integral is returned. The common function `sfc_ellint_3` handles the following special cases:

$$\Pi(\varphi, \nu, k) = \begin{cases} 0 & \varphi = 0, \\ \varphi & \nu = 0, k = 0, \\ F(\varphi, k) & \nu = 0, k \neq 0, \\ \tan \varphi & \nu = 1, k = 0. \end{cases}$$

If  $|k| > 1$  the reciprocal-modulus transformation from [30, (19.7.4)] gives

$$\begin{aligned} \Pi(\varphi, \nu, k) &= \Pi(\arcsin(k \sin \varphi), \nu/k^2, 1/k)/k \\ &= \text{EllipticPi}(k \sin \varphi, \nu/k^2, 1/k)/k \end{aligned}$$

For  $|k| < 1$  the argument  $\varphi$  is reduced mod  $\pi$ , i.e.  $\varphi = n\pi + \varphi'$  with  $|\varphi'| \leq \pi/2$ . Using the quasi-periodicity of  $\Pi$  and the abbreviations  $s = \sin \varphi'$  and  $c = \cos \varphi'$ , the total integral can be calculated with the complete integral  $\Pi(\nu, k)$  and the Carlson functions, see [12, (4.3)]:

$$\Pi(\varphi, \nu, k) = 2n\Pi(\nu, k) + sR_F(c^2, 1 - k^2s^2, 1) + \frac{\nu}{3}s^3R_J(c^2, 1 - k^2s^2, 1, 1 - \nu s^2)$$

### 3.2.7 Carlson style elliptic integrals

The Carlson style elliptic integrals are a complete alternative group to the classical Legendre style integrals. They are symmetric and the numerical calculation is usually performed by duplication as described in Carlson [12].

Note that using the **AMath** implementation for Carlson's elliptic integrals with the very old Turbo Pascal 5.0 compiler may generate FPU exceptions; if really needed the source code sequences have to be broken into smaller pieces.

#### Degenerate elliptic integral $R_C$

---

```
function ell_rc(x, y: double): double;
function ell_rcx(x, y: extended): extended;
```

---

These functions compute the value of Carlson's degenerate<sup>5</sup> elliptic integral  $R_C$  for  $x \geq 0, y \neq 0$ :

$$R_C(x, y) = R_F(x, y, y) = \frac{1}{2} \int_0^\infty (t+x)^{-1/2} (t+y)^{-1} dt.$$

---

<sup>5</sup>  $R_C$  is not a true elliptic integral, but a logarithm or inverse hyperbolic function for  $y < x$  and an inverse circular function for  $y > x$ .

If  $y < 0$  the returned result is the Cauchy principal value

$$R_C(x, y) = \left( \frac{x}{x-y} \right)^{1/2} R_C(x-y, -y),$$

otherwise  $R_C$  is calculated with Carlson's [12] Algorithm 2, see also Press et al. [13, 6.11], function `rc`.

### Integral of the 1st kind $R_F$

---

```
function ell_rf(x, y, z: double): double;
function ell_rfx(x, y, z: extended): extended;
```

---

These functions return Carlson's elliptic integral of the 1st kind

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty ((t+x)(t+y)(t+z))^{-1/2} dt,$$

with  $x, y, z \geq 0$ , at most one may be zero.  $R_F$  is computed with Carlson's [12] Algorithm 1 (see also [13, 6.11], function `rf`). Note that Carlson's original  $s^6$  error scaling does not give accurate results for extended precision<sup>6</sup>, therefore **AMath** scales the error with  $s^{4.5}$ .

### Integral of the 2nd kind $R_D$

---

```
function ell_rd(x, y, z: double): double;
function ell_rdx(x, y, z: extended): extended;
```

---

These functions return Carlson's elliptic integral of the 2nd kind

$$R_D(x, y, z) = R_J(x, y, z, z) = \frac{3}{2} \int_0^\infty ((t+x)(t+y))^{-1/2} (t+z)^{-3/2} dt$$

with  $z > 0$ ,  $x, y \geq 0$ , at most one of  $x, y$  may be zero.  $R_D$  is computed with Carlson's [12] Algorithm 4 (see also [13, 6.11], function `rd`).

### Integral of the 3rd kind $R_J$

---

```
function ell_rj(x, y, z, r: double): double;
function ell_rjx(x, y, z, r: extended): extended;
```

---

These functions calculate Carlson's elliptic integral of the 3rd kind

$$R_J(x, y, z, r) = \frac{3}{2} \int_0^\infty (t+r)^{-1} ((t+x)(t+y)(t+z))^{-1/2} dt$$

with  $x, y, z \geq 0$ , at most one may be zero and  $r \neq 0$ . If  $r > 0$  then  $R_F$  is computed with Carlson's [12] Algorithm 3 (see also [13, 6.11], function `rj`). For  $r < 0$  the Cauchy principal value [12, 2.22] is returned.

## 3.2.8 Bulirsch style elliptic integrals

Bulirsch's integrals are linear combinations of the Legendre integrals. They are computed with algorithms based on the Bartky transformation and use the complementary modulus  $k_c = k'$  as input parameter for numerical stability reasons. They are described in a series of articles ([10, 11] and two others).

<sup>6</sup> Also noticed in [19], function `ellint_rf.hpp`: Boost uses  $s^{4.25}$ .

### Complete integral of the 1st kind cel1

---

```
function cel1(kc: double): double;  
function cel1x(kc: extended): extended;
```

---

These functions return Bulirsch's complete elliptic integral of the first kind

$$\text{cel1}(k_c) = \int_0^\infty \frac{dt}{\sqrt{(1+t^2)(1+k_c^2 t^2)}}$$

with  $k_c \neq 0$ . The common function `sfc_cel1` is a Pascal port of the ALGOL procedure `cel1` in [10].

### Complete integral of the 2nd kind cel2

---

```
function cel2(kc, a, b: double): double;  
function cel2x(kc, a, b: extended): extended;
```

---

These functions return Bulirsch's complete elliptic integral of the second kind

$$\text{cel2}(k_c, a, b) = \int_0^\infty \frac{a + bt^2}{(1+t^2)\sqrt{(1+t^2)(1+k_c^2 t^2)}} dt$$

with  $k_c \neq 0$ . If  $ab \geq 0$  the common function `sfc_cel1` uses a Pascal port of the ALGOL procedure `cel2` in [10]. In order to achieve optimal accuracy with extended precision the integral is evaluated as `cel(kc, 1, a, b)` for  $ab < 0$ .

### General complete integral cel

---

```
function cel(kc, p, a, b: double): double;  
function celx(kc, p, a, b: extended): extended;
```

---

These functions evaluate Bulirsch's general complete elliptic integral

$$\text{cel}(k_c, p, a, b) = \int_0^\infty \frac{a + bt^2}{(1+pt^2)\sqrt{(1+t^2)(1+k_c^2 t^2)}} dt$$

with  $k_c \neq 0$ . If  $p < 0$  the Cauchy principle value is returned. The common function `sfc_cel` is a Pascal port of the ALGOL procedure `cel` in [11].

### Incomplete integral of the 1st kind el1

---

```
function el1(x, kc: double): double;  
function el1x(x, kc: extended): extended;
```

---

These functions evaluate Bulirsch's incomplete elliptic integral of the first kind

$$\text{el1}(x, k_c) = \int_0^x \frac{dt}{\sqrt{(1+t^2)(1+k_c^2 t^2)}}.$$

If  $k_c = 0$  the common function `sfc_el1` returns

$$\text{el1}(x, 0) = \text{arcsinh } x,$$

otherwise a Pascal port of the ALGOL procedure `el1` from [10] is used.

### Incomplete integral of the 2nd kind `el2`

---

```
function el2(x, kc, a, b: double): double;  
function el2x(x, kc, a, b: extended): extended;
```

---

These functions evaluate Bulirsch's incomplete elliptic integral of the second kind

$$\text{el2}(x, k_c, a, b) = \int_0^x \frac{a + bt^2}{(1+t^2)\sqrt{(1+t^2)(1+k_c^2 t^2)}} dt$$

If  $k_c = 0$  the common function `sfc_el2` returns

$$\text{el2}(x, 0, a, b) = \frac{(a-b)x}{\sqrt{1+x^2}} + b \operatorname{arcsinh} x,$$

otherwise a Pascal port of the ALGOL procedure `el2` from [10] is used.

### Incomplete integral of the 3rd kind `el3`

---

```
function el3(x, kc, p: double): double;  
function el3x(x, kc, p: extended): extended;
```

---

These functions evaluate Bulirsch's incomplete elliptic integral of the third kind

$$\text{el3}(x, k_c, p) = \int_0^x \frac{1+t^2}{(1+pt^2)\sqrt{(1+t^2)(1+k_c^2 t^2)}} dt$$

Bulirsch's `el3` in [11] replaces a former inadequate version, but it is still suboptimal; therefore the common function `sfc_el3` uses the Carlson form [12, 4.18]:

$$\text{el3}(x, k_c, p) = x R_F(1, 1 + k_c^2 x^2, 1 + x^2) + \frac{1}{3}(1-p)x^3 R_J(1, 1 + k_c^2 x^2, 1 + x^2, 1 + px^2)$$

## 3.2.9 Maple style elliptic integrals

### Complete integral of the 1st kind `EllipticK`

---

```
function EllipticK(k: double): double;  
function EllipticKx(k: extended): extended;
```

---

These functions compute the complete elliptic integral of the first kind

$$\text{EllipticK}(k) = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-k^2 t^2)}}$$

for with  $|k| < 1$ . The common function `sfc_EllipticK` returns

$$\text{EllipticK}(k) = \frac{\pi/2}{\operatorname{agm}(1, \sqrt{1-k^2})},$$

where the AGM is computed in-line.

### Complementary complete integral of the 1st kind EllipticCK

---

```
function EllipticCK(k: double): double;  
function EllipticCKx(k: extended): extended;
```

---

These functions compute the complementary complete elliptic integral of the first kind

$$\text{EllipticCK}(k) = \text{EllipticK}(k_c) = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-k_c^2 t^2)}}$$

with  $|k| < 1$ . The common function `sfc.EllipticCK` returns

$$\text{EllipticCK}(k) = \frac{\pi/2}{\text{agm}(1, |k|)}.$$

### Complete integral of the 2nd kind EllipticEC

---

```
function EllipticEC(k: double): double;  
function EllipticECx(k: extended): extended;
```

---

These functions compute the complete elliptic integral of the second kind

$$\text{EllipticEC}(k) = \int_0^1 \frac{\sqrt{1-k^2 t^2}}{\sqrt{1-t^2}} dt$$

for  $|k| \leq 1$ . The common function `sfc.EllipticEC` returns

$$E(k) = \text{cel2}(k_c, 1, k_c^2)$$

with  $k_c = \sqrt{1-k^2}$ , c.f. Bulirsch[10, p.80].

### Complementary complete integral of the 2nd kind EllipticCE

---

```
function EllipticCE(k: double): double;  
function EllipticCEx(k: extended): extended;
```

---

These functions compute the complementary complete elliptic integral of the second kind

$$\text{EllipticCE}(k) = \text{EllipticEC}(k_c) = \int_0^1 \frac{\sqrt{1-k_c^2 t^2}}{\sqrt{1-t^2}} dt.$$

The common function `sfc.EllipticCE` returns

$$\text{EllipticCE} = \text{cel2}(|k|, 1, k^2).$$

### Complete integral of the 3rd kind EllipticPiC

---

```
function EllipticPiC(nu, k: double): double;  
function EllipticPiCx(nu, k: extended): extended;
```

---

These functions compute the value of complete elliptic integral of the third kind  $\text{EllipticPiC}(\nu, k)$  with  $|k| < 1$ ,  $\nu \neq 1$

$$\text{EllipticPiC}(\nu, k) = \int_0^1 \frac{dt}{(1-\nu t^2)\sqrt{(1-t^2)(1-k^2 t^2)}}$$

The common function `sfc.EllipticPiC` returns

$$\text{EllipticPiC}(\nu, k) = \text{cel}(k_c, 1-\nu, 1, 1)$$

with  $k_c = \sqrt{1-k^2}$ , c.f. Bulirsch[11, 1.2.2].

### Complementary complete integral of the 3rd kind `EllipticCPi`

---

```
function EllipticCPi(nu,k: double): double;  
function EllipticCPix(nu,k: extended): extended;
```

---

These functions compute the complementary complete elliptic integral of the third kind with  $|k| \neq 0$ ,  $\nu \neq 1$

$$\text{EllipticCPi}(k) = \text{EllipticPiC}(k_c) = \int_0^1 \frac{dt}{(1-\nu t^2)\sqrt{(1-t^2)(1-k_c^2 t^2)}}.$$

The common function `sfc.EllipticCPi` returns

$$\text{EllipticCPi}(\nu, k) = \text{cel}(k, 1-\nu, 1, 1)$$

### Incomplete integral of the 1st kind `EllipticF`

---

```
function EllipticF(z,k: double): double;  
function EllipticFx(z,k: extended): extended;
```

---

These functions compute the incomplete elliptic integral of the first kind with  $|z| \leq 1$ ,  $|kz| \leq 1$

$$\text{EllipticF}(z, k) = \int_0^z \frac{dt}{\sqrt{(1-t^2)(1-k^2 t^2)}}$$

The common function `sfc.EllipticF` returns  $\arcsin z$  if  $k = 0$ , and

$$\text{EllipticF}(z, k) = \text{el1}\left(\frac{z}{\sqrt{1-z^2}}, \sqrt{1-k^2}\right)$$

for  $|k| \leq 1$ . Otherwise the reciprocal-modulus transformation [30, 19.7.4] is used with a recursive call:  $\text{EllipticF}(z, k) = \text{EllipticF}(zk, 1/k)/k$ .

### Incomplete integral of the 2nd kind `EllipticE`

---

```
function EllipticE(z,k: double): double;  
function EllipticEx(z,k: extended): extended;
```

---

These functions compute the incomplete elliptic integral of the first kind with  $|z| \leq 1$ ,  $|kz| \leq 1$

$$\text{EllipticE}(z, k) = \int_0^z \frac{\sqrt{1-k^2 t^2}}{\sqrt{1-t^2}} dt.$$

The common function `sfc.EllipticE` returns  $\arcsin z$  if  $k = 0$ ,  $z$  if  $|k| = 1$ . For  $|k| < 1$  the integral is computed as

$$\text{EllipticE}(z, k) = \text{el2}\left(\frac{z}{\sqrt{1-z^2}}, k_c, 1, k_c^2\right)$$

with  $k_c = \sqrt{1-k^2}$ , c.f. Bulirsch[10] p.80, and for  $|k| > 1$  the reciprocal-modulus transformation [30, 19.7.4] is used with a call to `el2(.,.,1,0)`.

### Incomplete integral of the 3rd kind EllipticPi

---

```
function EllipticPi(z,nu,k: double): double;  
function EllipticPix(z,nu,k: extended): extended;
```

---

These functions return the incomplete elliptic integral of the third kind with  $|z| \leq 1$ ,  $|kz| \leq 1$

$$\text{EllipticPi}(z, k) = \int_0^z \frac{dt}{(1 - \nu t^2) \sqrt{(1 - t^2)(1 - k^2 t^2)}}$$

The common function `sfc_EllipticPi` computes the integral as a substitution of Carlson integrals [12, 4.18] for `el3` in the formula (1.1.3) of Bulirsch[11].

### 3.2.10 Heuman's Lambda function

---

```
function heuman_lambda(phi,k: double): double;  
function heuman_lambdax(phi,k: extended): extended;
```

---

These functions compute Heuman's Lambda function for  $|k| \leq 1$  (c.f. Abramowitz and Stegun [30, 17.4.39/17.4.40], where a slightly different notation is used)

$$\begin{aligned} \Lambda_0(\varphi, k) &= \frac{F(\varphi, k')}{K(k')} + \frac{2}{\pi} K(k) Z(\varphi, k') \\ &= \frac{2}{\pi} (K(k) E(\varphi, k') - (K(k) - E(k)) F(\varphi, k')) \end{aligned}$$

The common function `sfc_hlambda` returns zero for  $\varphi = 0$  and  $2\varphi/\pi$  for  $|k| = 1$ . Otherwise  $\varphi$  is reduced modulo  $\pi$  as  $\varphi = \varphi' + n\pi$  with  $|\varphi'| \leq \pi/2$ . Since the Jacobi Zeta function  $Z(\varphi, k)$  has period  $\pi$  and

$$F(\varphi + n\pi, k') = 2nK(k') + F(\varphi, k'),$$

Heuman's function  $\Lambda_0$  is quasi-periodic with

$$\Lambda_0(\varphi + n\pi, k) = 2n + \Lambda_0(\varphi, k).$$

If  $|\varphi'| = \pi/2$ , i.e.  $\varphi$  is an odd multiple  $m$  of  $\pi/2$ , then  $\Lambda_0 = m$ . For  $|\varphi'| < \pi/2$  the result is calculated with a single call to the `cel` function, see Bulirsch [11, 1.2.3]:

$$\Lambda_0(\varphi, k) = 2n + \frac{2}{\pi} \sqrt{p} \sin \varphi' \text{cel}(k_c, p, 1, k_c^2), \quad p = 1 + k^2 \tan^2 \varphi'.$$

### 3.2.11 Jacobi Zeta function

---

```
function jacobi_zeta(phi,k: double): double;  
function jacobi_zetax(phi,k: extended): extended;
```

---

These functions return the Jacobi Zeta function for  $|k| \leq 1$

$$Z(\varphi, k) = E(\varphi, k) - \frac{E(k)}{K(k)} F(\varphi, k).$$

Zeta is periodic<sup>7</sup>  $Z(\varphi + \pi, k) = Z(\varphi, k)$ ,  $Z(\pi/2, k) = 0$ , and for  $|k| = 1$  we have<sup>8</sup>

$$Z(\varphi, k) = \sin \varphi, \quad \text{for } |k| = 1, |\varphi| < \pi/2.$$

---

<sup>7</sup> c.f. <http://functions.wolfram.com/08.07.04.0004.01>

<sup>8</sup> c.f. <http://functions.wolfram.com/08.07.03.0002.01>

The common function `sfc_jzeta` handles this special case, and for  $|k| < 1$  it uses a variation of Bulirsch's cel algorithm together with his formulas [11, 1.2.4.a] and

$$Z(\varphi, k) = k^2 \frac{\sin \varphi \cos \varphi}{K(k)} \operatorname{cel}(k_c, p, 0, \sqrt{p}), \quad p = \cos^2 \varphi + k_c^2 \sin^2 \varphi.$$

### 3.2.12 Elliptic modulus

---

```
function EllipticModulus(q: double): double;
function EllipticModulusx(q: extended): extended;
```

---

These routines return the elliptic modulus  $k(q)$  as a function of the nome  $|q| \leq 1$ . The modulus  $k$  is often used as argument of elliptic integrals and Jacobi elliptic functions, the nome  $q$  is used with Jacobi Theta functions.  $k(q)$  is explicitly given by

$$k(q) = \frac{\theta_2(q)^2}{\theta_3(q)^2}.$$

For  $|q| \leq 0.125$  the common function `sfc_ellmod` uses a Chebyshev approximation for  $k(q)/(4q^{1/2})$  calculated with Maple. For  $|q| \geq 0.8125$  the result is 1.0 accurate to extended precision, otherwise  $k$  is computed with the Theta functions.

### 3.2.13 Elliptic nome

---

```
function EllipticNome(k: double): double;
function EllipticNomex(k: extended): extended;
```

---

These routines return the elliptic nome  $q(k)$  as a function of the modulus  $|k| < 1$ :

$$q(k) = \exp\left(-\pi \frac{K'(k)}{K(k)}\right)$$

For  $|k| \leq 0.125$  the common function `sfc_ellnome` uses a Chebyshev approximation calculated with Maple, else if  $|k| < 0.99999999$  the above formula is implemented, where  $K'/K$  is computed with two in-line AGM iterations. Otherwise the result is<sup>9</sup>

$$q(k) = \exp\left(\frac{\pi^2}{r}\right) \left(1 - \frac{t\pi^2}{2r^2} \left(1 + \frac{13}{32}t\right)\right) \quad \text{with } t = 1 - k^2, r = \ln(t/16).$$

### 3.2.14 Jacobi amplitude

---

```
function jacobi_am(x, k: double): double;
function jacobi_amx(x, k: extended): extended;
```

---

The Jacobi amplitude function  $\operatorname{am}(x, k)$  for a given modulus  $k$  is the inverse function of Legendre's elliptic function of the first kind:  $\operatorname{am}(F(x, k)) = x$ . When  $|k| < 1$ ,  $\operatorname{am}(x, k)$  is a monotone quasi-periodic function [30, 22.16.2]

$$\operatorname{am}(x + 2K(k), k) = \operatorname{am}(x, k) + \pi,$$

with the special case  $\operatorname{am}(x, 0) = x$ . When  $|k| > 1$ ,  $\operatorname{am}(x, k)$  is periodic with period  $4K(1/k)/k$ , and if  $|k| = 1$ , then it is equal to the Gudermannian function  $\operatorname{am}(x, \pm 1) = \operatorname{gd}(x)$ .

---

<sup>9</sup> C.f. <http://functions.wolfram.com/09.53.06.0003.01>: **AMath** uses only the most significant terms of the given expansion.

The common function `sfc_jam` handles the special cases, and for  $|k| > 1$  it calls `sfc_sncndn` and returns<sup>10</sup>

$$\operatorname{am}(x, k) = \operatorname{arctan2}(\operatorname{sn}(x, k), \operatorname{cn}(x, k)).$$

If  $k < 1$  then  $x$  is decomposed as  $x = 2nK + z$ , with  $K = K(k)$ ,  $|z| \leq K$ , and the result is computed from the quasi-periodicity

$$\operatorname{am}(x, k) = \operatorname{arctan2}(\operatorname{sn}(z, k), \operatorname{cn}(z, k)) + n\pi.$$

### 3.2.15 Jacobi elliptic functions

---

```
procedure sncndn(x, mc: double; var sn, cn, dn: double);
procedure sncndnx(x, mc: extended; var sn, cn, dn: extended);
```

---

These procedures return the Jacobi elliptic functions `sn`, `cn`, `dn` for argument  $x$  and complementary parameter  $m_c$ , the implementation is based on Bulirsch's [10] Algorithm 5 and his ALGOL procedure `sncndn`. A convenient implicit definition of the functions is

$$x = \int_0^{\operatorname{sn}} \frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}}, \quad \operatorname{sn}^2 + \operatorname{cn}^2 = 1, \quad k^2 \operatorname{sn}^2 + \operatorname{cn}^2 = 1$$

with  $k^2 = 1 - m_c$ . The common function `sfc_sncndn` computes all three function simultaneously by Gauß/AGM transformation; parameters  $m_c < 0$  are made positive by Jacobi's real transformation [1, 16.11].

There are a lot of equivalent definitions of the Jacobi elliptic functions, e.g with the Jacobi amplitude function (see e.g. Olver et al. [30], 22.16.11/12)

$$\operatorname{sn}(x, k) = \sin(\operatorname{am}(x, k)), \quad \operatorname{cn}(x, k) = \cos(\operatorname{am}(x, k)),$$

or with Jacobi theta functions (c.f. [30, 22.2]).

#### Jacobi elliptic function `sn`

---

```
function jacobi_sn(x, k: double): double;
function jacobi_snx(x, k: extended): extended;
```

---

These functions return the Jacobi elliptic function  $\operatorname{sn}(x, k)$ . The common function `sfc_jacobi_sn` calls `sfc_sncndn` with  $m_c = (1 - k)(1 + k)$ .

#### Jacobi elliptic function `cn`

---

```
function jacobi_cn(x, k: double): double;
function jacobi_cnx(x, k: extended): extended;
```

---

These functions return the Jacobi elliptic function  $\operatorname{cn}(x, k)$ . The common function `sfc_jacobi_cn` calls `sfc_sncndn` with  $m_c = (1 - k)(1 + k)$ .

---

<sup>10</sup>The calculation with `arctan2` is slightly more accurate than `arcsin(sn)`, and because `sn` and `cn` are computed simultaneously there is no time penalty.

### Jacobi elliptic function dn

---

```
function jacobi_dn(x,k: double): double;  
function jacobi_dnx(x,k: extended): extended;
```

---

These functions return the Jacobi elliptic function  $\text{dn}(x, k)$ . The common function `sfc_jacobi_dn` calls `sfc_sncndn` with  $m_c = (1 - k)(1 + k)$ .

### 3.2.16 Inverse Jacobi elliptic functions

#### Inverse Jacobi elliptic function arcn

---

```
function jacobi_arcsn(x,k: double): double;  
function jacobi_arcsnx(x,k: extended): extended;
```

---

These functions compute the inverse Jacobi elliptic function  $\text{arcsn}(x, k)$  for  $|x| \leq 1$  and  $|kx| \leq 1$ . The common function `sfc_jacobi_arcsn` returns

$$\text{arcsn}(x, k) = F(\arcsin(x), k).$$

#### Inverse Jacobi elliptic function arccn

---

```
function jacobi_arccn(x,k: double): double;  
function jacobi_arccnx(x,k: extended): extended;
```

---

These functions compute the inverse Jacobi elliptic function  $\text{arccn}(x, k)$  for  $x \leq 1$  and  $|kx| \leq 1$ . The common function `sfc_jacobi_arccn` returns

$$\text{arccn}(x, k) = F(\arccos(x), k).$$

#### Inverse Jacobi elliptic function arcdn

---

```
function jacobi_arcdn(x,k: double): double;  
function jacobi_arcdnx(x,k: extended): extended;
```

---

These functions compute the inverse Jacobi elliptic function  $\text{arcdn}(x, k)$  for  $|x| \leq 1$  and  $k^2 + x^2 > 1$ . The common function `sfc_jacobi_arcdn` returns

$$\text{arcdn}(x, k) = F\left(\arcsin\left(\frac{\sqrt{1-x^2}}{k}\right), k\right).$$

### 3.2.17 Jacobi theta functions

---

```
function jacobi_theta(n: integer; x,q: double): double;  
function jacobi_thetax(n: integer; x,q: extended): extended;
```

---

These functions return the Jacobi theta functions  $\theta_n(x, q)$  for  $n = 1..4$  and  $|q| < 1$ . For all real  $x$  they are defined by the series (see Abramowitz and Stegun [1, 16.27] or

the NIST handbook [30, 20.2]):

$$\begin{aligned}\theta_1(x, q) &= 2q^{1/4} \sum_{k=0}^{\infty} (-1)^k q^{k(k+1)} \sin(2k+1)x \\ \theta_2(x, q) &= 2q^{1/4} \sum_{k=0}^{\infty} q^{k(k+1)} \cos(2k+1)x \\ \theta_3(x, q) &= 1 + 2 \sum_{k=1}^{\infty} q^{k^2} \cos 2kx \\ \theta_4(x, q) &= 1 + 2 \sum_{k=1}^{\infty} (-1)^k q^{k^2} \cos 2kx\end{aligned}$$

With respect to  $x$  the functions are periodic with period  $2\pi$  or  $\pi$ :

$$\theta_{1,2}(x + \pi, q) = -\theta_{1,2}(x, q), \quad \theta_{3,4}(x + \pi, q) = \theta_{3,4}(x, q).$$

For  $|q| \leq 0.25$  the common function `sfc_jtheta` computes the  $\theta_n(x, q)$  with these series, otherwise the "Transformations of Lattice Parameter" from Olver et al. [30, 20.7(viii)] are used. For non-zero  $x$  these transformations become somewhat complicated, explicit expressions can be found at the Wolfram function site<sup>11</sup>

$$\theta_1(z, q) = -\frac{2\sqrt{\pi}}{\sqrt{-\ln q}} e^{\frac{4z^2 + \pi^2}{4\ln q}} \sum_{k=0}^{\infty} (-1)^k e^{\frac{k(k+1)\pi^2}{\ln q}} \sinh\left(\frac{(2k+1)\pi z}{\ln q}\right)$$

and similar for the other  $\theta_n$ . The transformed series used by **AMath** are:

$$\begin{aligned}\theta_1(x, q) &= 2\left(-\frac{\pi}{p}\right)^{1/2} e^{(x^2 + \pi^2/4)/p} \sum_{k=0}^{\infty} (-1)^k e^{k(k+1)y} \sinh((2k+1)w) \\ \theta_2(x, q) &= \left(-\frac{\pi}{p}\right)^{1/2} e^{x^2/p} \left(1 + 2 \sum_{k=1}^{\infty} (-1)^k e^{k^2 y} \cosh(2kw)\right) \\ \theta_3(x, q) &= \left(-\frac{\pi}{p}\right)^{1/2} e^{x^2/p} \left(1 + 2 \sum_{k=1}^{\infty} e^{k^2 y} \cosh(2kw)\right) \\ \theta_4(x, q) &= 2\left(-\frac{\pi}{p}\right)^{1/2} e^{(x^2 + \pi^2/4)/p} \sum_{k=0}^{\infty} e^{k(k+1)y} \cosh((2k+1)w),\end{aligned}$$

with the definitions  $p = \ln q$ ,  $y = \pi^2 / \ln q$ , and  $w = \pi|x / \ln q|$ . In the actual calculations only three terms of the sums are used (this requires  $q \gtrsim 0.25$ ) and precautions are taken to avoid overflow/underflow of the separate exp and sinh/cosh terms.

### 3.2.18 Jacobi theta functions at zero

This subsection documents how the Jacobi theta functions  $\theta_i(q) = \theta_i(0, q)$  for  $i = 2, 3, 4$  are computed in **AMath**; obviously  $\theta_1(q) = 0$  for all  $q$ , and therefore  $\theta'_1(q) = \partial\theta_1(z, q)/\partial z|_{z=0}$  is provided.

As described in the NIST handbook [30, 20.7(viii)] "Transformations of Lattice Parameter", for the theta  $q$  series can be restricted to very small  $q$ , theoretically to  $0 \leq q \leq \exp(-\pi) = 0.0432139\dots$  for  $z = 0$ ; see the example in [30, 20.14] "Methods of Computation" for  $\theta_3(0.9)$ .

<sup>11</sup><http://functions.wolfram.com/09.01.06.0042.01>

In practice **AMath** uses the following transformations for  $q \geq 0.1$ :

$$\begin{aligned}\theta_2(q) &= (-\pi/\ln q)^{1/2} \theta_4(\exp(\pi^2/\ln q)) \\ \theta_3(q) &= (-\pi/\ln q)^{1/2} \theta_3(\exp(\pi^2/\ln q)) \\ \theta_4(q) &= (-\pi/\ln q)^{1/2} \theta_2(\exp(\pi^2/\ln q))\end{aligned}$$

Additionally for  $q < 0$  the relation  $\theta_4(q) = \theta_3(-q)$  is applied.

The transformation formula for  $\theta_1$  from [30, 20.7.30] implies, that  $\theta'_1(q)$  has the functional equation

$$\theta'_1(q) = (-\pi/\ln q)^{3/2} \theta'_1(\exp(\pi^2/\ln q)).$$

For the actual calculations the following  $q$  series are used internally for small  $q$

$$\begin{aligned}\theta_1^{(s)}(q) &= 2q^{1/4} \sum_{n=0}^{\infty} (-1)^n (2n+1) q^{n(n+1)} \\ \theta_2^{(s)}(q) &= 2q^{1/4} \sum_{n=0}^{\infty} q^{n(n+1)} \\ \theta_3^{(s)}(q) &= 1 + 2 \sum_{n=1}^{\infty} q^{n^2}\end{aligned}$$

Although these series are convergent for all  $|q| < 1$  and are often used to compute the functions over the whole  $q$  range, the convergence of the transformed functions is much better for  $q \gtrsim 0.1$ .

### Jacobi theta1p(q)

---

```
function theta1p(q: double): double;
function theta1px(q: extended): extended;
```

---

These functions return  $\theta'_1(q) = \partial\theta_1(z, q)/\partial z$  at  $z = 0$  for  $0 \leq q < 1$ . The common function `sfc.theta1p` uses the  $q$  series to compute

$$\theta'_1(q) = \begin{cases} \theta_1^{(s)}(q) & q \leq 0.1, \\ (-\pi/\ln q)^{3/2} \theta_1^{(s)}(\exp(\pi^2/\ln q)) & q > 0.1. \end{cases}$$

For  $q \geq 0.7$  only one term of the series is needed, but special care is taken to avoid underflow of partial results.

### Jacobi theta2(q)

---

```
function theta2(q: double): double;
function theta2x(q: extended): extended;
```

---

These functions return  $\theta_2(q) = \theta_2(0, q)$  for  $0 \leq q < 1$ . The common function `sfc.theta2` uses the  $q$  series to compute

$$\theta_2(q) = \begin{cases} \theta_2^{(s)}(q) & q \leq 0.1, \\ (-\pi/\ln q)^{1/2} \theta_3^{(s)}(-\exp(\pi^2/\ln q)) & q > 0.1. \end{cases}$$

### Jacobi theta3(q)

---

```
function theta3(q: double): double;  
function theta3x(q: extended): extended;
```

---

These functions return  $\theta_3(q) = \theta_3(0, q)$  for  $-1 < q < 1$ . The common function `sfc_theta3` uses the  $q$  series to compute

$$\theta_3(q) = \begin{cases} \theta_4(-q) & q < 0, \\ \theta_3^{(s)}(q) & 0 \leq q \leq 0.1, \\ (-\pi/\ln q)^{1/2} \theta_3^{(s)}(\exp(\pi^2/\ln q)) & q > 0.1. \end{cases}$$

### Jacobi theta4(q)

---

```
function theta4(q: double): double;  
function theta4x(q: extended): extended;
```

---

These functions return  $\theta_4(q) = \theta_4(0, q)$  for  $-1 < q < 1$ . The common function `sfc_theta4` uses the  $q$  series to compute

$$\theta_4(q) = \begin{cases} \theta_3(-q) & q < 0, \\ \theta_3^{(s)}(-q) & 0 \leq q \leq 0.1, \\ (-\pi/\ln q)^{1/2} \theta_2^{(s)}(\exp(\pi^2/\ln q)) & q > 0.1. \end{cases}$$

For  $q \geq 0.7$  only one term of the series is needed, but special care is taken to avoid underflow of partial results.

### 3.3 Error function and related

The Pascal unit **sferf** implements the common code for the error and related functions.

#### 3.3.1 Dawson's integral

---

```
function dawson(x: double): double;  
function dawsonx(x: extended): extended;
```

---

These functions return the value of Dawson's integral for  $x$ . The Dawson integral is defined by

$$F(x) = e^{-x^2} \int_0^x e^{t^2} dt$$

The common function **sfc.dawson** uses three Chebyshev approximations and is based on the routines from Fullerton [14, 20] (file ddaws.f).

#### 3.3.2 Error function erf

---

```
function erf(x: double): double;  
function erfxx(x: extended): extended;
```

---

These functions compute the error function

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

For  $|x| \leq 1$  two rational approximations from Cephes [7] are used (file ldouble/ndtrl.c), for  $|x| > 1$  the result is  $\operatorname{erf}(x) = 1 - \operatorname{erfc}(x)$ .

#### 3.3.3 Complementary error function erfc

---

```
function erfc(x: double): double;  
function erfcxx(x: extended): extended;
```

---

These functions compute the complementary error function

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt.$$

For  $|x| < 1$  the result is  $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$  and for  $|x| \geq 1$  two rational approximations from Cephes [7] are used (file ldouble/ndtrl.c).

#### 3.3.4 Imaginary error function erfi

---

```
function erfi(x: double): double;  
function erfiox(x: extended): extended;
```

---

These functions return the imaginary error function

$$\operatorname{erfi}(x) = \frac{1}{i} \operatorname{erf}(ix).$$

**erfi** is computed using the Dawson integral as

$$\operatorname{erfi}(x) = \frac{2}{\sqrt{\pi}} e^{x^2} \operatorname{dawson}(x),$$

where  $e^{x^2}$  is evaluated with the **AMath** function **expx2** in order to minimize error amplification for large  $x$ .

### 3.3.5 Exponentially scaled complementary error function

---

```
function erfce(x: double): double;  
function erfcex(x: extended): extended;
```

---

These functions return  $\text{erfce}(x) = \text{erfc}(x)(\exp x^2)$ , the exponentially scaled complementary error function. If  $|x| \leq 1$  the result is  $(1 - \text{erf } x)\exp x^2$ , and (accurate to extended precision)  $2 \exp x^2$  if  $x < -6.75$ . For  $1 < x \leq 128$  and  $-6.75 \leq x < -1$  two rational approximations from Cephes [7] are used (file `ldouble/ndtrl.c`), for  $x > 128$  the function is computed with an asymptotic expansion ( $y = 1/(2x^2)$ ):

$$\text{erfce}(x) = (1 - y + 3y^2 - 15y^3 + 105y^4 - 945y^5 + O(y^6))/(x\sqrt{\pi})$$

### 3.3.6 Inverse function of erf

---

```
function erf_inv(x: double): double;  
function erf_invx(x: extended): extended;
```

---

These functions return the functional inverse of erf, i.e.

$$\text{erf}(\text{erf\_inv}(x)) = x, \quad -1 < x < 1.$$

The common function `sfc_erf_inv` is based on the Boost [19] routine `erf_inv` in `erf_inv.hpp`; it uses seven rational approximations.

### 3.3.7 Inverse function of erfc

---

```
function erfc_inv(x: double): double;  
function erfc_invx(x: extended): extended;
```

---

These functions return the functional inverse of erfc, i.e.

$$\text{erfc}(\text{erfc\_inv}(x)) = x, \quad 0 < x < 2.$$

The common function `sfc_erfc_inv` is based on the Boost [19] routine `erfc_inv` in `erf_inv.hpp`; it uses seven rational approximations.

### 3.3.8 expint3

---

```
function expint3(x: double): double;  
function expint3x(x: extended): extended;
```

---

These functions return the integral

$$\int_0^x e^{-t^3} dt, \quad (x \geq 0).$$

The calculation is based on the MISCFUN [22] routine EXP3; it uses two Chebyshev approximations. The function is related to the incomplete Gamma function:  $\text{expint3}(x) = \Gamma(4/3) - \Gamma(1/3, x^3)/3$ ; it is listed here, because a normalised version is tabulated in the error function chapter of the Handbook of Mathematical Functions [1] (Table 7.6).

### 3.3.9 Goodwin-Staton integral

---

```
function gsi(x: double): double;  
function gsix(x: extended): extended;
```

---

These functions return the Goodwin-Staton integral

$$G(x) = \int_0^x \frac{e^{-t^2}}{t+x} dt$$

for  $x > 0$ . The common function `sfc.gsi` is based on MacLeod's MISCFUN [22] routine GOODST; it uses two Chebyshev approximations.

### 3.3.10 Fresnel integrals

---

```
procedure fresnel(x: double; var s,c: double);  
procedure fresnelx(x: extended; var s,c: extended);
```

---

These procedures return the Fresnel integrals

$$C(x) = \int_0^x \cos\left(\frac{1}{2}\pi t^2\right) dt,$$
$$S(x) = \int_0^x \sin\left(\frac{1}{2}\pi t^2\right) dt.$$

For  $|x| < 1.5$  the integrals are calculated with the series

$$C(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \left(\frac{\pi}{2}\right)^{2n}}{(2n)!(4n+1)} x^{4n+1},$$
$$S(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \left(\frac{\pi}{2}\right)^{2n+1}}{(2n+1)!(4n+3)} x^{4n+3}.$$

In the medium range  $1.5 \leq |x| < 1.562e6$  a continued fraction algorithm is used; c.f. Press et al.[13] (Ch. 6.9, function `frenel`).<sup>12</sup> For larger  $|x|$  the values are given by the asymptotic expressions:

$$C(x) = \frac{1}{2} - \frac{\cos\left(\frac{1}{2}\pi x^2\right)}{\pi x},$$
$$S(x) = \frac{1}{2} + \frac{\sin\left(\frac{1}{2}\pi x^2\right)}{\pi x}.$$

---

<sup>12</sup>The continued fraction evaluates  $C(x) + iS(x)$  using the complex error function, the complex arithmetic is performed in-line in procedure `fresnel_cfrac`.

## 3.4 Exponential integrals and related

The Pascal unit **sfExpInt** implements the common code for the exponential integrals and related functions.

### 3.4.1 Hyperbolic cosine integral Chi

---

```
function chi(x: double): double;  
function chix(x: extended): extended;
```

---

These functions return the hyperbolic cosine integral for  $x > 0$

$$\text{Chi}(x) = \gamma + \ln(x) + \int_0^x \frac{\cosh(t) - 1}{t} dt,$$

and  $\text{Chi}(x) = \text{Chi}(-x)$  for  $x < 0$ . In the common function **sfc.chi** the integral is calculated using the relation [30, 6.5.4]

$$\text{Chi}(x) = \frac{1}{2}(\text{Ei}(x) - \text{E}_1(x)), \quad (x > 0).$$

### 3.4.2 Cosine integral Ci

---

```
function ci(x: double): double;  
function cix(x: extended): extended;
```

---

These functions return the cosine integral for  $x > 0$

$$\begin{aligned} \text{Ci}(x) &= - \int_x^\infty \frac{\cos(t)}{t} dt \\ &= \gamma + \ln(x) + \int_0^x \frac{\cos(t) - 1}{t} dt \\ &= \gamma + \ln(x) - \text{Cin}(x) \end{aligned}$$

and  $\text{Ci}(x) = \text{Ci}(-x)$  for  $x < 0$ . The calculation is based on the routines of Fullerton [20] (files **dcin.f** and **d9sifg.f**). A Chebyshev approximation is used for  $0 < x \leq 4$  and for  $x > 4$  the function is given by

$$\text{Ci}(x) = f(x) \sin(x) - g(x) \cos(x)$$

Where the auxiliary functions **f** and **g**

$$\begin{aligned} f(x) &= + \text{Ci}(x) \sin(x) - \text{si}(x) \cos(x), \\ g(x) &= - \text{Ci}(x) \cos(x) - \text{si}(x) \sin(x) \end{aligned}$$

are computed with five Chebyshev approximations (for details see procedure **auxfg** in unit **sfExpInt**), **si(x)** is the shifted sine integral **ssi** (3.4.12).

### 3.4.3 Entire cosine integral Cin

---

```
function cin(x: double): double;  
function cinx(x: extended): extended;
```

---

These functions return the entire cosine integral

$$\text{Cin}(x) = \int_0^x \frac{1 - \cos(t)}{t} dt.$$

For  $0 \leq x \leq 4$  the function is computed with a Chebyshev approximation from Fullerton [20] (file `dcin.f`), for  $x > 4$  it returns

$$\text{Cin}(x) = (g(x) \cos(x) - f(x) \sin(x)) + \ln(x) + \gamma,$$

and for  $x < 0$  the result is  $\text{Cin}(-x)$ ; the auxiliary functions `f` and `g` are defined in the `Ci` section 3.4.2.

### 3.4.4 Entire hyperbolic cosine integral `Cinh`

---

```
function cinh(x: double): double;
function cinhx(x: extended): extended;
```

---

These functions return the entire hyperbolic cosine integral

$$\text{Cinh}(x) = \int_0^x \frac{\cosh(t) - 1}{t} dt,$$

For  $0 \leq x \leq 3$  the common function `sfc_cinh` uses a Chebyshev approximation from Fullerton [20] (file `dcinh.f`), for  $x > 3$  it returns  $\text{Chi}(x) - \ln(x) - \gamma$ , and for  $x < 0$  the result is  $\text{Cinh}(-x)$ .

### 3.4.5 Exponential integral `E1`

---

```
function e1(x: double): double;
function e1x(x: extended): extended;
```

---

These functions return the exponential integral  $E_1(x)$  for  $x \neq 0$

$$E_1(x) = \int_1^\infty \frac{e^{-xt}}{t} dt$$

For  $x > 0$  the common function `sfc_e1` is based on a Boost [19] routine from `expint.hpp`; it uses two rational approximations. For  $x < 0$  the integral is calculated as  $E_1(x) = -\text{Ei}(-x)$ .

### 3.4.6 Exponential integral `Ei`

---

```
function ei(x: double): double;
function eix(x: extended): extended;
```

---

These functions return the exponential integral  $\text{Ei}(x)$  for  $x \neq 0$

$$\text{Ei}(x) = -\text{PV} \int_{-x}^\infty \frac{e^{-t}}{t} dt = \text{PV} \int_{-\infty}^x \frac{e^t}{t} dt.$$

For  $x > 0$  the common function `sfc_ei` uses rational approximations from Boost [19] (`expint.hpp`). The Boost code preserves the root at 0.37250741... but is suboptimal for very small  $x$ . If  $0 < x < \text{eps}_x$ , the approximation  $\text{Ei}(x) = \gamma + \ln(x)$  is used in `AMath`. For  $x < 0$  the integral is calculated as  $\text{Ei}(x) = -E_1(-x)$ .

### 3.4.7 Entire exponential integral Ein

---

```
function ein(x: double): double;
function einx(x: extended): extended;
```

---

These functions return the entire exponential integral  $\text{Ein}(x)$

$$\text{Ein}(x) = \int_0^x \frac{1 - e^{-t}}{t} dt.$$

If  $|x| < 1/4$  the common function `sfc.ein` uses a Chebyshev approximation calculated with Maple V from the series [30, 6.6.4]:

$$\text{Ein}(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^n}{n!n},$$

otherwise the function is computed as

$$\text{Ein}(x) = \begin{cases} \ln(x) + \gamma & x \geq 45 \\ \ln(x) + \gamma + E_1(x) & 1/4 < x < 45 \\ \ln(-x) + \gamma - \text{Ei}(-x) & -50 < x < 1/4 \\ -\text{Ei}(-x) & x \leq -50 \end{cases}$$

The first and last cases are simplifications of the complete cases and are accurate to extended precision.

### 3.4.8 Exponential integrals $E_n$

---

```
function en(n: longint; x: double): double;
function enx(n: longint; x: extended): extended;
```

---

These functions return the exponential integrals

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt \quad (n \geq 0),$$

with  $x > 0$  for  $n < 2$ , and  $x \geq 0$  for  $n \geq 2$ . Special values are

$$E_n(0) = \frac{1}{n-1} \quad (n > 1),$$

$$E_0(x) = \frac{e^{-x}}{x}.$$

These special cases and  $E_1(x)$  are handled separately in the common function `sfc.en`. If  $n > 10^5$  the approximation (5.1.52) from Abramowitz and Stegun [1] is accurate to extended precision:

$$E_n(x) = \frac{e^{-x}}{x+n} \left( 1 + \frac{n}{(x+n)^2} + \frac{n(n-2x)}{(x+n)^4} + \frac{n(6x^2 - 8nx + n^2)}{(x+n)^6} + R(n, x) \right)$$

$$-0.36n^{-4} \leq R(n, x) \leq \left( 1 + \frac{1}{x+n-1} \right) n^{-4} \quad (x > 0)$$

For  $x > 1$  the continued fraction [1, 5.1.22]

$$E_n(x) = e^{-x} \left( \frac{1}{x+} \frac{n}{1+} \frac{1}{x+} \frac{n+1}{1+} \frac{2}{x+} \dots \right)$$

is used, c.f. the Cephes [7] function expn. For  $x \leq 1$  the convergence of the continued fraction is not fast enough and the integral is computed with the series expansion (5.1.12) from [1]

$$E_n(x) = \frac{(-x)^{n-1}}{(n-1)!} (-\ln x + \psi(n)) - \sum_{\substack{m=0 \\ m \neq n-1}}^{\infty} \frac{(-x)^m}{(m-n+1)m!}.$$

### 3.4.9 Logarithmic integral li

---

```
function li(x: double): double;
function lix(x: extended): extended;
```

---

These functions return the logarithmic integral  $\text{li}(x)$  for  $x \geq 0$

$$\text{li}(x) = \text{PV} \int_0^x \frac{1}{\ln(t)} dt \quad (x \neq 1).$$

If  $x \neq 0$ , the common function `sfc_li` computes  $\text{li}(x) = \text{Ei}(\ln(x))$ , where special care is taken to avoid inaccuracies for  $x > 6$ .

### 3.4.10 Hyperbolic sine integral Shi

---

```
function shi(x: double): double;
function shix(x: extended): extended;
```

---

These functions return the hyperbolic sine integral

$$\text{Shi}(x) = \int_0^x \frac{\sinh(t)}{t} dt.$$

For  $|x| \leq 0.375$  the common function `sfc_shi` uses a Chebyshev approximation from Fullerton [20] (file `dshi.f`), for  $x > 0.375$  it returns

$$\text{Shi}(x) = \frac{1}{2} (\text{Ei}(x) + \text{E}_1(x)),$$

and for  $x < 0$  the result is  $-\text{Shi}(-x)$ .

### 3.4.11 Sine integral Si

---

```
function si(x: double): double;
function six(x: extended): extended;
```

---

These functions return the sine integral

$$\text{Si}(x) = \int_0^x \frac{\sin(t)}{t} dt.$$

For small  $|x| < 10^{-5}$  the first two terms  $x - x^3/18$  of the Maclaurin series give an accurate result. If  $x > 4$  the relation

$$\text{Si}(x) = \frac{\pi}{2} - f(x) \cos(x) - g(x) \sin(x)$$

is used, where the auxiliary functions `f` and `g` are defined in the `Ci` section 3.4.2. For  $x \leq 4$  the integral is computed with a Chebyshev approximation from Fullerton [20] (file `dsi.f`), and for  $x < 0$  the result is  $\text{Si}(x) = -\text{Si}(-x)$ .

### 3.4.12 Shifted sine integral si

---

```
function ssi(x: double): double;  
function ssix(x: extended): extended;
```

---

These functions return the shifted sine integral

$$\text{si}(x) = \text{Si}(x) - \frac{1}{2}\pi.$$

The evaluation is similar to that of Si, except that for  $|x - x_0| < 0.25$  a Chebyshev approximation calculated with Maple V is used, where  $x_0 = 1.92644766\dots$  is the smallest positive root of si.

## 3.5 Gamma function and related

The unit `sfGamma` implements the common code for the gamma and related functions.

### 3.5.1 Gamma functions

At first some import internal routines of the unit `sfGamma` are described, which are used by the interfaced functions.

```
function sfc_gamma_medium(x: extended): extended;
```

This function computes  $\Gamma(x)$  when  $|x| \leq 13$ ; it is based on the Cephes [7] function `gammal`. The recurrence formula  $\Gamma(x+1) = x\Gamma(x)$  is used to bring the argument into the interval  $[2, 3)$  where a rational approximation is used. Or, if during this iteration an argument `x` is in  $[-1/32, 1/32]$  the function `sfc_gaminv_small` is called.

```
function sfc_gaminv_small(x: extended): extended;
```

This function computes  $1/\Gamma(x)$  for  $|x| \leq 1/32$ . It uses two polynomial approximations c.f. Cephes [7] `gammal`.<sup>13</sup>

```
function stirf(x: extended): extended;
```

This function computes  $\Gamma(x)$  for  $x > 13$  using Stirling's formula for  $x > 1024$  and otherwise the approximation

$$(2\pi)^{1/2} x^{x-1/2} e^{-x} \left(1 + \frac{1}{x} P\left(\frac{1}{x}\right)\right)$$

where the polynomial is from Cephes [7] function `stirf`.

```
function sfc_lngcorr(x: extended): extended;
```

This function returns the  $\ln \Gamma$  correction term for  $x \geq 8$

$$\ln \Gamma(x) - ((x - 1/2) \ln(x) - x + \ln \sqrt{2\pi})$$

and is computed with a polynomial approximation in Cephes [7], function `lgaml`.

```
function lngamma_small(x, xm1, xm2: extended;  
                      useln1p: boolean): extended;
```

This function returns  $\ln \Gamma(x)$  for  $0 < x < 8$ ; the arguments `xm1 = x - 1`, and `xm2 = x - 2` are externally supplied for increased precision, `useln1p` should be true if `xm1` is supposed to be more accurate than `x - 1`. The implementation is based on the Boost [19] function `lgamma_small.hpp` and uses three rational approximations.

### Gamma function $\Gamma(x)$

---

```
function gamma(x: double): double;  
function gammax(x: extended): extended;
```

---

These functions compute the gamma function for  $x \neq 0, -1, -2, \dots$  defined by

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

and by analytic continuation if  $x < 0$ . The common function `sfc_gamma` returns `sfc_gamma_medium(x)` when  $|x| \leq 13$ , otherwise the result is calculated with the internal `stirf` function and the reflection formula if  $x < 0$

$$\Gamma(x)\Gamma(1-x) = \pi / \sin(\pi x)$$

---

<sup>13</sup> Label `small` in file `ldouble/gammal.c`

## Gamma function $\Gamma(1+x) - 1$

---

```
function gamma1pm1(x: double): double;  
function gamma1pm1x(x: extended): extended;
```

---

These functions return  $\Gamma(1+x) - 1$  accurate even for  $x \approx 0$ . The common function `sfc_gamma1pm1` returns  $-\gamma x$  if  $|x| < \text{eps}_x$ . For  $x < -0.5$  or  $x > 2$  the result is computed directly from the definition, otherwise `expm1`, `ln1p`, and `lngamma_small` are used.

## Temme's regulated gamma function $\Gamma^*(x)$

---

```
function gammastar(x: double): double;  
function gammastarx(x: extended): extended;
```

---

These functions calculate Temme's regulated function  $\Gamma^*(x)$  defined in [26] by:

$$\Gamma(x) = \sqrt{2\pi} e^{-x} x^{x-1/2} \Gamma^*(x), \quad x > 0.$$

For  $x > 8$  the common function `sfc_gstar` returns `exp(sfc_lngcorr(x))`, in the range  $1 \leq x \leq 8$  two Chebyshev expansions computed with Maple are used<sup>14</sup>, and if  $x < 1$  Temme's recursion formula is applied to make  $x > 1$ :

$$\Gamma^*(x) = e^{-1} \left( \frac{x+1}{x} \right)^{x+1/2} \Gamma^*(x+1)$$

## Logarithm of the gamma function

---

```
function lngamma(x: double): double;  
function lngammax(x: extended): extended;
```

---

These functions compute  $\ln|\Gamma(x)|$  for  $x \neq 0, -1, -2, \dots$ <sup>15</sup>. If  $x < 0$  the common function `sfc_lngamma` uses the logarithmic form of the reflection formula. For  $x > 8$  the result is

$$\ln|\Gamma(x)| = (x - 0.5) \ln x - x + \ln \sqrt{2\pi} + \text{sfc_lngcorr}(x),$$

otherwise the value `lngamma_small(x, x - 1, x - 2, false)` is returned.

## Logarithm of $\Gamma(1+x)$

---

```
function lngamma1p(x: double): double;  
function lngamma1px(x: extended): extended;
```

---

These functions compute  $\ln|\Gamma(1+x)|$  with increased accuracy for  $x \approx 0$ . For  $x < -1$  or  $x > 7$  the result is calculated as `lngamma(1+x)` otherwise the value `lngamma_small(x+1, x, x - 1, true)` is returned.

---

<sup>14</sup> The rational approximations from [26] not suitable for extended precision.

<sup>15</sup> `siggamma` can be used if the sign of  $\Gamma(x)$  is needed.

## Reciprocal gamma function

---

```
function rgamma(x: double): double;  
function rgamma(x: extended): extended;
```

---

These routines calculate the reciprocal gamma function  $1/\Gamma(x)$ , which is an entire function with simple zeros at the points  $x = 0$  and the negative integers. If  $|x| < 0.03125$  the common function `sfc_rgamma` returns `sfc_gaminv_small(x)`, for  $x \in [-0.5, 8]$  the result is computed with the `sfc_gamma_medium` function just from the definition, and otherwise

$$1/\Gamma(x) = \text{sign}(\Gamma(x)) \exp(-\ln \Gamma(x)).$$

## Sign of gamma function

---

```
function signgamma(x: double): double;  
function signgamma(x: extended): extended;
```

---

These functions return the sign of  $\Gamma(x)$ , which is  $+1$  if  $x > 0$  or if  $\lfloor x \rfloor$  is even,  $-1$  otherwise<sup>16</sup>.

### 3.5.2 Incomplete gamma functions

The incomplete gamma functions are defined as

$$\begin{aligned}\gamma(a, x) &= \int_0^x t^{a-1} e^{-t} dt & (a > 0), \\ \Gamma(a, x) &= \int_x^\infty t^{a-1} e^{-t} dt.\end{aligned}$$

They are used in **AMath** only in their normalised forms

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)} \quad Q(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

An important internal routine for the incomplete gamma code is:

```
function sfc_igprefix(a, x: extended): extended;
```

This function returns  $x^a \exp(-x)/\Gamma(a)$ , which is the most sensible single term controlling the rounding errors occurring in computation of the incomplete gamma functions, see [26, 3.3]. It is implemented with the Lanczos sum<sup>17</sup> from Boost[19] file `gamma.hpp`, function `regularised_gamma_prefix`.

## Normalised incomplete gamma functions

---

```
procedure incgamma(a, x: double; var p, q: double);  
procedure incgamma(a, x: extended; var p, q: extended);
```

---

These procedures simultaneously compute the normalised incomplete gamma functions

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

---

<sup>16</sup> The sign is meaningless for  $x = 0$  or negative integers.

<sup>17</sup> Not for TP 5.0!

for  $a \geq 0$  and  $x \geq 0$ . The implementation of the common function `sfc.incgamma`<sup>18</sup> is heavily based on Temme's paper and algorithms [26], except in the asymptotic case  $a \sim x$  for large  $a$ , where the algorithm from Boost[19] `igamma_large.hpp` is used.<sup>19</sup>

---

```
function igammap(a,x: double): double;
function igammapx(a,x: extended): extended;
```

---

These functions return the normalised incomplete gamma function  $P(a, x)$

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

for  $a \geq 0$  and  $x \geq 0$  using a call to `sfc.incgamma`. If  $a = x = 0$  then  $P = 0.5$ , otherwise  $P = 0$  if  $x = 0$  and  $P = 1$  if  $a = 0$ .

---

```
function igammaq(a,x: double): double;
function igammaqx(a,x: extended): extended;
```

---

These functions return the normalised incomplete gamma function  $Q(a, x)$

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

for  $a \geq 0$  and  $x \geq 0$  using a call to `sfc.incgamma`. If  $a = x = 0$  then  $Q = 0.5$ , otherwise  $Q = 1$  if  $x = 0$  and  $Q = 0$  if  $a = 0$ .

### Inverse normalised incomplete gamma functions

This section describes the procedures and functions for the functional inverses of the normalised incomplete gamma functions; in **AMath** they are used for inverting the gamma and chi-square distributions. The classical reference is the article and FORTRAN code by A.R. Didonato and A.H. Morris [27].

---

```
procedure incgamma_inv(a,p,q: double; var x: double;
                    var ierr: integer);
procedure incgamma_invx(a,p,q: extended; var x: extended;
                    var ierr: integer);
```

---

These procedures return the inverse normalised incomplete gamma function, i.e. they calculate  $x$  with  $P(a, x) = p$  and  $Q(a, x) = q$ . The input parameters are  $a > 0$ ,  $p \geq 0$ ,  $q \geq 0$ , and  $p + q = 1$ . The output parameter `ierr` is  $\geq 0$  for success, and  $< 0$  for input errors or iterations failures:

$\geq 0$	iteration count
$= -2$	if $a \leq 0$
$= -4$	if $p < 0$ , $q \leq 0$ , or $ p + q - 1  > \text{eps.d}$
$= -6$	if 10 iterations were performed
$= -7$	iteration failed
$= -8$	$x$ is calculated with unknown accuracy

<sup>18</sup> Actually in `sfc.incgamma_ex` where the prefix calculation is optional.

<sup>19</sup> Temme's `pqasymp` with it's  $\approx 10$  digit accuracy is not suitable for extended precision.

The common procedure `sfc_incgamma_inv` is divided into three parts: In the first part the input parameters are checked; note that the condition  $p + q = 1$  is evaluated as  $|p + q - 1| \leq \text{eps\_d}$  because the parameters may be casts from double precision. The second (main) part is the sophisticated guessing of the initial value for  $x$ , and finally this value is improved by Schröder or Newton iteration steps; note that the iteration count may be zero if the initial guess is good enough. In the **AMath** Pascal source code the original text and equations from Didonato and Morris [27] are used for easy reference<sup>20</sup>.

---

```
function igamma_inv(a,p,q: double): double;
function igamma_invx(a,p,q: extended): extended;
```

---

These functions return the inverse normalised incomplete gamma function, i.e. they calculate  $x$  with  $P(a, x) = p$  and  $Q(a, x) = q$ . The input parameters are  $a > 0$ ,  $p \geq 0$ ,  $q \geq 0$ , and  $p + q = 1$ . The routines finally use `sfc_incgamma_inv` and raise error conditions instead of returning negative error codes.

---

```
function igammap_inv(a,p: double): double;
function igammap_invx(a,p: extended): extended;
```

---

These functions return the inverse normalised incomplete gamma function, i.e. they calculate  $x$  with  $P(a, x) = p$  with  $a > 0$ ,  $0 \leq p < 1$ . The routines finally use `sfc_incgamma_inv` and raise error conditions instead of returning negative error codes.

---

```
function igammaq_inv(a,q: double): double;
function igammaq_invx(a,q: extended): extended;
```

---

These functions return the inverse normalised incomplete gamma function, i.e. they calculate  $x$  with  $Q(a, x) = q$  with  $a > 0$ ,  $0 < q \leq 1$ . The routines finally use `sfc_incgamma_inv` and raise error conditions instead of returning negative error codes.

### 3.5.3 Beta functions

An important internal routine for the incomplete beta function and related code is:

```
function sfc_ibetaprefix(a,b,x,y: extended): extended;
```

This function returns  $x^a y^b / B(a, b)$  using Lanczos approximation<sup>21</sup> from Boost[19] file `beta.hpp`, function `ibeta_power_terms`.

---

```
function beta(x,y: double): double;
function betax(x,y: extended): extended;
```

---

These functions calculate the beta function, which is defined as

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}.$$

If one argument is 1, the inverse of the other is returned. In the general case when  $x + y$  is not an non-positive integer the common function `sfc_beta` computes<sup>22</sup>

$$B(x, y) = \text{sign } B(x, y) \exp(\text{lnbeta}(x, y)).$$

---

<sup>20</sup> Together with the Boost[19] interpretation from `igamma_inverse.hpp`

<sup>21</sup> Not for TP 5.0!

<sup>22</sup>  $\text{sign } B(x, y) = \text{sign } \Gamma(x) \text{sign } \Gamma(y) / \text{sign } \Gamma(x + y)$

Special considerations are needed if  $x + y$  is a non-positive integer: If  $x$  and  $y$  are no integers then  $B = 0$ . Since  $B(x, y)$  is symmetric, let  $x \leq y$ . If  $y > 0$  then  $B(x, y) = \Gamma(y)/(x)_y$ , otherwise the result is undefined.

### Logarithm of the beta function

---

```
function lnbeta(x, y: double): double;
function lnbetax(x, y: extended): extended;
```

---

These functions compute the logarithm of the beta function  $\ln |B(x, y)|$  with  $x, y \neq 0, -1, -2, \dots$ . Since the function is symmetric, let  $x \leq y$ . The implementation is based on the SLATEC [14] routine dlbeta.f. Let  $s = x + y$ . If  $x \geq 8$  then

$$\ln |B(x, y)| = -0.5 \ln y + \ln \sqrt{2\pi} + c_1 + (x - 0.5) \ln(x/s) + y \ln 1p(-x/s)$$

with  $c_1 = \text{sfc.lngcorr}(x) + \text{sfc.lngcorr}(y) - \text{sfc.lngcorr}(s)$ . If  $y \geq 8$  and  $s \geq 8$  then

$$\ln |B(x, y)| = \ln |\Gamma(x)| + c_2 + x - x \ln(s) + (y - 0.5) \ln 1p(-x/s)$$

with  $c_2 = \text{sfc.lngcorr}(y) - \text{sfc.lngcorr}(s)$ , otherwise

$$\ln |B(x, y)| = \ln |\Gamma(x)| + \ln |\Gamma(y)| - \ln |\Gamma(s)|.$$

### Normalised incomplete beta function

---

```
function ibeta(a, b, x: double): double;
function ibetax(a, b, x: extended): extended;
```

---

These functions return the normalised incomplete beta function  $I_x(a, b)$  for  $a > 0$ ,  $b > 0$ , and  $0 \leq x \leq 1$ :

$$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)}, \quad B_x(a, b) = \int_0^x t^{a-1} (1-t)^{b-1} dt.$$

There are some special cases

$$I_0(a, b) = 0, \quad I_1(a, b) = 1, \quad I_x(a, 1) = x^a,$$

and the symmetry relation  $I_x(a, b) = 1 - I_{1-x}(b, a)$ , which is used for  $x > a/(a+b)$ . The common function `sfc_ibeta` is based on the Cephes [7] routine incbetl. If  $bx \leq 1$  and  $x \leq 0.95$  (or the corresponding symmetry transformed relations) the result is computed with a hypergeometric series, c.f. NIST handbook [30, 8.17.7], otherwise  $I_x$  is evaluated with a continued fraction expansion.<sup>23</sup> If  $a$  and  $b$  are large and  $x \approx a/(a+b)$  then an asymptotic expansion [37, subroutine BASYM] from DiDonato and Morris is used.

## 3.5.4 Factorials, Pochhammer symbol, binomial coefficient

### Factorial

---

```
function fac(n: integer): double;
function facx(n: integer): extended;
```

---

These functions return the factorial  $n! = n \times (n-1) \times \dots \times 2 \times 1$ . If  $n < 0$  the result is  $\infty$ , for  $n < 25$  the value is taken from a table, and otherwise  $n! = \Gamma(n+1)$  is computed with the stirf function.

<sup>23</sup> Actually the two almost identical CF from [7] are merged into a single **AMath** routine.

## Double factorial

---

```
function dfac(n: integer): double;  
function dfacx(n: integer): extended;
```

---

These functions return the double factorial  $n!!$ , for even  $n < 0$  the result is  $\infty$ . For positive  $n$  the double factorial is defined as

$$n!! = \begin{cases} 1 \cdot 3 \cdot 5 \cdots n & \text{if } n \text{ is odd,} \\ 2 \cdot 4 \cdot 6 \cdots n & \text{if } n \text{ is even.} \end{cases}$$

The common function `sfc.dfac` computes the result depending on one of the following cases<sup>24</sup> with integer  $k \geq 0$ :

$$\begin{aligned} (2k)!! &= 2^k k! \\ (2k+1)!! &= (2k+1)! / (2^k k!) = \frac{2^{k+1}}{\sqrt{\pi}} \Gamma\left(k + \frac{3}{2}\right) \\ (-2k-1)!! &= (-1)^k / (2k-1)!! \end{aligned}$$

## Logarithm of factorials

---

```
function lnfac(n: longint): double;  
function lnfacx(n: longint): extended;
```

---

These functions return  $\ln(n!)$ . If  $n < 0$  the result is  $\infty$ , for  $n < 25$  it is computed from a table, otherwise  $\ln(n!) = \text{sfc.lngamma}(n+1)$ .

## Binomial coefficient

---

```
function binomial(n,k: integer): double;  
function binomialx(n,k: integer): extended;
```

---

These functions return the binomial coefficient (" $n$  choose  $k$ ") defined as

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)}$$

for  $k \geq 0$ . If  $n < 0$  or  $k < 0$  there are two non-trivial ranges: For  $n < 0$  and  $k \geq 0$  the coefficients are calculated with Knuth's [32] identity 1.2.6 (17)

$$\binom{n}{k} = (-1)^k \binom{k-n-1}{k}$$

and for  $k \leq n < 0$  with [32, 1.2.6 (19)]

$$\binom{n}{k} = (-1)^{n-k} \binom{-k-1}{n-k}.$$

In the common function `sfc.binomial` these cases are handled with recursive calls. In the 'normal' range  $0 \leq k \leq n$  the result is calculated as  $n!/(n-k)!/k!$  if  $n < 1755$ , i.e.  $n!$  does not overflow. If  $k \leq 4000$ <sup>25</sup> and a rough estimate shows that there will

---

<sup>24</sup>  $\Gamma$  is called explicitly for odd  $n > 25$ .

<sup>25</sup>  $k$  is replaced by  $n-k$  if it is  $> n/2$ .

be no overflow the value is computed as a product of quotients as in the definition. Otherwise the `lnbeta` function is used to return

$$\binom{n}{k} = \left( k \exp(\text{lnbeta}(k, n - k + 1)) \right)^{-1}.$$

The unusual complicated form is chosen to minimize overflow risk while maintaining reasonable accuracy. The table shows the RMS and peak relative errors calculated for random input samples:

Range (n)	Samples	RMS	Peak (n,k)
0.. 1000	20000	1.2087E-19	5.9631E-19 for (820,415)
1000.. 1700	20000	1.4733E-19	6.5052E-19 for (1678,331)
1700.. 8000	20000	1.4177E-18	7.4801E-18 for (7185,4779)
8000..16000	10000	2.0377E-16	1.2166E-15 for (14557,6857)
16000..32000	10000	1.8608E-16	1.2806E-15 for (18717,4636)

### Pochhammer symbol

---

```
function pochhammer(a,x: double): double;
function pochhammerx(a,x: extended): extended;
```

---

These functions return the Pochhammer symbol  $(a)_x = \Gamma(a+x)/\Gamma(a)$ . By convention  $(0)_x = 0$  and  $(a)_0 = 1$  for  $a \neq 0$ . In the special case that  $x = n$  is a positive integer,  $(a)_n = a(a+1)(a+2)\cdots(a+n-1)$  is often called "rising factorial".

If  $a$  or  $a+x$  are negative integers special care must be taken: If only  $a$  is an integer then the result is zero. If  $a+x$  is also a negative integer the Pochhammer symbol is computed from the limiting form of the  $\Gamma$  reflection formula

$$(a)_x = (-1)^x \Gamma(1-a)/\Gamma(1-a-x),$$

and otherwise the function is undefined. If  $a$  and  $a+x$  are no negative integers,  $(a)_x$  is calculated from the definition.

The common function `sfc_pochhammer` computes both gamma function ratios with the `gamma_delta_ratio` function.

### Ratio of gamma functions

---

```
function gamma_delta_ratio(x,d: extended): extended;
function gamma_delta_ratio(x,d: double): double;
```

---

These functions compute  $\Gamma(x)/\Gamma(x+d)$ , accurate even for  $|d| \ll |x|$ . The Pascal code uses Lanczos sums and is based on the corresponding Boost [19] function in `gamma.hpp`.

---

```
function gamma_ratio(x,y: double): double;
function gamma_ratio(x,y: extended): extended;
```

---

These routines return the gamma function ratio  $\Gamma(x)/\Gamma(y)$ . The common function simply calls `gamma_delta_ratio(x, y - x)`.

### 3.5.5 Psi and polygamma functions

#### Digamma function $\psi(x)$

---

```
function psi(x: double): double;  
function psix(x: extended): extended;
```

---

These functions return the digamma or  $\psi$  function, which is defined as

$$\psi(x) = d(\ln \Gamma(x))/dx = \Gamma'(x)/\Gamma(x), \quad x \neq 0, -1, -2, \dots$$

For  $x > 12$  the common function `sfc_psi` evaluates digamma with the asymptotic expansion from Abramowitz and Stegun [1, 6.3.18]

$$\psi(x) \sim \ln x - \frac{1}{2x} - \sum_{n=1}^{\infty} \frac{B_{2n}}{2nx^{2n}}$$

If  $x < 0$  it is transformed to positive values with the reflection formula [1, 6.3.7]

$$\psi(1-x) = \psi(x) + \pi \cot \pi x,$$

and for  $0 < x \leq 12$  the recurrence formula [1, 6.3.5]

$$\psi(x+1) = \psi(x) + \frac{1}{x}$$

is used to make  $x > 12$ . There are two special cases: First, if  $|x - x_0| < 0.2$ , where  $x_0 = 1.4616321\dots$  is the positive zero of  $\psi$ , a Padé approximation calculated with Maple V is used, and second: if  $x \leq 12$  is a positive integer  $n$ , then the result is computed directly (in reversed order) with [1, 6.3.2]:

$$\psi(n) = -\gamma + \sum_{k=1}^{n-1} k^{-1}$$

#### Trigamma function $\psi'(x)$

---

```
function trigamma(x: double): double;  
function trigamma(x: extended): extended;
```

---

These functions return the trigamma function  $\psi'(x)$ ,  $x \neq 0, -1, -2, \dots$ . The common function `sfc_trigamma` returns the Hurwitz zeta value  $\zeta(2, x)$  if  $x$  is positive; for  $x < 0$  the polygamma reflection formula [1, 6.4.7] for  $n = 1$  is used to compute the result

$$\psi'(x) = \left(\frac{\pi}{\sin \pi x}\right)^2 - \zeta(2, 1-x).$$

#### Tetragamma function $\psi''(x)$

---

```
function tetragamma(x: double): double;  
function tetragamma(x: extended): extended;
```

---

These functions return the tetragamma function  $\psi''(x)$ ,  $x \neq 0, -1, -2, \dots$ . The common function `sfc_tetragamma` returns the Hurwitz zeta value  $-2\zeta(3, x)$  if  $x$  is positive; for  $x < 0$  the polygamma reflection formula [1, 6.4.7] for  $n = 2$  is used for the result

$$\psi''(x) = -2\pi^3 \cot(\pi x)(1 + \cot^2(\pi x)) - 2\zeta(3, 1-x).$$

## Pentagamma function $\psi'''(x)$

---

```
function pentagamma(x: double): double;  
function pentagamax(x: extended): extended;
```

---

These functions return the pentagamma function  $\psi'''(x)$ ,  $x \neq 0, -1, -2, \dots$ . The common function `sfc_pentagamma` returns the Hurwitz zeta value  $6\zeta(4, x)$  if  $x$  is positive; for negative  $x$  the polygamma reflection formula [1, 6.4.7] for  $n = 3$  is used

$$\psi'''(x) = 2\pi^4(1 + 4\cot^2(\pi x) + 3\cot^4(\pi x)) - 6\zeta(4, 1 - x).$$

## Polygamma function $\psi^{(n)}(x)$

---

```
function polygamma(n: integer; x: double): double;  
function polygamax(n: integer; x: extended): extended;
```

---

These functions return the polygamma function  $\psi^{(n)}(x)$ , i.e. the  $n^{\text{th}}$  derivative of the  $\psi$  function,  $n \geq 0$ .  $x$  must be  $\neq 0, -1, -2, \dots$ , and for  $n > 10$  the current implementation requires  $x > 0$ . For  $n < 4$  the common function `sfc_polygamma` just calls the di-, tri-, tetra-, or pentagamma functions.

For  $x > 0$  the result is calculated with the series expansion [1, 6.4.10]

$$\psi^{(n)}(x) = (-1)^{n+1}n! \sum_{k=0}^{\infty} (x+k)^{-n-1} = (-1)^{n+1}n! \zeta(n+1, x).$$

If the actual computed extended value  $\zeta(n+1, x)$  of the Hurwitz zeta function is zero, the asymptotic formula [1, 6.4.11] is used:

$$\psi^{(n)}(x) \sim (-1)^{n-1} \left( \frac{(n-1)!}{x^n} + \frac{n!}{2x^{n+1}} + \sum_{k=1}^{\infty} B_{2k} \frac{(2k+n-1)!}{(2k)!x^{2k+n}} \right)$$

If in extreme cases a term in the asymptotic formula would cause overflow (e.g. for `polygamma(5000, 200)  $\approx -0.149687668e4819$` ) a "no convergence" condition is raised and the result is NaN.

For  $x < 0$  and  $n \leq 10$  the polygamma reflection formula [1, 6.4.7]

$$\psi^{(n)}(1-x) + (-1)^{n+1}\psi^{(n)}(x) = (-1)^n \pi \frac{d^n}{dx^n} \cot(\pi x)$$

is used with the pre-calculated coefficients of the polynomials  $p_n(\cot(\pi x))$  from the  $n^{\text{th}}$  derivative of  $\cot(\pi x)$ .

## 3.6 Zeta functions and polylogarithms

The unit **sfZeta** implements the common code for the zeta functions, polylogarithms, and related functions.

There are two important basic internal functions, which are called by the other common routines in the  $\eta/\zeta$  function group. In this section let  $\eta_\epsilon = 10^{-9}$ , for  $|s| \leq \eta_\epsilon$  the eta functions are linear (accurate to extended precision).

```
function etam1pos(s: extended): extended;
```

This function returns the Dirichlet  $\eta(s) - 1$  function for  $s \geq -\eta_\epsilon$

$$\eta(s) - 1 = - \sum_{k=2}^{\infty} \frac{(-1)^k}{k^s}.$$

If  $|s| \leq \eta_\epsilon$  the result is  $(s \ln(\pi/2) - 1)/2$ . Otherwise the powers  $k^s$  are computed with prime powers, but only if necessary. If  $s < 19.5$  the powers up to  $k^{27}$  are used with P. Borwein's [36] convergence acceleration technique.<sup>26</sup>

```
function zetap(s, sc: extended): extended;
```

This function returns the Riemann zeta function for  $s > 0, s \neq 1$ . The parameter  $sc = 1 - s$  is externally supplied to increase the accuracy. The routine is based on Boost's [19] zeta.hpp and uses six rational approximations in the range  $0 < s < 42$ .

### 3.6.1 Riemann zeta function

---

```
function zeta(s: double): double;  
function zetax(s: extended): extended;
```

---

These functions calculate the Riemann zeta function  $\zeta(s)$  for  $s \neq 1$ , defined by

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}, \quad s > 1,$$

and by analytic continuation for  $s < 1$ . When  $s = 0$  the result is  $\zeta(0) = -1/2$ , and for  $s > 0$  the common function **sfz\_zeta** returns  $\text{zetap}(s, 1 - s)$ . If  $s < 0$  the reflection formula [30, 25.4.2] is used:

$$\zeta(s) = 2(2\pi)^{s-1} \sin(\frac{1}{2}\pi s) \Gamma(1 - s) \zeta(1 - s)$$

#### Riemann $\zeta(n)$ for integer arguments

---

```
function zetaint(n: integer): double;  
function zetaintx(n: integer): extended;
```

---

These functions return the Riemann zeta function  $\zeta(n)$  for integer arguments  $n \neq 1$ . For  $n > 63$  the result is 1, for  $0 \leq n \leq 63$  the value is taken from table, otherwise the Bernoulli numbers are used:  $\zeta(n) = B_{1-n}/(n - 1)$  for  $n < 0$ .

---

<sup>26</sup>The coefficients are from Algorithm 2, but re-calculated, scaled, and shifted with Maple.

### Riemann $\zeta(1+x)$

---

```
function zeta1p(x: double): double;  
function zeta1px(x: extended): extended;
```

---

These functions calculate the Riemann zeta function  $\zeta(1+x)$  for  $x \neq 0$ . Normally used with  $|x| \ll 1$  for increased accuracy near the pole of  $\zeta(s)$  at  $s = 1$ . The common function `sfc.zeta1p` returns `zetap(1+x, -x)` for  $|x| < 1$ , and  $\zeta(1+x)$  otherwise.

### Riemann $\zeta(s) - 1$

---

```
function zetam1(s: double): double;  
function zetam1x(s: extended): extended;
```

---

These functions evaluate the Riemann function  $\zeta(s) - 1$  for  $s \neq 1$ . They are provided as a separate routines because  $\zeta(s) \rightarrow 1$  for large  $s$ , in fact  $\zeta(s) = 1$  to extended precision for  $s \geq 64$ . The common function `sfc.zetam1` returns  $\zeta(s) - 1$  for  $s \leq 2$ , and  $2^{-s}$  if  $s \geq 120$ , otherwise the result is computed as

$$\zeta(s) - 1 = \frac{1 + (\eta(s) - 1)2^{s-1}}{2^{s-1} - 1}.$$

## 3.6.2 Prime zeta function

---

```
function primezeta(x: double): double;  
function primezetax(x: extended): extended;
```

---

These functions return the prime zeta function

$$P(x) = \sum_{p \text{ prime}} p^{-x}, \quad x > 1.$$

For  $x \geq 27$  the common function `sfc.pz` uses the first significant terms of the sum up to  $7^{-x}$ , otherwise the result is computed from<sup>27</sup>

$$P(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} \ln(\zeta(nx)),$$

where  $\mu$  is the Möbius function<sup>28</sup>.  $\zeta(nx)$  will soon be very close to 1, therefore (in order to minimise rounding and truncation errors) the sum is evaluated with special **AMath** functions:

$$P(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} \ln1p(\text{zetam1}(nx)).$$

---

<sup>27</sup> See e.g. <http://www.math.u-bordeaux.fr/~cohen/hardy1w.dvi>, section 2.1; for TP6 or newer the speed-up is used with  $A = 7$ .

<sup>28</sup>  $\mu(n) = 1$  if  $n = 1$  or  $n$  is a product of an even number of distinct primes,  $\mu(n) = -1$  if  $n$  is a product of an odd number of distinct primes, and  $\mu(n) = 0$  if  $n$  has a multiple prime factor.

### 3.6.3 Dirichlet eta function

---

```
function eta(s: double): double;
function etax(s: extended): extended;
```

---

These functions return the Dirichlet function  $\eta(s)$ , also known as the alternating zeta function, defined for  $s > 0$  as

$$\eta(s) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^s}$$

and by analytic continuation for  $s \leq 0$ . The important relation to the Riemann zeta function is  $\eta(s) = (1 - 2^{1-s})\zeta(s)$ , which is directly evaluated for  $s \leq -8$ . In the range  $-8 < s < -\eta_\epsilon$  the reflection formula<sup>29</sup> for  $\eta$  is used:

$$\eta(s) = \frac{2(1 - 2^{1-s})\Gamma(1-s)\cos(\frac{1}{2}\pi(1-s))}{(1-2^s)(2\pi)^{1-s}}\eta(1-s)$$

and for  $s \geq -\eta_\epsilon$  the common function `sfc_eta` returns  $1 + \text{etam1pos}(s)$ .

#### Dirichlet $\eta(s) - 1$

---

```
function etam1(s: double): double;
function etam1x(s: extended): extended;
```

---

These functions return the Dirichlet function  $\eta(s) - 1$ , it is provided as a separate routine because  $\eta(s) \rightarrow 1$  for large  $s$ , in fact  $\eta(s) = 1$  to extended precision for  $s \geq 65$ . The common function `sfc_etam1` returns  $\eta(s) - 1$  if  $s < -\eta_\epsilon$  and `etam1pos(s)` otherwise.

### 3.6.4 Dirichlet beta function

---

```
function DirichletBeta(s: double): double;
function DirichletBetax(s: extended): extended;
```

---

These functions return the Dirichlet function  $\beta(s)$ , defined for  $s > 0$  as

$$\beta(s) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^s}$$

and by analytic continuation for  $s \leq 0$ . For positive  $s$  the common function `sfc_dbeta` adds up to three terms of the sum if  $s > 22.8$ , otherwise the relation

$$\beta(s) = 2^{-s}\Phi(-1, s, \frac{1}{2})$$

is used. For negative  $s$  the result is computed with the reflection formula

$$\beta(s) = \left(\frac{\pi}{2}\right)^{s-1} \Gamma(1-s) \cos\left(\frac{\pi s}{2}\right) \beta(1-s).$$

For large negative  $s$  values the extended precision  $\Gamma$  function will overflow and  $\infty$  is returned, and if  $s$  is close to 0 then

$$\beta(s) \approx \frac{1}{2} + s\beta'(0), \quad \beta'(0) = \ln \frac{\Gamma^2(1/4)}{2\pi\sqrt{2}} = 0.39159439270683677647\dots$$

---

<sup>29</sup> It can be derived easily from the reflection formula for  $\zeta$ .

### 3.6.5 Dirichlet lambda function

---

```
function DirichletLambda(s: double): double;
function DirichletLambdax(s: extended): extended;
```

---

These functions return the Dirichlet function  $\lambda(s)$ , defined for  $s > 1$  as

$$\lambda(s) = \sum_{n=0}^{\infty} (2n+1)^{-s}$$

and by analytic continuation for  $s < 1$ . The common function `sfc_dlambda` returns

$$\lambda(s) = (1 - 2^{-s})\zeta(s) = -\exp2m1(-s)\zeta(s).$$

### 3.6.6 Hurwitz zeta function

---

```
function zetaH(s, a: double): double;
function zetaHx(s, a: extended): extended;
```

---

These functions calculate the Hurwitz zeta function  $\zeta(s, a)$  defined as

$$\zeta(s, a) = \sum_{k=0}^{\infty} \frac{1}{(k+a)^s} \quad (s > 1, a \neq 0, -1, -2, \dots),$$

and by continuation to  $s < 1$ . **Note:** The current **AMath** implementation restricts the parameters to  $s \geq 0$ ,  $s \neq 1$ , and  $a > 0$ . If  $a = 1$  then  $\zeta(s)$  is returned, and if  $s = 0$  the result is  $0.5 - a$ .

The common function `sfc_zetaH` is based on Cephes [7, zeta.c] and uses Euler-Maclaurin summation<sup>30</sup>

$$\zeta(s, a) = \sum_{k=0}^n \frac{1}{(a+k)^s} + \frac{(a+n)^{1-s}}{s-1} + \sum_{k=0}^{\infty} \frac{B_{k+1}}{(k+1)!} \frac{(s+k-1)_k}{(a+n)^{s+k}}$$

or with  $B_1 = -1/2$  and  $B_{2k+1} = 0$

$$\zeta(s, a) = \sum_{k=0}^n \frac{1}{(a+k)^s} + \frac{(a+n)^{1-s}}{s-1} - \frac{1}{2(a+n)^s} + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!} \frac{s(s+1)\cdots(s+2k)}{(a+n)^{s+2k+1}}.$$

As the upper bound of the first sum the value  $n = 8$  is used if  $s < 0.5$  and  $n = 9$  otherwise, in the second sum the values  $B_{2k}/(2k)!$  are pre-calculated up to  $k = 21$ . Either summation is terminated if a term is small compared to the partial sum.

### 3.6.7 Legendre's Chi-function

---

```
function LegendreChi(s, x: double): double;
function LegendreChix(s, x: extended): extended;
```

---

These functions return Legendre's Chi-function  $\chi_s(x)$  defined for  $s \geq 0, |x| \leq 1$  by<sup>31</sup>

$$\chi_s(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)^s}.$$

<sup>30</sup> See e.g. <http://functions.wolfram.com/10.02.06.0020.01>

<sup>31</sup> Additionally  $x \neq 1$  if  $s \leq 1$ .

The function can be expressed as

$$\chi_s(x) = 2^{-s} x \Phi(z^2, s, \frac{1}{2}) = \frac{1}{2} (\text{Li}_s(x) - \text{Li}_s(-x))$$

For large  $s > 22.8$  the common function `sfc_lchi` adds up to three terms of the sum, for  $s = 0$  or  $s = 1$  the  $\text{Li}_s$  relation is used (with elementary functions, see 3.6.9), otherwise the result is computed with Lerch's transcendent.

### 3.6.8 Lerch's transcendent

---

```
function LerchPhi(z,s,a: double): double;
function LerchPhix(z,s,a: extended): extended;
```

---

These functions calculate the Lerch transcendent (sometimes called Hurwitz-Lerch zeta function)  $\Phi(z, s, a)$  defined for complex arguments as [30, 25.14.1]

$$\Phi(z, s, a) = \sum_{n=0}^{\infty} \frac{z^n}{(a+n)^s}, \quad a \neq 0, -1, -2, \dots, |z| < 1; \quad \Re s > 1, |z| = 1.$$

and by continuation for other  $z$  values. The current **AMath** implementation is restricted to real arguments  $|z| \leq 1, s \geq 0, a > 0$ .

For  $s = 0$  the function is just the geometric series

$$\Phi(z, 0, a) = \sum_{n=0}^{\infty} z^n = \frac{1}{1-z},$$

otherwise the common function `sfc_lerch` uses three methods of computation for several parameter regions: Direct summation, convergence acceleration for alternating series, and asymptotic expansion for large  $a$ , see the Pascal source code for the definitive list of cases.

The implemented convergence acceleration for alternating series is based on the C code `lerchphi.c` by S.V. Aksenov and U.D. Jentschura, the method is described in Aksenov et al. [40]. For  $-1 < z < -0.5$  a Levin transformation is used for the alternating series, for  $0.5 < z < 1$  a Van Wijngaarden transformation is applied to the non-alternating series and the new series is processed with the Levin transformation<sup>32</sup>.

The asymptotic expansion for large  $a$  implements the formula given in **Theorem 1** by Ferreira and López [41], which can be specialised for the **AMath** context as

$$\Phi(z, s, a) = \frac{1}{1-z} \frac{1}{a^s} + \sum_{n=1}^{N-1} \frac{(-1)^n \text{Li}_{-n}(z)}{n!} \frac{(s)_n}{a^{n+s}} + R_N(z, s, a), \quad N = 1, 2, 3, \dots$$

with the error term  $R_N(z, s, a) = O(a^{-N-s})$  as  $a \rightarrow \infty$ .

For  $z = 1, s > 1$  the result is  $\zeta(s, a)$ . If  $z = -1$  the convergence acceleration method is used, although Aksenov and Jentschura do not allow  $z = -1$ . But this is a classical series transformation scenario and their code gives good results<sup>33</sup>.

---

<sup>32</sup> If  $|z| \leq 0.5$  the framework of `lerchphi.c` essentially performs a sophisticated direct summation, this part is skipped by the driver function in **AMath**.

<sup>33</sup> The theoretical alternative  $\Phi(-1, s, a) = (\zeta(s, a/2) - \zeta(s, (a+1)/2))/2^s$  suffers from "catastrophic cancellation" with a zero result for  $a > 2/\text{eps}_x!$

### 3.6.9 Polylogarithms of integer order

---

```
function polylog(n: integer; x: double): double;
function polylogx(n: integer; x: extended): extended;
```

---

These functions return the polylogarithm function of integer order  $n$

$$\text{Li}_n(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^n}, \quad n \in \mathbb{Z}, |x| < 1,$$

or it's analytic continuation<sup>34</sup>; for  $n > 0$  there is the argument restriction  $x \leq 1$ .

The Pascal implementation in the common function `sfc_polylog` uses some ideas from the Cephes[7] library (file `polylog.c`), but most code is based on the formulas and transformations in R. Crandall's note [31].

There are a few special cases that are handled separately. For fixed  $n$ :

$$\begin{aligned} \text{Li}_0(x) &= \frac{x}{1-x} \\ \text{Li}_1(x) &= -\ln(1-x) \\ \text{Li}_2(x) &= \text{dilog}(x) \end{aligned}$$

and for fixed  $x$ :

$$\begin{aligned} \text{Li}_n(1) &= \zeta(n) \\ \text{Li}_n(-1) &= -\eta(n) \end{aligned}$$

Otherwise for  $n > 0$  the function is computed with the series from the definition if  $-0.875 < x < 0.5$ . For  $x < -1$  the inversion formula<sup>35</sup> is implemented:

$$\text{Li}_n(-z) + (-1)^n \text{Li}_n(-z^{-1}) = -\frac{\ln^n(z)}{n!} - 2 \sum_{k=1}^{\lfloor n/2 \rfloor} \frac{\ln^{n-2k}(z)}{(n-2k)!} \eta(2k)$$

For  $-1 < x \leq -0.875$  the recursive formula [31, 1.2]

$$\text{Li}_n(x) + \text{Li}_n(-x) = 2^{1-n} \text{Li}_n(x^2)$$

is used and for  $0.5 \leq x < 1$ :

$$\text{Li}_n(z) = \sum_{\substack{m=0 \\ m-n \neq 1}}^{\infty} \frac{\zeta(n-m)}{m!} \ln^m(x) + \frac{\ln^{n-1}(x)}{(n-1)!} (H_{n-1} - \ln(-\ln z)).$$

All the above formulas are valid only for  $n > 0$ . For negative  $n$  and  $x \neq 1$  the polylogarithms are rational-polynomial functions:

$$\text{Li}_{-n}(x) = \frac{1}{(1-x)^{n+1}} \sum_{m=1}^n \left( \sum_{k=1}^m (-1)^{k+1} \binom{n+1}{k-1} (m-k+1)^n \right) x^m \quad (n > 0)$$

In `sfc_polylog` the numerator polynomials for  $-10 < n < 0$  are hard-coded; otherwise for  $|x| \leq 0.25$  the function is computed with the series definition. The cases  $x > 4$  and  $x < -11.5$  are transformed to the previous with

$$\text{Li}_{-n}(x) = (-1)^{n-1} \text{Li}_{-n}\left(\frac{1}{x}\right), \quad (n > 0),$$

---

<sup>34</sup>The function can also be defined for complex orders  $n$ .

<sup>35</sup>See Cephes[7] or <http://functions.wolfram.com/10.08.17.0060.01>. I use  $-\eta(2k)$  instead of  $\text{Li}_{2k}(-1)$ .

and the other arguments are handled with Crandall's relation [31, 1.5]:

$$\operatorname{Li}_n(z) = (-n)!(-\ln x)^{n-1} - \sum_{k=0}^{\infty} \frac{B_{k-n+1}}{k!(k-n+1)} \ln^k(x), \quad (n \leq 0),$$

where in-line complex arithmetic is used if  $x < 0$ .

### 3.6.10 Polylogarithms of real order

---

```
function polylogr(s, x: double): double;
function polylogrx(s, x: extended): extended;
```

---

These functions return the polylogarithm function of real order  $s \geq 0$

$$\operatorname{Li}_s(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^s}, \quad s \geq 0, |x| \leq 1,$$

for  $s \leq 1$  there is the additional argument restriction  $x \neq 1$ . The common function handles the special case  $\operatorname{Li}_s(0) = 0$ , and adds one or two terms of the sum for  $s \geq 40.5$ . If  $s = n$  is an integer the value `sfc.polylog(n, x)` is returned, otherwise Lerch's transcendent is used (c.f. [30, 25.14.3]):

$$\operatorname{Li}_s(x) = x \Phi(x, s, 1)$$

### 3.6.11 Dilogarithm function

---

```
function dilog(x: double): double;
function dilogx(x: extended): extended;
```

---

These functions return the dilogarithm function

$$\operatorname{dilog}(x) = \Re \operatorname{Li}_2(x) = -\Re \int_0^x \frac{\ln(1-t)}{t} dt.$$

For  $x \leq 1$  the relation  $\operatorname{dilog}(x) = \operatorname{polylog}(2, x)$  is valid, but `dilog` is implemented even for  $x > 1$ .

Note that there is some confusion about the naming: some authors and/or computer algebra systems use  $\operatorname{dilog}(x) = \operatorname{Li}_2(1-x)$  and then call  $\operatorname{Li}_2(x)$  Spence function/integral or similar.

The common function `sfc_dilog` uses a Chebyshev approximation and several linear transformations from Fullerton [20, 14] (file `dspenc.f`), see also Maximon [35], section 3(a).

### 3.6.12 Clausen function

---

```
function cl2(x: double): double;
function cl2x(x: extended): extended;
```

---

These functions return the Clausen function

$$\operatorname{Cl}_2(x) = \Im \operatorname{Li}_2(e^{ix}) = -\int_0^x \ln|2 \sin(\frac{t}{2})| dt.$$

See MacLeod's MISCFUN [22] (function clausn) for formulas and hints. As pointed out, only absolute accuracy can be guaranteed close to the zeroes. Observed relative errors for a reference Pascal translation of `clausn` are e.g. 1700 eps\_x for  $|x - \pi| \approx 6.7 \cdot 10^{-4}$ , but even for  $|x - \pi| \approx 0.03$  they are still about 128 eps\_x.

Therefore two separate Chebyshev expansions are computed for the reduced argument  $z = x \bmod 2\pi$ . The first for  $z \in (-\pi/2, \pi/2)$  is based on [22, (3)] or [1, 27.8.2]:

$$\text{Cl}_2(x) = -x \ln|x| + x - \sum_{k=1}^{\infty} \frac{(-1)^k B_{2k} x^{2k+1}}{2k(2k+1)(2k)!},$$

and the second for the function  $\text{Cl}_2(x + \pi)$  uses

$$\text{Cl}_2(x) = \Im \text{Li}_2(e^{ix}) = -\frac{i}{2} (\text{Li}_2(e^{ix}) - \text{Li}_2(e^{-ix})).$$

The calculations are done using Maple<sup>36</sup> and the **MPArith** calculator **T\_RCalc** to convert to Hex/Extended. Both approximations are for even functions, and there are only the even Chebyshev polynomials, so the argument for `CSEvalX` is  $2(2x/\pi)^2 - 1$  with the calculated coefficients.

For  $z = 0$  or  $z = \pi$  the common function `sfc_c12` returns zero.

### 3.6.13 Inverse-tangent integral

---

```
function ti2(x: double): double;
function ti2x(x: extended): extended;
```

---

These functions return the inverse-tangent integral

$$\text{Ti}_2(x) = \int_0^x \frac{\tan^{-1}t}{t} dt.$$

MISCFUN [22] (function atnint) and GSL [21] (function specfunc/atanint.c) give Chebyshev approximations, but both are not suitable for extended precision.

For  $0 \leq x \leq 1$  the integral is computed with a Chebyshev approximation calculated with Maple<sup>37</sup>, for  $x > 1$  the relation [35, 8.6]

$$\text{Ti}_2(x) = \text{Ti}_2\left(\frac{1}{x}\right) + \frac{\pi}{2} \ln x$$

is used, and for  $x < 0$  the result is  $\text{Ti}_2(x) = -\text{Ti}_2(-x)$ .

---

<sup>36</sup>Note that Maple's `polylog(2,x)` must be used for  $\text{Li}_2(x)$ .

<sup>37</sup>`chebyshev(int(arctan(t)/t, t=0..x)/x, x=-1..1, 0.5e-20);`

## 3.7 Orthogonal polynomials and related

Orthogonal polynomials are families of polynomials  $p_n(x)$  which are defined over an interval  $(a, b)$  and have an orthogonality relation with respect to a weight function  $w(x) \geq 0$ , i.e.  $\int_a^b w(x)p_n(x)p_m(x)dx = 0$  for  $n \neq m$ .

The Pascal unit **sfPoly** implements the common code for most of the (unshifted) classical orthogonal polynomials and related functions; also included are the Zernike radial polynomials.

### 3.7.1 Chebyshev polynomials of the first kind

---

```
function chebyshev_t(n: integer; x: double): double;  
function chebyshev_tx(n: integer; x: extended): extended;
```

---

These functions return  $T_n(x)$ , the Chebyshev polynomial of the first kind of degree  $n$ . The  $T_n(x)$  are orthogonal on the interval  $[-1, 1]$ , with respect to the weight function  $w(x) = (1 - x^2)^{-1/2}$ .

For  $0 \leq n \leq 64$  the common function **sfc\_chebyshev\_t** evaluates  $T_n(x)$  with the standard recurrence formulas [1, 22.7.4]:

$$\begin{aligned}T_0(x) &= 1 \\T_1(x) &= x \\T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x)\end{aligned}$$

If  $n > 64$  the following trigonometric and hyperbolic identities [1, 22.3.15]:

$$T_n(x) = \begin{cases} \cos(n \arccos(x)) & |x| < 1 \\ \cosh(n \operatorname{arccosh}(x)) & |x| > 1 \end{cases}$$

are used, and the special cases  $|x| = 1$  are handled separately. If  $n < 0$  the function result is  $T_n(x) = T_{-n}(x)$ .

### 3.7.2 Chebyshev polynomial of the second kind

---

```
function chebyshev_u(n: integer; x: double): double;  
function chebyshev_ux(n: integer; x: extended): extended;
```

---

These functions return  $U_n(x)$ , the Chebyshev polynomial of the second kind of degree  $n$ . The  $U_n$  are orthogonal on the interval  $[-1, 1]$ , with respect to the weight function  $w(x) = (1 - x^2)^{1/2}$ .

For  $0 \leq n \leq 64$  the common function **sfc\_chebyshev\_u** evaluates  $U_n(x)$  with the standard recurrence formulas [1, 22.7.5]:

$$\begin{aligned}U_0(x) &= 1 \\U_1(x) &= 2x \\U_{n+1}(x) &= 2xU_n(x) - U_{n-1}(x)\end{aligned}$$

If  $n > 64$  the trigonometric and hyperbolic identities [1, 22.3.16]:

$$U_n(x) = \begin{cases} \frac{\sin((n+1) \arccos(x))}{\sin(\arccos(x))} & |x| < 1 \\ \frac{\sinh((n+1) \operatorname{arccosh}(x))}{\sinh(\operatorname{arccosh}(x))} & |x| > 1 \end{cases}$$

are used, and the special cases  $|x| = 1$  are handled separately. If  $n < 0$  the function results are  $U_{-1}(x) = 0$  and  $U_n(x) = -U_{-n-2}(x)$

### 3.7.3 Gegenbauer (ultraspherical) polynomials

---

```
function gegenbauer_c(n: integer; a, x: double): double;
function gegenbauer_cx(n: integer; a, x: extended): extended;
```

---

These functions return  $C_n^{(a)}(x)$ , the Gegenbauer (ultraspherical) polynomial of degree  $n$  with parameters  $a$ . The degree  $n$  must be non-negative;  $a$  should be  $> -1/2$ . The Gegenbauer polynomials are orthogonal on the interval  $[-1, 1]$ , with respect to the weight function  $w(x) = (1 - x^2)^{a-1/2}$ .

If  $a \neq 0$  the common function `sfc.gegenbauer_c` uses the standard recurrence formulas [1, 22.7.3]:

$$\begin{aligned}C_0^{(a)}(x) &= 1 \\C_1^{(a)}(x) &= 2ax \\nC_n^{(a)}(x) &= 2(n + a - 1)x C_{n-1}^{(a)}(x) - (n + 2a - 2)C_{n-2}^{(a)}(x)\end{aligned}$$

For  $a = 0$  the result can be expressed in Chebyshev polynomials:

$$C_0^0(x) = 1, \quad C_n^0(x) = 2/nT_n(x)$$

Note that  $a > -1/2$  is not checked in the Pascal function. It seems that this requirement is related to the exponent  $a - 1/2$  in the weight function. For a formal definition of the Gegenbauer polynomials with the recurrence relation it is obviously not needed.

### 3.7.4 Hermite polynomials

---

```
function hermite_h(n: integer; x: double): double;
function hermite_hx(n: integer; x: extended): extended;
```

---

These functions return  $H_n(x)$ , the Hermite polynomial of degree  $n \geq 0$ . The  $H_n$  are orthogonal on the interval  $(-\infty, \infty)$ , with respect to the weight function  $w(x) = \exp(-x^2)$ . They are computed in the common function `sfc.hermite_h` with the standard recurrence formulas [1, 22.7.13]:

$$\begin{aligned}H_0(x) &= 1 \\H_1(x) &= 2x \\H_n(x) &= 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x).\end{aligned}$$

### 3.7.5 Jacobi polynomials

---

```
function jacobi_p(n: integer; a, b, x: double): double;
function jacobi_px(n: integer; a, b, x: extended): extended;
```

---

These functions return  $P_n^{(a,b)}(x)$ , the Jacobi polynomial of degree  $n \geq 0$  with parameters  $(a, b)$ .  $a, b$  should be greater than  $-1$ , and  $a + b$  must not be an integer less than  $-1$ . Jacobi polynomials are orthogonal on the interval  $[-1, 1]$ , with respect to the weight function  $w(x) = (1 - x)^a(1 + x)^b$ , if  $a, b$  are greater than  $-1$ .

In the common function `sfc.jacobi_p` the cases  $n \leq 1$  are computed with the explicit formulas

$$P_0^{(a,b)} = 1, \quad 2P_1^{(a,b)} = (a - b) + (a + b + 2)x,$$

and for  $n > 1$  there is the somewhat complicated recurrence relation from [30] (18.9.1) and (18.9.2):

$$\begin{aligned}
 P_{n+1}^{(a,b)} &= (A_n x + B_n) P_n^{(a,b)} - C_n P_{n-1}^{(a,b)} \\
 A_n &= \frac{(2n + a + b + 1)(2n + a + b + 2)}{2(n + 1)(n + a + b + 1)} \\
 B_n &= \frac{(a^2 - b^2)(2n + a + b + 1)}{2(n + 1)(n + a + b + 1)(2n + a + b)} \\
 C_n &= \frac{(n + a)(n + b)(2n + a + b + 2)}{(n + 1)(n + a + b + 1)(2n + a + b)}
 \end{aligned}$$

### 3.7.6 Generalized Laguerre polynomials

---

```

function laguerre(n: integer; a, x: double): double;
function laguerrex(n: integer; a, x: extended): extended;

```

---

These functions return  $L_n^{(a)}(x)$ , the generalized Laguerre polynomials of degree  $n \geq 0$  with parameter  $a$ ;  $x \geq 0$  and  $a > -1$  are the standard ranges. These polynomials are orthogonal on the interval  $[0, \infty)$ , with respect to the weight function  $w(x) = e^{-x} x^a$ .

If  $x < 0$  and  $a > -1$  the common function `sfc_laguerre` tries to avoid inaccuracies and computes the result with Kummer's confluent hypergeometric function, see [1, 22.5.34]

$$L_n^{(a)}(x) = \binom{n+a}{n} M(-n, a+1, x),$$

otherwise the standard recurrence formulas from [1, 22.7.12] are used:

$$\begin{aligned}
 L_0^{(a)}(x) &= 1 \\
 L_1^{(a)}(x) &= -x + 1 + a \\
 nL_n^{(a)}(x) &= (2n + a - 1 - x)L_{n-1}^{(a)}(x) - (n + a - 1)L_{n-2}^{(a)}(x)
 \end{aligned}$$

### 3.7.7 Laguerre polynomials

---

```

function laguerre_l(n: integer; x: double): double;
function laguerre_lx(n: integer; x: extended): extended;

```

---

These functions return  $L_n(x)$ , the Laguerre polynomials of degree  $n \geq 0$ . The  $L_n$  are just special cases of the generalized Laguerre polynomials

$$L_n(x) = L_n^{(0)}(x),$$

the common function `sfc_laguerre_l` simple calls `sfc_laguerre(n,0,x)`.

### 3.7.8 Associated Laguerre polynomials

---

```

function laguerre_ass(n, m: integer; x: double): double;
function laguerre_assx(n, m: integer; x: extended): extended;

```

---

These functions return  $L_n^m(x)$ , the associated Laguerre polynomials of degree  $n \geq 0$  and order  $m \geq 0$ , defined as

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x).$$

The  $L_n^m$  are computed using the generalized Laguerre polynomials

$$L_n^m(x) = L_n^{(m)}(x).$$

### 3.7.9 Legendre polynomials

---

```
function legendre_p(l: integer; x: double): double;
function legendre_px(l: integer; x: extended): extended;
```

---

These functions return  $P_l(x)$ , the Legendre polynomials (sometimes called Legendre functions of the first kind) of degree  $l \geq 0$  and  $|x| \leq 1$ . The Legendre polynomials are orthogonal on the interval  $[-1, 1]$ .

The  $P_l$  can be written as special cases of the Gegenbauer or Jacobi polynomials, but the common function `sfc_legendre_p` invokes a function (also called by the Legendre functions of the second kind) which uses the recurrence formulas

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ (l+1)P_{l+1}(x) &= (2l+1)xP_l(x) - lP_{l-1}(x) \end{aligned}$$

### 3.7.10 Associated Legendre polynomials

---

```
function legendre_plm(l,m: integer; x: double): double;
function legendre_plmx(l,m: integer; x: extended): extended;
```

---

These functions return  $P_l^m(x)$ , the associated Legendre polynomials<sup>38</sup> of degree  $l$  and order  $m$  for  $|x| \leq 1$ . They are also known as Ferrer's functions of the first kind (see e.g. [30] Ch.14), and are defined as

$$P_l^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_l(x).$$

Note that the factor  $(-1)^m$  is not always included in the literature, the **AM-ath** definition agrees with Abramowitz and Stegun[1], the NIST handbook[30], Press et al.[13], and Boost[19].

Negative  $l$  or  $m$  are mapped to positive values with [1] (8.2.1) and (8.2.5)

$$\begin{aligned} P_{-l-1}^m(x) &= P_l^m(x), \\ P_l^{-m}(x) &= (-1)^m \frac{(l-m)!}{(l+m)!} P_l^m(x). \end{aligned}$$

If (after these mappings)  $m > l$ , then the function result is zero; if  $m = 0$  the Legendre polynomial  $P_l(x)$  is returned.

Otherwise the common function `sfc_legendre_plm` invokes a function<sup>39</sup>, which is based on the recurrence formula for varying degree [1, 8.5.2]

$$(l-m+1)P_{l+1}^m(x) = (2l+1)P_l^m(x) - (l+m)P_{l-1}^m(x)$$

and uses some tweaks from [19] (file `legendre.hpp`) and [13] Ch.6.8.

---

<sup>38</sup>True polynomials only for even  $|m|$ .

<sup>39</sup>Also called by the spherical harmonic functions.

### 3.7.11 Legendre functions of the second kind

---

```
function legendre_q(l: integer; x: double): double;  
function legendre_qx(l: integer; x: extended): extended;
```

---

These functions return  $Q_l(x)$ , the Legendre functions of the second kind of degree  $l$  and  $|x| < 1$ . The common function `sfc.legendre_q` invokes a function, which shares code for the Legendre functions of the first kind and uses the recurrence formulas<sup>40</sup>

$$Q_0(x) = \frac{1}{2} \ln \frac{x+1}{x-1}$$
$$Q_1(x) = \frac{x}{2} \ln \frac{x+1}{x-1} - 1$$
$$(l+1)Q_{l+1}(x) = (2l+1)xQ_l(x) - lQ_{l-1}(x)$$

### 3.7.12 Spherical harmonic functions

---

```
procedure spherical_harmonic(l, m: integer;  
    theta, phi: double; var yr, yi: double);  
procedure spherical_harmonicx(l, m: integer;  
    theta, phi: extended; var yr, yi: extended);
```

---

The procedures return the real and imaginary parts of the spherical harmonic function  $Y_{lm}(\theta, \phi)$ . These functions are closely related to the associated Legendre polynomials:

$$Y_{lm}(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} P_l^m(\cos \theta) e^{im\phi}.$$

Note that the spherical harmonics are periodic with period  $\pi$  in  $\theta$ , see also the Boost [19] function `spherical_harmonic.hpp` and the note in the  $P_l^m$  section.

The common function `sfc.spherical_harmonic` maps negative  $m$  to positive values, reduces the ranges of  $\theta, \phi$ , and keeps track of the sign changes related to these transformations. Then the amplitude  $A = |Y_{lm}(\theta, \phi)|$  is calculated using the code shared with the associated Legendre polynomials. The results are (after possible sign adjustments):

$$\text{yr} = \Re Y_{lm}(\theta, \phi) = A \cos(m\phi),$$
$$\text{yi} = \Im Y_{lm}(\theta, \phi) = A \sin(m\phi).$$

### 3.7.13 Zernike radial polynomials

---

```
function zernike_r(n, m: integer; r: double): double;  
function zernike_rx(n, m: integer; r: extended): extended;
```

---

These functions return the Zernike radial polynomials  $R_n^m(r)$  with  $r \geq 0$  and  $n \geq m \geq 0$ ,  $n-m$  even, zero otherwise. The orthogonality relation is

$$\int_0^1 R_n^m(r) R_{n'}^m(r) r dr = \frac{1}{2(n+1)} \delta_{nn'}.$$

The common function `sfc.zernike_r` evaluates the  $R_n^m$  as special cases of the Jacobi polynomials<sup>41</sup>

$$R_n^m(r) = (-1)^{(n-m)/2} r^m P_{(n-m)/2}^{(m,0)}(1-2r^2).$$

---

<sup>40</sup>See Abramowitz and Stegun[1] (8.4.2) and (8.4.4) for the cases  $l = 0, 1$ .

<sup>41</sup>See e.g. <http://mathworld.wolfram.com/ZernikePolynomial.html>, (6)

## 3.8 Statistical distributions

This section describes the statistical distributions that are implemented in **AMath**, the source code unit is **sfsdist**. For each continuous distribution there are three functions: The *probability density function* (PDF), generally denoted by  $f(x) \geq 0$  with its support interval  $[A, B]$ , the *cumulative distribution function* (CDF)

$$F(x) = \int_A^x f(t)dt, \quad F(x) = 1 \text{ for } x \geq B,$$

and the *inverse cumulative distribution function* (ICDF), normally the functional inverse  $F^{-1}$  of  $F$ , i.e.  $F(F^{-1}(y)) = y$  for  $y \in (0, 1)$ . For the implemented discrete distributions there are *probability mass functions* (PMF) and CDF.

The Pascal functions have the suffixes `'_pdf'`, `'_pmf'`, `'_cdf'`, and `'_inv'` (plus the usual `'x'` for the extended versions).

There are two additional references for this section, they are not used in actual implementation but provide formulas and background information:

- H. Rinne, Taschenbuch der Statistik, 4.Auflage, Frankfurt 2008
- Wikipedia, the free encyclopedia. List of probability distributions with links to many specific probability distributions: [http://en.wikipedia.org/wiki/List\\_of\\_probability\\_distributions](http://en.wikipedia.org/wiki/List_of_probability_distributions)

### 3.8.1 Beta distribution

---

```
function beta_pdf(a, b, x: double): double;
function beta_cdf(a, b, x: double): double;
function beta_inv(a, b, y: double): double;
function beta_pdfx(a, b, x: extended): extended;
function beta_cdfx(a, b, x: extended): extended;
function beta_invx(a, b, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the beta distribution with parameters  $a > 0$  and  $b > 0$  and the support interval  $(0, 1)$ :

$$\text{PDF: } f(x) = x^{a-1}(1-x)^{b-1}/B(a, b).$$

$$\text{CDF: } F(x) = I_x(a, b) = \text{ibeta}(a, b, x)$$

The ICDF is based on the Cephes [7] function `incbil.c`, it is computed with Newton iterations using an initial value from the ICDF of the standard normal distribution.

### 3.8.2 Binomial distribution

---

```
function binomial_pmf(p: double; n, k: longint): double;
function binomial_cdf(p: double; n, k: longint): double;
function binomial_pmf_x(p: extended; n, k: longint): extended;
function binomial_cdf_x(p: extended; n, k: longint): extended;
```

---

These functions return PMF and CDF of the (discrete) binomial distribution with number of trials  $n \geq 0$  and success probability  $0 \leq p \leq 1$ .

$$\text{PMF: } f(k) = \binom{n}{k} p^k (1-p)^{n-k} = \text{beta\_pdf}(k+1, n-k+1, p)/(n+1)$$

$$\text{CDF: } F(k) = I_{1-p}(n-k, k+1) = \text{ibeta}(n-k, k+1, 1-p)$$

### 3.8.3 Cauchy distribution

---

```
function cauchy_pdf(a, b, x: double): double;
function cauchy_cdf(a, b, x: double): double;
function cauchy_inv(a, b, y: double): double;
function cauchy_pdfx(a, b, x: extended): extended;
function cauchy_cdfx(a, b, x: extended): extended;
function cauchy_invx(a, b, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the Cauchy distribution with parameters  $a$  (location) and  $b > 0$  (scale) and the support interval  $(-\infty, +\infty)$ :

$$\text{PDF: } f(x) = \frac{1}{\pi b(1 + ((x - a)/b)^2)}$$
$$\text{CDF: } F(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x - a}{b}\right)$$

The inverse cumulative distribution function is

$$F^{-1}(y) = \begin{cases} a - b/\tan(\pi y) & y < 0.5, \\ a & y = 0.5, \\ a - b/\tan(\pi(1 - y)) & y > 0.5. \end{cases}$$

### 3.8.4 Chi-square distribution

---

```
function chi2_pdf(nu: longint; x: double): double;
function chi2_cdf(nu: longint; x: double): double;
function chi2_inv(nu: longint; p: double): double;
function chi2_pdfx(nu: longint; x: extended): extended;
function chi2_cdfx(nu: longint; x: extended): extended;
function chi2_invx(nu: longint; p: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the chi-square distribution with  $\nu \in \mathbb{N}$  degrees of freedom and the support interval  $(0, +\infty)$ :

$$\text{PDF: } f(x) = \frac{(\frac{1}{2}x)^{\nu/2-1} \exp(-\frac{1}{2}x)}{2\Gamma(\nu/2)} = \text{sfc\_igprefix}(\frac{1}{2}\nu, \frac{1}{2}x)/x$$

$$\text{CDF: } F(x) = P(\frac{1}{2}\nu, \frac{1}{2}x) = \text{igammap}(\frac{1}{2}\nu, \frac{1}{2}x)$$

$$\text{ICDF: } F^{-1}(y) = 2 \cdot \text{igammap\_inv}(\frac{1}{2}\nu, y)$$

### 3.8.5 Exponential distribution

---

```
function exp_pdf(a, alpha, x: double): double;
function exp_cdf(a, alpha, x: double): double;
function exp_inv(a, alpha, y: double): double;
function exp_pdfx(a, alpha, x: extended): extended;
function exp_cdfx(a, alpha, x: extended): extended;
function exp_invx(a, alpha, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the exponential distribution with location  $a$  and rate  $\alpha > 0$  and the support interval  $(a, +\infty)$  :

$$\begin{aligned} \text{PDF: } f(x) &= \alpha \exp(-\alpha(x - a)) \\ \text{CDF: } F(x) &= 1 - \exp(-\alpha(x - a)) = \text{expm1}(-\alpha(x - a)) \\ \text{ICDF: } F^{-1}(y) &= a - \ln(1 - y) / \alpha \end{aligned}$$

### 3.8.6 F-distribution

---

```
function f_pdf(nu1, nu2: longint; x: double): double;
function f_cdf(nu1, nu2: longint; x: double): double;
function f_inv(nu1, nu2: longint; y: double): double;
function f_pdfx(nu1, nu2: longint; x: extended): extended;
function f_cdfx(nu1, nu2: longint; x: extended): extended;
function f_invx(nu1, nu2: longint; y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the F- (or Fisher-Snedecor) distribution with  $\nu_1, \nu_2 \in \mathbb{N}$  degrees of freedom and the support interval  $(0, +\infty)$ .

$$\begin{aligned} \text{PDF: } f(x) &= \frac{x^{\frac{1}{2}\nu_1 - 1}}{B(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2)} \left(\frac{\nu_1}{\nu_2}\right)^{\frac{1}{2}\nu_1} \left(1 + \frac{\nu_1}{\nu_2}x\right)^{-\frac{1}{2}(\nu_1 + \nu_2)} \\ \text{CDF: } F(x) &= I_{\frac{\nu_1 x}{\nu_1 x + \nu_2}}(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2) = \text{ibeta}(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2, \frac{\nu_1 x}{\nu_1 x + \nu_2}) \end{aligned}$$

The ICDF is computed with the ICDF of the beta distribution, where the first equation is used if  $y < I_{1/2}(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2)$ , and the second otherwise.

$$\begin{aligned} F^{-1}(y) &= \frac{\nu_2 x}{\nu_1(1 - x)} \quad \text{with } x = \text{beta.inv}(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2, y), \\ F^{-1}(y) &= \frac{\nu_2(1 - x)}{\nu_1 x} \quad \text{with } x = \text{beta.inv}(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2, 1 - y). \end{aligned}$$

### 3.8.7 Gamma distribution

---

```
function gamma_pdf(a, b, x: double): double;
function gamma_cdf(a, b, x: double): double;
function gamma_inv(a, b, p: double): double;
function gamma_pdfx(a, b, x: extended): extended;
function gamma_cdfx(a, b, x: extended): extended;
function gamma_invx(a, b, p: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the gamma distribution with shape  $a > 0$ , and scale  $b > 0$  and the support interval  $(0, +\infty)$ .

$$\begin{aligned} \text{PDF: } f(x) &= \frac{x^{a-1} e^{-x/b}}{\Gamma(a) b^a} \\ \text{CDF: } F(x) &= P(a, x/b) = \text{igammap}(a, x/b) \\ \text{ICDF: } F^{-1}(y) &= b \cdot \text{igammap.inv}(a, y) \end{aligned}$$

### 3.8.8 Hypergeometric distribution

---

```
function hypergeo_pmf(n1, n2, n, k: longint): double;  
function hypergeo_cdf(n1, n2, n, k: longint): double;  
function hypergeo_pmf_x(n1, n2, n, k: longint): extended;  
function hypergeo_cdf_x(n1, n2, n, k: longint): extended;
```

---

These functions return PMF and CDF of the (discrete) hypergeometric distribution; the PMF gives the probability that among  $n$  randomly chosen samples from a container with  $n_1$  type<sub>1</sub> objects and  $n_2$  type<sub>2</sub> objects there are exactly  $k$  type<sub>1</sub> objects<sup>42</sup>:

$$\text{PMF: } f(k) = \frac{\binom{n_1}{k} \binom{n_2}{n-k}}{\binom{n_1+n_2}{n}} \quad (n, n_1, n_2 \geq 0, n \leq n_1 + n_2).$$

$f(k)$  is computed with the R trick [39], which replaces the binomial coefficients by binomial PMFs with  $p = n/(n_1 + n_2)$ . There is no explicit formula for the CDF, it is calculated as  $\sum f(i)$ , using the lower tail if  $k < nn_1/(n_1 + n_2)$  and the upper tail otherwise with one value of the PMF and the recurrence formulas:

$$f(k+1) = \frac{(n_1 - k)(n - k)}{(k+1)(n_2 - n + k + 1)} f(k)$$
$$f(k-1) = \frac{k(n_2 - n + k)}{(n_1 - k + 1)(n - k + 1)} f(k)$$

### 3.8.9 Laplace distribution

---

```
function laplace_inv(a, b, y: double): double;  
function laplace_pdf(a, b, x: double): double;  
function laplace_cdf_x(a, b, x: extended): extended;  
function laplace_inv_x(a, b, y: extended): extended;  
function laplace_pdf_x(a, b, x: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the Laplace distribution with location  $a$ , scale  $b > 0$  and the support interval  $(-\infty, +\infty)$ :

$$\text{PDF: } f(x) = \exp(-|x - a|/b)/(2b)$$
$$\text{CDF: } F(x) = \begin{cases} \frac{1}{2} - \frac{1}{2} \expm1(-\frac{x-a}{b}) & x \geq a \\ \frac{1}{2} \exp(-\frac{x-a}{b}) & x < a \end{cases}$$
$$\text{ICDF: } F^{-1}(y) = \begin{cases} a + b \ln(2y) & y < 0.5 \\ a - b \ln(2(1-y)) & y \geq 0.5 \end{cases}$$

### 3.8.10 Logistic distribution

---

```
function logistic_pdf(a, b, x: double): double;  
function logistic_cdf(a, b, x: double): double;  
function logistic_inv(a, b, y: double): double;  
function logistic_pdf_x(a, b, x: extended): extended;  
function logistic_cdf_x(a, b, x: extended): extended;  
function logistic_inv_x(a, b, y: extended): extended;
```

---

<sup>42</sup> There are different definitions; the **AMath** version is compatible with Maple and R.

These functions return PDF, CDF, and ICDF of the logistic distribution with parameters  $a$  (location) and  $b > 0$  (scale) and the support interval  $(-\infty, +\infty)$  :

$$\begin{aligned} \text{PDF: } f(x) &= \frac{1}{b} \frac{\exp(-\frac{x-a}{b})}{\left(1 + \exp(-\frac{x-a}{b})\right)^2} \\ \text{CDF: } F(x) &= \frac{1}{1 + \exp(\frac{a-x}{b})} \\ \text{ICDF: } F^{-1}(y) &= a - b \ln\left(\frac{1-y}{y}\right) \end{aligned}$$

### 3.8.11 Lognormal distribution

---

```
function lognormal_pdf(a, b, x: double): double;
function lognormal_cdf(a, b, x: double): double;
function lognormal_inv(a, b, y: double): double;
function lognormal_pdfx(a, b, x: extended): extended;
function lognormal_cdfx(a, b, x: extended): extended;
function lognormal_invx(a, b, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the log-normal distribution with location  $a$ , scale  $b > 0$  and the support interval  $(0, +\infty)$  :

$$\begin{aligned} \text{PDF: } f(x) &= \frac{1}{bx\sqrt{2\pi}} \exp\left(-\frac{(\ln x - a)^2}{2b^2}\right) \\ \text{CDF: } F(x) &= \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{\ln x - a}{b\sqrt{2}}\right)\right) \\ \text{ICDF: } F^{-1}(y) &= \exp\left(a + b \cdot \operatorname{normstd\_inv}(y)\right) \end{aligned}$$

### 3.8.12 Negative binomial distribution

---

```
function negbinom_pmf(p, r: double; k: longint): double;
function negbinom_cdf(p, r: double; k: longint): double;
function negbinom_pmf_x(p, r: extended; k: longint): extended;
function negbinom_cdf_x(p, r: extended; k: longint): extended;
```

---

These functions return PMF and CDF of the (discrete) negative binomial distribution<sup>43</sup> with target for number of successful trials  $r > 0$  and success probability  $0 \leq p \leq 1$  :

$$\begin{aligned} \text{PMF: } f(k) &= \frac{\Gamma(k+r)}{k!\Gamma(r)} p^r (1-p)^k = \frac{p}{r+k} \operatorname{beta\_pdf}(r, k+1, p) \\ \text{CDF: } F(k) &= I_{1-p}(r, k+1) = \operatorname{ibeta}(r, k+1, 1-p) \end{aligned}$$

If  $r = n$  is a positive integer the name **Pascal** distribution is used, and for  $r = 1$  it is called **geometric** distribution.

---

<sup>43</sup> There are different definitions for the negative binomial distribution; the **AMath** version is compatible with Maple, R, GSL, Boost.

### 3.8.13 Gaussian Normal distribution

---

```
function normal_pdf(mu, sd, x: double): double;  
function normal_cdf(mu, sd, x: double): double;  
function normal_inv(mu, sd, y: double): double;  
function normal_pdfx(mu, sd, x: extended): extended;  
function normal_cdfx(mu, sd, x: extended): extended;  
function normal_invx(mu, sd, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the Gaussian normal distribution with mean  $\mu$ , standard deviation  $\sigma > 0$  and the support interval  $(-\infty, +\infty)$ :

$$\begin{aligned}\text{PDF: } f(x) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\ \text{CDF: } F(x) &= \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right) = \operatorname{normstd.cdf}\left(\frac{x-\mu}{\sigma}\right) \\ \text{ICDF: } F^{-1}(y) &= \sigma \cdot \operatorname{normstd.inv}(y) + \mu\end{aligned}$$

### 3.8.14 Standard Normal distribution

---

```
function normstd_pdf(x: double): double;  
function normstd_cdf(x: double): double;  
function normstd_inv(y: double): double;  
function normstd_pdfx(x: extended): extended;  
function normstd_cdfx(x: extended): extended;  
function normstd_invx(y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the standard normal distribution with on the support interval  $(-\infty, +\infty)$ :

$$\begin{aligned}\text{PDF: } f(x) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \\ \text{CDF: } F(x) &= \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = \frac{1}{2} \operatorname{erfc}\left(-\frac{x}{\sqrt{2}}\right) \\ \text{ICDF: } F^{-1}(y) &= -\sqrt{2} \operatorname{erfc.inv}(2y)\end{aligned}$$

### 3.8.15 Pareto distribution

---

```
function pareto_pdf(k, a, x: double): double;  
function pareto_cdf(k, a, x: double): double;  
function pareto_inv(k, a, y: double): double;  
function pareto_pdfx(k, a, x: extended): extended;  
function pareto_cdfx(k, a, x: extended): extended;  
function pareto_invx(k, a, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the Pareto distribution with minimum

(real) value  $k > 0$  and shape  $a > 0$  and the support interval  $(k, +\infty)$  :

$$\text{PDF: } f(x) = \frac{a}{x} \left(\frac{k}{x}\right)^a$$

$$\text{CDF: } F(x) = 1 - \left(\frac{k}{x}\right)^a = -\text{powm1}(k/x, a)$$

$$\text{ICDF: } F^{-1}(y) = k(1 - y)^{-1/a}$$

### 3.8.16 Poisson distribution

---

```
function poisson_pmf(mu: double; k: double): double;
function poisson_cdf(mu: double; k: longint): double;
function poisson_pmf_x(mu: extended; k: extended): extended;
function poisson_cdf_x(mu: extended; k: longint): extended;
```

---

These functions return PMF and CDF of the Poisson distribution with mean  $\mu \geq 0$ .

$$\text{PMF: } f(k) = \frac{\mu^k}{k!} e^{-\mu} = \text{sfc.igprefix}(1 + k, \mu)$$

$$\text{CDF: } F(k) = e^{-\mu} \sum_{i=0}^k \frac{\mu^i}{i!} = \text{igammaq}(1 + k, \mu)$$

### 3.8.17 t-distribution

---

```
function t_pdf(nu: longint; x: double): double;
function t_cdf(nu: longint; t: double): double;
function t_inv(nu: longint; p: double): double;
function t_pdf_x(nu: longint; x: extended): extended;
function t_cdf_x(nu: longint; t: extended): extended;
function t_inv_x(nu: longint; p: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the Student's t-distribution with  $\nu \in \mathbb{N}$  degrees of freedom and the support interval  $(-\infty, +\infty)$  :

$$\text{PDF: } f(x) = \frac{\Gamma(1 + \frac{1}{2}\nu)}{\sqrt{\nu\pi} \Gamma(\frac{1}{2}\nu)} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{1}{2}(\nu+1)} = \frac{1}{\sqrt{\nu} B(\frac{1}{2}, \frac{1}{2}\nu)} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{1}{2}(\nu+1)}$$

$$\text{CDF: } F(x) = I_z(\frac{1}{2}\nu, \frac{1}{2}\nu) \quad \text{with } z = \frac{x + \sqrt{x^2 + \nu}}{2\sqrt{x^2 + \nu}}$$

The actual implementation of the CDF, which is based on the Cephys [7, stdtrl.c], uses the normalised incomplete beta function  $I_x$  only for  $x < -1.6$  or  $\nu > 20000$ . Otherwise it is computed with the series expansions, c.f. [1, 26.7.3/4]. The ICDF uses `beta_inv` and distinguishes between several cases.

### 3.8.18 Triangular distribution

---

```
function triangular_pdf(a, b, c, x: double): double;
function triangular_cdf(a, b, c, x: double): double;
function triangular_inv(a, b, c, y: double): double;
function triangular_pdf_x(a, b, c, x: extended): extended;
function triangular_cdf_x(a, b, c, x: extended): extended;
function triangular_inv_x(a, b, c, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the triangular distribution on the support interval  $[a, b]$  with finite  $a < b$  and mode  $c$ ,  $a \leq c \leq b$ .

$$\text{PDF: } f(x) = \begin{cases} 0 & x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & a \leq x < c \\ \frac{2}{b-a} & x = c \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \leq b \\ 0 & x > b \end{cases}$$

$$\text{CDF: } F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a \leq x < c \\ \frac{c-a}{b-a} & x = c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c < x \leq b \\ 1 & x > b \end{cases}$$

and with  $t = (c - a)/(b - a)$

$$\text{ICDF: } F^{-1}(y) = \begin{cases} a + \sqrt{(b-a)(c-a)y} & y < t \\ c & y = t \\ b - \sqrt{(b-a)(b-c)(1-y)} & y > t \end{cases}$$

### 3.8.19 Uniform distribution

---

```
function uniform_pdf(a, b, x: double): double;
function uniform_cdf(a, b, x: double): double;
function uniform_inv(a, b, y: double): double;
function uniform_pdfx(a, b, x: extended): extended;
function uniform_cdfx(a, b, x: extended): extended;
function uniform_invx(a, b, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the uniform distribution on the support interval  $[a, b]$  with finite  $a < b$ :

$$\text{PDF: } f(x) = \frac{1}{b-a}$$

$$\text{CDF: } F(x) = \frac{x-a}{b-a}$$

$$\text{ICDF: } F^{-1}(y) = a + y(b-a)$$

### 3.8.20 Weibull distribution

---

```
function weibull_pdf(a, b, x: double): double;
function weibull_cdf(a, b, x: double): double;
function weibull_inv(a, b, y: double): double;
function weibull_pdfx(a, b, x: extended): extended;
function weibull_cdfx(a, b, x: extended): extended;
function weibull_invx(a, b, y: extended): extended;
```

---

These functions return PDF, CDF, and ICDF of the Weibull distribution with shape  $a > 0$ , and scale  $b > 0$  and the support interval  $(0, +\infty)$ .

$$\text{PDF: } f(x) = \frac{a}{x} \left(\frac{x}{b}\right)^a \exp(-(x/b)^a)$$

$$\text{CDF: } F(x) = 1 - \exp(-(x/b)^a) = -\text{expm1}(-(x/b)^a)$$

$$\text{ICDF: } F^{-1}(y) = b \left(\ln \frac{1}{1-y}\right)^{1/a} = b(-\ln 1p(-y))^{1/a}$$

## 3.9 Other special functions

Several other special function which do not fit into the previous categories are implemented in the `sfMisc` unit.

### 3.9.1 Arithmetic-geometric mean

---

```
function agm(x, y: double): double;  
function agmx(x, y: extended): extended;
```

---

These functions return the arithmetic-geometric mean of  $|x|$  and  $|y|$ . With  $a_0 = \max(|x|, |y|)$ ,  $b_0 = \min(|x|, |y|)$  the AGM is calculated using the recurrence formulas

$$\begin{aligned}a_{n+1} &= \frac{1}{2}(a_n + b_n) \\ b_{n+1} &= \sqrt{a_n b_n}\end{aligned}$$

The sequences converge quadratically to a common limit and  $a_n \geq b_n$ . In the function `sfc_agm` the iteration is terminated if  $a_n - b_n \leq \epsilon a_n$ , where  $\epsilon$  is less than `eps_x`<sup>1/2</sup>. The result is  $(a_n + b_n)/2$ .

```
function sfc_agm2(x, y: extended; var s: extended): extended;
```

This function additionally returns

$$s = \sum_{n=1}^{\infty} 2^n (a_n - b_n)^2,$$

which can be used e.g. for elliptic integrals.

### 3.9.2 Bernoulli numbers

---

```
function bernoulli(n: integer): double;  
function bernoullix(n: integer): extended;
```

---

These functions return the Bernoulli numbers  $B_n$ , which are defined by their generating function

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!}, \quad |t| < 2\pi.$$

If  $n < 0$  or if  $n$  is odd, the result is 0, and  $B_1 = -1/2$ . If  $n \leq 120$  the function value is taken from a pre-calculated table. For large  $n$  the asymptotic approximation [30, 24.11.1]

$$(-1)^{n+1} B_{2n} \sim \frac{2(2n)!}{(2\pi)^{2n}},$$

gives an asymptotic recursion formula

$$B_{2n+2} \sim -\frac{(2n+1)(2n+2)}{(2\pi)^2} B_{2n},$$

which is used for computing  $B_n$  for  $120 < n \leq 2312$  from a pre-calculated table of values  $B_{32k+128}$  ( $0 \leq k \leq 68$ ). The average iteration count<sup>44</sup> is 4, and the maximum relative error of 4.5 `eps_x` occurs for  $n = 878$ .

---

<sup>44</sup>Note that the recursion can go in both directions.

### 3.9.3 Debye functions

---

```
function debye(n: integer; x: double): double;  
function debyex(n: integer; x: extended): extended;
```

---

These functions return the Debye functions

$$D_n(x) = \frac{n}{x^n} \int_0^x \frac{t^n}{e^t - 1} dt \quad (n > 0, x \geq 0).$$

The calculation is based on section 27.1 of Abramowitz and Stegun[1]. For  $x \leq 4$  the formula [1, 27.1.1]

$$\int_0^x \frac{t^n}{e^t - 1} dt = x^n \left( \frac{1}{n} - \frac{x}{2(n+1)} + \sum_{k=1}^{\infty} \frac{B_{2k} x^{2k}}{(2k+n)(2k)!} \right)$$

is used, where the  $B_{2k}$  are the Bernoulli numbers 3.9.2. The series converges for  $x < 2\pi$ . If  $x > 4$  the partial sums of [1, 27.1.2]

$$\int_x^{\infty} \frac{t^n}{e^t - 1} dt = \sum_{k=1}^{\infty} e^{-kx} \left( \frac{x^n}{k} + \frac{nx^{n-1}}{k^2} + \frac{n(n-1)x^{n-2}}{k^3} + \dots + \frac{n!}{k^{n+1}} \right)$$

are subtracted from the complete integral [1, 27.1.3]

$$\int_0^{\infty} \frac{t^n}{e^t - 1} dt = n! \zeta(n+1).$$

With increasing  $n$  the cancellation caused by the subtraction is the main source of the decreasing accuracy. The extended version is accurate up to  $n = 6$  only; that's why the common function `sfc_debye` uses a double exponential automatic numerical quadrature algorithm<sup>45</sup> for  $x > 4, n > 6$ .

### 3.9.4 Lambert W functions

---

```
function LambertW(x: double): double;  
function LambertW1(x: double): double;  
function LambertWx(x: extended): extended;  
function LambertW1x(x: extended): extended;
```

---

The multivalued Lambert W function is defined as a solution of

$$W(x) \exp(W(x)) = x.$$

This function has two real branches for  $x < 0$  with a branch point at  $x = -1/e$ . `LambertW(x) =  $W_0(x)$`  is the principal branch with  $W_0(x) \geq -1$  for  $x < 0$ , and `LambertW1(x) =  $W_{-1}(x)$`  is the other real branch with  $W_{-1}(x) \leq -1$  for  $x < 0$ .

If  $x \leq -0.36767578125$  in the common function `sfc_LambertW`, the principal branch  $W_0$  is evaluated as a polynomial in  $z = (x - 1/e)^{1/2}$ , the coefficients are calculated with `MPArith` from the branch point series given by Corless et al.[23, 4.21–4.25].

For other arguments the defining equation is solved with Fritsch iterations (see e.g. Veberic [24] section 2.3). For  $x < 3$  the starting values are obtained by a Padé approximation<sup>46</sup>, and for  $x \geq 3$  from the asymptotic series [23, 4.19] or [24, 40].

---

<sup>45</sup>A customised version of the `AMTools` procedure `intde` based on [38]

<sup>46</sup>Computed with Maple V: `pade(LambertW(x), x, [3,2])`

The  $W_{-1}$  branch in the common function `sfc_LambertW1` is calculated with Halley iterations ([23, 5.9] or [24] section 2.2). The starting values are computed with the branch point series for  $x < -0.275$ , with another Padé approximation<sup>47</sup> for  $x < -0.125$ , and otherwise with the asymptotic series near zero [24, 40].

### 3.9.5 Riemann prime counting function

---

```
function RiemannR(x: double): double;
function RiemannRx(x: extended): extended;
```

---

These functions return the Riemann prime counting function

$$R(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} \operatorname{li}(x^{1/n}), \quad x > 0.$$

The Gram series for  $R$

$$R(x) = 1 + \sum_{n=1}^{\infty} \frac{(\ln x)^n}{nn! \zeta(n+1)}$$

is used in the common function `sfc_ri` if  $x < 10^{19}$ . For larger  $x$  values  $R(x)$  is computed with the li series (only the first term is needed for  $x \geq 10^{40}$ ). `sfc_ri` requires  $x \geq 0.0625$ , because for smaller values there are massive accuracy problems without multi-precision arithmetic.

---

<sup>47</sup>`pade(LambertW(-1, x), x=-0.2, [2, 2])`

# Appendix A

## Licenses

### A.1 Boost

Boost Software License - Version 1.0 - August 17th, 2003:

<http://www.boost.org/>

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### A.2 Cephys

Cephys Mathematical Library V2.8, (June 2000)  
Copyright 1984, 1991, 1998 by Stephen L. Moshier  
Author : Stephen L. Moshier  
Original-site : <http://www.moshier.net/#Cephys>  
Alternate-site: <http://netlib.org/cephys>  
Platform : Any  
Copying-policy: Freely distributable

### A.3 FDLIBM

FDLIBM (Freely Distributable LIBM) is a C math library for machines that support IEEE 754 floating-point arithmetic.

Copyright (C) 1993 by Sun Microsystems, Inc. All rights reserved. Developed at SunSoft, a Sun Microsystems, Inc. business. Permission to use, copy, modify, and distribute this software is freely granted, provided that this notice is preserved.

For a copy of FDLIBM see <http://www.netlib.org/fdlibm/>  
or <http://www.validlab.com/software/>

### A.4 SLATEC

SLATEC Common Mathematical Library, Version 4.1, July 1993. A comprehensive software library containing over 1400 general purpose mathematical and statistical routines written in Fortran 77. The library is in the public domain. Available from <http://www.netlib.org/slatec/>

# Bibliography

- [1] M. Abramowitz, I.A. Stegun. Handbook of Mathematical Functions. New York, 1970, <http://www.math.sfu.ca/~cbm/aands/>
- [2] Intel, IA-32 Architecture Software Developer's Manual Volume 2A: Instruction Set Reference, A-M <http://www.intel.com/products/processor/manuals/>
- [3] D. Goldberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic, 1991; extended and edited reprint via <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6768>
- [4] ISO/IEC 10967-2, Information technology: Language independent arithmetic, Part 2: Elementary numerical functions. [http://standards.iso.org/ittf/PubliclyAvailableStandards/c024427\\_ISO\\_IEC\\_10967-2\\_2001\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c024427_ISO_IEC_10967-2_2001(E).zip)
- [5] FDLIBM 5.3 (Freely Distributable LIBM), developed at Sun Microsystems, see <http://www.netlib.org/fdlibm/> or <http://www.validlab.com/software/fdlibm53.tar.gz>
- [6] K.C. Ng, Argument Reduction for Huge Arguments: Good to the Last Bit, Technical report, SunPro, 1992. Available from <http://www.validlab.com/arg.pdf>
- [7] Cephes Mathematical Library, Version 2.8, <http://www.moshier.net/#Cephes> or <http://www.netlib.org/cephes/>
- [8] T. Ogita, S.M. Rump, and S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput., 26 (2005), pp. 1955-1988. Available as <http://www.ti3.tu-harburg.de/paper/rump/OgRu0i05.pdf>
- [9] N.J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd ed., Philadelphia, 2002. <http://www.maths.manchester.ac.uk/~higham/asna/>
- [10] R. Bulirsch, Numerical Calculation of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7, 78-90, 1965
- [11] R. Bulirsch, Numerical Calculation of Elliptic Integrals and Elliptic Functions, part III. Numerische Mathematik 13, 305-315, 1969
- [12] B.C. Carlson, Computing Elliptic Integrals by Duplication. Numerische Mathematik 33, 1-16, 1979
- [13] W.H. Press et al., Numerical Recipes in C, 2nd ed., Cambridge, 1992. <http://www.nrbook.com/a/bookcpdf.html>

- [14] SLATEC Common Mathematical Library, Version 4.1, July 1993 (general purpose mathematical and statistical routines written in Fortran 77), <http://www.netlib.org/slatec/>
- [15] W. Kahan, On the Cost of Floating-Point Computation Without Extra-Precise Arithmetic, 2004. <http://www.eecs.berkeley.edu/~wkahan/Qdrtcs.pdf>
- [16] G.E. Forsythe, How do you solve a quadratic equation? Stanford University Technical Report no. CS40, 1966. <ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/66/40/CS-TR-66-40.pdf>
- [17] P.H. Sterbenz, Floating-Point Computation. Englewood Cliffs, 1974. Chap.9.3: Carefully written programs/quadratic equation, p.246ff
- [18] W.Y. Sit, Quadratic Programming? 1997. <http://www.mmrc.iss.ac.cn/ascm/ascm03/sample.pdf>
- [19] Boost C++ Libraries, Release 1.42.0, 2010. <http://www.boost.org/>
- [20] Special functions by Wayne Fullerton, <http://www.netlib.org/fn/>. Almost identical to the FNLIB subset of SLATEC [14].
- [21] GNU Scientific Library, GSL-1.14 (March 2010), <http://www.gnu.org/software/gsl/>
- [22] A.J. MacLeod, MISCFUN: A software package to compute uncommon special functions. ACM Trans. on Math. Soft. 22 (1996), pp. 288-301. Fortran source: <http://netlib.org/toms/757>
- [23] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, D.E. Knuth, On the Lambert W Function, Adv. Comput. Math., 5 (1996), pp. 329-359. <http://www.apmaths.uwo.ca/~rcorless/frames/PAPERS/LambertW/LambertW.ps> or <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3583>
- [24] D. Veberic, Having Fun with Lambert W(x) Function, 2010. <http://arxiv.org/abs/1003.1628>
- [25] I. Smith, Examples.xls/txt Version 3.3.4, Personal communication, 2010
- [26] N.M. Temme, A Set of Algorithms for the Incomplete Gamma Functions, Probability in the Engineering and Informational Sciences, 8 (1994), pp. 291-307. Available from <http://repository.cwi.nl/search/fullrecord.php?publnr=10080>
- [27] A.R. Didonato, A.H. Morris, Computation of the Incomplete Gamma Function Ratios and their Inverse. ACM TOMS, Vol 12, No 4, Dec 1986, pp. 377-393. Fortran source: ACM TOMS 13 (1987) pp. 318-319; available from <http://netlib.org/toms/654>
- [28] R.P. Brent, Algorithms for Minimization without Derivatives, Englewood Cliffs, 1973. Scanned copy available from the author's site: <http://maths.anu.edu.au/~brent/pub/pub011.html>
- [29] G.E. Forsythe, M.A. Malcolm, C.B. Moler, Computer Methods for Mathematical Computations, Englewood Cliffs, 1977. Fortran code from <http://www.netlib.org/fmm/>

- [30] [NIST]: F.W.J. Olver, D.W. Lozier, R.F. Boisvert, C.W. Clark, NIST Handbook of Mathematical Functions, Cambridge, 2010. Online resource: NIST Digital Library of Mathematical Functions, <http://dlmf.nist.gov/>
- [31] R.E. Crandall, Note on fast polylogarithm computation, 2006. <http://www.reed.edu/~crandall/papers/Polylog.pdf>
- [32] D.E. Knuth: The Art of computer programming; Volume 1, Fundamental Algorithms, 3rd ed., 1997; Volume 2, Seminumerical Algorithms, 3rd ed., 1998; <http://www-cs-faculty.stanford.edu/~knuth/taocp.html>
- [33] <http://functions.wolfram.com/>: Formulas and graphics about mathematical functions for the mathematical and scientific community. Also used: <http://mathworld.wolfram.com/> ("*the web's most extensive mathematical resource*") and Wolfram Alpha - Computational Knowledge Engine at <http://www.wolframalpha.com/> for online calculation of multi-precision special function reference values.
- [34] R. Piessens, E. de Doncker-Kapenga, C.W. Überhuber, D. Kahaner, QUADPACK: A subroutine package for automatic integration (1983). Public domain Fortran source: <http://www.netlib.org/quadpack/>
- [35] L.C. Maximon, The dilogarithm function for complex argument, 2003, Proc. R. Soc. Lond. A, **459**, 2807-2819, doi:10.1098/rspa.2003.1156 <http://rspa.royalsocietypublishing.org/content/459/2039/2807.full.pdf>
- [36] P. Borwein, An Efficient Algorithm for the Riemann Zeta Function, CMS Conference Proc. 27 (2000), pp. 29-34. Available as <http://www.cecm.sfu.ca/personal/pborwein/PAPERS/P155.pdf>
- [37] A.R. DiDonato, A.H. Morris, Algorithm 708: Significant digit computation of the incomplete beta function ratios, ACM TOMS, Vol. 18, No. 3, 1992, pp. 360-373. Fortran source available from <http://netlib.org/toms/708>
- [38] T. Ooura's Fortran and C source code for automatic quadrature using Double Exponential transformation; available from <http://www.kurims.kyoto-u.ac.jp/~ooura/intde.html>
- [39] R: A Language and Environment for Statistical Computing, Version 2.11.1, <http://www.r-project.org/>
- [40] S.V. Aksenov et al., Application of the combined nonlinear-condensation transformation to problems in statistical analysis and theoretical physics. Computer Physics Communications, 150, 1-20, 2003. Available from [http://dx.doi.org/10.1016/S0010-4655\(02\)00627-6](http://dx.doi.org/10.1016/S0010-4655(02)00627-6) or as e-print: <http://arxiv.org/pdf/math/0207086v1>.  
C source and user guide for lerchphi.c is available from <http://aksenov.freeshell.org/lerchphi.html>.
- [41] C. Ferreira, J.L. López, Asymptotic expansions of the Hurwitz-Lerch zeta function, J. Math. Anal. Appl. 298 (2004), no. 1, 210-224. <http://dx.doi.org/10.1016/j.jmaa.2004.05.040> or from <http://pcmap.unizar.es/~chelo/investig/publicaciones/chelo/Hurwitz-Lerch/original.pdf>

[Temme76a] N.M. Temme, On the Numerical Evaluation of the Ordinary Bessel Function of the Second Kind. J. Comput. Phys., **21**(3): 343-350 (1976), section 2. Available as

<http://repository.cwi.nl/search/fullrecord.php?publnr=10710a>

[Temme76b] N.M. Temme, On the Numerical Evaluation of the Modified Bessel Function of the Third Kind, 2nd edition. Preprint, available from

<http://repository.cwi.nl/search/fullrecord.php?publnr=7885>